INTERNATIONAL STANDARD

ISO 13584-32

First edition
2010-12-15

Industrial automation systems and integration — Parts library —

Part 32: Implementation resources: OntoML: Product ontology markup language

Systèmes d'automatisation industrielle et intégration — Bibliothèque de composants —

Partie 32: Resources d'implémentation: OntoML: Langage de marquage ontologique

Reference number
ISO 13584-32:2010(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Figures

Tables

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

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.

ISO 13584-32 was prepared by Technical Committee ISO/TC 184, *Automation systems and integration*, Subcommittee SC 4, *Industrial data*.

ISO 13584 consists of the following parts under the general title *Industrial automation systems and integration — Parts library:*

— *Part 1: Overview and fundamental principles*

— *Part 20: Logical resource: Logical model of expressions*

— *Part 24: Logical resource: Logical model of supplier library*

— *Part 25: Logical resource: Logical model of supplier library with aggregate values and explicit content*

— *Part 26: Logical resource: Information supplier identification*

— *Part 31: Implementation resources: Geometric programming interface*

— *Part 32: Implementation resources: OntoML: Product ontology markup language*

— *Part 35: Implementation resources: Spreadsheet interface for parts library* [Technical Specification]

— *Part 42: Description methodology: Methodology for structuring parts families*

— *Part 101: Geometrical view exchange protocol by parametric program*

— *Part 102: View exchange protocol by ISO 10303 conforming specification*

— *Part 501: Reference dictionary for measuring instruments — Registration procedure*

— *Part 511: Mechanical systems and components for general use — Reference dictionary for fasteners*

The structure of ISO 13584 is described in ISO 13584-1. The numbering of the parts of ISO 13584 reflects its structure:

— Parts 10 to 19 specify the conceptual descriptions;

— Parts 20 to 29 specify the logical resources;

— Parts 30 to 39 specify the implementation resources;

— Parts 40 to 49 specify the description methodology;

— Parts 100 to 199 specify the view exchange protocols;

— Parts 500 to 599 specify the reference dictionaries.

A complete list of parts of ISO 13584 is available from the following URL:

<http://www.tc184-sc4.org/titles/PLIB_Titles.htm>

Introduction

ISO 13584 is an International Standard for the computer-interpretable representation and exchange of parts library data. The objective is to provide a neutral mechanism capable of transferring parts library data, independent of any application that is using a parts library data system. The nature of this description makes it suitable not only for the exchange of files containing parts, but also as a basis for implementing and sharing databases of parts library data.

ISO 13584 is organized as a series of parts, each published separately. The parts of ISO 13854 fall into one of the following series: conceptual descriptions, logical resources, implementation resources, description methodology, view exchange protocol, and reference dictionaries. The series are described in ISO 13584-1. This part of ISO 13584 is a member of the implementation resources series.

This part of ISO 13584 specifies an XML-based exchange structure for ISO 13584/IEC 61360-compliant data. It provides a set of constructs allowing to represent both an ontology, possibly together with its external resources, and a description of a set of products that reference ontologies and that constitute a library or catalogue. This exchange structure is called OntoML. It is advisable to be familiar with ISO/IEC Guide 77-2 when making use of this part of ISO 13584.

Industrial automation systems and integration — Parts library —

Part 32: Implementation resources: OntoML: Product ontology markup language

1 Scope

This part of ISO 13584 specifies the use of the Extensible Markup Language (XML) and the XML Schema specification for representing data according to the ISO 13584 data model.

The following are within the scope of this part of ISO 13584:

— representation of the common ISO 13584/IEC 61360 model using UML notations;

— definition of two levels of implementation of the common ISO 13584/IEC 61360 model, respectively called the simple level and the advanced level;

— specification of XML markup declarations that enable both simple ontologies and advanced ontologies compliant with the common ISO 13584/IEC 61360 model to be exchanged using XML;

— specification of XML markup declarations that enable the exchange of both ontologies compliant with the common ISO 13584/IEC 61360 model and families of products whose characterizations are defined by means of these ontologies;

NOTE 1 In this part of ISO 13584, such an exchange context is called an OntoML library.

NOTE 2 The information model for exchanging families of products whose characterizations are defined by means of the common ISO 13584/IEC 61360 model compliant ontologies is defined in ISO 13584-25.

— specification of XML global elements allowing to use OntoML as an exchange format for representing responses to queries performed using the ISO/TS 29002-20 concept dictionary resolution mechanism;

— specification of a formal mapping allowing to associate each OntoML elements and attributes of the corresponding entities and attributes of the common ISO 13584/IEC 61360 model EXPRESS data model.

The following are outside the scope of this part of ISO 13584:

— rules used to build OntoML from the common ISO 13584/IEC 61360 model;

— the specification of the program intended to interpret all the mapping operators defined in OntoML for building the corresponding ISO 10303-21 instances of the EXPRESS representation of the common ISO 13584/IEC 61360 model;

— the exchange of individual products whose characterizations are defined by means of ontologies compliant with the common ISO 13584/IEC 61360 model.

NOTE 3 Individual products can be exchanged using the ISO/TS 29002-10 product exchange format.

2 Normative references

The following referenced documents are indispensable for the application 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 10303-11:1994, *Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual*

ISO/IEC 14977, *Information technology — Syntactic metalanguage — Extended BNF*

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

ISO/TS 29002-10:2009, *Industrial automation systems and integration — Exchange of characteristic data — Part 10: Characteristic data exchange format*

Multipurpose Internet Mail Extensions (MIME) Part One*: Format of Internet Message Bodies. Internet Engineering Task Force RFC 2045 November 1996 [cited 2000-08-15].* Available from World Wide Web: <http://www.ietf.org/rfc/rfc2045.txt>

Uniform Resource Identifiers (URI)*: Generic Syntax. Internet Engineering Task Force RFC 2396 August 1998 [cited 2000-08-07].* Available from World Wide Web: http://www.ietf.org/rfc/rfc2396.txt

Extensible Markup Language (XML) 1.0*. Fourth Edition. World Wide Web Consortium Edited Recommendation 14 June 2006.* Available from World Wide Web: <http://www.w3.org/TR/2006/PER-xml-20060614>

XML Schema Part 1*: Structures. Second Edition. World Wide Web Consortium Recommendation 28 October 2004.* Available from World Wide Web: <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>

XML Schema Part 2*: Datatypes. Second Edition. World Wide Web Consortium Recommendation 28 October 2004.* Available from World Wide Web: <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>

XML Path Language (XPath) 1.0*. World Wide Web Consortium Recommendation 16 November 1999.* Available from World Wide Web: <http://www.w3.org/TR/1999/REC-xpath-19991116>

Namespaces in XML 1.0. Second Edition. World Wide Web Consortium Recommendation 14 June 2006. Available from World Wide Web: <http://www.w3.org/TR/2006/PER-xml-names-20060614>

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1
class
abstraction of a set of similar products

NOTE Adapted from ISO 13584-42:2010, definition 3.6.

3.2
class member
product that complies with the abstraction defined by a class

[ISO 13584-42:2010, definition 3.8]

3.3
common ISO 13584/IEC 61360 model
data model for product ontology, using the information modelling language EXPRESS, resulting from a joint effort between ISO/TC 184/SC4/WG2 and IEC SC3D

NOTE 1 Adapted from ISO 13584-42:2010, definition 3.10.

NOTE 2 The previous version of the common ISO 13584/IEC 61360 ontology model is published in both IEC 61360-5 and ISO 13584-25:2004. A new version, in line with this version of OntoML and with ISO 13584-42:2010 is under work.

3.4
CIIM ontology concept
basic unit of knowledge represented in an ontology based on the common ISO 13584/IEC 61360 ontology model, CIIM ontology concepts are information source (supplier), class, property, data type and document

NOTE 1 Each CIIM ontology concept is associated with a global identifier allowing to reference it externally to an exchange file.

NOTE 2 The same CIIM ontology concept can be referenced several times in the same exchange file. Therefore, a referencing mechanism is defined in OntoML.

3.5
EXPRESS attribute
data element for the computer-sensible description of a property, a relation or a class

NOTE An attribute describes only a single detail of a property, of a class or of a relation.

EXAMPLE The name of a property, the code of a class, the measure unit in which values of a property are provided are examples of attributes.

3.6
EXPRESS entity
class of information defined by common properties

[ISO 10303-11:1994, definition 3.2.5]

3.7
global identifier
code providing an unambiguous and universally unique identification of some concepts or objects

NOTE All CIIM ontology concepts are associated with a global identifier.

3.8
is-a relationship
class inclusion relationship associated with inheritance: if A1 is-a A, then each product belonging to A1 belongs to A, and all that is described in the context of A is automatically duplicated in the context of A1

NOTE 1 This mechanism is usually called “inheritance”.

NOTE 2 In the common ISO 13584/IEC 61360 dictionary model, the is-a relationship can only be defined between characterization classes. It is advisable that it define a single hierarchy and it ensures that both visible and applicable properties are inherited.

[ISO 13584-42:2010, definition 3.23]

3.9
is-case-of relationship
property importation mechanism: if A1 is case-of A, then the definition of A products also covers A1 products, thus A1 can import any property from A

NOTE 1 The goal of the case-of relationship is to allow connecting together several class inclusion hierarchies while insuring that referenced hierarchies can be updated independently.

NOTE 2 There is no constraint stating that the case-of relationship is intended to define single hierarchies.

NOTE 3 In the common ISO 13584/IEC 61360 dictionary model, the case-of relationship can be used in particular in four cases: (1) to link a characterization class to a categorization class, (2) to import, in the context of some standardized reference dictionaries, some properties already defined in other standardized reference dictionaries, (3) to connect a user reference dictionary to one or several standardized reference dictionaries, (4) to describe a product using the properties of different classes: when products of class A1 fulfil two different functions, and are thus logically described by properties associated with two different classes, A and B, A1 can be connected by is-a to, for example, A, and by case-of to B.

NOTE 4 Adapted from ISO 13584-42:2010, definition 3.24.

3.10
is-view-of
relationship providing a formal expression of the fact that an object is a representation of another object according to a particular perspective

EXAMPLE A set of geometric entities might provide a drafting representation of a particular screw. If both the set of geometric entities and the particular screw are represented as objects, the is-view-of relationship holds between the former object and the latter object (in a drafting perspective).

[ISO 13584-24:2003, definition 3.64]

3.11
library
representation of a set of products by their product characterizations, possibly associated with the ontology where product characterization classes and properties are defined

NOTE 1 A library is also called a catalogue.

NOTE 2 In the OntoML schemas, to highlight the difference between the ontology part and the content part, the ontology part is embedded in the “dictionary” XML element, and the content part is embedded in the “library” XML element.

3.12
OntoML document instance
XML document that complies with the OntoML XML Schema

3.13
product categorization
part categorization
categorization
recursive partition of a set of products into subsets for a specific purpose

NOTE 1 Subsets which appear in a product categorization are called product categorization classes, or product categories.

NOTE 2 A product categorization is not a product ontology. It cannot be used for characterizing products.

NOTE 3 No property is associated with categorizations.

NOTE 4 Several categorizations of the same set of products are possible according to their target usage.

EXAMPLE The UNSPSC classification, defined by the United Nations, is an example of product categorization that was developed for spend analysis.

NOTE 5 Using the is-case-of relationship, several product characterization class hierarchies can be connected to a categorization hierarchy to generate a single structure.

[ISO 13584-42:2010, definition 3.32]

3.14
product categorization class
part categorization class
categorization class
class of products that constitutes an element of a categorization

EXAMPLE *Manufacturing Components and Supplies and Industrial Optics*, are examples of a product categorization class defined in the UNSPSC.

NOTE 1 No rule is given in this part of ISO 13584 about how to select categorization classes. This concept is introduced (1) to clarify its difference with characterization class, and (2) to explain that the same characterization class can be connected to any number of categorization classes.

NOTE 2 There is no property associated with a categorization class.

[ISO 13584-42:2010, definition 3.33]

3.15
product characterization
part characterization
description of a product by means of a product characterization class to which it belongs and a set of property value pairs

EXAMPLE *Hexagon_head_bolts_ISO_4014* (*Product grades* = A, *thread type* = M, *length* = 50, *Diameter* = 8) is an example of product characterization.

NOTE In the example, *Hexagon_head_bolts_ISO_4014* stands for the identifier of the “Hexagon head bolts” product characterization class defined by ISO 4014. All the names in italics between parentheses stand for the identifier of the bolt properties defined in ISO 4014.

[ISO 13584-42:2010, definition 3.34]

3.16
product characterization class
part characterization class
characterization class
class of products that fulfil the same function and that share common properties

NOTE Product characterization classes can be defined at various levels of details, thus defining a class inclusion hierarchy.

EXAMPLE *Metric threaded bolt/screw* and *hexagon head bolt* are examples of product characterization classes defined in ISO 13584-511. The first characterization class is included in the second one. *Transistor* and *bipolar power transistor* are examples of product characterization classes defined in IEC 61360-4-DB. The second one is included in the first one.

[ISO 13584-42:2010, definition 3.35]

3.17
product ontology
part ontology
ontology
model of product knowledge, done by a formal and consensual representation of the concepts of a product domain in terms of identified characterization classes, class relations and identified properties

NOTE 1 Product ontologies are based on a class-instance model that allows one to recognize and to designate the sets of products, called characterization classes, that have a similar function (e.g. *ball bearing*, *capacitor*), but also to discriminate within a class the various subsets of products, called instances, that are considered identical. It is advisable that the rules defined in ISO 1087-1 be used for formulating designations and definitions of characterization classes. Instances have no definitions. They are designated by the class to which they belong and a set of property-value pairs.

NOTE 2 Ontologies are not concerned with words but with concepts, independent of any particular language.

NOTE 3 “Consensual” means that the conceptualization is agreed upon in some community.

NOTE 4 “Formal” means that the ontology is intended to be machine interpretable. Some level of machine reasoning is logically possible over ontology, e.g. consistency checking or inferencing.

EXAMPLE 1 Consistency checking is a kind of machine reasoning.

NOTE 5 “Identified” means that each ontology characterization class and properties is associated with a globally unique identifier allowing one to reference this concept from any context.

NOTE 6 In OntoML, advanced ontologies are those ontologies that use all the modelling mechanisms defined in the common ISO 13584/IEC 61360 ontology model. OntoML also defines a simple functional subset of this model allowing to define simple ontologies.

NOTE 7 In this part of ISO 13584, each product ontology addressing a particular product domain compliant with the common ISO 13584/IEC 61360 dictionary model is called a reference dictionary for that domain.

EXAMPLE 2 The product ontology defined in IEC 61360 is agreed upon by all member bodies of IEC SC3D. A corporate ontology is agreed upon by experts designated by management on behalf of the company.

NOTE 8 Adapted from ISO 13584-42:2010, definition 3.36.

3.18
property
defined parameter suitable for the description and differentiation of products

NOTE 1 A property describes one aspect of a given object.

NOTE 2 A property is defined by the totality of its associated attributes. The types and number of attributes that describe a property with high accuracy are documented in this part of ISO 13584.

NOTE 3 The term “property” used in this part of ISO 13584 and the term “data element type” used in IEC 61360 are synonyms.

NOTE 4 Adapted from ISO 13584-42:2010, definition 3.37.

3.19
reference dictionary
product ontology compliant with the common ISO 13584/IEC 61360 dictionary model

NOTE In the ISO 13584 standard series, a product ontology that addresses a particular product domain, based on the common ISO 13584/IEC 61360 dictionary model, is called a reference dictionary for that domain.

[ISO 13584-42:2010, definition 3.41]

3.20
XML attribute
XML construct included in an element and defined by a name and a simple value pair

NOTE 1 Adapted from XML 1.0 Recommendation.

NOTE 2 In this part of ISO 13584, the name of an XML attribute will be prefixed by “@” to stipulate that the corresponding piece of information is represented as an attribute and not as an XML embedded element.

3.21
XML complex type
set of XML element and/or attribute declarations describing an XML element content model

3.22
XML element
XML structure including a start tag, an end tag, information between these tags, and, possibly, a set of XML attributes

NOTE 1 Adapted from XML 1.0 Recommendation.

NOTE 2 The information structure between start tag and end tag is defined by either by an XML simple type or an XML complex type.

NOTE 3 An XML element can contain other XML elements defined by either an XML simple type or an XML complex type.

3.23
XML simple type
set of constraints applicable to the value of an XML attribute or to the value of an XML element without any XML child element

NOTE An XML simple type applies to the values of attributes and the text-only content of elements.

4 Abbreviated terms

CIIM Common ISO 13584/IEC 61360 Model

IRDI International Registration Data Identifier

SI Système International d'Unités (International System of Units)

STEP Standard for The Exchange of Product model data

UNSPSC classification of products and services defined by the United Nations

URI Uniform Resource Identifier

URN Uniform Resource Name

XML Extensible Markup Language

5 OntoML implementation levels

The CIIM includes a number of concepts and of modelling mechanisms allowing to characterize not only items, such that products, but also (1) multi point of view discipline specific representations of items, and (2) characterization of the various possible point of views. Such advanced concepts may not be necessary in a number of applications.

Therefore, this part of ISO 13584 identifies a subset of all the modelling mechanisms defined in the CIIM that should prove useful in most application contexts. This subset defines allowed levels of implementation of OntoML. These levels are denoted as “simple” in this part of ISO 13584.

All the modelling mechanisms that do not belong to the simple level are referenced as “advanced” when they are presented. Clause 9.5 summarizes those OntoML constructs that belong to simple levels and those that belong to advanced level.

NOTE 1 Simple levels being defined as a consistent functional subset, reading and understanding advanced mechanisms is not needed to understand and to use the simple levels.

Moreover, OntoML makes it possible to model two kinds of information:

— ontologies,

— libraries, which are set of instances data possibly associated with their ontology definitions.

Depending on the application context, not all those kinds of information may prove useful. Therefore, four subsets of OntoML are defined as allowed levels of implementation:

— simple ontology,

— advanced ontology,

— simple library,

— advanced library.

A simple instance data level does not exist, because this kind of information exchange will be addressed by ISO/TS 29002-10.

NOTE 2 ISO/TS 29002 is developed as a joint effort of several standardization committees to promote interoperability between the various standards that require product characterization.

NOTE 3 Only simple ontology and advanced ontology subsets comply with the CIIM. Representation of libraries complies with extensions of the CIIM defined in ISO 13584-24:2003 and ISO 13584-25:2004.

6 Overview of OntoML ontology representation

In this clause, CIIM ontology concepts are defined and their underlying structure is presented. Additionally, graphical notations used to illustrate every CIIM ontology concept structure are introduced.

6.1 CIIM ontology concepts

According to the CIIM, an ontology consists of five kinds of main concepts:

— supplier,

— class,

— property,

— identified data types,

— document.

Each of these CIIM ontology concepts carries two kinds of information:

— its identification that is a global identifier. This identifier allows to reference this concept from within or outside the OntoML document that defines the concept. The structure of CIIM ontology concept global identifiers is defined in Clause 9.1.

NOTE The OntoML global identifier of a CIIM ontology concept contains the same information as the ones defined in the other parts of ISO 13584, and known as basic semantic unit.

— its definition that consists of a set of pieces of information specified in the CIIM.

6.2 OntoML structure of a CIIM ontology concept

Each CIIM ontology concept definition includes a number of pieces of information and of relations with other CIIM ontology concepts. In the XML representation, each CIIM ontology concept is represented by one XML element. Then, the pieces of information that contribute to the definition of the CIIM ontology concept are represented either externally or internally to its associated XML element, depending on the relationships involved:

— external representation are used to reference any piece of information that represents another CIIM ontology concept, through its own identifier;

— internal representations are used to reference any other piece of information.

Figure 1 illustrates these two kinds of representation.

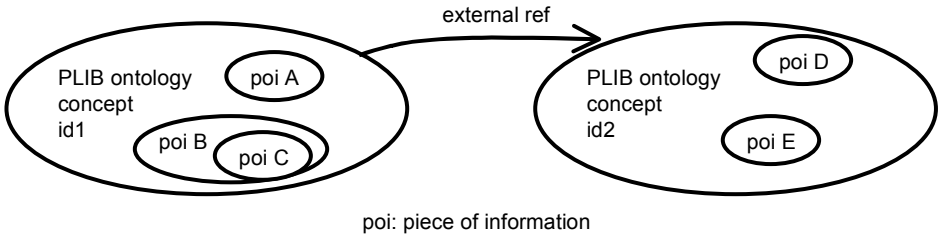

Figure 1 — CIIM ontology concepts description

Assumes that two CIIM ontology concepts are defined. They are both unambiguously identified (*id1* and *id2* identifiers). Additionally, both are defined by several pieces of information (*poi*) that are embedded within the XML representation of the CIIM ontology concepts. In turn, these pieces of information can themselves embed some others pieces of information. Finally, the CIIM ontology concept identified by *id1* references the CIIM ontology concept identified by *id2*.

NOTE Internal representation that consists in embedding the referenced piece of information within the XML element that represents a CIIM ontology concept may result in a duplication of some pieces of information. Anyway, it does not change the semantics of the underlying CIIM EXPRESS data model.

6.3 UML-like graphical representation of OntoML constructs

In this part of ISO 13584, OntoML is described using UML notations. The basic UML notations are enriched in order to:

— highlight the difference between an XML element and an XML attribute;

— explicitly represent references between XML complex type represented in different diagrams;

— represent references(s) between CIIM ontology concepts using the specified identification mechanism;

— simplify the diagram when representing references between CIIM ontology concepts.

These graphical notations are called UML-like notations.

This clause presents the UML-like graphical representation of OntoML constructs. It also describes the mechanism used for representing in OntoML references between CIIM ontology concepts together with its graphical representation.

After presenting the referencing mechanism used to provide for external reference between CIIM ontology concepts, this clause presents the structure of the various CIIM ontology concepts defined in OntoML using UML diagrams.

6.3.1 Graphical notations

In the remaining part of the document, the following conventions will be used to represent the OntoML structure using UML-like diagrams.

6.3.1.1 Representation of an XML complex type

An XML complex type is represented as a box split in two parts: the XML complex type name at the top, the XML attribute(s) and/or the embedded XML element(s) below. It is illustrated in Figure 2.

EXAMPLE 1 In Figure 2, an XML complex type called **PROPERTY_Type** is represented.

Figure 2 — UML-like representation of an XML complex type

If the XML complex type is abstract, its name is represented using italic font style.

A complex type can also be represented as a rounded thin line box: it means that the XML attribute(s) and/or the embedded XML element(s) specifying its content model are defined elsewhere.

EXAMPLE 2 Figure 3 specifies a reference to a **PROPERTY_Type** XML complex type.

Figure 3 — UML-like representation of a reference to an XML complex type

6.3.1.2 Representation of references to external information elements

OntoML is using external XML Schema resources for defining its own content. For that purpose, a graphical notation has been introduced. Because it is a reference to an XML complex type, it is firstly represented as a rounded thin line box. Secondly, because it is an external reference, this box is filled with a light gray colour. It is illustrated in Figure 4 below.

EXAMPLE In Figure 4, a complex type called **Content** is referenced from another XML schema identified by the **cat** prefix.

Figure 4 — UML-like representation of an external reference to an XML complex type

NOTE The prefix is defined according to the XML namespaces mechanism and allows to recognize XML definitions specified in some other external XML Schema vocabularies.

6.3.1.3 Representation of XML attributes and XML elements whose content models are XML simple types

Both XML attributes and XML elements whose contents are an XML simple type embedded within an XML complex type are represented by their name and their type. To distinguish XML attributes and XML (nested) elements, the name of the former is prefixed by the "@" character.

NOTE The '@' character is not part of the attribute name. Therefore, it is not represented in the OntoML XML Schema.

In Figure 5 below, a **PROPERTY_Type** is an abstract XML complex type that embeds a **revision** element whose type is the **REVISION_TYPE_Type** XML simple type, and an **id** XML attribute whose type is the **PropertyId** XML simple type.

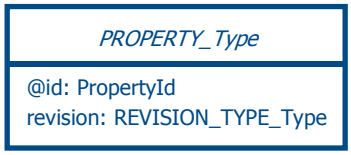

Figure 5 — UML-like representation of XML attributes and simple type XML elements

EXAMPLE Figure 6 below shows the XML document instance corresponding to Figure 5.

```
<xs:complexType name="PROPERTY_Type" abstract="true">
    <xs:sequence>
        <xs:element name="revision" type="REVISION_TYPE_Type"/>
    </xs:sequence>
    <xs:attribute name="id" type="PropertyId" use="required"/>
</xs:complexType>
```

Figure 6 — XML representation of XML attributes and simple type XML elements

6.3.1.4 Representation of an XML element whose content model is an XML complex type

An XML element whose content model is an XML complex type is represented as a relationship between the complex type XML element, and the complex type that embeds the content model of this XML element. This relationship is represented by a filled diamond followed by a plain line whose end is an arrow. The label associated to the relationship represents the XML element name.

NOTE 1 The filled diamond is used to denote a composition relationship.

NOTE 2 The default direct relationship cardinality is exactly one.

EXAMPLE In Figure 7, an XML element called **domain** is defined: it represents an embedded element of the **PROPERTY_Type** XML complex type, and its own content model is the abstract **ANY_TYPE_Type** XML complex type.

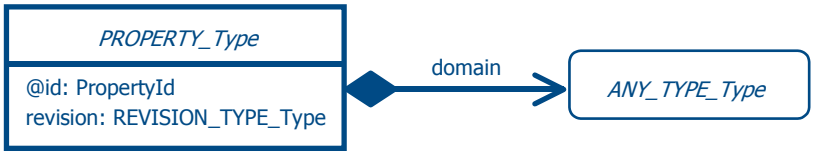

Figure 7 — UML-like representation of an XML complex type XML element

Figure 8 below shows the XML document instance corresponding to Figure 7.

```
<xs:complexType name="PROPERTY_Type" abstract="true">
    <xs:sequence>
        <xs:element name="revision" type="REVISION_TYPE_Type"/>
        <xs:element name="domain" type="ANY_TYPE_Type"/>
    </xs:sequence>
    <xs:attribute name="id" type="PropertyId" use="required"/>
</xs:complexType>
```

Figure 8 — XML representation of an XML complex type element

6.3.1.5 Representation of the cardinality of embedded XML elements

XML elements cardinality is specified using the UML-like notation: *minimum cardinality .. maximum cardinality*.

EXAMPLE 1 In Figure 9, a minimum cardinality equal to 0 and a maximum cardinality equal to 1 are assigned both to the **is_deprecated** and **icon** XML elements.

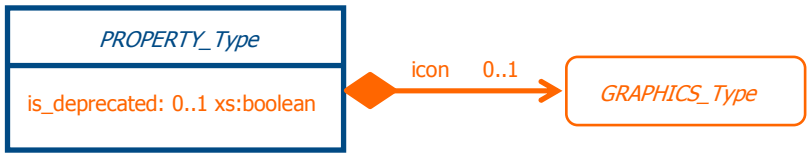

Figure 9 — UML-like representation of XML elements cardinality

NOTE 1 The cardinality relationships expressed in Figure 9 stand for optionality.

NOTE 2 Colour conventions are defined in Clause 6.3.3.

EXAMPLE 2 Figure 10 below shows the XML document instance corresponding to Figure 9.

```
<xs:complexType name="PROPERTY_Type" abstract="true">
    <xs:sequence>
        <xs:element name="is_deprecated" type="xs:boolean" minOccurs="0"/>
        <xs:element name="icon" type="GRAPHICS_Type" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
```

Figure 10 — XML representation of XML elements cardinality

6.3.1.6 Representation of XML complex type extensions

XML complex type extensions are represented using the usual triangle UML inheritance symbol.

EXAMPLE 1 In Figure 11 below, the **NON_DEPENDENT_P_DET_Type** XML complex type is defined as an extension of the **PROPERTY_Type** abstract XML complex type.

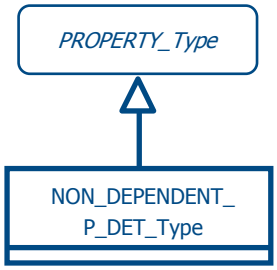

Figure 11 — UML-like representation of XML complex type extensions

EXAMPLE 2 Figure 12 below shows the XML document instance corresponding to Figure 11.

```
<xs:complexType name="NON_DEPENDENT_P_DET_Type">
    <xs:complexContent>
        <xs:extension base="PROPERTY_Type"/>
    </xs:complexContent>
</xs:complexType>
```

Figure 12 — XML representation of XML complex type extensions

6.3.2 Reference mechanism between CIIM ontology concepts

This clause presents the graphical notations that are used to represent the identification of a CIIM ontology concept and its reference from another CIIM ontology concept. Moreover, it introduces graphical notations for representing multi-references from one CIIM ontology concept to a set of other CIIM ontology concepts.

6.3.2.1 Identification of a CIIM ontology concept

The CIIM specifies how to associate a global identifier with any CIIM ontology concept.

In OntoML, for identifying each particular type of CIIM ontology concept, a particular XML complex type is defined:

— the names of these XML complex types reflect the names of their target concept types,

— each of these XML complex types contains an attribute whose value is the global identifier of the particular CIIM ontology concept it identifies,

— the name of this attribute also reflects the name of the target type.

Such elements are intended to be embedded within CIIM ontology concepts.

The name of these XML complex types is given according to the following rule:

— supplier: it is of **SUPPLIER_Type** XML complex type; the type of the identifier is **SupplierId**;

— class: it is of **CLASS_Type** XML complex type; the type of the identifier is **ClassId**;

— property: it is of **PROPERTY_Type** XML complex type; the type of the identifier is **PropertyId**;

— data type: it is of **DATATYPE_Type** XML complex type; the type of the identifier is **DatatypeId**;

— document: it is of **DOCUMENT_Type** XML complex type; the type of the identifier is **DocumentId**.

NOTE The structure of these identifier types are given in Clause 9.1.

The name of this identification attribute is “id”. This name will be represented as **@id** throughout this document to specify that it is an XML attribute and not an XML embedded element.

EXAMPLE In Figure 13, the global identifier of a class ontology concept, represented by the **CLASS_Type** XML complex type, contains a **@id** attribute whose data type is **ClassId**.

Figure 13 — Identification of a CIIM ontology concept

6.3.2.2 OntoML representation of a reference to a CIIM ontology concept

For referencing each particular type of CIIM ontology concept, a particular XML complex type is defined:

— the name of these XML complex types reflect the name of their target data type,

— each reference element contains an attribute whose value is the global identifier of the CIIM ontology concept it references,

— the name of this attribute also reflects the name of the target type.

The XML reference complex type and the XML reference attribute name, prefixed by “@” to stipulate that it is an XML attribute, are defined according to the CIIM ontology concepts referenced:

— **SUPPLIER_REFERENCE_Type** XML complex type and **supplier_ref** XML attribute (whose data type is **SupplierId**): reference to a supplier;

— **CLASS_REFERENCE_Type** XML complex type and **class_ref** XML attribute (whose data type is **ClassId**): reference to a class;

— **PROPERTY_REFERENCE_Type** XML complex type and **property_ref** XML attribute (whose data type is **PropertyId**): reference to a property;

— **DATATYPE_REFERENCE_Type** XML complex type and **datatype_ref** XML attribute (whose data type is **DatatypeId**): reference to a data type;

— **DOCUMENT_REFERENCE_Type** XML complex type and **document_ref** XML attribute (whose data type is **DocumentId**): reference to a document;

NOTE The reference attribute type is defined to match the type of the referenced CIIM ontology concept.

EXAMPLE The reference to a class ontology concept is performed using a **CLASS_REFERENCE_Type** XML complex type and an XML reference attribute called **@class_ref** (whose data type is **ClassId**), as illustrated in Figure 14.

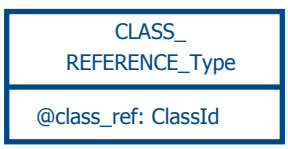

Figure 14 — CIIM ontology concept reference

6.3.2.3 OntoML representation of simple and multi-valued references between CIIM ontology concepts

When the reference from a CIIM ontology concept to another CIIM ontology concept is multi-valued, an additional XML complex type is created as shown in Figure 15.

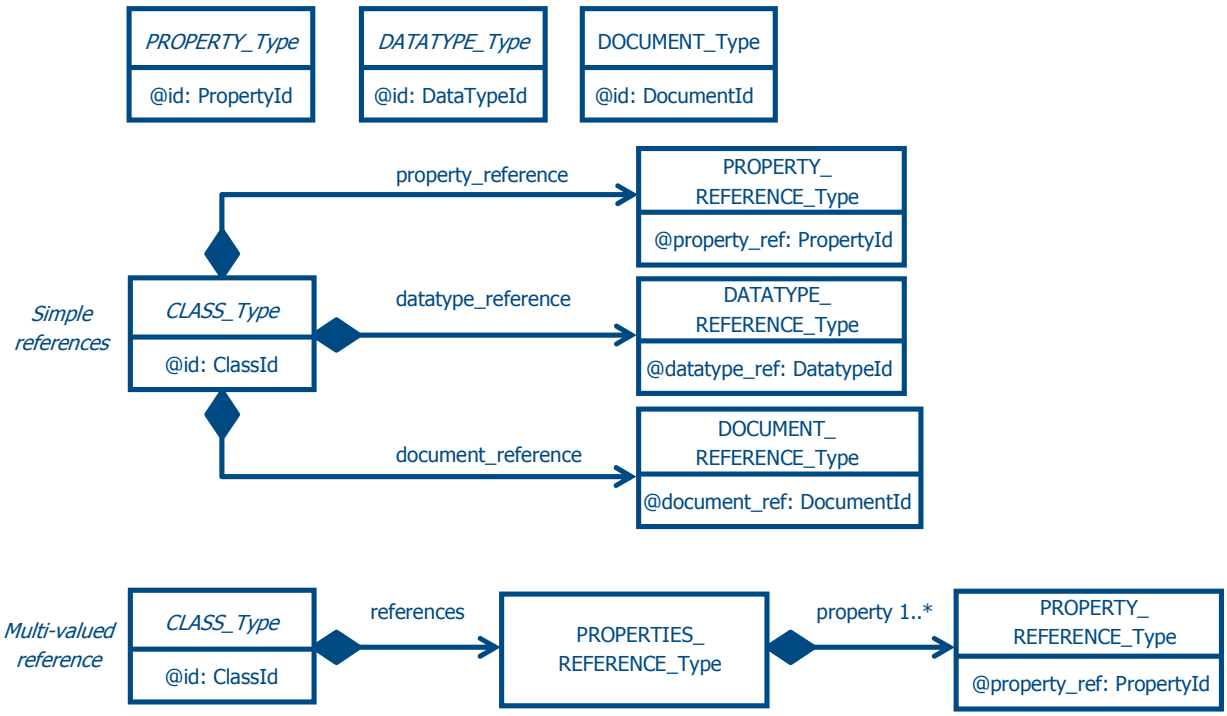

Figure 15 — Reference between CIIM ontology concepts

In this figure:

— a filled diamond is used to represent the composition relationship that denotes the underlying XML nested structure,

— the arrow specifies the corresponding relationship orientation.

Moreover, in this figure, a property, a data type, a document and a class ontology concepts, respectively represented by a **PROPERTY_Type**, **DATATYPE_Type**, **DOCUMENT_Type** and a **CLASS_Type** XML complex types are defined. All are identified by an **id** XML attribute whose type depends on the identified CIIM ontology concept. Two cases of relationships are presented:

— *simple reference*: it is a one to one relationship from a class to a property, a data type or a document. The relationship is represented by an XML element (respectively *property_reference*, *datatype_reference* and *document_reference*) whose content definition is respectively a **PROPERTY_REFERENCE_Type**, **DATATYPE_REFERENCE_Type** and a **DOCUMENT_REFERENCE_Type** XML complex type;

— *multi-valued reference*: it is a one to many relationship from a class to a set of properties. In this case, the relationship is represented by an XML element (*references*) that acts as a container, and its content definition is a **PROPERTIES_REFERENCE_Type** XML complex type.

6.3.2.4 Simplified graphical representation of references between CIIM ontology concepts

As explained in Clause 6.3.2.3, references between CIIM ontology concepts involve a complex chain of one or two composition relationships followed by an identifier / reference matching. To simplify its graphical representation, a particular graphical notation is introduced. This representation is as follows. A reference between CIIM ontology concepts will be represented by a filled diamond followed by a dashed line that joins the referencing XML complex type to the referenced CIIM ontology concept.

The target CIIM ontology concept is represented in a dashed box by its corresponding name, in capital letters, as follows:

— supplier: **SUPPLIER**;

— class: **CLASS**;

— property: **PROPERTY**;

— data type: **DATATYPE**;

— document: **DOCUMENT**.

When different from a one to one relationship, the relationship cardinality is represented like in UML.

EXAMPLE 1 Figure 16 represents a simple reference between two CIIM ontology concepts together with its meaning using the previous notation.

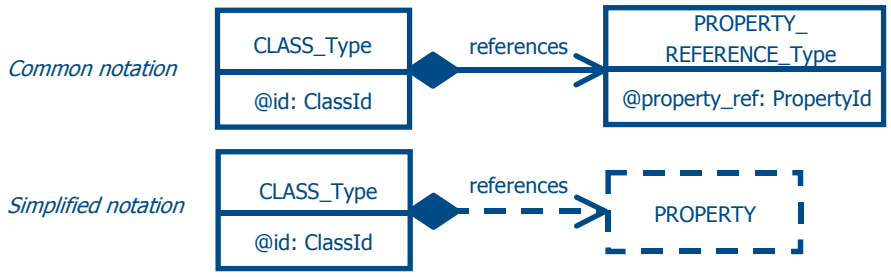

Figure 16 — UML-like representation of a simple reference between CIIM ontology concepts

EXAMPLE 2 Figure 17 below shows the XML document instance corresponding to Figure 16.

```
<xs:complexType name="CLASS_Type">
      <xs:sequence>
            <xs:element name="references" type="PROPERTY_REFERENCE_Type"/>
      </xs:sequence >
</xs:complexType>

<xs:complexType name="PROPERTY_REFERENCE_Type">
      <xs:attribute name="property_ref" type="PropertyId" use="required"/>
</xs:complexType>
```

Figure 17 — XML representation of a simple reference between CIIM ontology concepts

EXAMPLE 3 Figure 18 represents a multi-valued reference between CIIM ontology concepts.

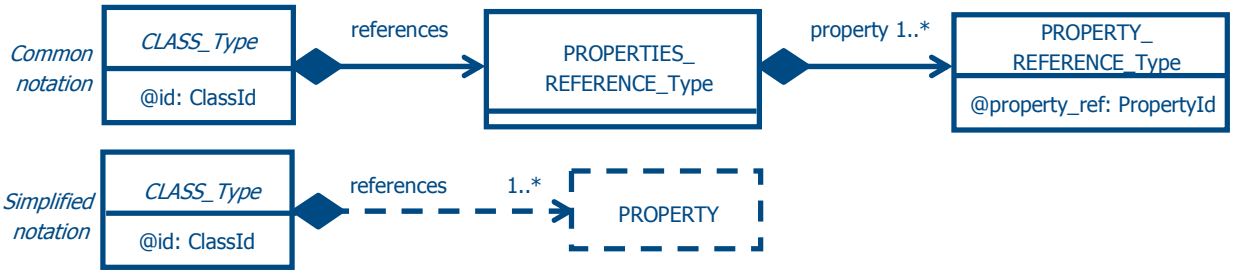

Figure 18 — UML-like representation of a multi-valued reference between CIIM ontology concepts

EXAMPLE 4 Figure 19 below shows the XML document instance corresponding to Figure 18.

```
<xs:complexType name="CLASS_Type">
      <xs:sequence>
            <xs:element name="references" type=" PROPERTIES_REFERENCE_Type"/>
      </xs:sequence >
</xs:complexType>
<xs:complexType name="PROPERTIES_REFERENCE_Type">
      <xs:sequence>
            <xs:element name="property"
                  type="PROPERTY_REFERENCE_Type" maxOccurs="unbounded"/>
      </xs:sequence>
</xs:complexType>
<xs:complexType name="PROPERTY_REFERENCE_Type">
      <xs:attribute name="property_ref" type="PropertyId" use="required"/>
</xs:complexType>
```

Figure 19 — XML representation of a multi-valued reference between CIIM ontology concepts

6.3.3 UML diagrams colour conventions

In UML diagrams, a colour convention is used to highlight those XML attributes and XML elements that are mandatory for the description of any information elements (CIIM ontology concepts or pieces of information). The convention is:

— black line / text when the information element is mandatory,

— gray line / text when the information element is optional.

6.3.4 Description of the structure of all OntoML complex types

In the following, the structure and content of all the OntoML complex type is defined through a set of clauses. Each clause focuses on one particular XML complex type or possibly a small number of related XML complex types not embedded within a single XML type. This XML type or these XML types are called the clause main type or types. The clause main type or types are clearly identified by the name and the header of the clause.

EXAMPLE Clause 6.6 is titled “Root element of an ontology" and the header says “In OntoML, every ontology pieces of information are gathered into a general structure that is a **DICTIONARY_TYPE** XML complex type”. **DICTIONARY_TYPE** is then clause main type.

To make the description more synthetic, the same clause also defines the content and structure of a number of other OntoML complex types that are connected with the clause main type or types, either by inheritance or by composition.

The description of these set of complex type is as follows.

6.3.4.1 Graphical presentation

In each clause, the complete structure of the clause main type or types is defined graphically using the UML-like notations presented above. The clause main type or types can include embedded XML elements whose content models are defined by complex types.

Some of these XML complex types are represented as squared-boxes. This means that the complete structure and content of these complex types are also defined in the current clause. Their structures are defined in the same figure as the one that defines their embedding complex-type.

EXAMPLE 1 In Figure 21, the **HEADER_Type** embeds an **ontoml_information** XML element, whose content model is defined by an **INFORMATION_Type** XML complex type. This type is represented as a squared-box. Thus, its complete structure is defined in Figure 21: it embeds different XML elements: **synonnymous_names**, **preferred_name**, **short_name**, **icon**, **remark** and **note** XML elements.

The content of such an XML complex type is defined under the "Internal type definition" header of the same clause.

EXAMPLE 2 In Clause 6.5 the content of the **INFORMATION_Type** is defined under the "Internal type definition" header as follows: "the list of class descriptions contained in the dictionary".

Some other of these XML complex types are represented as rounded boxes. Their structure and content are defined in another clause of the document, whose number is defined under the "External type definition" header of the current clause.

EXAMPLE 3 In Clause 6.5, the content model of the **icon** embedded XML element is a **GRAPHICS_Type** XML complex type. The corresponding box is rounded. This means that this complex type is defined in another clause. Under the "External type definition" header of Clause 6.5, it is specified that **GRAPHICS_TYPE** is defined in Clause 8.2.2.2.

6.3.4.2 Internal item definition

Under the "Internal item definition" header of each clause, all the XML attributes, and all the XML elements that are embedded within all the XML complex type defined in this clause are defined. Those items, XML attributes or embedded XML elements, that belong to the clause main type are listed by their names. Those items that belong to an embedded XML element whose complex type is represented by a squared box are identified using a path notation starting from the main element. The path separator is slash ('/')

EXAMPLE 1 Under the "Internal item definition" header of Clause 6.6, the definition of the **class** XML element that is embedded within the **contained_classes** element of **DICTIONARY_Type** is associated with the following identifier: **contained_classes/class**.

When the clause addresses several clause main types connected by inheritance relationship, those items that belong to the root of the inheritance hierarchy are not qualified, those items that belong to children classes are qualified by the name of the class between parentheses.

EXAMPLE 2 Clause 5.7.3 defines the simple ontology-level properties that include **PROPERTY_TYPE**, **NON_DEPENDENT_P_DET_Type**, **CONDITION_DET_type** and **DEPENDENT_P_DET_Type**. Under the "Internal item definition" header of this clause, the definition of the **depends_on** XML element that is embedded within the **DEPENDENT_P_DET_Type** subtype of **PROPERTY_Type** is associated with the following identifier: **depends_on** (**DEPENDENT_P_DET_Type**).

6.3.4.3 Internal type definition

Under the "Internal type definition" header of the clause, the content of the following type are defined:

— types of the attributes of all the XML complex types defined in the clause;

— types of all the XML elements that are embedded within one of the XML complex types defined in the clause and whose XML type are simple types;

— types of those XML elements that are embedded within one of the XML complex types defined in the clause and whose XML type are XML complex types represented as squared boxes in the figure.

6.3.4.4 External type definition

Under the "External type definition" header of the clause, each XML complex type represented in the figure as a rounded box is associated with the clause number of the clause where they are defined.

6.3.4.5 Constraint specification

Under the “Constraint specification” header of the clause, additional constraints that can not be represented using the UML conventions are listed.

6.4 OntoML general structure

An OntoML compliant XML document instance allows to represent data describing an ontology, instances, or both. The upper level of a OntoML document instance is defined through the **ONTOML_Type** XML complex type, as is illustrated in Figure 20.

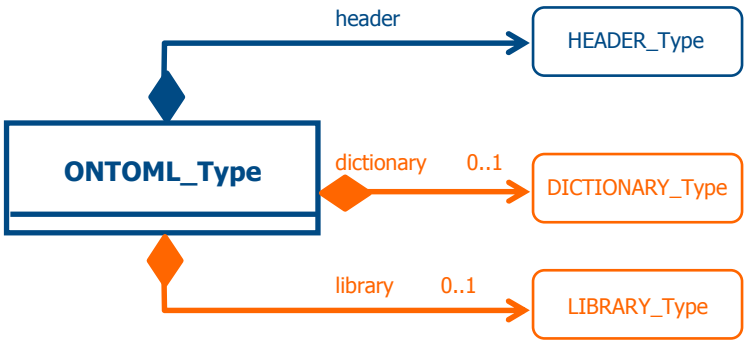

Figure 20 — Ontology structure UML diagram

Internal item definitions:

dictionary: the CIIM ontology concepts that constitute the exchanged ontology.

header: general information about the file that is exchanged.

library: the set of product descriptions that constitute the content of the exchanged library.

External type definitions:

DICTIONARY_Type: the specification of the OntoML ontology, see 6.6.

HEADER_Type: the specification of the OntoML XML document header, see 6.5.

LIBRARY_Type: the specification of the OntoML library, see 7.

Constraint specification:

Either a dictionary XML element exists, or a library XML element exists or both exist.

6.5 OntoML header

The OntoML header provides the version of this part of ISO 13584 used to create the OntoML document instance and the human readable information about this document instance. Additionally, it gives information about the general structure of the OntoML document instance. It is represented by the **HEADER_Type** XML complex type as illustrated in Figure 21.

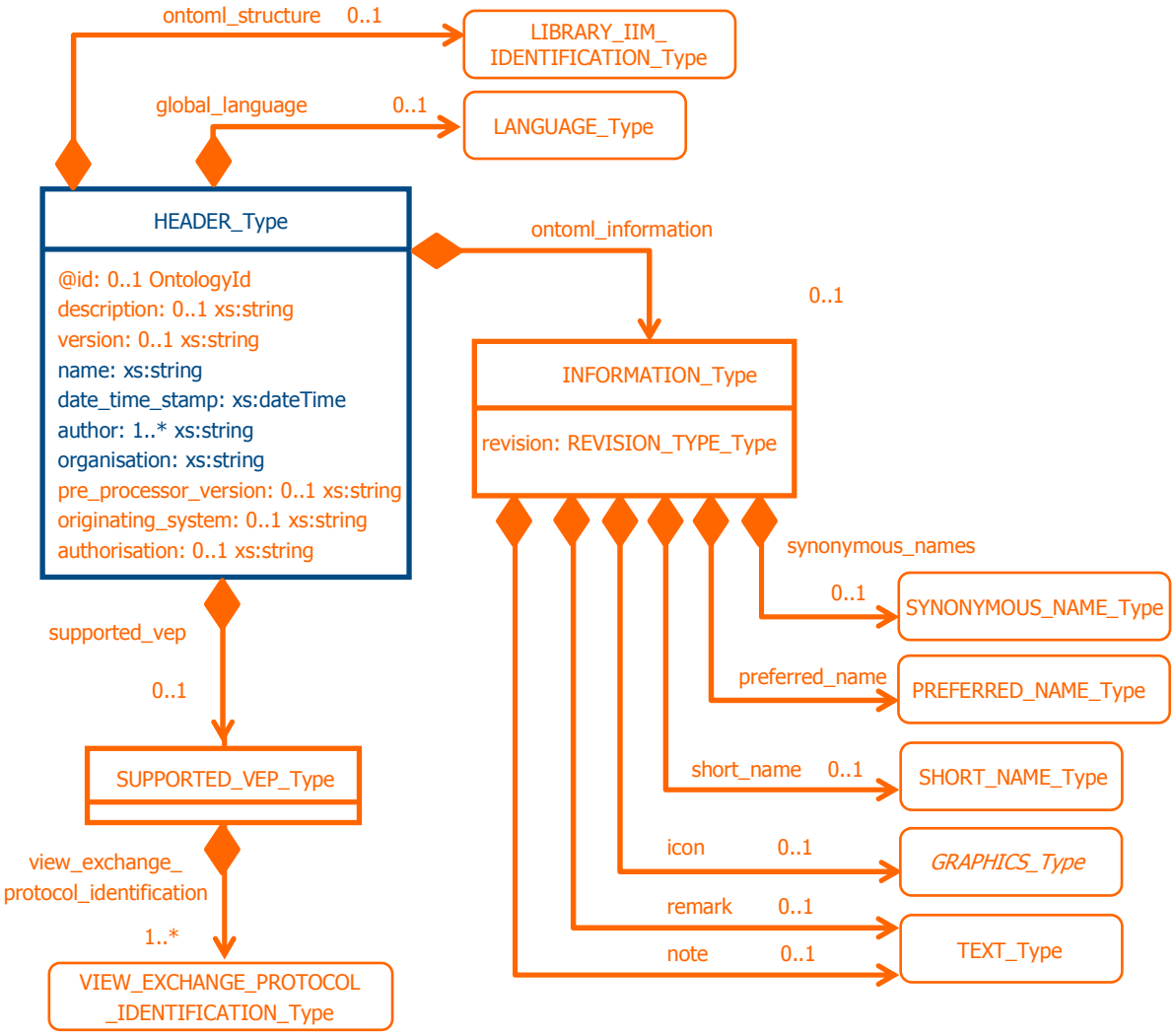

Figure 21 — Ontology header structure

Internal item definitions:

@id: the possible identifier of the dictionary to which the classes defined belong to.

author: the name and mailing address of the persons responsible for creating the exchange structure.

authorisation: the name and mailing address of the person who authorized the sending of the exchange structure.

date_time_stamp: the date and time specifying when the exchange structure was created.

description: an informal description of the content of the OntoML document instance.

global_language: the possible global language used to describe non translated information associated to any CIIM ontology concept.

name: the string used to name this particular OntoML document instance.

ontoml_information/icon: an optional graphics which represents the description associated with the different names provided for describing the OntoML dictionary and / or library.

ontoml_information/note: further information on any part of the dictionary and / or library, which is essential to the understanding.

ontoml_information/preferred_name: the name of the dictionary and / or library that is preferred for use.

ontoml_information/remark: explanatory text further clarifying the meaning of this dictionary and / or library.

ontoml_information/revision: the dictionary and / or library revision number.

ontoml_information/short_name: the abbreviation of the preferred name.

ontoml_information/synonymous_names: the set of synonymous names.

ontoml_structure: the library integrated information model that the OntoML dictionary and / or library realizes.

organisation: the group or organisation which is responsible for the ontology exchange structure / ontology document instance.

originating_system: the system from which the data in this exchange structure originated.

pre_processor_version: the system used to create the exchange structure, including the system product name and version.

revision: the revision of the OntoML schema to which the exchange structure conforms.

supported_vep: the list of view exchange protocols supported by the dictionary and / or library.

supported_vep/view_exchange_protocol_identification: a view exchange protocol supported by the dictionary and / or library.

version: the version of the OntoML schema to which the exchange structure conforms.

Internal type definitions:

INFORMATION_Type: clear text information, possibly translated, of the delivered dictionary and / or library.

REVISION_TYPE_Type: a string (**xs:string** XML Schema data type) that represents the values allowed for a revision. Its value length shall not exceed 3 characters.

SUPPORTED_VEP_Type: the specification of the view exchange protocols supported by the dictionary and / or library.

External type definitions:

OntologyId: see 9.1.

GRAPHICS_Type: see 8.2.2.2.

LANGUAGE_Type: see 8.1.1.

LIBRARY_IIM_IDENTIFICATION_Type: see 8.7.1.

PREFERRED_NAME_Type: see 8.1.2.

SHORT_NAME_Type: see 8.1.2.

SYNONYMOUS_NAME_Type: see 8.1.2.

TEXT_Type: see 8.1.2.

VIEW_EXCHANGE_PROTOCOL_IDENTIFICATION_Type: see 8.7.2.

6.6 Root element of an ontology

In OntoML, every ontology pieces of information are gathered into a general structure that is a **DICTIONARY_TYPE** XML complex type. It is illustrated in Figure 22.

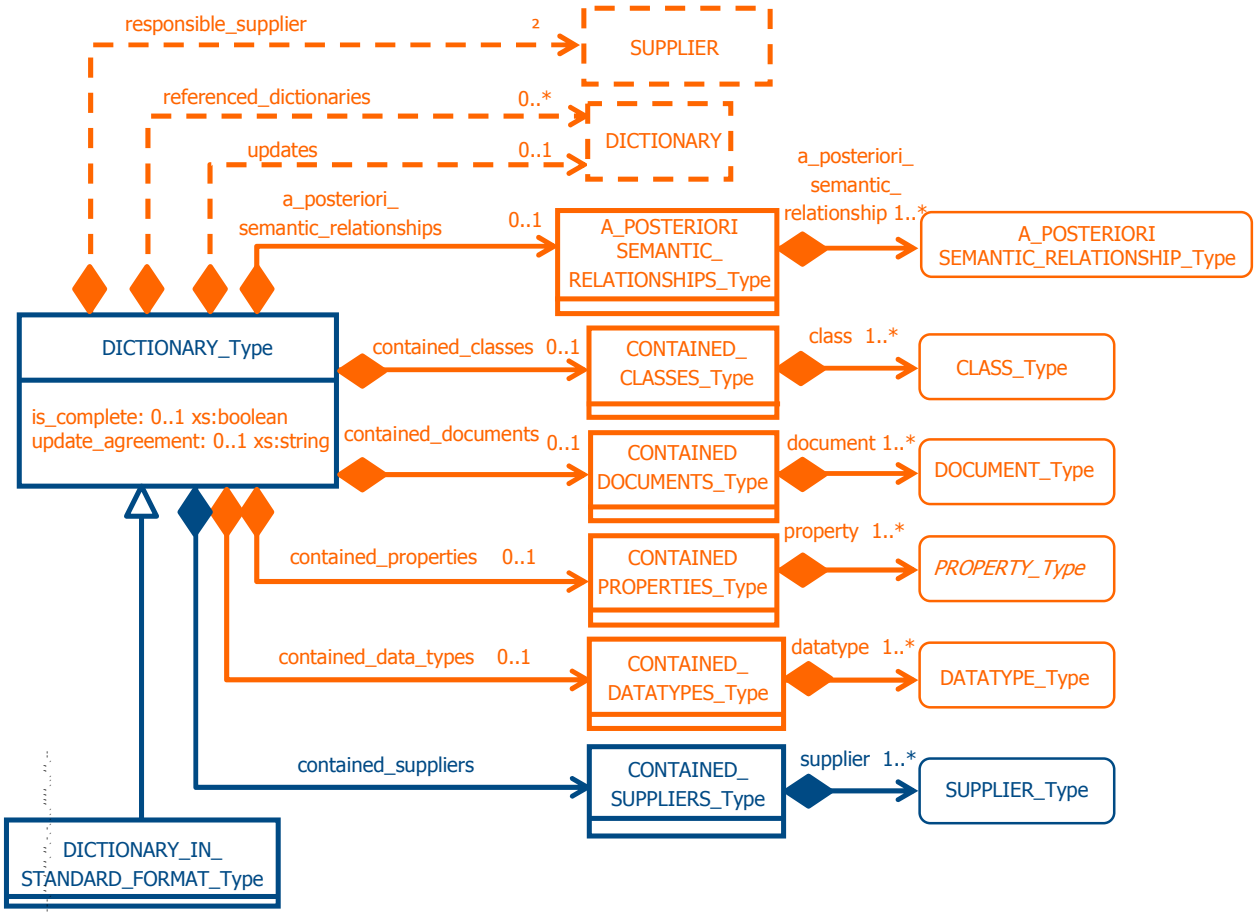

Figure 22 — Root element of an ontology

Internal item definitions:

a_posteriori_semantic_relationships: the list of *a posteriori* relationships contained in the dictionary.

a_posteriori_semantic_relationships/a_posteriori_semantic_relationship: an *a posteriori* relationship contained in the dictionary.

contained_classes: the list of class descriptions contained in the dictionary.

contained_classes/class: a class description contained in the dictionary.

contained_datatypes: the list of data type descriptions contained in the dictionary.

contained_datatypes/datatype: a data type description contained in the dictionary.

contained_documents: the list of document descriptions contained in the dictionary.

contained_documents/document: a document description contained in the dictionary.

contained_properties: the list of property descriptions contained in the dictionary.

contained_properties/property: a property description contained in the dictionary.

contained_suppliers: the list of supplier descriptions contained in the dictionary.

contained_suppliers/supplier: a supplier description contained in the dictionary.

is_complete: specifies whether the dictionary describes completely the exchanged ontology or only its changes.

NOTE 1 The **is_complete** XML element is only used when the dictionary is identified through its **@id** XML attribute.

referenced_dictionaries: the dictionary identifiers, if any, referencing the other dictionaries for which some classes are referenced in this dictionary.

responsible_supplier: the possible data supplier responsible for the ontology concepts.

NOTE 2 The supplier of all or part of the dictionary content is referenced as the **responsible_supplier** only when he/she is the responsible of the OntoML document instance. Else, he/she is referenced in the **contained_supplier** XML element.

update_agreement: the identifier, if any, that identifies the process to be used for creating the dictionary on the receiving system from the list of dictionary defined in the **updates** XML element. The **update_agreement** may only be used when the **updates** XML element is itself used.

updates: dictionary identification, if any, of the dictionary that is supposed to be already available on the receiving system to be able to create the complete content of this dictionary.

NOTE 3 The **updates** XML element can only exist when the **identified_by** XML element exists, and when the **is_compelete** XML element is valued to false.

Internal type definitions:

CONTAINED_CLASSES_Type: sequence of class descriptions.

CONTAINED_DATATYPES_Type: sequence of data type descriptions.

CONTAINED_DOCUMENTS_Type: sequence of document descriptions.

CONTAINED_PROPERTIES_Type: sequence of property descriptions.

CONTAINED_SUPPLIERS_Type: sequence of supplier descriptions.

DICTIONARY_IN_STANDARD_FORMAT_Type: a dictionary that only uses external file protocols that are allowed either by the library integrated information model indicated by the **library_structure** XML element or the view exchange protocols referenced in the **supported_vep** XML element, both defined in the **HEADER_Type** XML complex type.

External type definitions:

A_POSTERIORI_SEMANTIC_RELATIONSHIP_Type: see 8.6.

CLASS_Type: dictionary class description, see 6.7.2.

DATATYPE_Type: dictionary datatype description, see 6.7.6.

DOCUMENT_Type: dictionary document description, see 6.7.7.

PROPERTY_Type: dictionary property description, see 6.7.4.

SUPPLIER_Type: supplier description, see 6.7.1.

Constraint specifications:

When the header part of the OntoML exchange file provides a product ontology identifier (**id** XML attribute of the **HEADER_Type** XML complex type), the **is_complete** XML element information must be provided.

When the header part of the OntoML exchange file provides a product ontology identifier (**id** XML attribute of the **HEADER_Type** XML complex type), the referenced **responsible_supplier** shall be the same than the supplier identified in the ontology identifier.

Updates shall not exist when the product ontology is not identified (**id** XML attribute of the **HEADER_Type** XML complex type), or when the **is_complete** XML element is set to true.

If both the product ontology is identified (**id** XML attribute of the **HEADER_Type** XML complex type) and **updates** has been defined, the identified product ontology shall have the same code and the same supplier as the one referenced by the **updates** XML element, and it shall have a version greater than the one that appears in the **updates** referenced dictionary.

6.7 OntoML representation of CIIM ontology concepts

In this clause, the OntoML representation of the various CIIM ontology concepts is defined.

6.7.1 Supplier

The supplier ontology concept stands for the description of an organization responsible for some information identified in an OntoML document instance. It is represented as illustrated in the UML diagram of Figure 23.

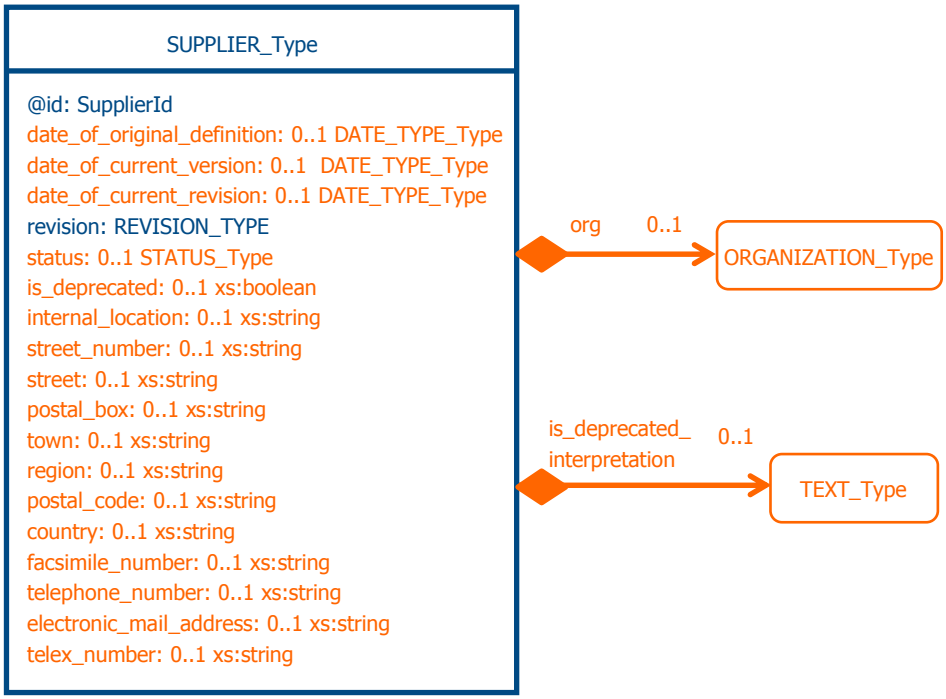

Figure 23 — Supplier ontology concept UML diagram

Internal item definitions:

@id: the supplier identifier.

country: the name of a country.

date_of_original_definition: the date associated to the first stable version of the supplier definition.

date_of_current_version: the date associated to the present version of the supplier definition.

date_of_current_revision: the date associated to the present revision of the supplier definition.

electronic_mail_address: the electronic address at which electronic mail can be received.

facsimile_number: the number at which facsimiles can be received.

internal_location: organization-defined address for internal mail delivery.

is_deprecated: a Boolean that specifies, when true, that the supplier definition shall no longer be used.

is_deprecated_interpretation: specifies the deprecation rationale and how instance values of the deprecated supplier, and of its corresponding identifier, should be interpreted.

org: organizational data of this supplier.

postal_box: the number of a postal box.

postal_code: the code that is used by the country's postal service.

region: the name of a region.

revision: the revision number of the present supplier definition.

status: defines the life cycle state of the supplier definition.

NOTE 1 Allowed **status** values are defined by private agreement between the dictionary supplier and dictionary users.

NOTE 2 If the **status** XML element is not provided, and if this supplier definition is not deprecated as denoted by a possible **is_deprecated** XML element, then the supplier definition has the same standardization status as the whole ontology into which it is used. In particular, if the ontology is standardized, this supplier definition is part of the current edition of the standard.

street: the name of a street.

street_number: the number of a building in a street.

telephone_number: the number at which telephone calls can be received.

telex_number: the number at which telex messages can be received.

town: the name of a town.

Internal type definitions:

DATE_TYPE_Type: identifies the values allowed for a date (the specific **xs:date** XML Schema datatype).

REVISION_TYPE_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a revision. Its value length shall not exceed 3 characters.

STATUS_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a status. This string shall not contain any hyphen « - » or space characters.

External type definitions:

SupplierId: see 9.1.

ORGANIZATION_Type: see 8.8.1.

Constraint specification:

At least **internal_location**, **street_number**, **street**, **postal_box**, **town**, **region**, **postal_code**, **country**, **facsimile_number**, **telephone_number**, **electronic_mail_address** or **telex_number** shall have a value.

When **is_deprecated** exists, **is_deprecated_interpretation** shall exist.

Instance values of i**s_deprecated_interpretation** element shall be defined at the time where deprecation decision was taken.

6.7.2 Simple-level ontology class

OntoML defines three subtypes of the generic and abstract concept of class as simple classes:

— *item class*: allows to characterize any kind of items, and in particular products, by a class belonging and a set of property value pairs. Item classes belong to a single *is-a* hierarchy associated with inheritance.

— *categorization class*: allows to classify an item characterized as an item class in various classification systems. Such a classification does not imply any additional properties.

— *item class case-of*: a special kind of item class that, besides inheriting properties from its possible is-a parent, borrows some properties from some other existing classes that encompass the item class case-of within their own scope.

NOTE Product characterization class and categorization class are defined in Clause 5 of ISO/IEC Guide 77-2:2008.

6.7.2.1 Item class

Figure 24 describes both the structure of a class and of an item class.

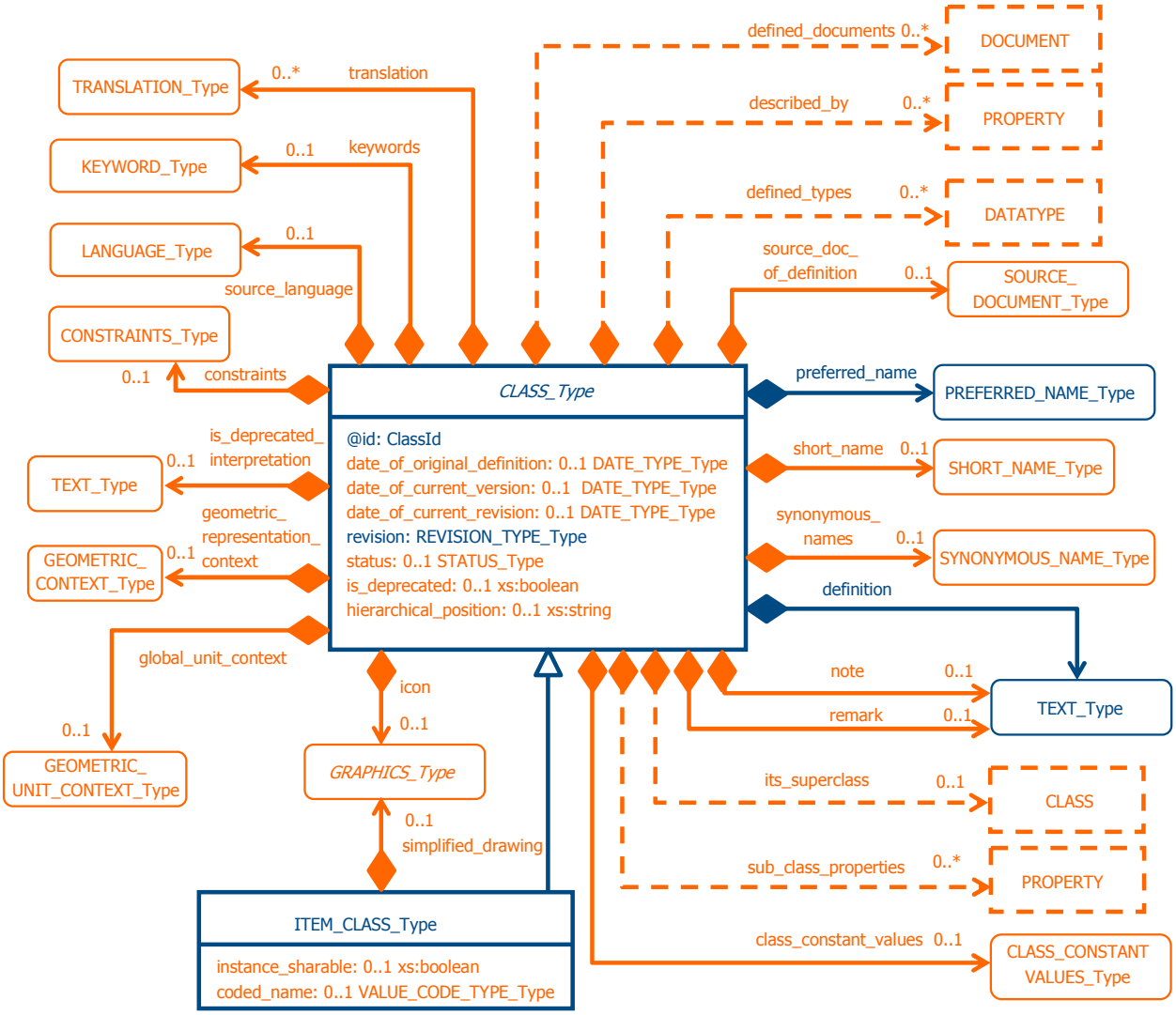

Figure 24 — Simple class ontology concept UML diagram

An item class (represented by the **ITEM_CLASS_Type** XML complex type) inherits the XML content description defined in the abstract **CLASS_Type** XML complex type.

NOTE 1 The most basic representation of classes only requires to define an identifier (**@id**), a **revision** number, a **preferred_name** and a **definition**.

Internal item definitions:

@id: the class identifier.

class_constant_values: assignments in the current class for class-valued properties declared in superclasses.

NOTE 2 **class_constant_values** defines class selectors, as specified in Clause 5.5 of ISO/IEC Guide 77-2:2008.

coded_name (**ITEM_CLASS_Type**): a possible coded name of the class.

constraints: the set of constraints that restrict the target domains of values of some visible properties of the class to some subsets of their inherited domains of values.

NOTE 3 Each constraint in the **constraints** collection must be fulfilled by class instances. Thus the **constraints** collection is a conjunction of constraints.

date_of_current_revision: the date associated to the present revision of the class definition.

date_of_current_version: the date associated to the present version of the class definition.

date_of_original_definition: the date associated to the first stable version of the class definition.

definition: the text describing this class, possibly translated.

defined_documents: the set of references to the additional documents that can be used throughout the inheritance tree descending from this class.

NOTE 4 Every document referenced in the **defined_documents** collection is said applicable to the class.

defined_types: the set of references to the additional types that can be used for various properties throughout the inheritance tree descending from this class.

NOTE 5 Every data type referenced in the **defined_types** collection is said applicable to the class.

described_by: the list of references to the additional properties available for use in the description of the instances within this class, and any of its subclasses.

NOTE 6 Every property referenced in the **described_by** collection is said applicable to the class.

NOTE 7 A property may also be applicable to a class when this property is imported from another class through an **ITEM_CLASS_CASE_OF_Type** class (see 6.7.2.3), a **FUNCTIONAL_MODEL_CLASS_Type** (see 6.7.3.2) or a **FM_CLASS_VIEW_OF_Type** (see 6.7.3.3). Therefore the properties referenced by the **described_by** attribute do not define all the applicable properties for a class.

NOTE 8 The list order is the presentation order of the properties suggested by the supplier.

geometric_representation_context: the specification of the reference coordinate system for every property of the class whose datatype is a STEP positioning entity, i.e., either a **PLACEMENT_TYPE_Type**, an **AXIS1_PLACEMENT_TYPE_Type**, an **AXIS2_PLACEMENT_2D_TYPE_Type** or an **AXIS2_PLACEMENT_3D_TYPE_Type**.

NOTE 9 STEP positioning entities are defined in Annex E, Clause E1.

NOTE 10 The positioning of the reference coordinate system with respect to the object defined by the class is described informally in the **description** element of the **geometric_representation_context**.

EXAMPLE 1 Consider an item class that describes cupboards whose top faces are rectangular. The supplier wants to define by **placement**s the eight vertex of the shipping box for each cupboard. The geometric representation context, allowing then to define the eight vertex, might be defined by the following description: "the origin of the reference coordinate system is the intersection of the two diagonal lines of the cupboard top face, z axis moves upward, x axis is horizontal in the front direction of the cupboard".

NOTE 11 The **geometric_representation_context** is the OntoML representation of the **geometric_representation_context** defined in ISO 10303-42 for geometric representations. In ISO 10303-42, this context applies to all geometric representation items referenced by the representation. In OntoML, this context applies to all STEP positioning entities that are values of properties of the class where the **geometric_representation_context** is defined.

global_unit_context: the specification of the length unit, and possibly angle unit, that are assigned to the geometric representation context of all the STEP positioning entities of the class.

NOTE 12 The **global_unit_context** is the OntoML representation of the **global_unit_assigned_context** defined in ISO 10303-42 for geometric representations. In ISO 10303-42, this context applies to all geometric representation items referenced by the representation. In OntoML, this context applies to all STEP positioning entities that are values of properties of the class where the **global_unit_context** is defined.

NOTE 13 If **global_unit_context** is not provided, the default value for length measure is millimetre and for planar angle measure it is degree.

hierarchical_position: the coded representation of the class position in a class inclusion hierarchy to which it belongs

NOTE 14 This kind of coded name is used in particular in product categorization hierarchies for representing the class inclusion structure through some coding convention.

EXAMPLE 2 In UNSPSC, Manufacturing Components and Supplies has the hierarchical position 31000000, Hardware has the hierarchical position 31160000 and Bolt the hierarchical position 31161600. By convention, this representation of the hierarchical position allows to represent that Manufacturing Components and Supplies is at the first level of the hierarchy, that Hardware is at the second level of the hierarchy and is included in Manufacturing Components and Supplies, and that Bolt is at the third level of the hierarchy and is included in Hardware.

NOTE 15 A **hierarchical_position** of a class changes when the class structure of an ontology is changed. Thus it cannot be used as a stable identifier for classes.

icon: a graphics representing the description associated with the names.

instance_sharable (**ITEM_CLASS_Type**): when false, it specifies that instances of the item class are features; when not provided or true it specifies that instances of the item class are stand-alone items.

NOTE 16 In the common ISO13584/IEC61360 dictionary model, it is implementation dependent to decide whether several real world items modelled by the same set of property-values pairs should be represented in the data exchange file by several XML item description constructs or by the same XML item description construct. Thus, a single XML item description construct whose **instance_sharable** equals false and that is referenced by several XML item description constructs at the data model level is interpreted as representing several real world items.

is_deprecated: a Boolean that specifies, when true, that the class definition shall no longer be used.

is_deprecated_interpretation: specifies the deprecation rationale and how instance values of the deprecated class, and of its corresponding identifier, should be interpreted.

its_superclass: reference to the class the current one is a subclass of.

keywords: a set of keywords, possibly in several languages, allowing to retrieve the class**.**

note: further information on any part of the class, which is essential to its understanding, possibly translated.

preferred_name: the name of the class that is preferred for use, possibly translated.

remark: explanatory text further clarifying the meaning of this class, possibly translated.

revision: the revision number of the present class definition.

short_name: the abbreviation of the preferred name, possibly translated.

simplified_drawing: drawing that can be associated to the described class.

source_doc_of_definition: the possible source document from which the definition comes.

source_language: the language in which the class definition was initially defined and that provides the reference meaning in case of translation discrepancy.

status: defines the life cycle state of the class definition.

NOTE 17 Allowed **status** values are defined by private agreement between the dictionary supplier and dictionary users.

NOTE 18 If the **status** XML element is not provided, and if this class definition is not deprecated as denoted by a possible **is_deprecated** XML element, then the class definition has the same standardization status as the whole ontology into which it is used. In particular, if the ontology is standardized, this class definition is part of the current edition of the standard.

sub_class_properties: declares properties as class-valued, i.e. in subclasses one single value will be assigned per class.

NOTE 19 **sub_class_properties** defines class selectors, as specified in Clause 5.5 of ISO/IEC Guide 77-2:2008.

synonymous_names: the set of synonymous names of the preferred name, possibly translated.

translation: the possible set of translations information provided for the translatable items.

Internal type definitions:

DATE_TYPE_Type: identifies the values allowed for a date (the specific **xs:date** XML Schema datatype).

REVISION_TYPE_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a revision. Its value length shall not exceed 3 characters.

STATUS_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a status.

VALUE_CODE_TYPE_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a value code. Its value length shall not exceed 35 characters.

External type definitions:

ClassId: see 9.1.

ConstraintId: see 9.1.

CLASS_CONSTANT_VALUES_Type: see 6.7.2.4.

CONSTRAINT_Type: see 8.5.

GEOMETRIC_CONTEXT: see 8.8.3.

GEOMETRIC_UNIT_CONTEXT: see 8.8.4.

GRAPHICS_Type: see 8.2.2.2.

KEYWORD_Type: see 8.1.2.

LANGUAGE_Type: see 8.1.1.

PREFERRED_NAME_Type: see 8.1.2.

SHORT_NAME_Type: see 8.1.2.

SOURCE_DOCUMENT_Type: see 8.2.2.1.

SYNONYMOUS_NAME_Type: see 8.1.2.

TEXT_Type: see 8.1.2.

TRANSLATION_Type: see 8.1.3.

Constraint specifications:

The inheritance structure defined by the class hierarchy (defined through the **its_superclass** XML element inherited from the **CLASS_Type** XML complex type) shall not contain cycles.

Only those properties that are visible for a class may become applicable to this class by virtue of being referenced in the list specified by the **described_by** collection.

Only those data types that are visible for a class may become applicable to this class by virtue of being referenced in the list specified by the **defined_types** collection.

Only those properties that are not applicable for a class by inheritance may become applicable to this class by virtue of being referenced in the list specified by the **described_by** collection.

Only those data types that are not applicable for a class by inheritance may become applicable to this class by virtue of being referenced in the list by specified the **defined_types** collection.

Only context dependent properties (**DEPENDENT_P_DET_Type**, see 6.7.4) whose all context parameters (**CONDITION_DET_Type**, see 6.7.4) are applicable in the class may become applicable for this class by virtue of being referenced by its **described_by** XML element.

The **constraints** collection shall define restrictions that are compatible with the domain of values of the properties to which they apply.

All the properties referenced in the **precondition** collection of a constraint for which base type is a **CONFIGURATION_CONTROL_CONSTRAINT_Type** shall be applicable to the class.

All the properties referenced in the **constraints** collection shall be either visible or applicable to the class.

The properties referenced in the **sub_class_properties** collection shall also be referenced in the **described_by** collection.

The properties referenced in the **class_constant_values** collection were declared as class-valued in some superclass of the current class, or in the current class itself.

If a property referenced in the **class_constant_values** collection was already assigned a value in a superclass, the value assigned in the current class should be the same.

When **is_deprecated** exists, **is_deprecated_interpretation** shall exist.

Instance values of i**s_deprecated_interpretation** element shall be defined at the time where deprecation decision was taken.

If **PLACEMENT_TYPE_Type** or **AXIS1_PLACEMENT_TYPE_Type** or **AXIS2_PLACEMENT_2D_TYPE_Type** or **AXIS2_PLACEMENT_3D_TYPE_Type** is the datatype assigned to a property referenced in the **described_by** collection, the corresponding **geometric_representation_context** XML element shall be provided.

6.7.2.2 Categorization class

A categorization class enables the modelling of a grouping of a set of objects that constitutes an element of a categorization.

EXAMPLE 1 Manufacturing Components and Supplies, Industrial optics, are example of product categorization class defined in UNSPSC.

Neither properties nor data types, nor constraints are associated with such a class. Moreover, categorization classes may not be related to each other by the is-a inheritance relationship, but they may only be related to each other through the is-case-of class relationship. A specific XML element, called **categorization_class_superclasses** allows to record the categorization classes that are superclasses of a categorization class in a case-of hierarchy.

NOTE Using the case-of resource constructs, item classes may also be connected to categorization classes.

EXAMPLE 2 The following example shows how characterization classes and categorization classes may be connected to achieve some particular goals. A ball bearing supplier wants to design its own ontology and to make it easy to retrieve and easy to use. To achieve these goals, he/she wants to use standard properties and to be connected to standard classifications. The supplier provides only ball bearings, but some bearings are sealed, some others are not. Particular properties may be associated with sealed bearings and with not sealed bearings, but these categories do not exist as classes in standard bearing ontologies. Thus, the bearing supplier processes as follows. (1) He/she designs a proprietary ontology consisting of three characterization classes: *my_bearing, my_sealed_bearings, my_non_sealed_bearing*. The two latter are connected to the former by the is-a inheritance relationship, and all the properties assigned to the former are inherited by the latter. (2) To use some of the properties defined in the future ISO/TS 23768-1, the bearing supplier specifies that his/her class *my_bearings* is case-of the standard bearing class *ball bearing* defined in ISO/TS 23768. Through this case-of relationship, he/she may import in his/her class *my_bearings* the standard-defined properties: *bore diameter, outside diameter, ISO tolerance class.* Moreover, he/she creates those needed properties that are not defined in the standard. (3) To facilitate the retrieval of the server that display the supplier's catalogue, he/she represents a small fragment of the UNSPSC classification, and a case-of relationship between the UNSPSC class *ball_bearings* and its own class *my_bearing*. The result is presented in Figure 25 below.

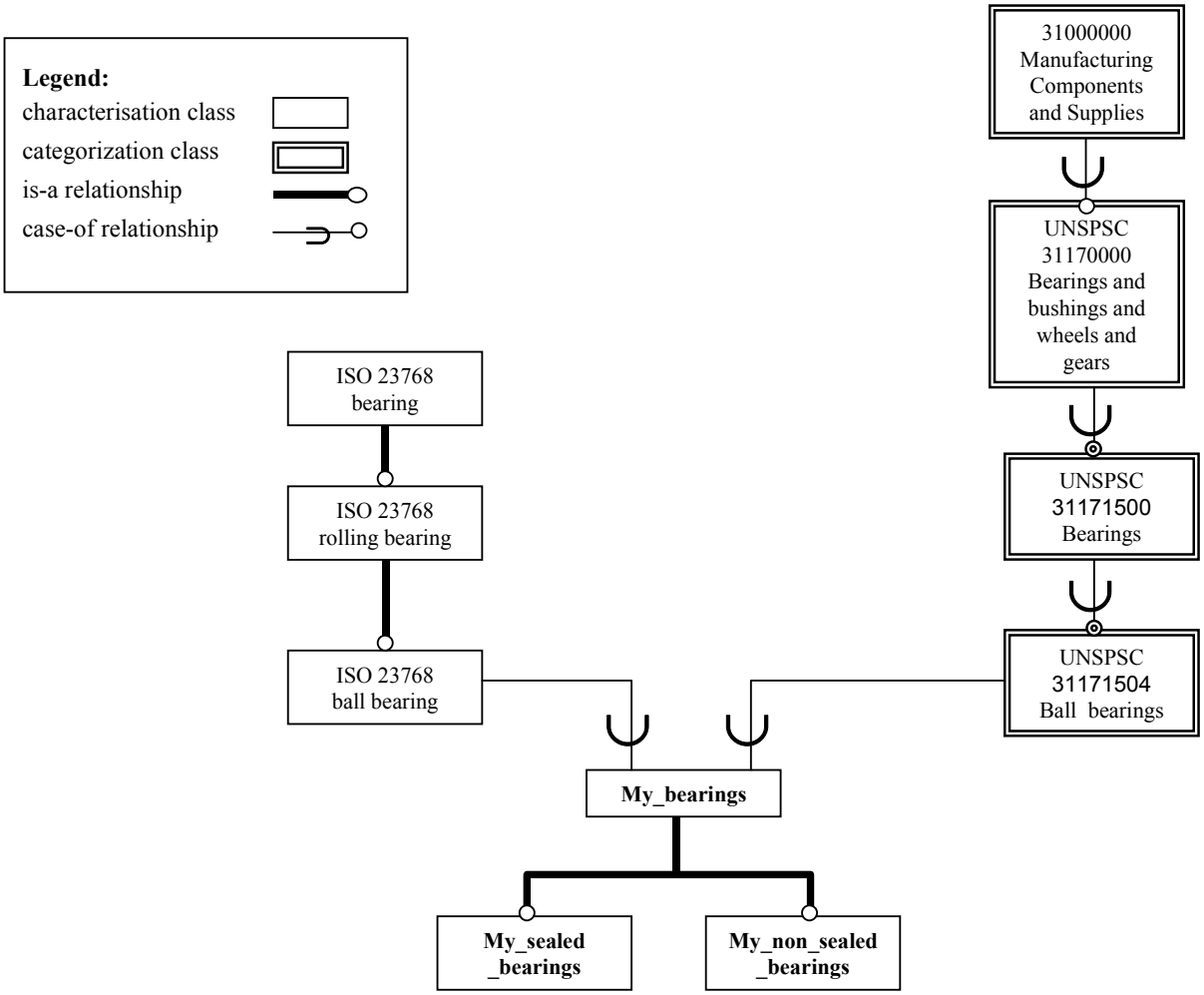

Figure 25 — Example of a supplier ontology using categorization classes

Figure 26 describes the structure of a categorization class.

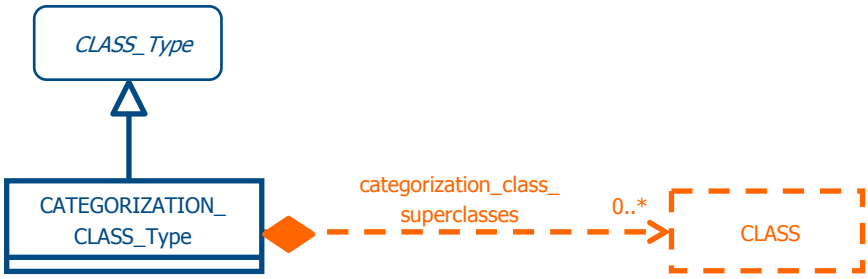

Figure 26 — Categorization class

A categorization class (represented by the **CATEGORIZATION_CLASS_Type** XML complex type) inherits the XML content description defined in the abstract **CLASS_Type** XML complex type.

Internal item definitions:

categorization_class_superclasses: the categorization classes that are one step above the categorization class in a case-of class hierarchy.

Constraint specifications:

Only identifiers corresponding to classes for which the base type is **CATEGORIZATION_CLASS_Type** may be referenced in the **categorization_class_superclasses** collection.

The inherited **its_superclass** XML element shall not be specified when defining a categorization class.

The inherited **described_by** XML element shall not be specified when defining a categorization class.

The inherited **defined_types** XML element shall not be specified when defining a categorization class.

The inherited **sub_class_properties** XML element shall not be specified when defining a categorization class.

The inherited **class_constant_values** XML element shall not be specified when defining a categorization class.

The inherited **constraints** XML element shall not be specified when defining a categorization class.

A categorization class shall not be the property definition class of any property.

6.7.2.3 Item class case-of

Standard product ontologies are designed to represent precisely and formally the product classes and the product properties on the definition for which experts of some product domain have agreed.

In each particular organization:

— only a few of all the standard product characterization classes may be useful;

— these standard product characterization classes may be defined in different standard product ontologies;

— some organization-specific classes may exist;

— only a few of the standard properties may be considered as needed, and

— some organization specific properties may be in use.

Thus, for representing such a domain using directly the standard product ontology within each organization would be not efficient.

The case-of relationship allows each organization to define its own class hierarchy while allowing to import all needed standard properties, thus providing for data integration and for data exchange with other organizations.

The case-of relationship may also be used within standard dictionaries.

In OntoML, the **ITEM_CLASS_CASE_OF_Type** XML complex type is used for such a purpose. It is illustrated in Figure 27.

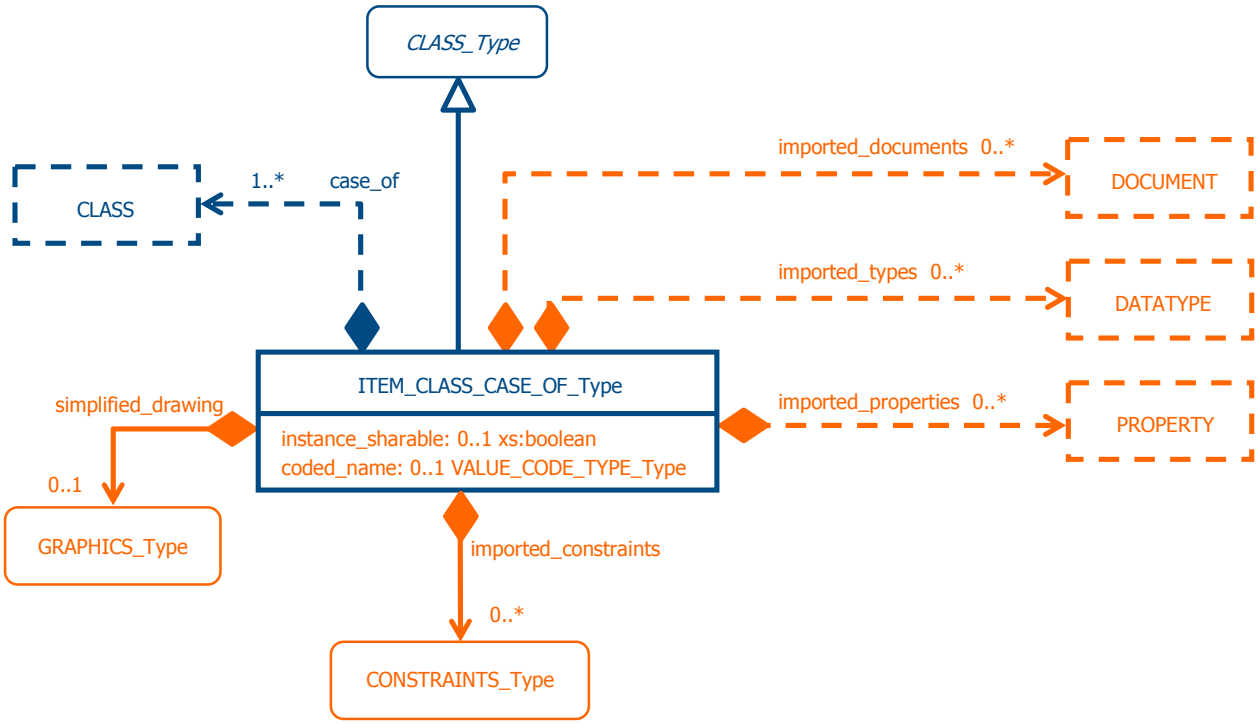

Figure 27 — Item class case-of UML diagram

Internal item definitions:

case_of: the class(es) for which the described class is case-of; it is represented by a reference to the corresponding class(es) ontology concept identifier(s).

NOTE 1 The case-of relationship and its use are described in Clause 3.5.2 of ISO/IEC Guide 77-2:2008.

imported_constraints: the set of constraints that are imported from the item class(es) the defined item class is case-of.

NOTE 2 Unlike other imported entities, the **imported_constraints** constraints cannot be selected when an item class case of is designed. These constraints are all the constraints that restrict the domains of any of the properties defined in the **imported_properties** collection in the classes of the **case_of** XML elment from which they are imported.

imported_documents: the imported document(s) from the **case_of** class(es) that are required to describe the current class; it is represented by a reference to the corresponding document(s) ontology concept identifier(s).

imported_properties: the imported property(ies) from the **case_of** class(es) that are required to describe the current class; it is represented by a reference to the corresponding property(ies) ontology concept identifier(s).

imported_types: the imported type(s) from the **case_of** class(es) that are required to describe the current class; it is represented by a reference to the corresponding type(s) ontology concept identifier(s).

instance_sharable (**ITEM_CLASS_Type**): when false, it specifies that instances of the item class case-of are features; when not provided or true it specifies that instances of the item class case-of are stand-alone items.

NOTE 3 In the common ISO13584/IEC61360 dictionary model, it is implementation dependent to decide whether several real world instances of features modeled by the same set of property-values pairs are represented by several EXPRESS pieces of data or by the same piece of data in the data exchange file. Thus, an instance of an item class case-of whose **instance_sharable** equals *false* and that is referenced by several instances of item classes case-of at the data model level is interpreted as several real world instances of the same feature.

simplified_drawing: drawing that can be associated to the described class.

Internal type definitions:

VALUE_CODE_TYPE_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a value code. Its value length shall not exceed 35 characters.

External type definitions:

CLASS_Type: see 6.7.2.1.

CLASS_VALUE_ASSIGNMENT_Type: see 6.7.2.4.

CONSTRAINTS_Type:: see 8.5.1.

GRAPHICS_Type: see 8.2.2.2.

Constraint specifications:

The inheritance structure defined by the class hierarchy (defined through the **its_superclass** XML element inherited from the **CLASS_Type** XML complex type) shall not contain cycles.

Only those properties that are visible for a class may become applicable to this class by virtue of being referenced in the list specified by the **described_by** collection.

Only those data types that are visible for a class may become applicable to this class by virtue of being referenced in the list specified by the **defined_types** collection.

Only those properties that are not applicable for a class by inheritance may become applicable to this class by virtue of being referenced in the list specified by the **described_by** collection.

Only those data types that are not applicable for a class by inheritance may become applicable to this class by virtue of being referenced in the list by specified the **defined_types** collection.

The **constraints** collection shall define restrictions that are compatible with the domain of values of the properties to which they apply.

All the properties referenced in the **precondition** collection of a constraint for which base type is a **CONFIGURATION_CONTROL_CONSTRAINT_Type** shall be applicable to the class.

All the properties referenced in the **constraints** collection shall be either visible or applicable to the class.

The properties referenced in the **class_constant_values** collection were declared as class-valued in some superclass of the current class, or in the current class itself.

If a property referenced in the **class_constant_values** collection was already assigned a value in a superclass, the value assigned in the current class should be the same.

Only an identifier corresponding to a class for which the base type is **ITEM_CLASS_Type** or an **ITEM_CLASS_CASE_OF_Type** may be referenced in the **its_superclass** XML element.

Only an identifier corresponding to a class for which the base type is **ITEM_CLASS_Type**, an **ITEM_CLASS_CASE_OF_Type** or a **CATEGORIZATION_CLASS_Type** may be referenced in the **case_of** collection.

The properties referenced in the **sub_class_properties** collection shall be referenced either in the **described_by** collection, or in the **imported_properties** collection.

All the class valued properties declared by being referenced in the **sub_class_properties** collection that are also referenced in the **imported_properties** collection shall be class valued properties in all the **case_of** classes where they are applicable.

The values assigned to an imported property in the **class_constant_value** collection shall not be different than the possible value assigned to the same property in the referenced classes.

All the properties referenced in the **imported_properties** collection that are class valued properties in a class referenced from the **case_of** collection, must be class valued properties in the current class.

All the properties referenced in the **imported_properties** that are assigned a class constant value in a class from the **case_of** collection, must be assigned the same class constant value class value in the current class.

Each constraint specified in the **imported_constraints** collection through the **constraint** XML element shall be defined either as a referenced constraint using the **constraint_ref** XML attribute or as a defined constraint using the **constraint_definition** XML element, not both.

6.7.2.4 Class valued property

A property may be specified as taking a single value for a given class. Such a property is called a class valued property. The definition, of such a property is twofold:

— it is declared in a given class; the property is described as any other property, but with one restriction: the property is referenced in the **sub_class_properties** XML element of the associated class.

— a typed value may be assigned in every subclasses of this given class: the value assignment is inherited by all the subclasses of the class where it is defined.

The assignment of a class valued property value to a class is represented by the **CLASS_CONSTANT_VALUES_Type** XML complex type as illustrated in Figure 28.

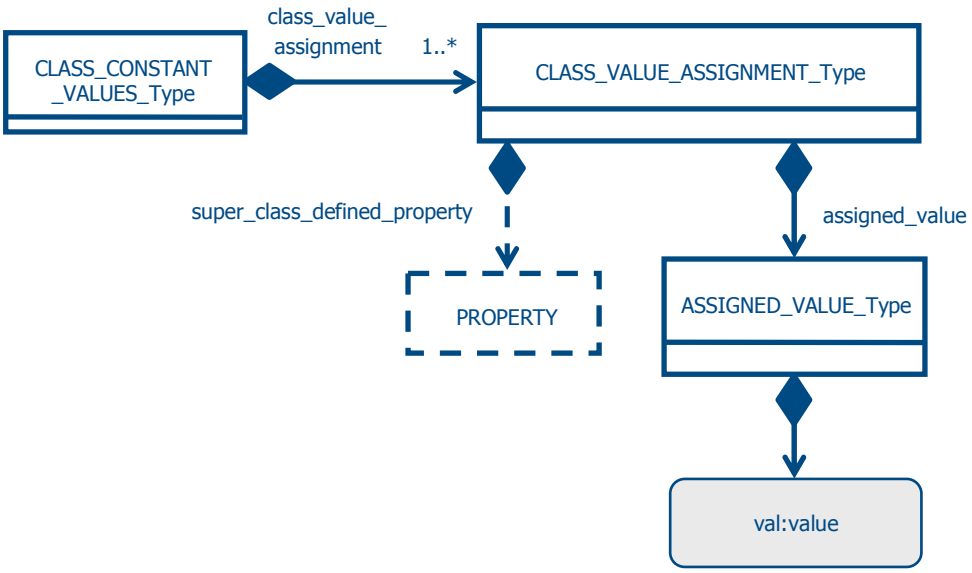

Figure 28 — Class value assignment structure

Internal item definitions:

class_value_assignment: the specification of all the class value assignments.

class_value_assignment/assigned_value: the value assigned to the property, valid for the whole class referring this class value assignment in its **class_constant_values** collection.

NOTE 1 The **assigned_value** belongs to referenced subclass property value domain.

class_value_assignment/super_class_defined_property: reference to the property (defined in a superclass as being a subclass property) to which value (**assigned_value** XML element) is assigned.

NOTE 2 A property is defined as being a subclass property when it appears in the **sub_class_properties** XML element defined in an **ITEM_CLASS_Type** or **ITEM_CLASS_CASE_OF_Type** XML complex type.

Internal type definitions:

ASSIGNED_VALUE_Type: the value to be assigned.

CLASS_VALUE_ASSIGNMENT_Type: the specification of a class value assignment.

VALUE_CODE_TYPE_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a value code. Its value length does not exceed 35 characters.

External type definitions:

val:value: specification of a typed value.

NOTE 3 **val:value** is defined in the ISO/TS 29002-10 product exchange format.

Constraint specification:

The value assigned by means of the **assigned_value** XML element shall be type compatible with the value domain of the associated property (referenced by the **super_class_defined_property** XML element).

6.7.3 Advanced-level ontology class

This clause defines modelling constructs that may only be used within the advanced subset of OntoML.

Characterization classes allow to capture the various kinds of items of an application domain, and to identify those characteristic properties that provide for discriminating items of the class by their values. Identical values for all characteristics properties means that items are identical. Different values for some properties means that items are different. Therefore, characteristic properties are supposed to be rigid (invariant for a given item).

EXAMPLE 1 When looking at the mechanical fasteners domain, we may identify the *metric threaded screw* as a characterization class of fasteners, and properties like *threaded length*, *total length*, *threaded diameter*, *material*, *part number* as characteristic properties allowing to characterize different ~~subsets of identical~~ items.

But properties of an item are not only characteristic properties. According to the discipline-oriented point of view we have on an item, a number of other properties prove useful. If an ontology user is in charge of *procurement* of screws, the properties *price* and *delivery delay* are necessary. If an ontology user is responsible for *inventory management*, the *inventory size* (the number of particular screws that are currently available in the inventory) and the *quantity of order* of this screw are relevant properties. If an ontology user designs products using a given CAD system, and wants to insert one screw in the current product, the *geometric shape* of the screw is necessary to make the design process efficient.

Compared with characteristic properties, the properties considered above have two differences that suggest criteria for identifying them:

— each of these properties makes sense only for some particular points of view, usages or disciplines of the item user;

— most of these properties are not characteristics of an item: they may change without changing the target item.

To allow an ontology designer to separate this kind of properties from the characteristic properties and to structure all the discipline-oriented properties that may be associated with an item, OntoML provides two additional categories of classes:

— functional view classes that provide for representing discipline-oriented points of view on items;

EXAMPLE 2 Procurement, inventory, marketing; geometry (3D, simplified) or geometry (2D, front view, precise), are examples of points of view.

NOTE 1 As shown in the geometry example, specifying a point of view may require not only a *name* ("geometry"), but also values of some variables such that *geometry_level* (2D, 3D), *side* (top, front, …) or *level_of_detail*. Such variables are called view control variables.

— functional model classes that provide for representing items of some characterization class(es) according to the discipline-oriented point of view defined by a functional view class.

NOTE 2 Each functional model class references one functional view class for defining its discipline-oriented point of view.

NOTE 3 If a functional model class does not reference a characterization class in an exchanged ontology, it is intended to be associated with a characterization class on the target user system.

EXAMPLE 3 A *procurement* functional model class might contain values for the following properties: *part number* (mported from a *metric threaded screw* characterization class)*, price, delivery delay*, for each screw in the *metric threaded screw* class.

EXAMPLE 4 The set of 2D precise front geometrical representations of all the screws in a *metric threaded screw* class could constitute a functional model class that provides a geometrical representation of items of this class.

NOTE 4 In the context of OntoML, geometrical representation might be exchanged as http files whose URIs are defined by a property whose type is **URI_TYPE_Type**.

The ISO 13584 standard series does not define which properties should be represented in characterization classes or in functional model classes, nor the different functional view classes that may exist. It is the responsibility of ontology designers to decide whether they split properties of the same item into different classes and which functional view classes should be defined for characterizing the various discipline-oriented points of view.

Some recommendations may be considered when structuring an ontology:

— properties represented in a characterization class should allow to discriminate items that are not considered as identical by the community that will use the ontology;

EXAMPLE 5 A characterization class of *screw*s such that two screws having equal values of all their properties are not compatible to replace each other in a mechanical product is probably not adapted for mechanical user needs. For being adapted to mechanical user needs, a new property would be necessary in order to discriminate both screws.

— properties that are not characteristic of items of a characterisation class, i.e., whose values may change without changing the item, might be considered to be put in a functional model class.

EXAMPLE 6 The *price* of an item may change over the time without changing the item. This property, and possibly some other business-oriented properties, might be considered for building a functional model class.

— a functional model class should be created when some properties are interesting only for a specific category of users.

EXAMPLE 7 The properties that describe the disposal of an item of a mechanical item class could be considered for building a functional model class.

6.7.3.1 Functional view class

A functional view class allows to characterize a particular a discipline-oriented point of view (also called a representation category) that may prove useful for various classes of products.

NOTE 1 A discipline-oriented point of view may be the specification of a particular engineering discipline. In this case, the functional view class is a simple class without any property. The name and definition of the class describe the point of view. The definition should also either define the kind of properties that should be represented in functional model classes that reference this functional view class, or list explicitly the properties that should be represented.

If needed, a functional view class may contain properties, called view control variables, to further specify a particular representation sub-category.

EXAMPLE 1 A *geometry* functional view class specifies a geometrical point of view. But, such a point of view remains ambiguous. To remove this ambiguity, a *geometry_level* view control variable whose values would be *2D* or *3D* could be defined. A functional model that references this functional view class would import this view control variable to specify which kind of particular geometrical representations it provides.

EXAMPLE 2 A functional view class named *acceptable_environmental_condition* might be used to characterize in which environments products of some characterization classes may be safely used. The definition of this functional view class could for instance specify in its definition that functional model classes referencing this functional view class could only contain (1) properties imported from an item class and (2) level type-valued environmental properties having at least a *min* and *max* values, for specifying in which context products of the referenced characterization classes are guaranteed for usage. Such a functional view class may be defined without using view control variables.

EXAMPLE 3 A functional view class may also be used for describing the *inventory status* point of view. This functional view class may be defined without any view control variable if the ontology designers only want to define a unique *inventory status* point of view that will describe, for each product, the *inventory size*, i.e., the number of each part that are currently available in the inventory, and the *quantity of orde*r of this part.

A view control variable does not characterise an item but a representation of an item. Thus, it shall be defined by a **REPRESENTATION_P_DET** property defined in Clause 6.7.5.

In this part of ISO 13584, a functional view class is represented by a **NON_INSTANCIABLE_FUNCTIONAL_VIEW_CLASS_Type** XML complex type. It is illustrated in Figure 29.

NOTE 2 The name "non instanciable functional view class type" comes from ISO 13584-24:2003 that defines also a mechanism for creating item representation by program. Thus, in this case, views are said to be "instantiated". This mechanism being seldom used in the past, it is not introduced in OntoML, but the name "non instanciable functional view class type" is preserved to keep the same name for the same resource both in ISO 13584-24:2003 and in OntoML.

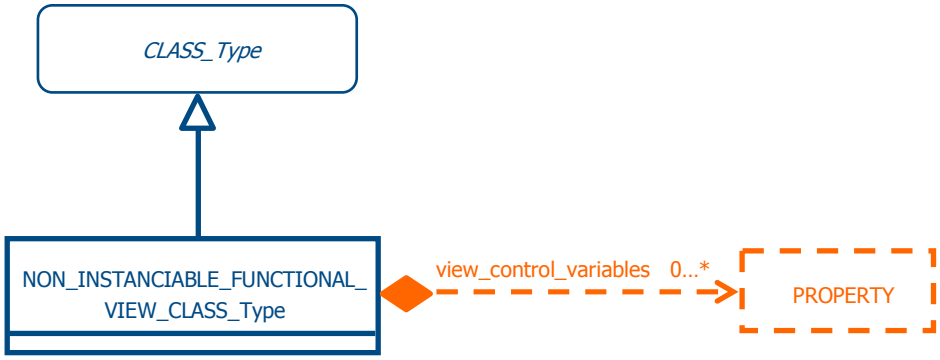

Figure 29 — Advanced-level ontology class concept UML diagram: functional view class

Internal item definitions:

view_control_variables: the list of properties that further defines the discipline-oriented point of view specified by the functional view class.

External type definitions:

CLASS_Type: see 6.7.2.1.

Constraint specifications:

Either the class has no referenced superclass (**its_superclass** inherited XML element), or if it has one, the referenced superclass shall be a **NON_INSTANCIABLE_FUNCTIONAL_VIEW_CLASS_Type**.

The type of each view control variable (**domain** XML element of the **PROPERTY_Type** XML complex type) shall be a **NON_QUANTITATIVE_INT_TYPE_Type** (see 8.3.8) whose values are successive integers.

Properties intended to be used as view control variables shall be defined as **REPRESENTATION_P_DET_Type** (see 6.7.5) and shall be referenced in the **described_by** collection of the non instanciable functional view class.

6.7.3.2 Functional model class

A functional model class is intended to provide the discipline-oriented point of view descriptions, specified by a functional view class, of items belonging to some item characterization class.

EXAMPLE 1 Assume that the *screw class* is the root item class of a hierarchy of screw classes and that this class declares the *part number* property allowing to identify a screw of any subclass of *screw class*; assume that *item price view* is a functional view class whose definition is "functional model classes that reference this view shall provide prices of items in Euros"; then a functional model class *screw price model* could be defined. It would reference both *screw class* (by **view_of** XML element, see Figure 31) to specify that it provides functional models of screws, and *item price view* (by **created_view**, see Figure 30 and Figure 31) to specify that it provides the *item price view* view. This functional model class could also import the *part number* property from *screw class* (through the **imported_properties_from_item** XML element, see Figure 31) and declare (using the **described_by** XML element) the *euro price* property as a **real_currency_type** property. With these assumptions, each instance of *screw price model* may contain a couple (*part number*, *euro price*) that specify the price of one of the screws of the *screw class* (or any of its subclass) in Euros.

NOTE 1 In Example 1, the prices may be automatically computed for each screw of the *screw* class by specifying that *part number* shall be represented both in the screw class and in the functional model class, and may be used to compute a left outer join between the contents of both classes (see **required_item_values** XML element in Clause 7.4)

EXAMPLE 2 With the same definition as in Example 1 for *screw class* and for *item price view,* a particular functional model class might be defined for each subclass of *screw class* that contain instances.

NOTE 2 In general, functional model classes are linked *a priori* with a product characterization class and, in this case, they are represented by the **FM_CLASS_VIEW_OF_Type** subtype. But a functional model class is not required to reference any product characterization class. This permits to link it *a posteriori* with existing characterization classes.

NOTE 3 This a posteriori link is specified using the **A_POSTERIORI_VIEW_OF_Type** XML complex type construct specified in clause 8.6.2.

A functional model class may import:

— properties and/or types and/or documents from the functional view class that specifies the point of view oriented descriptions it provides;

— properties and/or types and/or documents from the functional model class(es) for which the current functional model class could be case-of.

NOTE 4 A functional model class may also inherit properties and/or types and/or documents from its possible superclass. This allows to share the same properties and/or types and/or documents between a hierarchy of functional model classes.

Such a functional model class, represented by a **FUNCTIONAL_MODEL_CLASS_Type** XML complex type is depicted in Figure 30.

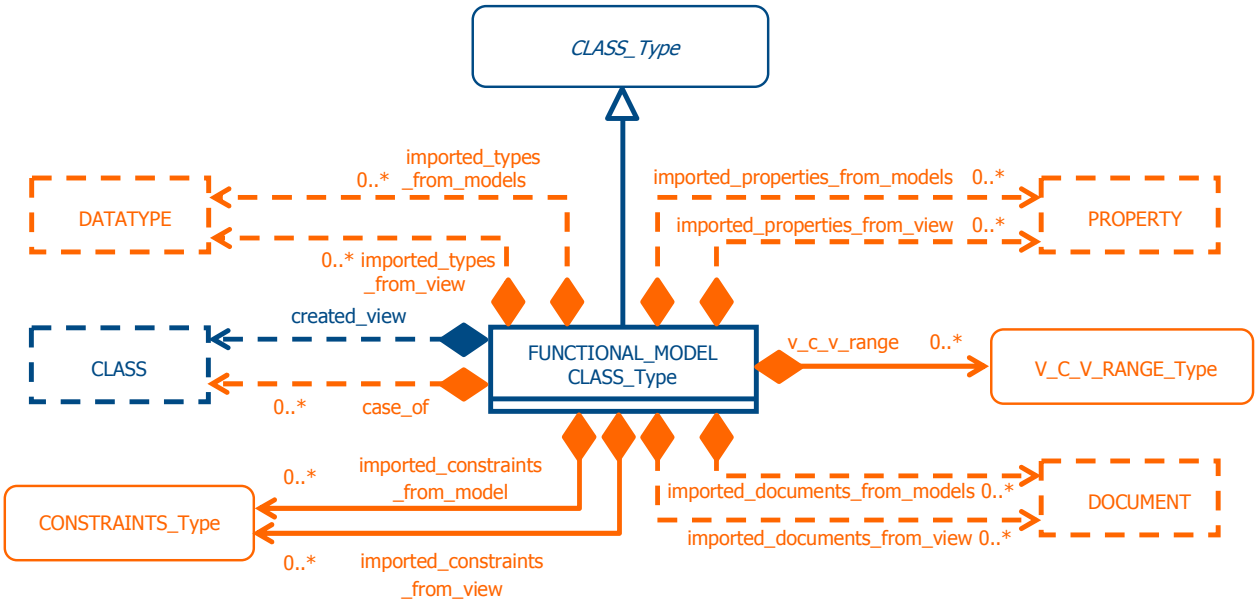

Figure 30 — Advanced class ontology concept UML diagram: functional model class

Internal item definitions:

case_of: the possible functional model classes the current functional model class is *case-of*.

NOTE 5 The case-of relationship and its use are described in Clause 3.5.2 of ISO/IEC Guide 77-2:2008.

created_view: the functional view class that characterizes the point of view addressed by the functional model class.

imported_constraints_from_model: the possible constraints that are imported from the *case-of* functional model classes.

NOTE 6 All the constraints that apply on the model properties that are imported by the functional model class are imported in this class.

imported_constraints_from_view: the possible constraints that are imported from the created view.

NOTE 7 All the constraints that apply on the view properties that are imported by the functional model class are imported in this class.

imported_documents_from_model: the possible documents that are imported from the *case-of* functional model classes.

imported_documents_from_view: the possible documents that are imported from the created view.

imported_properties_from_model: the possible properties that are imported from the *case-of* functional model classes.

imported_properties_from_view: the possible properties that are imported from the created view.

imported_types_from_model: the possible types that are imported from the *case-of* functional model classes.

imported_types_from_view: the possible types that are imported from the created view.

v_c_v_range: the list of the view control variable ranges that specify the various discipline-oriented sub-categories the functional model class is able to create. Each of them shall correspond to a view control variable of the functional view that is referenced by the **created_view** XML element. When a view control variable of the functional view defined by the **created_view** XML element is not represented in the **v_c_v_range** element, its range is its complete value domain.

NOTE 8 In most of the cases, a functional view classes has no view control variable. Thus, the **v_c_v_range** of each functional model class that references this functional view class is empty.

External type definitions:

CLASS_Type: see 6.7.2.1.

CONSTRAINTS_Type: see 8.5.1.

V_C_V_RANGE_Type: see 6.7.3.4.

Constraint specifications:

The class referenced by the **created_view** XML element shall have **NON_INSTANCIABLE_FUNCTIONAL_VIEW_CLASS_Type** as its base type.

Each view control variable used in the **v_c_v_range** XML element shall correspond to a view control variable of the functional view that is referenced by the **created_view** XML element.

Each view control variable defined in the referenced view (**created_view** XML element) and used in the **v_c_v_range** XML element shall be referenced in the **imported_properties_from_view** collection.

NOTE 9 Each view control variable whose **v_c_v_range** XML element does not restrict to a singleton is part of the key of the functional model class. This is specified in the constraints of **EXPLICIT_FUNCTIONAL_MODEL_CLASS_EXTENSION_Type** (see 7.4).

Either the class has no superclass, or the referenced superclass (**its_superclass** inherited XML element) shall have **FUNCTIONAL_MODEL_CLASS_Type** or a **FM_CLASS_VIEW_OF_Type** as its base type.

The **v_c_v_range** collection shall consist of a unique **view_control_variable_range** for each referenced property (**parameter_type** XML element)

Each property that is defined (**described_by** XML element) or inherited for the functional model class may be either a **NON_DEPENDENT_P_DET_Type**, a **DEPENDENT_P_DET_Type**, a **CONDITION_DET_Type** or a **REPRESENTATION_P_DET_Type** property.

NOTE 10 This constraint is less restrictive than the one defined in ISO 13584-24:2003 which restricted all properties of a functional model class to be **REPRESENTATION_P_DET_Type** properties.

Each class referenced through the **case_of** XML element shall be a **FUNCTIONAL_MODEL_CLASS_Type** or a **FM_CLASS_VIEW_OF_Type** class.

All the class valued properties declared by being referenced in the **sub_class_properties** collection that are also referenced in the **imported_properties_from_model** collection shall be class valued properties in all the **case_of** classes where they are applicable.

The values assigned to an imported property in the **class_constant_values** collection shall not be different than the possible value assigned to the same property in the referenced classes.

Each constraint specified in the **imported_constraints_from_model** and in the **imported_constraints_from_view** collections through the **constraint** XML element shall be defined either as a referenced constraint using the **constraint_ref** XML attribute or as a defined constraint using the **constraint_definition** XML element, not both.

6.7.3.3 Functional model class *view-of*

A functional model class *view-of* is a functional model class whose point-of-view-oriented descriptions are directly associated with the products of a particular product characterization class. In such a case, and in addition to the properties and/or types and/or documents possibly imported from the referenced functional view and, possibly, from the *case-of* functional model(s), properties and/or types and/or documents may also be imported from the product characterization class for which the current functional model class is specified.

NOTE 1 In particular, it is advisable that some properties applicable to the item characterization class be imported to connect each representation defined by the functional model class view-of with each item of the item class.

Such a functional model class is represented by the **FM_CLASS_VIEW_OF_Type** XML complex type as depicted in Figure 31.

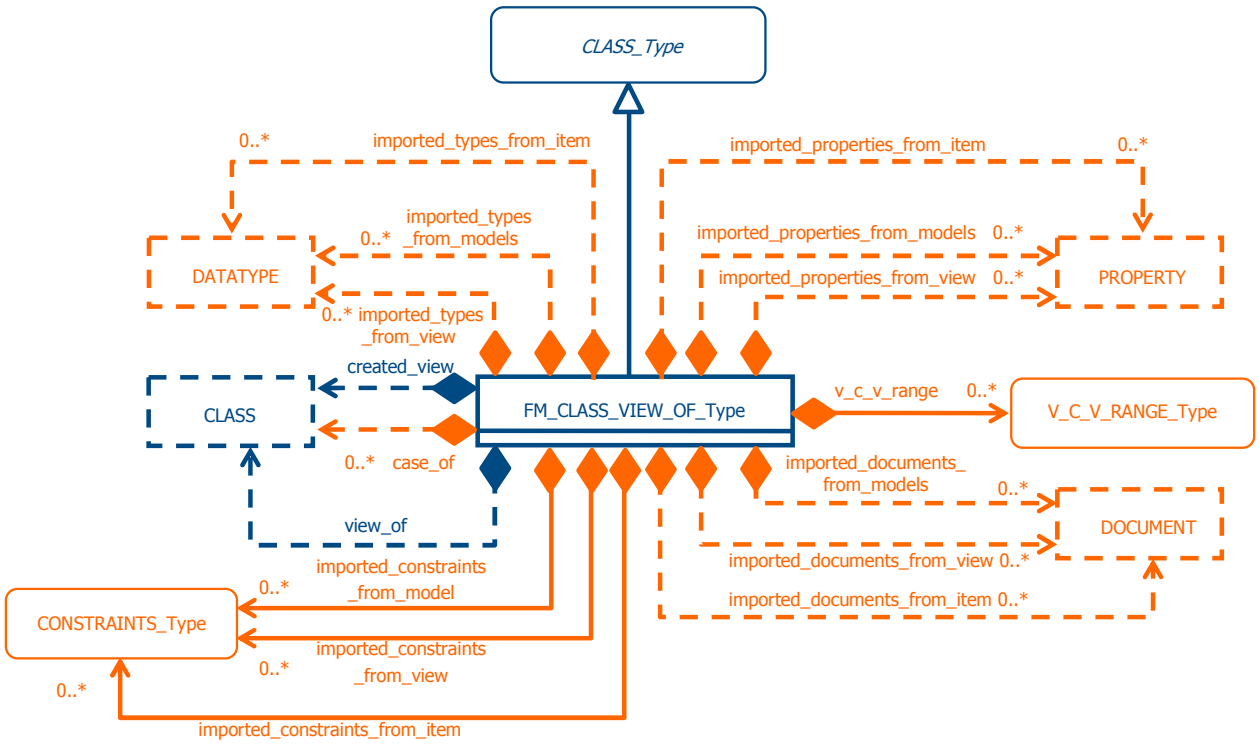

Figure 31 — Advanced class ontology concept UML diagram: functional model class *view-of*

Internal item definitions:

case_of: the possible other functional model classes the current functional model class is a *case-of*.

NOTE 2 The case-of relationship and its use are described in Clause 3.5.2 of ISO/IEC Guide 77-2:2008.

created_view: the functional view class that characterizes the user point of view addressed by the functional model class.

view_of: the product characterization class for which the described functional model class is able to define representations.

imported_constraints_from_item: the possible constraints that are imported from the item class for which the present class is able to generate the view.

NOTE 3 All the constraints that apply to item properties that are all imported by the functional model class are imported in this class.

imported_constraints_from_model: the possible constraints that are imported from the *case-of* functional model classes.

NOTE 4 All the constraints that apply to model properties that are imported by the functional model class are imported in this class.

imported_constraints_from_view: the possible constraints that are imported from the created view.

NOTE 5 All the constraints that apply to view properties that are imported by the functional model class are imported in this class

imported_properties_from_item: the possible properties that are imported from the item class for which the present class is able to generate the view.

imported_properties_from_model: the possible properties that are imported from the *case-of* functional model classes.

imported_properties_from_view: the possible properties that are imported from the created view.

imported_types_from_item: the possible types that are imported from the item class for which the present class is able to generate the view.

imported_types_from_model: the possible types that are imported from the *case-of* functional model classes.

imported_types_from_view: the possible types that are imported from the created view.

imported_documents_from_item: the possible documents that are imported from the item class for which the present class is able to generate the view.

imported_documents_from_model: the possible documents that are imported from the *case-of* functional model classes.

imported_documents_from_view: the possible documents that are imported from the created view.

v_c_v_range: the list of view control variable ranges that specify the various discipline-oriented sub-categories the functional model class is able to create. Each of them shall correspond to a view control variable of the functional view that is referenced by the **created_view** XML element When a view control variable of the functional view defined by the **created_view** XML element is not represented in the **v_c_v_range** element, its range is its complete value domain.

NOTE 6 In most of the cases, a functional view class has no view control variable. Thus, the **v_c_v_range** of the functional model classes view of that reference this functional view class are empty.

External type definitions:

CLASS_Type: see 6.7.2.1.

CONSTRAINTS_Type: see 8.5.1.

V_C_V_RANGE_Type: see 6.7.3.4.

Constraint specifications:

The class referenced by the **created_view** XML element shall have **NON_INSTANCIABLE_FUNCTIONAL_VIEW_CLASS_Type** as its base type.

Each view control variable used in the **v_c_v_range** XML element shall correspond to a view control variable of the functional view that is referenced by the **created_view** XML element.

Each view control variable defined in the referenced view (**created_view** XML element) and used in the **v_c_v_range** XML element shall be referenced in the **imported_properties_from_view** collection.

NOTE 7 Each view control variable whose **v_c_v_range** XML element does not restrict to a singleton is part of the key of the functional model class. This is specified in the constraints of **EXPLICIT_FUNCTIONAL_MODEL_CLASS_EXTENSION_Type** (see 7.4).

Either the class has no superclass, or the referenced superclass (**its_superclass** inherited XML element) shall have **FUNCTIONAL_MODEL_CLASS_Type** or a **FM_CLASS_VIEW_OF_Type** as its base type.

The **v_c_v_range** collection shall consist of a unique **view_control_variable_range** for each referenced property (**parameter_type** XML element).

Each property that is defined (**described_by** XML element) or inherited for the functional model class may be either **NON_DEPENDENT_P_DET_Type**, a **DEPENDENT_P_DET_Type**, a **CONDITION_DET_Type** or a **REPRESENTATION_P_DET_Type** property.

NOTE 8 This constraint is less restrictive than the one defined in ISO 13584-24:2003 which restricted all properties of a functional model class to be **REPRESENTATION_P_DET_Type** properties.

Each class referenced through the **case_of** XML element shall be a **FUNCTIONAL_MODEL_CLASS_Type** or a **FM_CLASS_VIEW_OF_Type**.

All the class valued properties declared by being referenced in the **sub_class_properties** collection that are also referenced in the **imported_properties_from_model** collection shall be class valued properties in all the **case_of** classes where they are applicable.

The values assigned to an imported property in the **class_constant_values** collection shall not be different than the possible value assigned to the same property in the referenced classes.

Each constraint specified in the **imported_constraints_from_model** and in the **imported_constraints_from_view** collections through the **constraint** XML element shall be defined either as a referenced constraint using the **constraint_ref** XML attribute or as a defined constraint using the **constraint_definition** XML element, not both

6.7.3.4 View control variable range

Functional views may be further specified using view control variables. Associated to a functional model that references a functional view, each view control variable range specifies which particular views are really described by that model.

EXAMPLE Assume that a *geometry* functional view defines a view control variable *detail_level* whose ordered domain of values is {*simplified, standard, extended*}. A particular functional model may describe only *simplified* and *standard* views. In this case, the range would be [*simplified : standard*].

NOTE When a view control variable of a functional view referenced by the functional model is not represented using a view control variable range, its range is its complete value domain.

A view control variable range is represented by the **VIEW_CONTROL_VARIABLE_RANGE_Type** XML complex type. All the view control variable ranges are gathered in a container specified by the **V_C_V_RANGE_Type**. It is illustrated in Figure 32.

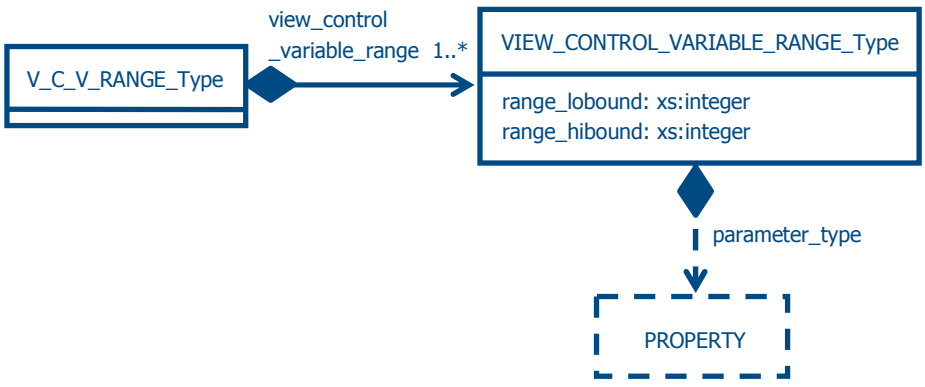

Figure 32 — View control variable structure

Internal item definitions:

view_control_variable_range: the specification of the functional model view control variable ranges.

view_control_variable_range/parameter_type: the reference to the property that is the view control variable for which the range applies.

view_control_variable_range/range_hibound: the integer that describes the high bound of the specified range.

view_control_variable_range/range_lobound: the integer that describes the low bound of the specified range.

Internal item definitions:

VIEW_CONTROL_VARIABLE_RANGE_Type: the specification of a view control variable range.

Constraint specifications:

The property referenced through the **parameter_type** XML element shall have a **NON_QUANTITATIVE_INT_TYPE_Type** type.

The **range_lobound** XML element value shall be less or equal to the **range_hibound** XML element value.

range_lobound and **range_hibound** shall belong to the domain of the referenced property (**parameter_type** XML element).

6.7.4 Simple-level ontology property

In the CIIM, values of a property are either simple values like integer or strings, or other class items. Moreover, the CIIM distinguishes:

— properties that may be used for characterizing an item, and

— properties that may only be used in functional model or functional view classes.

This clause defines the simple-level ontology property that are the properties used to characterize items.

The most commonly used properties in any ontology are the characterization properties that associates an item either with values or with other items. Such a property is represented by a **NON_DEPENDENT_P_DET_Type** XML complex type as illustrated in Figure 33.

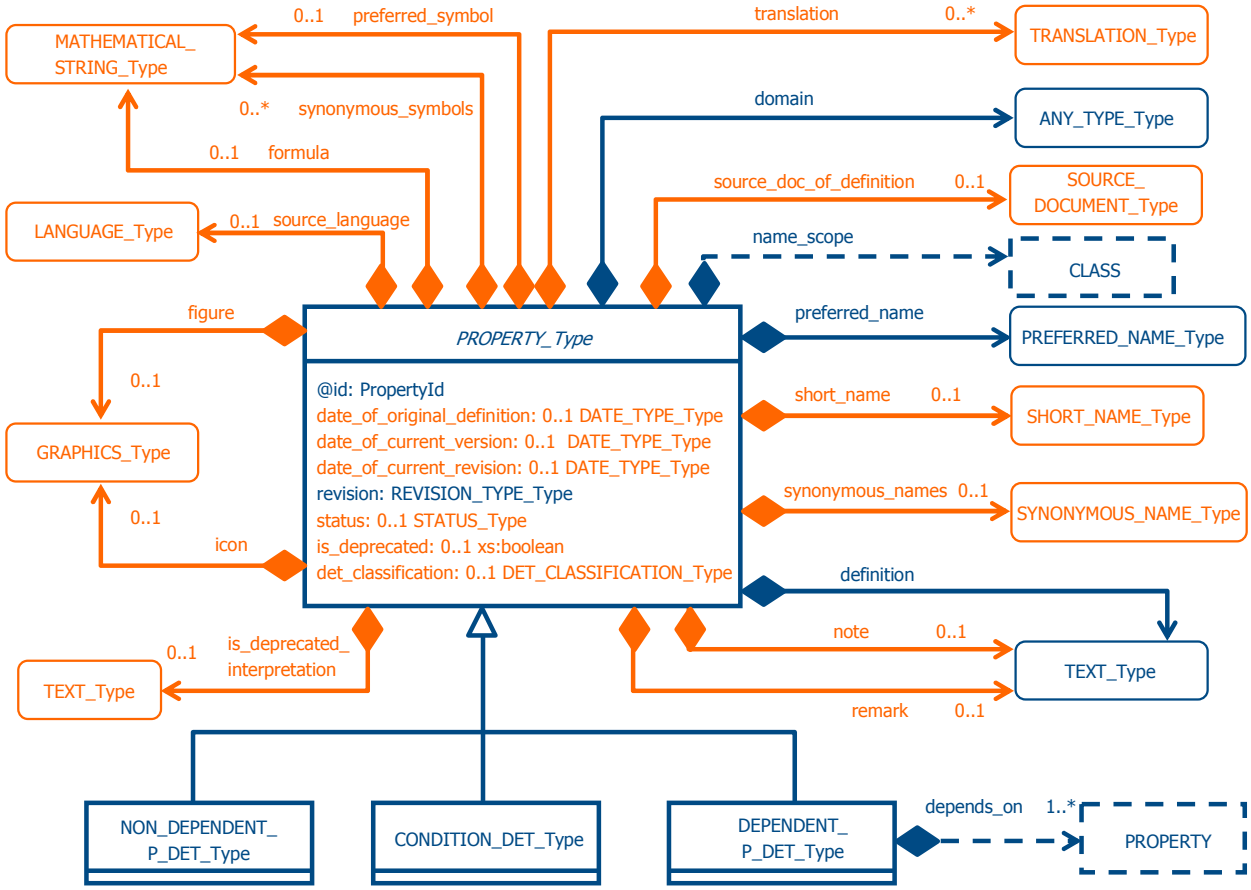

Figure 33 — Simple property ontology concept UML diagram

NOTE 1 Characterization property is defined in Clause 4 of ISO/IEC Guide 77-2:2008.

But, in the real world, no object can be considered as isolated from its environment. Quantitative properties that are measured, therefore, have to be related to conditions under which their values were obtained.

NOTE 2 Strictly speaking, any instance of a dimension could be accompanied by quoting the temperature at which the measurement was made. However, in practice, either this measuring context may be specified in the definition of a property, or it may be considered as not significant. But, for example, the *resistance* of an *electric thermistor* may strongly depend upon *ambient temperature*. Hence it is advisable that this information be always supplied.

Thus, in addition to the usual characterization properties, that may be considered as context independent properties, it is possible to define context parameters, through the **CONDITION_DET_Type** XML complex type, and context dependent properties, through the **DEPENDENT_P_DET_Type** XML complex type. For this latter kind of property, its context is defined (**depends_on** XML element) by reference to the context parameter(s) global identifier(s).

NOTE 3 Properties and context of evaluation is documented in Clause 4.4 of ISO/IEC Guide 77-2:2008 for specification of product properties and classes.

NOTE 4 The most basic representation of such a property requires only to define an identifier a **revision** number, a **preferred_name**, a **definition** and its domain of values (**domain** XML element).

Internal item definitions:

@id: the property identifier.

date_of_current_revision: the date associated to the present revision of the property definition.

date_of_current_version: the date associated to the present version of the property definition.

date_of_original_definition: the date associated to the first stable version of the property definition.

definition: the text describing this property, possibly translated.

depends_on (**DEPENDENT_P_DET_type**): set of references identifying the properties on which this property depends on.

det_classification: a code representing the ISO 80000 (formerly ISO 31) class for this property.

NOTE 5 ISO 13584-42:2010 specifies codes for ISO 80000 (formerly ISO 31) classes to which quantitative or non quantitative property may refer.

domain: the data type (the value domain) associated to the property.

figure: a possible graphics that describes the property.

formula: a mathematical expression for explaining the property.

icon: a graphics representing the description associated with the names.

is_deprecated: a Boolean that specifies, when true, that the property definition shall no longer be used.

is_deprecated_interpretation: specifies the deprecation rationale and how instance values of the property, and of its corresponding identifier, should be interpreted.

name_scope: the class domain of the property.

note: further information on any part of the property, which is essential to its understanding, possibly translated.

preferred_name: the name of the property that is preferred for use, possibly translated.

preferred_symbol: a shorter description of this property.

remark: explanatory text further clarifying the meaning of this property, possibly translated.

revision: the revision number of the present property definition.

short_name: the abbreviation of the preferred name, possibly translated.

source_doc_of_definition: the possible source document from which the definition comes.

source_language: the language in which the property definition was initially defined and that provides the reference meaning in case of translation discrepancy.

status: defines the life cycle state of the property definition.

NOTE 6 Allowed **status** values are defined by private agreement between the dictionary supplier and dictionary users.

NOTE 7 If the **status** XML element is not provided, and if this property definition is not deprecated as denoted by a possible **is_deprecated** XML element, then the property definition has the same standardization status as the whole ontology into which it is used. In particular, if the ontology is standardized, this property definition is part of the current edition of the standard.

synonymous_names: the set of synonymous names of the preferred name, possibly translated.

synonymous_symbols: the set of synonymous names of the preferred symbol of the property.

translation: the possible set of translations information provided for the translatable items.

Internal type definitions:

DATE_TYPE_Type: identifies the values allowed for a date (the specific **xs:date** XML Schema datatype).

REVISION_TYPE_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a revision. Its value length shall not exceed 3 characters.

STATUS_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a status.

External type definitions:

PropertyId: see 9.1.

ANY_TYPE_Type: see 8.3.

GRAPHICS_Type: see 8.2.2.2.

LANGUAGE_Type: see 8.1.1.

MATHEMATICAL_STRING: see 8.8.2

PREFERRED_NAME_Type: see 8.1.2.

SHORT_NAME_Type: see 8.1.2.

SOURCE_DOCUMENT_Type: see 8.2.2.1.

SYNONYMOUS_NAME_Type: see 8.1.2.

TEXT_Type: see 8.1.2.

TRANSLATION_Type: see 8.1.3.

Constraint specifications:

The **depends_on** collection shall only reference properties for which the base type is **CONDITION_DET_Type**.

The **depends_on** collection shall not contain duplicated property references.

When **is_deprecated** exists, **is_deprecated_interpretation** shall exist.

Instance values of i**s_deprecated_interpretation** element shall be defined at the time where deprecation decision was taken.

6.7.5 Advanced-level ontology property

In advanced-level ontology classes, i.e., functional view classes (see 6.7.3.1) and functional model classes (see 6.7.3.2 and 6.7.3.3), some properties are used that do not describe items, but that characterize representations of item. This is in particular the case for view control variables. These variables should be represented as **REPRESENTATION_P_DET_Type**. More precisely view control variables shall be represented as **REPRESENTATION_P_DET_Type**. Other properties defined in advanced-level ontology classes may be represented either as simple-level ontology property if they are considered as describing some aspect of an item or as **REPRESENTATION_P_DET_Type** if they characterize some representation of an item. **REPRESENTATION_P_DET_Type** is shown in Figure 34.

EXAMPLE Let's assume a property P1, defined in a functional model class, provides various 2D draftings of items of an item class, and property P2, defined in the same functional model class, describes the side of each drawing (top, bottom, front, rear,...) represented by P1. The value of P1 defines some aspect of an item. The value of P2 does not define any aspect of any item. Property P2 should be represented as **REPRESENTATION_P_DET_Type**. Property P1 could be represented as **non_dependent_P_DET**.

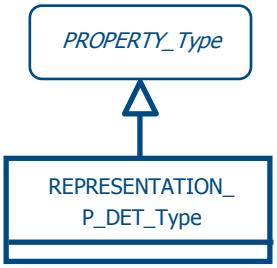

Figure 34 — Advanced property ontology concept UML diagram

NOTE The representation property concept is specified in ISO 13584-24:2003, Clause 11.15.1.

External type definitions:

PROPERTY_Type: see 6.7.4.

6.7.6 Identified data type

In some context, it proves useful to define domain of values that are associated with a global identifier and that could be re-used for several properties possibly in several ontologies.

EXAMPLE An ontology defining domain-specific units could be based on the use of ontology data types.

For that purpose, OntoML proposes a **DATATYPE_Type** XML complex type as illustrated in Figure 35.

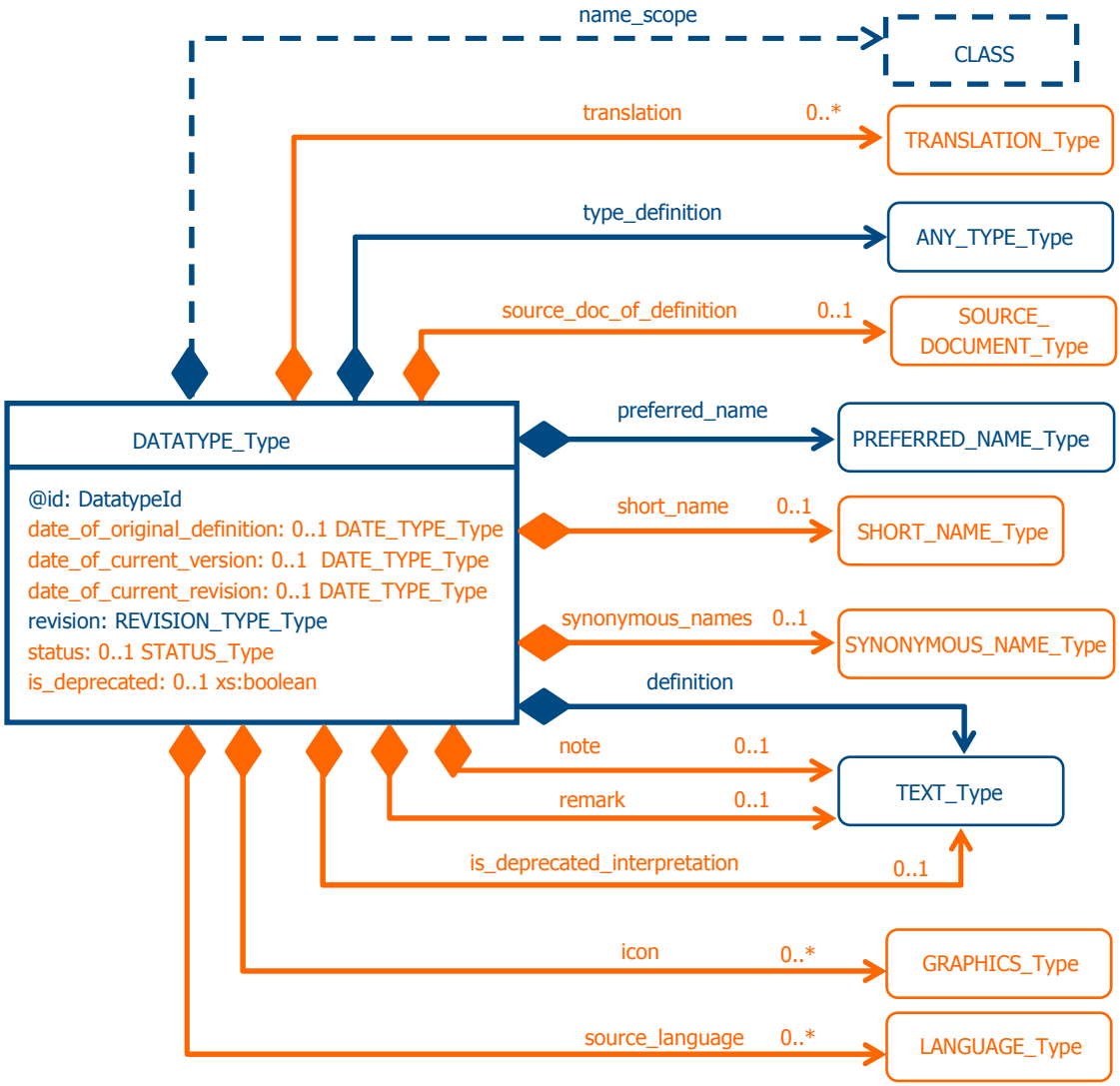

Figure 35 — Data type UML diagram

Internal item definitions:

@id: the datatype identifier.

date_of_current_revision: the date associated to the present revision of the datatype definition.

date_of_current_version: the date associated to the present version of the datatype definition.

date_of_original_definition: the date associated to the first stable version of the datatype definition.

definition: the text describing this datatype, possibly translated.

icon: a graphics representing the description associated with the names.

is_deprecated: a Boolean that specifies, when true, that the datatype definition shall no longer be used.

is_deprecated_interpretation: specifies the deprecation rationale and how instance values of the deprecated identified data type, and of its corresponding identifier, should be interpreted.

name_scope: the class domain of the datatype.

note: further information on any part of the datatype, which is essential to its understanding, possibly translated.

preferred_name: the name of the datatype that is preferred for use, possibly translated.

remark: explanatory text further clarifying the meaning of this datatype, possibly translated.

revision: the revision number of the present datatype definition.

short_name: the abbreviation of the preferred name, possibly translated.

source_doc_of_definition: the possible source document from which the definition comes.

source_language: the language in which the datatype definition was initially defined and that provides the reference meaning in case of translation discrepancy.

status: defines the life cycle state of the datatype definition.

NOTE 1 Allowed **status** values are defined by private agreement between the dictionary supplier and dictionary users.

NOTE 2 If the **status** XML element is not provided, and if this datatype definition is not deprecated as denoted by a possible **is_deprecated** XML element, then the datatype definition has the same standardization status as the whole ontology into which it is used. In particular, if the ontology is standardized, this datatype definition is part of the current edition of the standard.

synonymous_names: the set of synonymous names of the preferred name, possibly translated.

translation: the possible set of translations information provided for the translatable items.

type_definition: the description of the type carried by the data types.

Internal type definitions:

DATE_TYPE_Type: identifies the values allowed for a date (the specific **xs:date** XML Schema datatype).

REVISION_TYPE_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a revision. Its value length shall not exceed 3 characters.

STATUS_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a status.

External type definitions:

DatatypeId: see 9.1.

ANY_TYPE_Type: see 8.3.

GRAPHICS_Type: see 8.2.2.2.

LANGUAGE_Type: see 8.1.1.

PREFERRED_NAME_Type: see 8.1.2.

SHORT_NAME_Type: see 8.1.2.

SOURCE_DOCUMENT_Type: see 8.2.2.1.

SYNONYMOUS_NAME_Type: see 8.1.2.

TEXT_Type: see 8.1.2.

TRANSLATION_Type: see 8.1.3.

Constraint specifications:

When **is_deprecated** exists, **is_deprecated_interpretation** shall exist.

Instance values of **is_deprecated_interpretation** element shall be defined at the time where deprecation decision was taken.

6.7.7 Document

In OntoML, a document may be associated with a global identifier and considered as a CIIM ontology concept. For that purpose, OntoML proposes a **DOCUMENT_Type** XML complex type as illustrated in Figure 36.

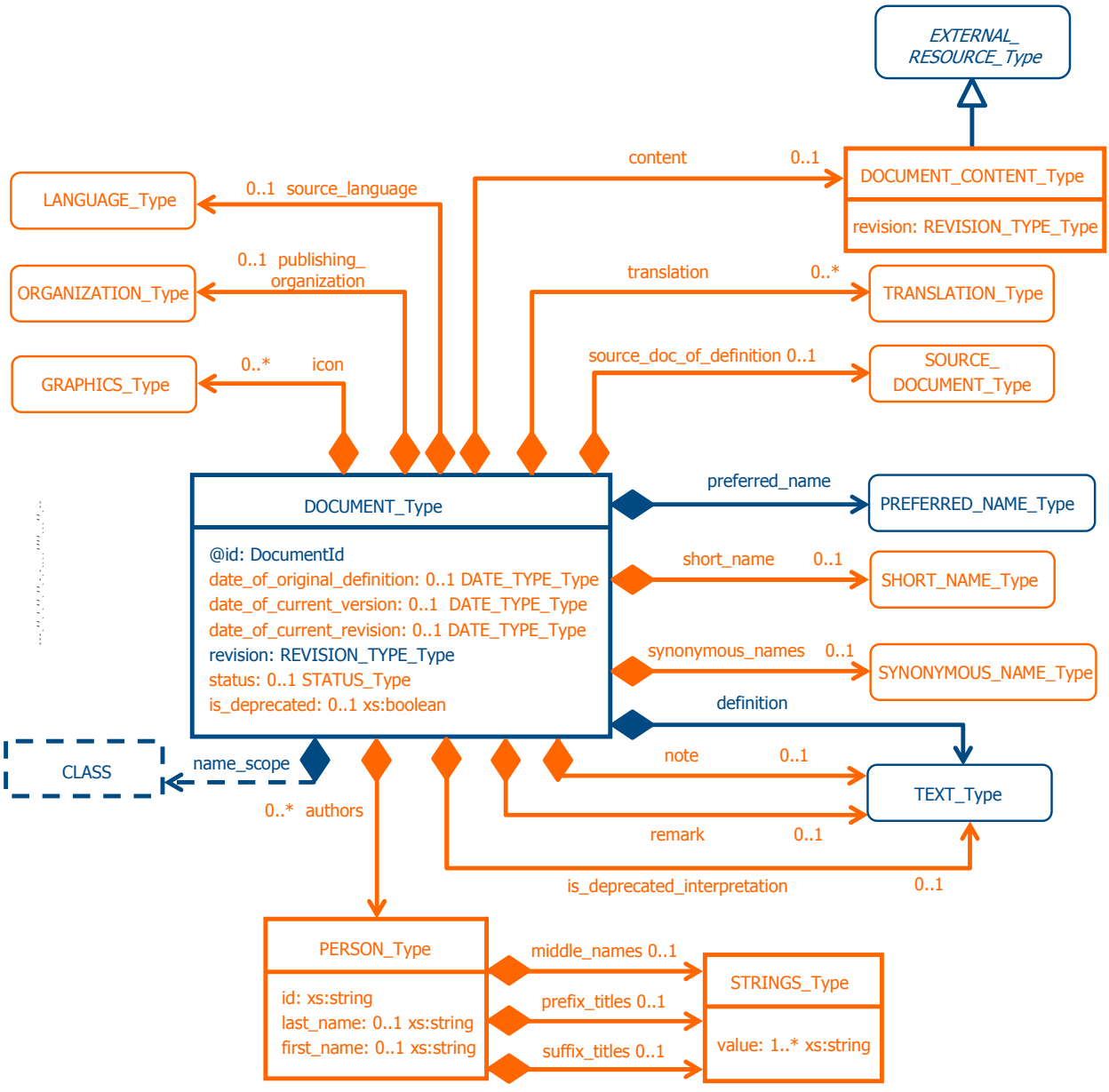

Figure 36 — Simple-level document UML diagram

Internal item definitions:

@id: the document identifier.

authors: the author(s) of the document.

authors/first_name: the first element of the person's list of forenames.

authors/id: a means by which the person may be identified.

authors/last_name: the person's surname.

authors/middle_names: the person's other forenames, if there are any.

authors/prefix_titles: the word, or group of words, which specify the person's social and/or professional standing and appear before his or her names.

authors/suffix_titles: the word, or group of words, which specify the person's social and/or professional standing and appear after his or her names.

authors/middle_names/value: a a middle name string in a midle name string collection.

authors/prefix_titles/value: a a prefix title string in a prefix title string collection.

authors/suffix_titles/value: a a suffix title string in a suffix title string collection.

content: the physical document for which the **DOCUMENT_Type** XML complex type gives a description.

NOTE 1 The content of the document is represented by the **DOCUMENT_CONTENT_Type** XML complex type. It is defined as a subtype of the **EXTERNAL_RESOURCE_Type** XML complex type defined in Clause 8.2. Thus, the inherited **file** XML element allows to reference, using a URI, the target document.

NOTE 2 The physical document is optional, and may, or not, be delivered in the same OntoML document instance.

content/revision: characterization of the updating of the physical document.

date_of_current_revision: the date associated to the present revision of the document definition.

date_of_current_version: the date associated to the present version of the document definition.

date_of_original_definition: the date associated to the first stable version of the document definition.

definition: the text describing this document, possibly translated.

icon: a graphics representing the description associated with the names.

is_deprecated: a Boolean that specifies, when true, that the document definition shall no longer be used.

is_deprecated_interpretation: specifies the deprecation rationale and how instance values of the deprecated document, and of its corresponding identifier, should be interpreted.

name_scope: the class domain of the document.

note: further information on any part of the document, which is essential to its understanding, possibly translated.

preferred_name: the name of the document that is preferred for use, possibly translated.

publishing_organization: the organisation that publishes the document.

remark: explanatory text further clarifying the meaning of this document, possibly translated.

revision: the revision number of the present document definition.

short_name: the abbreviation of the preferred name, possibly translated.

source_doc_of_definition: the possible source document from which the definition comes.

source_language: the language in which the document definition was initially defined and that provides the reference meaning in case of translation discrepancy.

status: defines the life cycle state of the document definition.

NOTE 3 Allowed **status** values are defined by private agreement between the dictionary supplier and dictionary users.

NOTE 4 If the **status** XML element is not provided, and if this document definition is not deprecated as denoted by a possible **is_deprecated** XML element, then the document definition has the same standardization status as the whole ontology into which it is used. In particular, if the ontology is standardized, this document definition is part of the current edition of the standard.

synonymous_names: the set of synonymous names of the preferred name, possibly translated.

translation: the possible set of translations information provided for the translatable items.

Internal type definitions:

DATE_TYPE_Type: identifies the values allowed for a date (the specific **xs:date** XML Schema datatype).

DOCUMENT_CONTENT_Type: the physical resource to which the document definition is related.

REVISION_TYPE_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a revision. Its value length shall not exceed 3 characters.

STATUS_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a status.

STRINGS_Type: a string (**xs:string** XML Schema datatype) collection container.

External type definitions:

DatatypeId: see 9.1.

EXTERNAL_RESOURCE_Type: see 8.2.1.

GRAPHICS_Type: see 8.2.2.2.

LANGUAGE_Type: see 8.1.1.

ORGANIZATION_Type: see 8.8.1.

PREFERRED_NAME_Type: see 8.1.2.

SHORT_NAME_Type: see 8.1.2.

SOURCE_DOCUMENT_Type: see 8.2.2.1.

SYNONYMOUS_NAME_Type: see 8.1.2.

TEXT_Type: see 8.1.2.

TRANSLATION_Type: see 8.1.3.

Constraint specifications:

When **is_deprecated** exists, **is_deprecated_interpretation** shall exist.

Instance values of i**s_deprecated_interpretation** element shall be defined at the time where deprecation decision was taken.

7 Overview of OntoML libraries representation

The library content part of OntoML provides resources for representing instances belonging to the classes defined in a given domain ontology. Such instances may be product characterizations if they belong to a product characterization class, or product representations if they belong to functional model class.

NOTE 1 Product is understood in a very generic sense that covers all items that may be characterized by OntoML item classes.

NOTE 2 A library content may, or not, be associated with an ontology describes in an OntoML document instance.

NOTE 3 A library content provides in particular for exchanging electronic catalogues.

7.1 Root element of a library

In OntoML, every library pieces of information are gathered into a general structure that is a **LIBRARY_TYPE** XML complex type. It is illustrated in Figure 37.

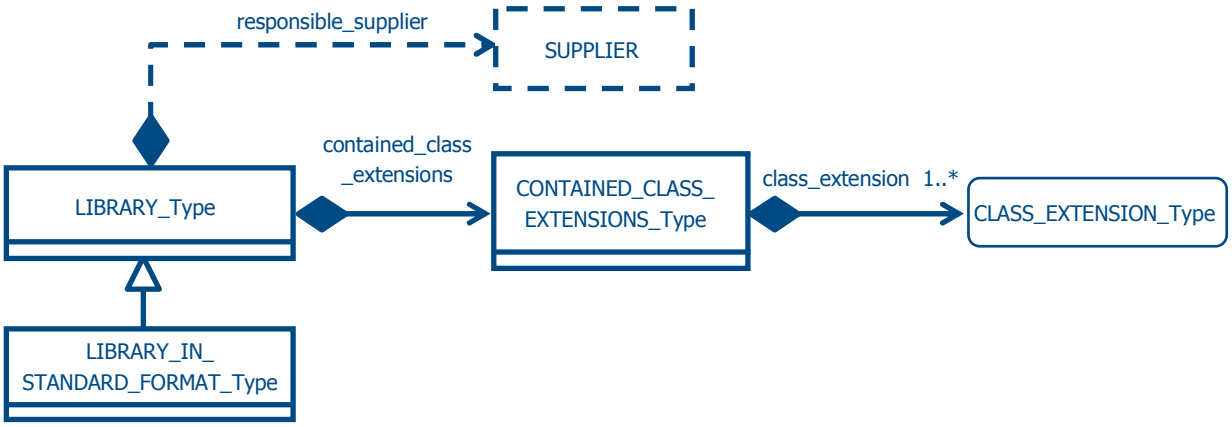

Figure 37 — Root element of library

Internal item definitions:

contained_class_extensions: the set of class extensions of the set of ontology classes.

contained_class_extensions/class_extension: a class extension contained in the dictionary.

responsible_supplier: the data supplier responsible for the library content.

Internal type definitions:

CONTAINED_CLASS_EXTENSIONS_Type: sequence of class extensions descriptions.

LIBRARY_IN_STANDARD_FORMAT_Type: a library that only uses external file protocols that are allowed either by the library integrated information model indicated by the **library_structure** XML element or the view exchange protocols referenced in the **supported_vep** XML element, both defined in the **HEADER_Type** XML complex type.

LIBRARY_Type: a container for representing library pieces of information.

External type definitions:

CLASS_EXTENSION_Type: see 7.2.

7.2 Class extension general structure

A class extension is represented by the **CLASS_EXTENSION_Type** abstract XML complex type, as illustrated in Figure 38. This type allows to specify in particular, for all kinds of class extensions, whether each instance is described by the same properties in the same order, thus as a row of a table (**table_like** XML element). It also allows to specify the set of class applicable properties that are necessary and sufficient to identify each instance belonging to the class extension. Thus, in the case of a table-like content structure, it corresponds to the key of this table. This is done by the **instance_identification** XML element as illustrated in Figure 38.

NOTE Properties that correspond to the table key shall be associated with values for all instances of the class. This is specified in the constraint specification sub-clause.

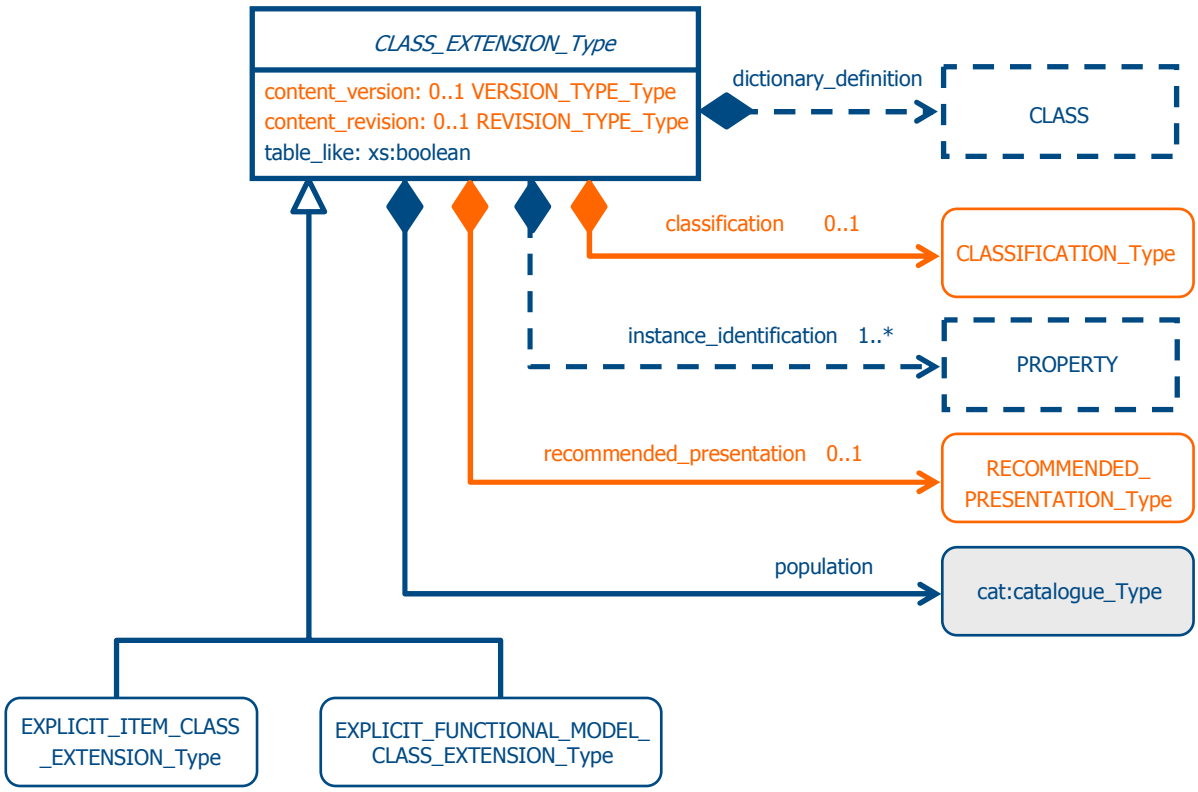

Figure 38 — General class extension structure

Internal item definitions:

classification: the possible reference to a classification of the properties used for describing class instances.

content_revision: the revision number that corresponds to the current description of the **content_version** version of a class extension.

content_version: the version number that characterizes the extension of a class, i.e., the set of all allowed instances.

dictionary_definition: the reference to the class extension dictionary definition.

instance_identification: the references to the properties that allow to identify unambiguously each instance belonging to a class.

population: the list of instances that describe the class population.

recommended_presentation: the recommended scaling factor, presentation units and value formats to be used when displaying the values of some referenced properties in the context of the referencing class.

table_like: a Boolean value that specifies whether all the class instances are characterized by the same properties in the same order, or not.

Internal type definitions:

CLASS_EXTENSION_Type: the abstract XML complex type, supertype of the various class extensions.

REVISION_TYPE_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a revision. Its value length shall not exceed 3 characters.

VERSION_TYPE_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a version. It shall contain only digits, and its value length shall not exceed 9 characters.

External type definitions:

cat:catalogue_Type: specification of instances as a set of property reference and value couples.

NOTE **cat:catalogue_Type** is defined in the ISO/TS 29002-10 product exchange format.

CLASSIFICATION_Type: see 7.2.1.

EXPLICIT_FUNCTIONAL_MODEL_CLASS_EXTENSION_Type: see 7.4.

EXPLICIT_ITEM_CLASS_EXTENSION_Type: see 7.3.

RECOMMENDED_PRESENTATION_Type: see 7.2.2.

Constraint specification:

For each instance that defines the class population, the properties that are referenced in the **instance_identification** collection shall never be associated with a null value.

From version to version of the same class, properties referenced in the **instance_identification** collection shall not change.

In the same class, whatever be the version, same values for **instance_identification** properties shall correspond to the same part.

content_version and **content_revision** shall exist together.

The **content_version** shall be incremented when, and only when, the extension of the class is changed (new instances become allowed), or previous instances are no longer allowed.

When the **content_version** is incremented, the **version** of its associated class definition (referenced by the **dictionary_definition** XML element) need not to be incremented.

The **content_revision** shall be incremented for any changes in the class extension description, but the changes that modify the allowed instances of this class.

When the **content_version** is incremented, the **content_revision** shall be reset to '0'.

If **table_like** XML element value is TRUE, the properties that describe each instance shall be the same properties given in the same order.

All the properties referenced in the **instance_identification** collection shall be applicable to the class.

The **instance_identification** collection shall not contain duplicated property references.

All the properties used to define any instance (through the **population** XML element) shall be applicable to the class

All the instances that define the class **population** shall be such as all the referenced **property_value**s whose values are **translated_string_value**s be translated in the same language(s).

All the instances that define the class **population** shall reference the same class than the one referenced by the **explicit_model_class_extension** through its inherited **dictionary_definition** XML element.

7.2.1 Property classification

Some properties participating to instance descriptions may be grouped using some classification resources. Each group is identified by an integer. The intent is to allow to process differently (on the receiving system) properties belonging to a given group.

NOTE 1 A property may belong to different groups.

NOTE 2 A property which is not associated with a classification value is not associated with any particular processing.

NOTE 3 This part of ISO 13584 does not make any assumption on how each classification value is interpreted on a receiving system. It may result from private agreement between the sender and the receiver or from latter standardization.

Properties classification is represented using the **CLASSIFICATION_Type** XML complex type as illustrated in Figure 39.

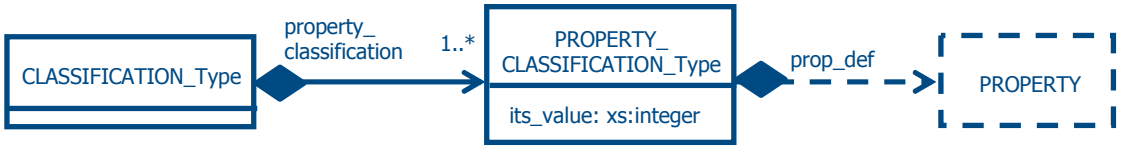

Figure 39 — Properties classification

Internal item definitions:

property_classification: the specification of the properties classifications.

property_classification/its_value: the classification value associated with the referenced **prop_def** property.

property_classification/prop_def: the reference to the property that is classified by means of the **its_value** value.

Internal type definitions:

CLASSIFICATION_Type: a property classification container.

PROPERTY_CLASSIFICATION_Type: an association that assigns a classification group to a property.

7.2.2 Properties presentation

When a particular property is used for describing instances of a particular class, it might prove useful to use a particular scaling factor, display unit and / or value format. Such a kind of recommendation is represented using **RECOMMENDED_PRESENTATION_Type** XML complex type as illustrated in Figure 40.

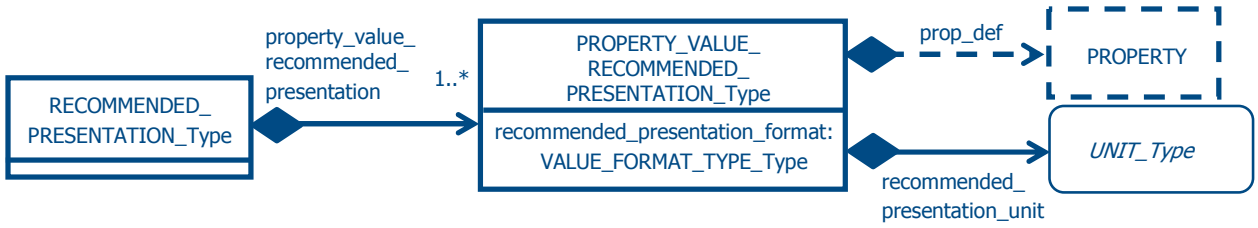

Figure 40 — Properties presentation

Internal item definitions:

property_value_recommended_representation: the specification of the properties recommended representations.

property_value_recommended_representation/prop_def: the reference to the property whose the library data supplier recommends to convert value for presentation purpose.

property_value_recommended_representation/recommended_presentation_format: the presentation format recommended by the library data supplier for presenting the values of the referenced **prop_def** property, if and only if these values are converted into the **recommended_presentation_unit** unit.

property_value_recommended_representation/recommended_presentation_unit: the particular unit in which the library data supplier recommends to convert data for presentation purpose.

Internal type definitions:

RECOMMENDED_PRESENTATION_Type: a recommended property presentation container.

PROPERTY_VALUE_RECOMMENDED_PRESENTATION_Type: the recommended property presentation format together with its presentation unit.

VALUE_FORMAT_TYPE_Type: identifies the values allowed for a value format.

NOTE **VALUE_FORMAT_TYPE_Type** values are defined according to Annex H.

External type definitions:

UNIT_Type: specification of a unit, see 8.4.

Constraint specification:

The length of a **VALUE_FORMAT_TYPE_Type** value shall not exceed 80 characters.

The unit specified by the **recommended_presentation_unit** XML element shall be compatible with the underlying data type associated to the property referenced by the **prop_def** XML element.

7.3 Simple-level library: content of classes of products

The description of products belonging to a given product characterization class is performed using the **EXPLICIT_ITEM_CLASS_EXTENSION_Type** XML complex type as illustrated in Figure 41. It provides also optional information that may be used on a receiving system to display the content of a product characterization class.

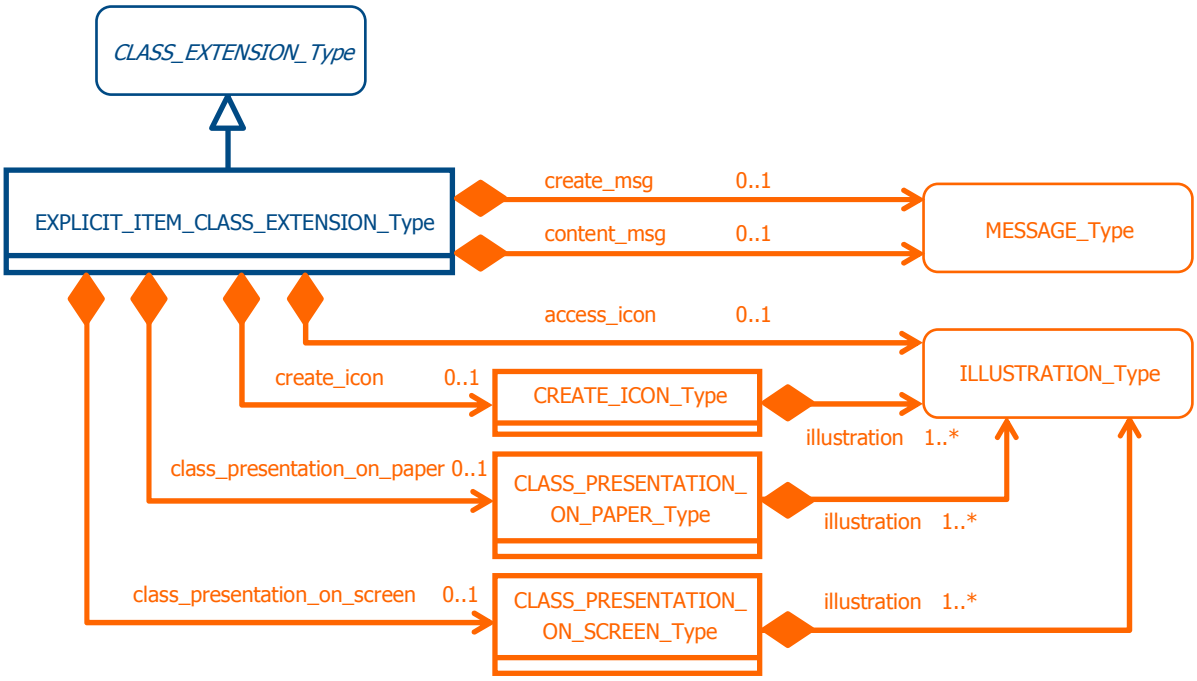

Figure 41 — Products representation structure

Internal item definitions:

access_icon: the optional icon that enables class presentation in a menu.

NOTE 1 The Icon is defined as A9 standardized size icon.

class_presentation_on_paper: the ordered set of illustrations that are recommended by the library data supplier to be presented to the user when the content of the class is presented on paper, for instance for printing the class content in a booklet.

class_presentation_on_paper/illustration: the class illustrations to be presented on paper, for instance for printing the class content in a booklet.

class_presentation_on_screen: the ordered set of illustrations that are recommended by the library data supplier to be presented to the user when the content of the class is presented on screen.

class_presentation_on_screen/illustration: the class illustrations to be displayed on screen.

content_msg: the message that describes the content of the class.

create_icon: the optional icon(s) that enable(s) visual presentation of the properties of class items and reference coordinate system.

NOTE 2 Icons are defined as A6 standardized size icons.

create_icon/illustration: the icon illustration to be displayed on screen.

create_msg: the optional message that describes the properties of a class items and their reference coordinate system.

Internal type definitions:

CLASS_PRESENTATION_ON_PAPER_Type: specifies the structure of a paper illustration.

CLASS_PRESENTATION_ON_SCREEN_Type: specifies the structure of a screen illustration.

CREATE_ICON_Type: specifies the visual presentation of the properties.

External type definitions:

CLASS_EXTENSION_Type: see 7.2.

ILLUSTRATION_Type: see 8.2.1.2.

MESSAGE_Type: see 8.2.1.3.

Constraint specification:

The class referenced by the inherited **dictionary_definition** XML element shall have as its base type **ITEM_CLASS_Type** or **ITEM_CLASS_CASE_OF_Type**.

Each **illustration** specified in the **class_presentation_on_paper illustration** collection shall set its **width** and **height** XML elements but not the **not_static_picture** value for the **kind_of_content** XML element.

Each **illustration** specified in the **class_presentation_on_screen illustration**s shall set **width** and **height** XML elements.

7.4 Advanced-level library: content of classes of product representations

Each instance of a functional model class, called a functional model, is a product representation that consists of a list of property-value pairs. The subset of these properties that is included in the **instance_identification** inherited XML element constitutes the key of these instances. **Instance_identification** contains the necessary information for identifying one instance, and discriminating it from other instances of the same class. This set of instances of the class is recorded in the **population** inherited XML element.

NOTE 1 Each functional model is a model of one (or several) product(s) of the class it is view of. The connection is done by the product property (or properties) referenced in the **required_item_values** XML element. These properies shall be properties imported from item, and they should be duplicated in both the functional model class and in the item class it is view of. These properties allow to make a joint between the item class and the functional model class.

When there is no view control variables defined in the functional view class referenced by the functional model class, the key of the functional model class, defined by the **instance_identification** inherited attribute, equals the collection of properties referenced in the **required_item_values** XML element. Thus the join operator associates with each item instance at most one functional model.

When some view control variables are defined in the functional view class referenced by the functional model class, the key of the functional model class, defined by the **instance_identification** inherited attribute, equals the union of the collection of properties referenced in the **required_item_values** XML element, and of the collection of properties referenced in the **view_control_variables** XML element of the functional view class. Thus the join operator associates with each item instance at most one functional model for each tuple of values of the set of view control variables.

EXAMPLE 1 Assume that a unique functional model class view-of is defined for the root of an item class of a library. Assume that this functional model class defines the *inventory status* of each product of each item class. The functional model class could be described by three properties: *the part number* (imported from the root item class), the *stock availability*, and the *quantity of order*. If the **required_item_values** XML element of the functional model class view-of only references the *part_number* property, then, when a product is selected, its inventory status may be automatically computed. Moreover, the set of inventory status of all the products may also be computed by the system just by a left outer joint between of this item class extension with the functional model class extension.

EXAMPLE 2 When a functional model class provides representations corresponding to several view control variables values, for each product, a particular value needs to be selected by the user for each view control variable to select the required product representation.

Functional model class extensions are represented using the **FUNCTIONAL_MODEL_CLASS_EXTENSION_Type** XML complex type as illustrated in Figure 42.

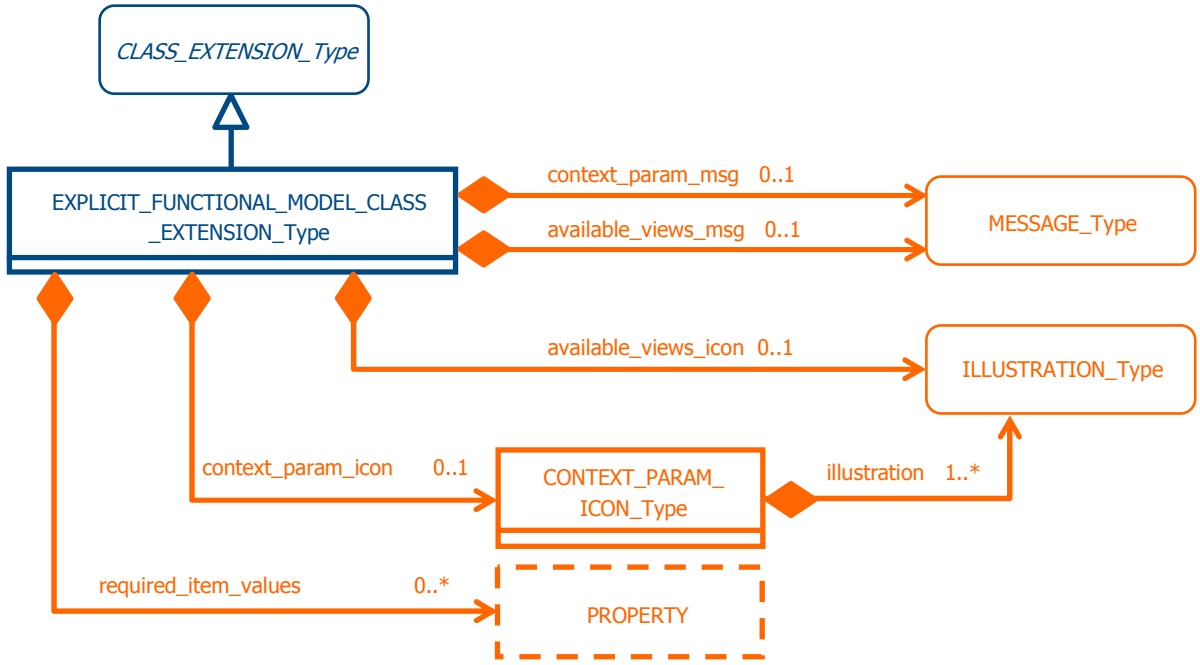

Figure 42 — Functional models structure UML diagram

Internal item definitions:

available_views_icon: possible illustrations that enable visual presentation of the various views that may be created by the functional model class.

NOTE 2 The icon is defined as an A6 standardized size icon.

available_views_msg: possible messages that describe the various views that may be created by the functional model class.

context_param_icon: possible icons that enable visual presentation of the properties that shall be assigned a value for selecting a particular view.

context_param_icon/illustration: the illustration that represents the **context_param_icon**.

NOTE 3 The illustration is defined as an A6 standardized size illustration.

context_param_msg: possible messages that explain the model properties that shall be assigned a value for selecting a particular view.

required_item_values: the item characteristics used for matching functional model instances with item class instances.

NOTE 4 The properties referenced in the **required_item_values** collection shall be represented both in the item class and in the functional model class. The matching is then done by a left outer join of the item class and the functional model class.

NOTE 5 Only the item characteristics required to instantiate the functional model class should appear.

NOTE 6 When the functional model class is not view-of a particular item class, the **required_item_values** collection is empty. When the functional model class is connected to an item class on the user system, an **A_POSTERIORI_VIEW_OF_Type** XML complex type (see 8.6.2) must be used to associate the functional model class with this item class, and to specify which properties of the functional model class must be mapped with the item class properties when making a join between item class and functional model class.

Internal type definitions:

CONTEXT_PARAM_ICON_Type: specifies the structure of context parameter icon.

External type definitions:

CLASS_EXTENSION_Type: see 7.2.

Constraint specification:

The class referenced by the inherited **dictionary_definition** XML element shall be either **FUNCTIONAL_MODEL_CLASS_Type** class or **FM_CLASS_VIEW_OF_Type** class.

If the **required_item_values** collection is not empty, the inherited **dictionary_definition** XML element shall have as its base type **FM_CLASS_VIEW_OF_Type**, and each property referenced in the **required_item_values** collection shall be **NON_DEPENDENT_P_DET_Type** properties.

If the **required_item_values** collection is not empty, the inherited **dictionary_definition** XML element shall have as its base type **FM_CLASS_VIEW_OF_Type**, and each property referenced in the **required_item_values** collection shall also belong to the **imported_properties_from_item** collection of this referenced **FM_CLASS_VIEW_OF_Type** class.

Each property referenced in the **required_item_values** collection shall also belong to the set of properties referenced in the **instance_identification** collection.

Each instance belonging to the **population** collection shall be described by all the view control variables that are defined in the functional view class referenced (**created_view** XML element) by the associated functional model class (inherited **dictionary_definition** XML element).

The inherited **instance_identification** collection shall reference both all the properties used to represent view control variables that are defined in the functional view class referenced (**created_view** XML element) by the associated functional model class (inherited **dictionary_definition** XML element), and the properties referenced in the **required_item_values** collection.

8 Other structured information elements

This clause specifies the other structured information element constructs that were referenced in Clause 7.

8.1 Translations

OntoML provides resources for translating clear text information and for managing translations.

8.1.1 Language specification

The specification of a language is twofold: the language specification, and possibly the associated country that further specify the language. It is illustrated in Figure 43.

Figure 43 — Language specification

Internal item definitions:

@country_code: the possible country code that further specifies the language code.

@language_code: the code of the language.

Internal type definitions:

COUNTRY_CODE_Type: the type of a language code. It is a string which contains 2 characters, and that defines a country according to ISO 3166-1.

LANGUAGE_CODE_Type: the type of a language code. It is a string which contains either 2 or 3 characters, and that defines a language according respectively to ISO 639-1 or ISO 639-2.

8.1.2 Translation of string valued elements

Every CIIM ontology concept is described using clear text information that may or not be translated. The corresponding information elements are:

— **preferred_names**;

— **short_names**;

— **synonymous_names**;

— **keywords**;

— **definitions, notes and remarks**;

— **source_doc_of_definition** (when it is specified as a document identifier, see 8.2.2.1.1).

OntoML provides constructs for representing these information elements as illustrated in Figure 44.

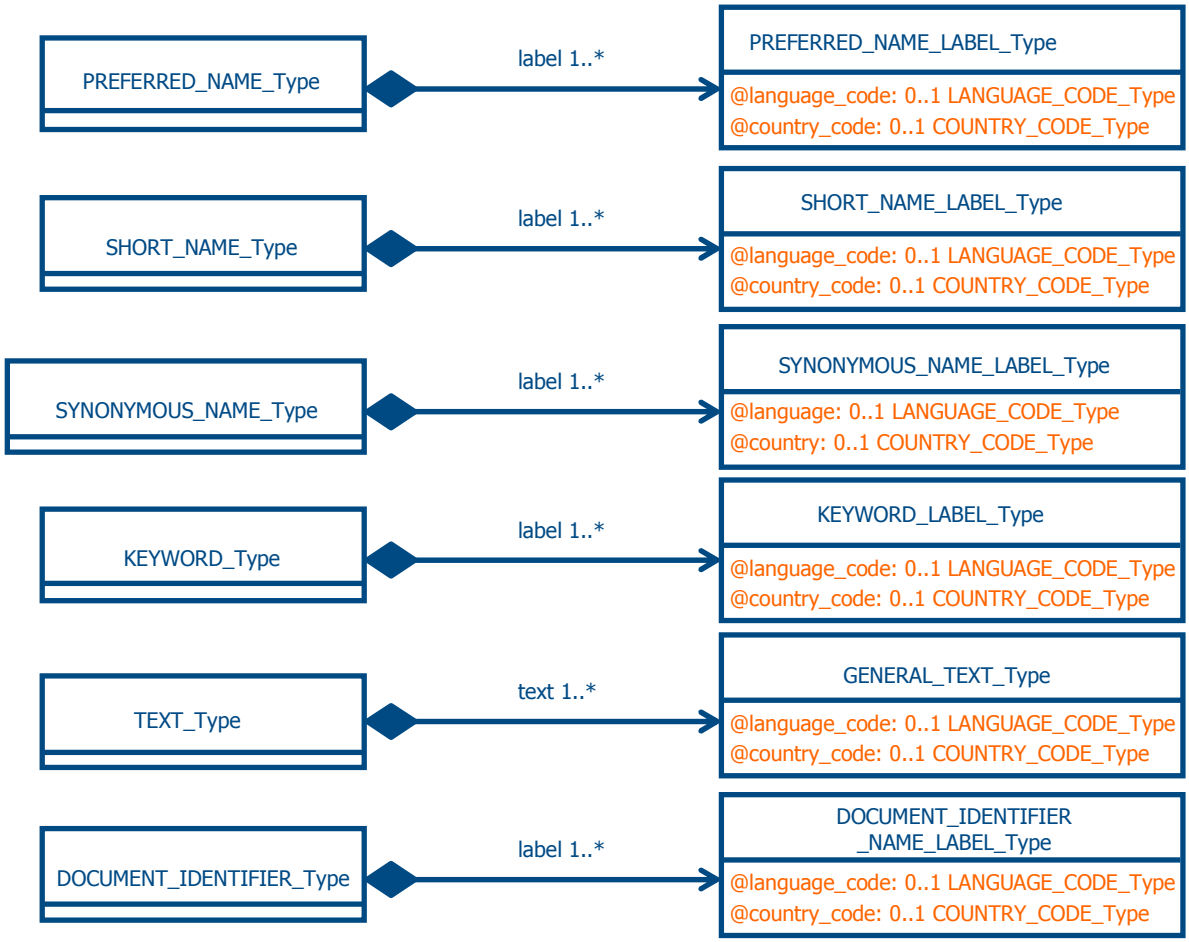

Figure 44 —Translation resources

NOTE 1 The minimum language information requirement consists in providing a value for the **language_code** XML attribute.

NOTE 2 When the ontology language is specified using the **global_language** XML element defined in the **HEADER_Type** XML complex type, neither **language_code** nor **country_code** XML attributes shall be provided.

NOTE 3 **HEADER_Type** is defined in Clause 6.5.

Internal item definitions:

label (**DOCUMENT_IDENTIFIER_Type**): defines the document labels that are possibly translated, together with their corresponding translation.

label (**KEYWORD_Type**): defines the keywords labels that are possibly translated, together with their corresponding translation.

label (**PREFERRED_NAME_Type**): defines the preferred name labels that are possibly translated, together with their corresponding translation.

label (**SHORT_NAME_Type**): defines the short name labels that are possibly translated, together with their corresponding translation.

label (**SYNONYMOUS_NAME_Type**): defines the synonymous name labels that are possibly translated, together with their corresponding translation.

text (**TEXT_Type**): defines the definitions, note and remark texts that are possibly translated, together with their corresponding translation.

Internal type definitions:

COUNTRY_CODE_Type: the type of a language code. It is a string which contains 2 characters, and that defines a country according to ISO 3166-1.

DOCUMENT_IDENTIFIER_Type: a possibly translated document identifier.

DOCUMENT_IDENTIFIER_NAME_LABEL_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a document label with its language (**@language_code** and possible **@country_code** XML attributes). Its value length shall not exceed 255 characters.

GENERAL_TEXT_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a definition, a note or a remark together with its language **@language_code** and possible **@country_code** XML attributes). Its value length is not constrained.

KEYWORD_LABEL_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a keyword with its language **@language_code** and possible **@country_code** XML attributes). Its value length shall not exceed 255 characters.

KEYWORD_Type: a possibly translated concept keyword.

LANGUAGE_CODE_Type: the type of a language code. It is a string which contains either 2 or 3 characters, and that defines a language according respectively to ISO 639-1 or ISO 639-2.

PREFERRED_NAME_LABEL_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a preferred name with its language **@language_code** and possible **@country_code** XML attributes). Its value length shall not exceed 255 characters.

PREFERRED_NAME_Type: a possibly translated concept preferred name.

SHORT_NAME_LABEL_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a short name with its language **@language_code** and possible **@country_code** XML attributes). Its value length shall not exceed 30 characters.

SHORT_NAME_Type: a possibly translated concept short name.

SYNONYMOUS_NAME_LABEL_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a synonymous name with its language **@language_code** and possible **@country_code** XML attributes). Its value length shall not exceed 255 characters.

SYNONYMOUS_NAME_Type: a possibly translated concept synonymous name.

TEXT_Type: a possibly translated concept text.

8.1.3 Translation management

Management information about the translation performed at the level of a CIIM ontology concept may be represented. For that purpose, OntoML provides the **TRANSLATION_Type** XML complex type. It is illustrated in Figure 45.

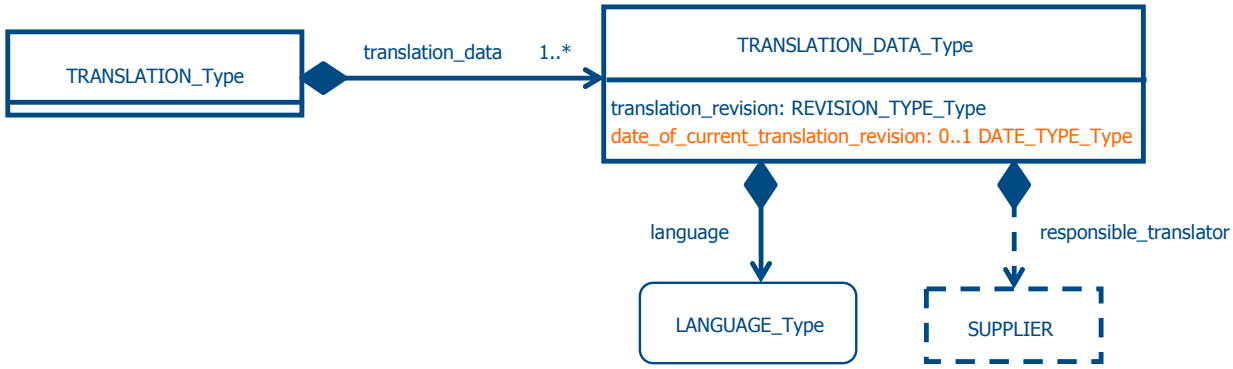

Figure 45 — Translation data structure

Internal item definitions:

translation_data: the set of translation information associated to a CIIM ontology concept.

translation_data/date_of_current_translation_revision: the date corresponding to the current revision of the translation in the **@language** language.

translation_data/language: the language for which the translation information is given.

translation_data/responsible_translator: a reference to the identifier of the organization who performed the translation in the **@language** language.

translation_data/translation_revision: the revision status of the translation in the **@language** language.

NOTE Change of **version** or change of **revision** of a dictionary element does not always require any change in their translations. If there is no change in a translation due to a change of **version** or change of **revision** of a dictionary element, the corresponding **translation_revision** shall not be changed. However any change of a translation will imply change of the corresponding **translation_revision**.

Internal type definitions:

DATE_TYPE_Type: identifies the values allowed for a date (the specific **xs:date** XML Schema datatype).

REVISION_TYPE_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a revision. Its value length shall not exceed 3 characters.

TRANSLATION_DATA_Type: translation information.

TRANSLATION_Type: translation information container.

External type definition:

LANGUAGE_Type: see 8.1.1.

8.2 External content

External contents allow to reference externally defined information from CIIM ontology concepts or from information elements. This reference may be performed in different ways:

— by specifying a URI (local or global) that references an external resource;

— by specifying a code identifying a document;

— by specifying a reference to a document or a documented graphics that is described and identified by a document ontology concept.

8.2.1 Simple-level ontology external resources

A piece of information may be provided as an external resource. This resource may be

— either a file (or a set of files) associated with the OntoML document instance and identified by a local URI,

— or an Internet resource, identified by a global URI.

Figure 46 illustrates the external resource root type, represented by the **EXTERNAL_RESOURCE_Type** abstract XML complex type.

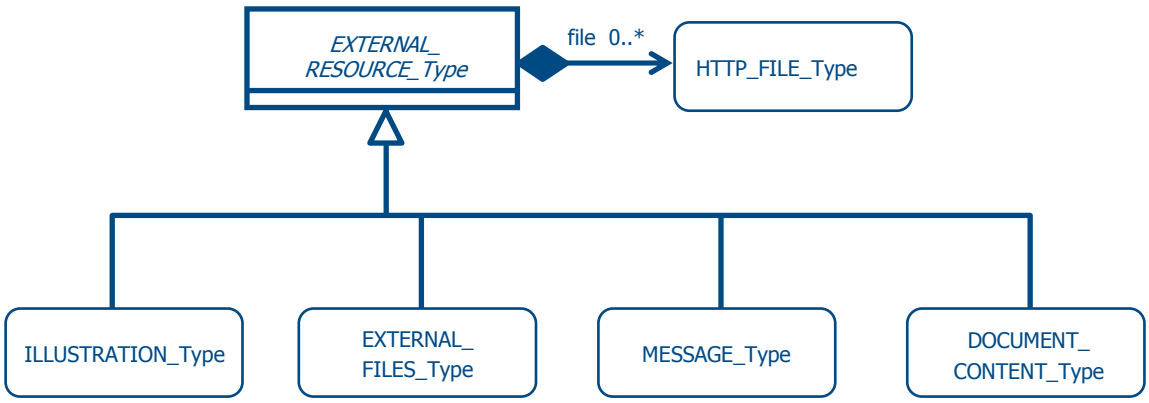

Figure 46 — Simple-level ontology external resources

Internal item definition:

file: a set of XML elements that describe and identify external resources represented by HTTP files.

Internl type definitions:

EXTERNAL_RESOURCE_Type: an abstract external resource.

External type definitions:

DOCUMENT_CONTENT_Type: see 6.7.7.

EXTERNAL_FILES_Type: see 8.2.1.4.

HTTP_FILE_Type: see 8.2.1.1.

ILLUSTRATION_Type: see 8.2.1.2.

MESSAGE_Type: see 8.2.1.3.

8.2.1.1 HTTP file

An HTTP file is the basic OntoML construct for referencing external information from an OntoML document instance.

An HTTP file is defined according to the **HTTP_FILE_Type** XML complex Type as presented in Figure 47.

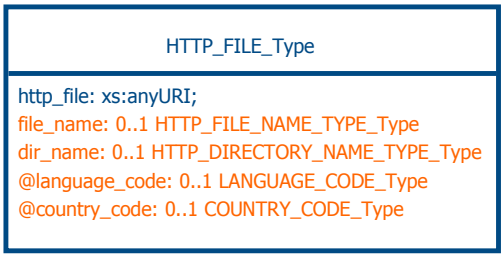

Figure 47 — Simple-level ontology external resources: HTTP file structure

Internal item definitions:

@country_code: the possible country code that further specifies the language.

@language_code: the possible language in which the information contained in the HTTP file is expressed.

dir_name: the possible target directory in which the HTTP file shall be stored on a receiving system if the http files include reference to each others.

NOTE 1 When **dir_name** exists, all directories of a same OntoML document instance are defined under a common root.

http_file: the URI locating the HTTP file.

NOTE 2 The MIME protocol interpretation completely defines the protocol to be used for processing any referenced external information.

http_file_name: the possible name of the HTTP file on a receiving system.

Internal type definitions:

COUNTRY_CODE_Type: the type of a language code. It is a string which contains 2 characters, and that defines a country according to ISO 3166-1.

HTTP_FILE_NAME_Type: the type of an **http_file_name**. Its representation fulfills constraints defined for representing URIs.

NOTE 3 URI is defined by [RFC 2396], and is amended by [RFC 2732].

HTTP_DIRECTORY_NAME_Type: the type of the name of an http file directory (**dir_name**). The length of the directory name shall not be greater than 128.

LANGUAGE_CODE_Type: the type of a language code. It is a string which contains either 2 or 3 characters, and that defines a language according respectively to ISO 639-1 or ISO 639-2.

8.2.1.2 Illustration

An illustration is an informative item that may be a schematic drawing, a realistic picture, or any other informative element that is not a static picture, whose content is defined by an http file possibly translated. An illustration may also be specified as being an *A6* or an *A9* standard format illustration. If such a standard size is not provided, the window size recommended to display the illustration may be specified. An illustration is represented by an **ILLUSTRATION_Type** XML complex type (see Figure 48).

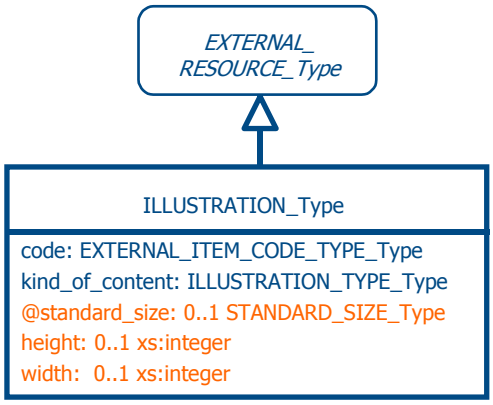

Figure 48 — Simple-level ontology external resources: illustration

Internal item definitions:

@standard_size: if provided, specifies that the illustration is either an *A6* or an *A9* standard format illustration.

code: a code that identifies the illustration.

height: in case of a non standardized format illustration, specifies the height of the window recommended by the library data supplier for viewing the illustration.

kind_of_content: the categorization of the illustration content, the value shall be schematic drawing, realistic picture or not static picture.

width: in case of a non standardized format illustration, specifies the width of the window recommended by the library data supplier for viewing the illustration.

NOTE The default **height** and **width** unit is millimeter.

Internal type definitions:

EXTERNAL_ITEM_CODE_TYPE_Type: an XML simple type specifying a string restriction: the string shall be less or equal to 18 and shall not contain any space.

ILLUSTRATION_TYPE_Type: an XML simple type specifying a string restriction: the string shall be equal either to “SCHEMATIC_DRAWING” or “REALISTIC_PICTURE” or “NOT_STATIC_PICTURE”.

STANDARD_SIZE_Type: an XML simple type specifying a string restriction: the string shall be equal either to “a6_illustration” or “a9_illustration”.

External type definitions:

EXTERNAL_RESOURCE_Type: see 8.2.1.

Constraint specification:

The illustration **code** value is unique in the class where it is used to define the illustration.

Either both **height** and **width** are defined or none is defined.

8.2.1.3 Message

A message is a short text, typically up to 255 characters, and that is supposed to be automatically displayed on the screen of a user library user within a clearly defined work context. A message is not associated with any particular window dimension. It may be provided in different languages, and it is stored in an http file (one file for each language). A message is represented by a **MESSAGE_Type** XML complex type (see Figure 49).

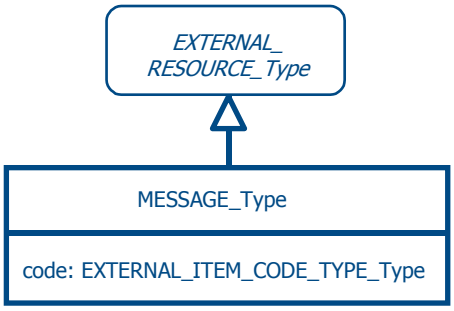

Figure 49 — Simple-level ontology external resources: message

Internal item definitions:

code: a code that identifies the message.

Internal type definitions:

EXTERNAL_ITEM_CODE_TYPE_Type: an XML simple type specifying a string restriction: the string shall be less or equal to 18 and shall not contain any space.

External type definitions:

EXTERNAL_RESOURCE_Type: see 8.2.1.

Constraint specification:

The message **code** value is unique in the class where it is used to define the illustration.

8.2.1.4 External files

An external file provides for exchanging graphical data by means of reference to external resources. It is represented by an **EXTERNAL_FILES_Type** XML complex type (see Figure 50)

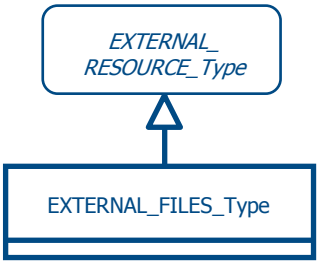

Figure 50 — Simple-level ontology external resources: external files

External type definitions:

EXTERNAL_RESOURCE_Type: see 8.2.1.

8.2.2 Source document and graphics

Documentation that may be associated to a CIIM ontology concept is twofold:

— source documents: general construct for referencing document-like documentation;

— graphics: general construct for referencing graphics-like documentation.

8.2.2.1 Source document

External resources that represent a source document are represented by a **SOURCE_DOCUMENT_Type** XML abstract complex type. It is illustrated in Figure 51.

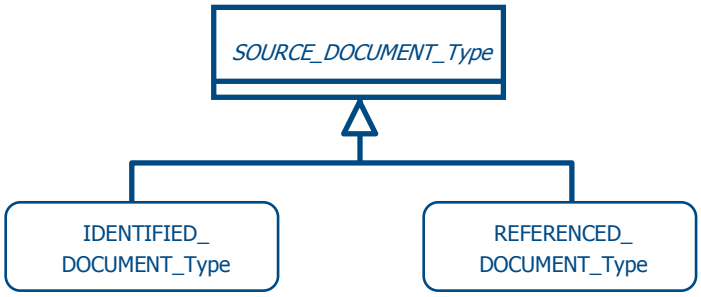

Figure 51 — External resources: source document

External type definitions:

IDENTIFIED_DOCUMENT_Type: see 8.2.2.1.1.

REFERENCED_DOCUMENT_Type: see 8.2.2.1.2.

8.2.2.1.1 Identified document

An identified document describes a document identified by its label. It is represented by an **IDENTIFIED_DOCUMENT_Type** XML complex type (see Figure 52).

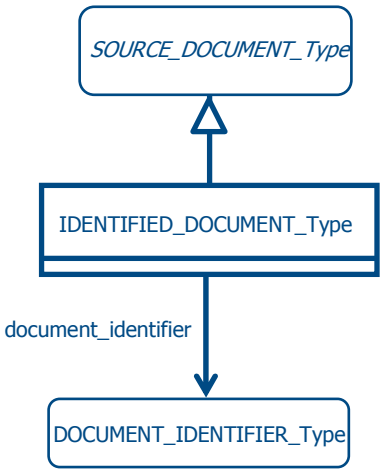

Figure 52 — External resources: identified document

Internal item definitions:

document_identifier: the label (possibly translated) of the described document.

External type definitions:

DOCUMENT_IDENTIFIER_Type: see 8.1.2.

SOURCE_DOCUMENT_Type: see 8.2.2.1.

8.2.2.1.2 Referenced document

A referenced document enables to reference a document that is described and identified by a document ontology concept. It is represented by a **REFERENCED_DOCUMENT_Type** XML complex type (see Figure 53).

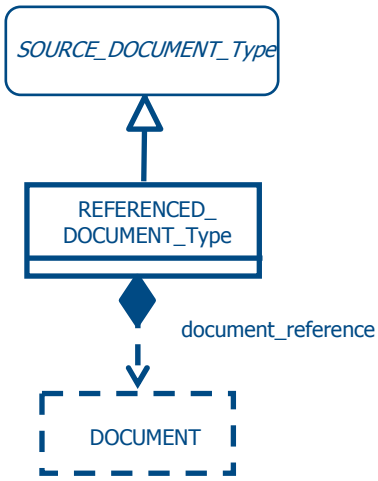

Figure 53 — External resources: referenced document

Internal item definitions:

document_reference: a reference to a document ontology concept identifier.

External type definitions:

SOURCE_DOCUMENT_Type: see 8.2.2.1.

8.2.2.2 Graphics

External resources that represent a graphics are represented by a **GRAPHICS_Type** XML abstract complex type. It is illustrated in Figure 54.

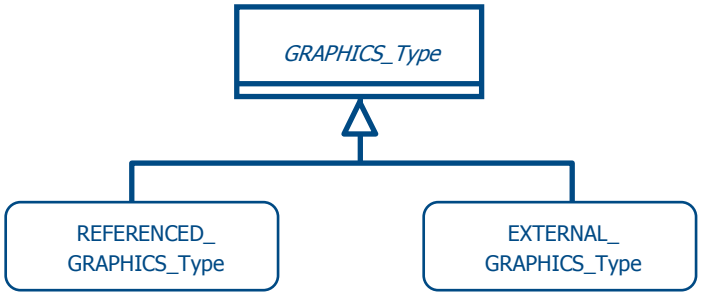

Figure 54 — External resources: graphics

External type definitions:

EXTERNAL_GRAPHICS_Type: see 8.2.2.2.1.

REFERENCED_GRAPHICS _Type: see 8.2.2.2.2.

8.2.2.2.1 External graphics

An external graphics enables to describe a graphics. It is represented by a **EXTERNAL_GRAPHICS_Type** XML complex type (see Figure 55).

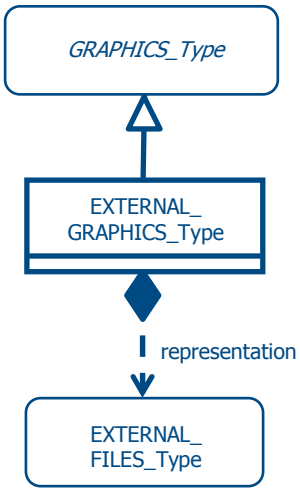

Figure 55 — External resources: external graphics

Internal item definitions:

representation: the external file(s) specifying the external graphics.

External type definitions:

EXTERNAL_FILES_Type: see 8.2.1.4.

GRAPHICS_Type: see 8.2.2.2.

Constraint specifications:

An **EXTERNAL_GRAPHICS_Type** shall provide a value to the inherited **file** XML element.

8.2.2.2.2 Referenced graphics

A referenced graphics enables to reference a graphics that is described and identified by a document ontology concept. It is represented by a **REFERENCED_GRAPHICS_Type** XML complex type (see Figure 56).

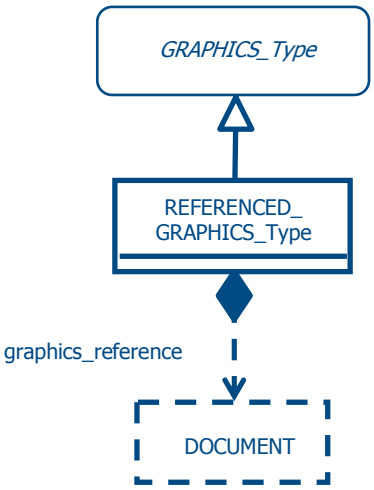

Figure 56 — External resources: referenced graphics

Internal item definitions:

graphics_reference: a reference to a document ontology concept identifier.

External type definitions:

GRAPHICS_Type: see 8.2.2.2.

Constraint specifications:

A **REFERENCED_GRAPHICS_Type** shall not provide a value to the inherited **file** XML element.

8.3 Data type system

OntoML provides resources for describing data types that allow to constrain domain of values assigned to properties. Simple (Boolean, real, string, …) to complex data types (named type, collections, classes, …) are available. Every datatype is defined as a subtype of the **ANY_TYPE_Type** XML complex type. Figure 57 gives an overview of the main data types available in OntoML.

NOTE 1 The lexical representation of the value of each datatype belonging to the OntoML data type system is defined in Annex D. In the next subclauses, each data type specification references its corresponding lexical representation.

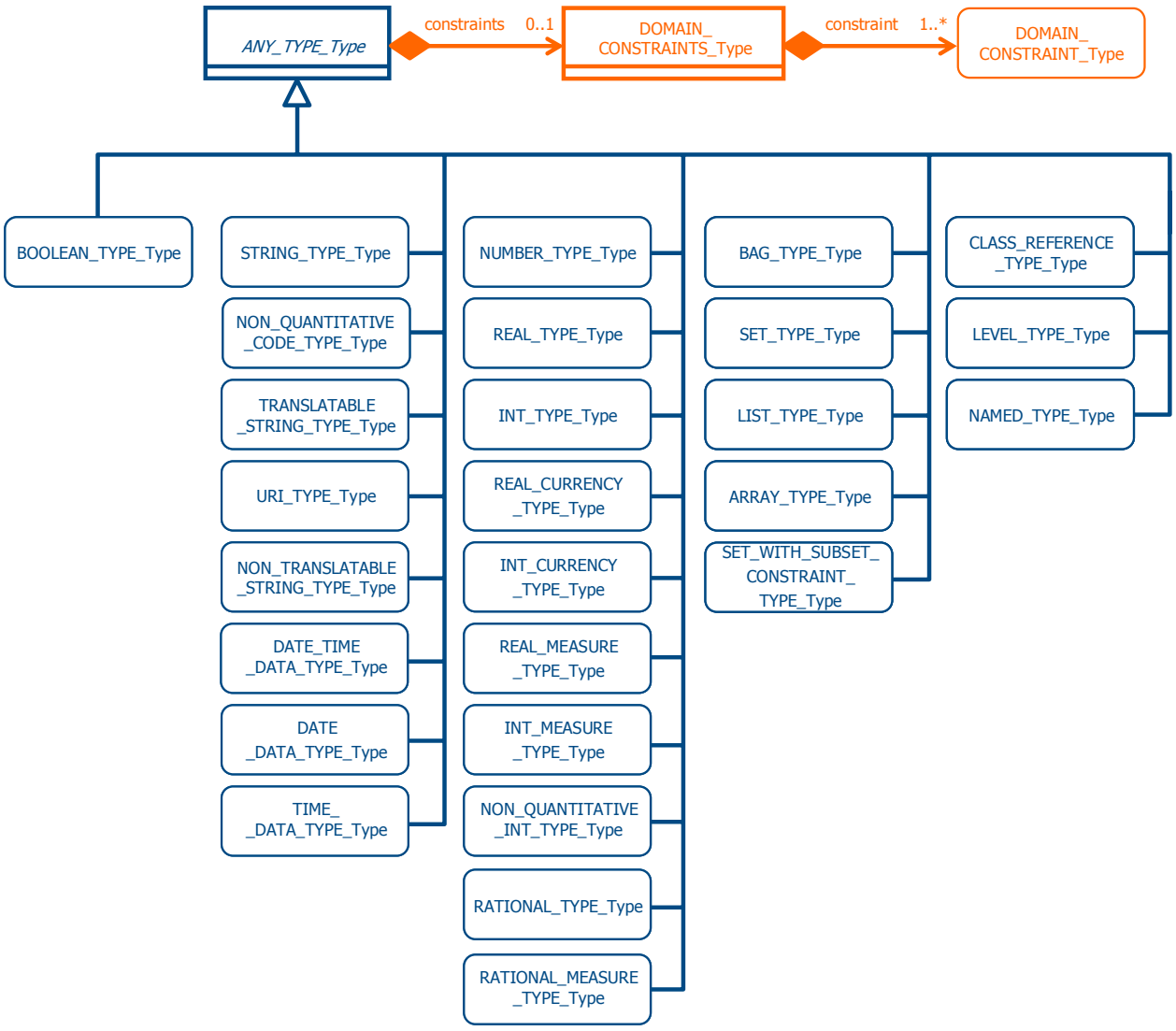

Figure 57 — OntoML datatype system

Internal element definitions:

constraints: the set of domain constraints that restrict the domain of values of the data type.

NOTE 2 Each domain constraint in the **constraints** collection must be fulfilled. Thus the **constraints** collection specifies a conjunction of constraints

Internal type definitions:

DOMAIN_CONSTRAINTS_Type: domain constraint collection.

External type definitions:

ARRAY_TYPE_Type: array collection, see 8.3.9.4.

BAG_TYPE_Type: bag collection, see 8.3.9.1.

BOOLEAN_TYPE_Type: Boolean type, see 8.3.1.

CLASS_REFERENCE_TYPE_Type: reference to an identified class type, see 8.3.10.

DATE_TIME_DATA_TYPE_Type: date and time type, see 8.3.3.

DATE_DATA_TYPE_Type: date type, see 8.3.3.

DOMAIN_CONSTRAINT_Type: data type constraint, see 8.5.3.3.

INT_CURRENCY_TYPE_Type: integer currency type, see 8.3.6.

INT_MEASURE_TYPE_Type: integer without unit type, see 8.3.7.

INT_TYPE_Type: integer without unit type, see 8.3.5.

LEVEL_TYPE_Type: level type, see 8.3.11.

LIST_TYPE_Type: list collection, see 8.3.9.3.

NAMED_TYPE_Type: reference to an identified data type, see 8.3.12.

NON_QUANTITATIVE_CODE_TYPE_Type: enumeration of string codes type, see 8.3.4.

NON_QUANTITATIVE_INT_TYPE_Type: enumeration of integer codes type, see 8.3.8.

NON_TRANSLATABLE_STRING_TYPE_Type: non translatable string, see 8.3.2.

NUMBER_TYPE_Type: number type, see 8.3.5.

RATIONAL_TYPE_Type: rational type, see 8.3.5.

RATIONAL_MEASURE_TYPE_Type: rational measure type, see 8.3.7.

REAL_TYPE_Type: real without unit type, see 8.3.5.

REAL_CURRENCY_TYPE_Type: real currency type, see 8.3.6.

REAL_MEASURE_TYPE_Type: real with unit type, see 8.3.7.

URI_TYPE_Type: string representing a URI, see 8.3.2.

SET_TYPE_Type: set collection, see 8.3.9.2.

SET_WITH_SUBSET_CONSTRAINT_TYPE_Type: explicitly defined set collection, see 8.3.9.

STRING_TYPE_Type: string type, see 8.3.2.

TIME_DATA_TYPE_Type: time type, see 8.3.3.

TRANSLATABLE_STRING_TYPE_Type: translatable string type, see 8.3.2.

8.3.1 Boolean type

A Boolean type is defined using the **BOOLEAN_TYPE_Type** XML complex type. It is represented in Figure 58.

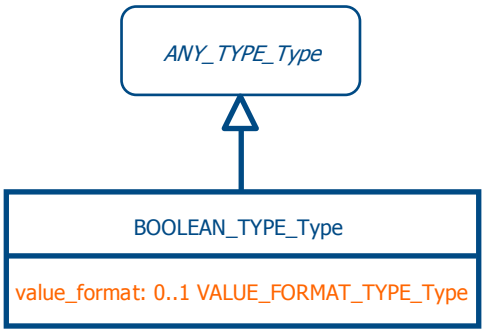

Figure 58 — Boolean type structure

Internal item definition:

value_format: the specification of the type and length of the recommended presentation for displaying the value of a property. If present, this attribute provides guidance to the system about how the value should be displayed.

Internal type definition:

VALUE_FORMAT_TYPE_Type: identifies the values allowed for a value format.

NOTE 1 No standard value is defined for this element.

Internal subtype definitions:

BOOLEAN_TYPE_Type: provides for values of properties or user types that are of type Boolean.

NOTE 2 The lexical representation of a value whose data type is **BOOLEAN_TYPE_type** is defined in Annex D, Clause D.1.1.

External type definition:

ANY_TYPE_Type: see 8.3.

Constraint specification:

The length of a **VALUE_FORMAT_TYPE_Type** value shall not exceed 80 characters.

8.3.2 String types

The value domain defined by strings is a sequence of any kind of characters. A basic OntoML string type is defined by the **STRING_TYPE_Type** XML complex type. It may be further qualified as a localized string (**TRANSLATABLE_STRING_TYPE_Type** XML complex type), a string that is not translatable (**NON_TRANSLATABLE_STRING_TYPE_Type** XML complex type) or a string that represents an HTTP address (**URI_TYPE_Type** XML complex type). Figure 59 illustrates these resources.

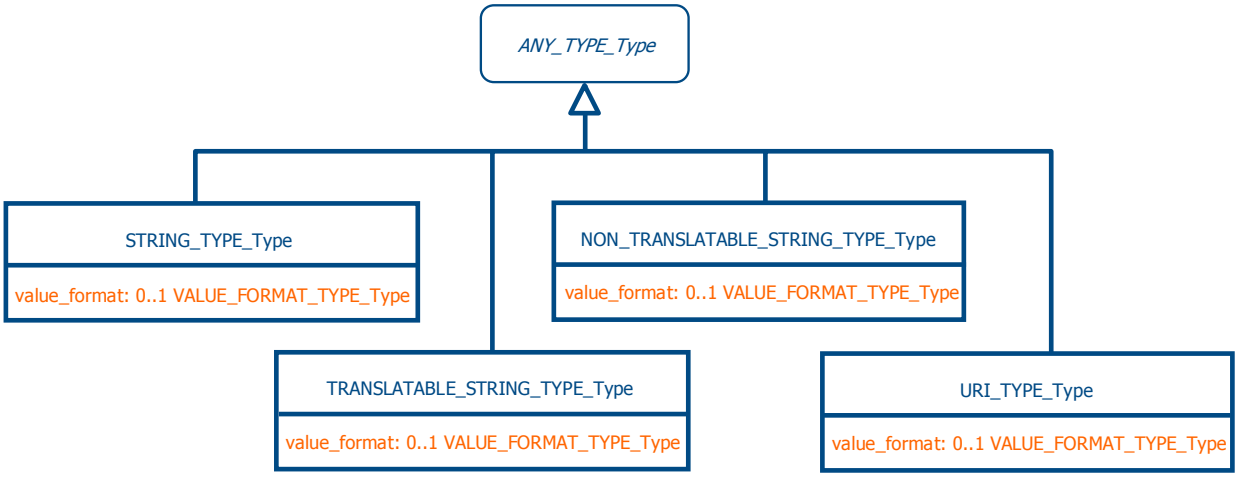

Figure 59 — String types structure

Internal item definition:

value_format (**STRING_TYPE_Type**, **TRANSLATABLE_STRING_TYPE_Type**, **NON_TRANSLATABLE_STRING_TYPE_Type**, **URI_TYPE_Type**): the specification of the type and length of the recommended presentation for displaying the value of a property. If present, this attribute provides guidance to the system about how the value should be displayed.

NOTE 1 **value_format** must not be used to restrict string type definitions.

NOTE 2 If **value_format** is not compatible with the associated string type definition, **value_format** will be ignored.

NOTE 3 If any string pattern constraint (see 8.5.3.3.2) applies to the value of a string type, then it takes precedence on the **value_format**.

Internal type definition:

VALUE_FORMAT_TYPE_Type: identifies the values allowed for a value format.

NOTE 4 **VALUE_FORMAT_TYPE_Type** values are defined according to Annex H.

Internal subtype definitions:

NON_TRANSLATABLE_STRING_TYPE_Type: provides for values of properties or user types that are of type string, but that are represented in the same way in any languages.

NOTE 5 The lexical representation of a value whose data type is **NON_TRANSLATABLE_STRING_TYPE_Type** is defined in Annex D, Clause D.1.6.

STRING_TYPE_Type: provides for values of properties or user types that are of type string.

NOTE 6 The lexical representation of a value whose data type is **STRING_TYPE_Type** is defined in Annex D, Clause D.1.2.

TRANSLATABLE_STRING_TYPE_Type: provides for values of properties or user types that are of type string, but that are supposed to be represented as different strings in different languages.

NOTE 7 The lexical representation of a value whose data type is **TRANSLATABLE_STRING_TYPE_Type** is defined in Annex D, Clause D.1.4.

NOTE 8 The **source_language** XML element (defined in the **PROPERTY_DET** XML complex type, see 6.7.4) of a property whose data type (**domain** XML element) is defined as a **TRANSLATABLE_STRING_TYPE_Type** plays the role of the root language in which string property values are converted to identify same values when represented in different language.

URI_TYPE_Type: provides for values of properties or user types that are of type string, but represents a URI.

NOTE 9 The lexical representation of a value whose data type is **URL_TYPE_Type** is defined in Annex D, Clause D.1.5.

External type definition:

ANY_TYPE_Type: see 8.3.

Constraint specification:

The length of a **VALUE_FORMAT_TYPE_Type** value shall not exceed 80 characters.

8.3.3 Date and time type

The value domain defined by date and time types is a sequence of characters that fulfills the representation rules defined in ISO 8601. A date and time type may represent either both a date and a time (**DATE_TIME_DATA_TYPE_Type** XML complex type), or a single date (**DATE_DATA_TYPE_Type** XML complex type) or a single time (**TIME_DATA_TYPE_Type** XML complex type). Figure 60 illustrates these resources.

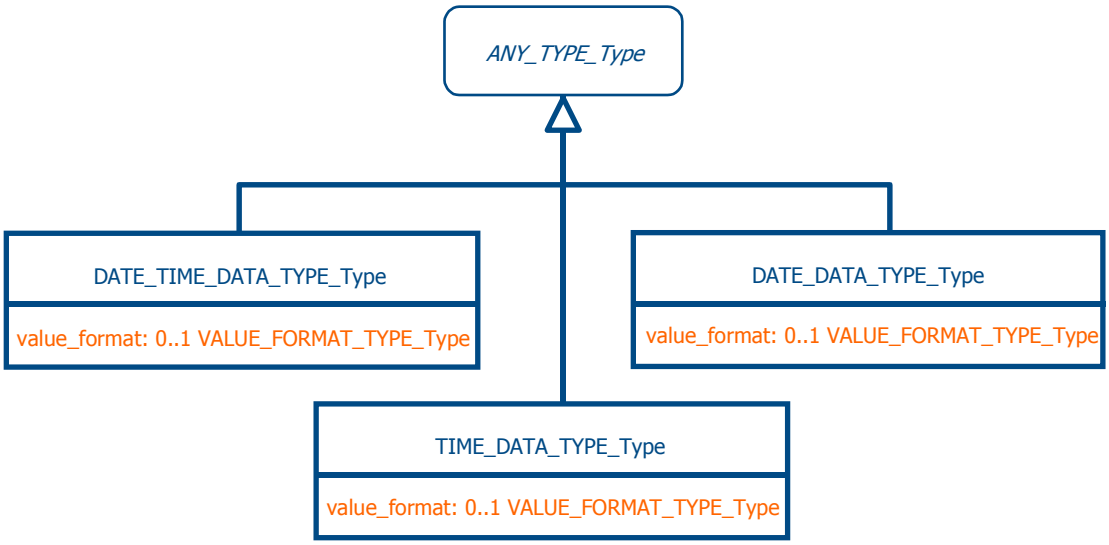

Figure 60 — Date and time types structure

Internal item definition:

value_format (**DATE_TIME_DATA_TYPE_Type**, **TIME_DATA_TYPE_Type** and **DATE_DATA_TYPE_Type**): the specification of the type and length of the recommended presentation for displaying the value of a property. If present, this attribute provides guidance to the system about how the value should be displayed.

NOTE 1 **value_format** must not be used to restrict date and time type definitions.

NOTE 2 If **value_format** is not compatible with the associated date and time type definition, **value_format** will be ignored.

NOTE 3 If any string pattern constraint (see 8.5.3.3.2) applies to the value of a date and time type, then it takes precedence on the **value_format**.

Internal type definition:

VALUE_FORMAT_TYPE_Type: identifies the values allowed for a value format.

NOTE 4 **VALUE_FORMAT_TYPE_Type** values are defined according to Annex H.

Internal subtype definitions:

DATE_DATA_TYPE_Type: provides for values of properties or user types that are of type date.

NOTE 5 The lexical representation of a value whose data type is **DATE_DATA_TYPE_Type** is defined in Annex D, Clause D.1.8.

DATE_TIME_DATA_TYPE_Type: provides for values of properties or user types that are of type date-time.

NOTE 6 The lexical representation of a value whose data type is **DATE_TIME_DATA_TYPE_Type** is defined in Annex D, Clause D.1.7.

TIME_DATA_TYPE_Type: provides for values of properties or user types that are of type time

NOTE 7 The lexical representation of a value whose data type is **TIME_DATA_TYPE_Type** is defined in Annex D, Clause D.1.9.

External type definition:

ANY_TYPE_Type: see 8.3.

Constraint specification:

The length of a **VALUE_FORMAT_TYPE_Type** value shall not exceed 80 characters.

8.3.4 Enumeration of string codes type

A string that shall take its value among a set of enumerated codes is represented by the **NON_QUANTITATIVE_CODE_TYPE_Type** XML complex type. Each of those codes is associated to a meaning. It is illustrated in Figure 61.

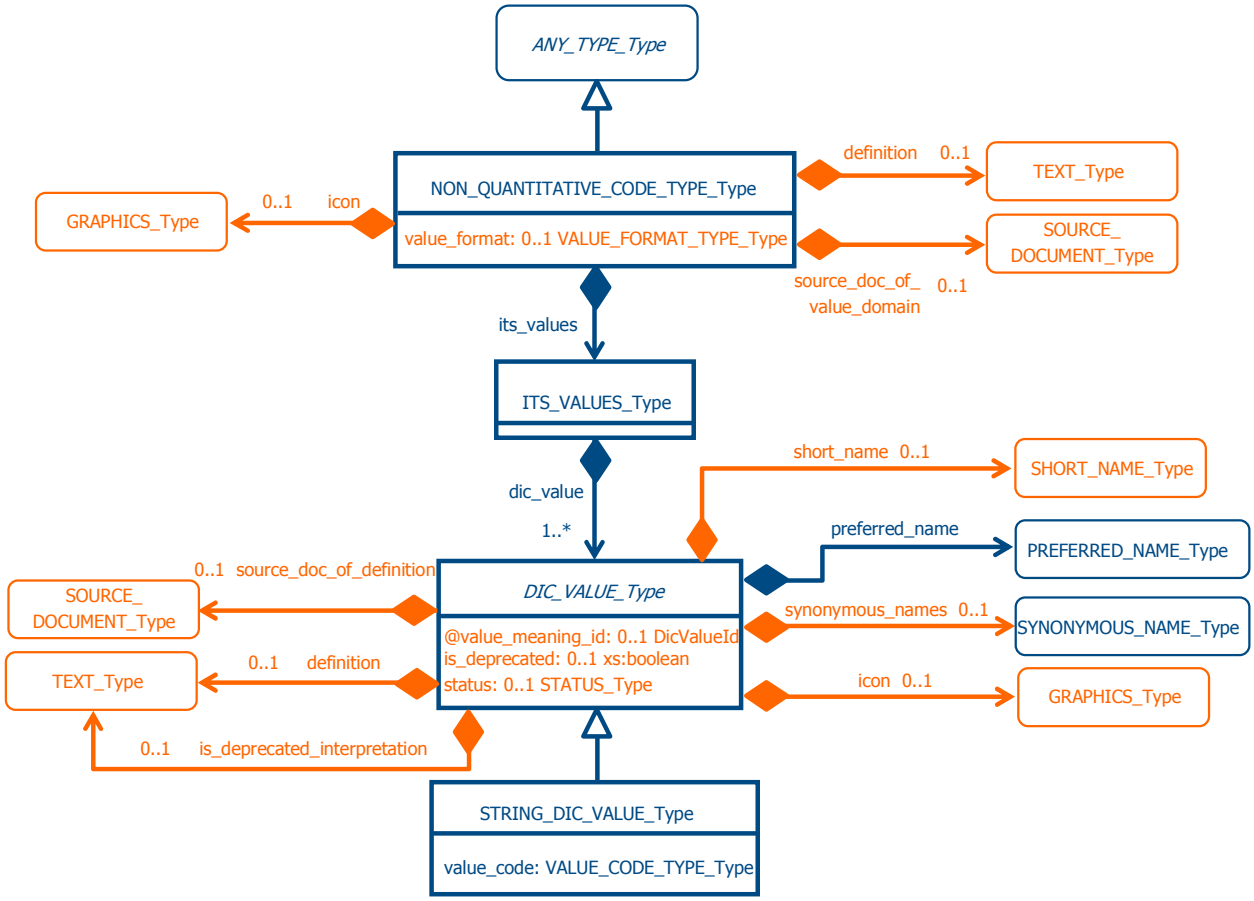

Figure 61 — Enumeration of string codes type structure

The enumeration items are defined in an XML element container called **its_values** whose content model is defined by is a **ITS_VALUES_Type** XML complex type. Each enumeration item is represented through a **dic_value** XML element. Its own content model is defined as a **DIC_VALUE_Type**, an more precisely, in case of the specification of an enumeration of string codes, by its specific **STRING_DIC_VALUE_Type** XML complex type subtype.

Internal item definition:

definition: the text describing this enumeration, possibly translated.

icon: a graphics representing the description associated with the enumeration.

its_values: enumeration items container.

its_values/dic_value: enumeration item.

NOTE 1 Each **dic_value** XML element content model is represented as a **STRING_DIC_VALUE_Type** XML complex type.

its_values/dic_value/definition: the text describing this enumeration item, possibly translated.

its_values/dic_value/icon: a graphics representing the description associated with the enumeration item names.

its_values/dic_value/is_deprecated: a Boolean that specifies, when true, that the enumeration item shall no longer be used.

NOTE 2 When **is_deprecated** is not defined, the **dic_value** is not deprecated.

NOTE 3 Deprecated **dic_value**s are left in the **its_values** collection for upward compatibility reasons.

its_values/dic_value/is_deprecated_interpretation: specify the deprecation rationale and how instance values of the deprecated element should be interpreted.

NOTE 4 Instance values of the deprecated element shall be defined at the time where deprecation decision was taken.

its_values/dic_value/preferred_name: the name of the enumeration item that is preferred for use, possibly translated.

its_values/dic_value/short_name: the abbreviation of the preferred name, possibly translated.

its_values/dic_value/source_doc_of_definition: the possible source document from which the enumeration item definition comes.

its_values/dic_value/status: defines the life cycle state of the enumeration item.

NOTE 5 Allowed **status** values are defined by private agreement between the dictionary supplier and dictionary users.

NOTE 6 If the **status** XML element is not provided, and if the enumeration item is not deprecated as denoted by a possible **is_deprecated** XML element, then the enumeration item has the same standardization status as the whole ontology into which it is used. In particular, if the ontology is standardized, this enumeration item is part of the current edition of the standard.

its_values/dic_value/synonymous_names: the set of synonymous names of the preferred name, possibly translated.

its_values/dic_value/value_code: the enumeration item value.

its_values/dic_value/@value_meaning_id: an identifier that is a global identifier of the value, independently of the value domain in which it is included.

NOTE 7 This identifier allows to reuse the same **dic_value** definition in various domains.

source_doc_of_value_domain: the possible source document from which the enumeration definition comes.

value_format: the specification of the type and length of the recommended presentation for displaying the value of a property. If present, this attribute provides guidance to the system about how the value should be displayed.

NOTE 8 **value_format** must not be used to restrict an enumeration of string code type definition.

NOTE 9 If **value_format** is not compatible with the associated enumeration of string code type definition, **value_format** will be ignored.

NOTE 10 If any string pattern constraint (see 8.5.3.3.2) applies to the value of an enumeration of string codes type, then it takes precedence on the **value_format**.

Internal type definition:

STATUS_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a status.

VALUE_FORMAT_TYPE_Type: identifies the values allowed for a value format.

NOTE 11 **VALUE_FORMAT_TYPE_Type** values are defined according to Annex H.

Internal subtype definition:

NON_QUANTITATIVE_CODE_TYPE_Type: provides for values of properties or user types that are of type enumeration of string codes.

NOTE 12 The lexical representation of a value whose data type is **NON_QUANTITATIVE_CODE_TYPE_Type** is defined in Annex D, Clause D.1.3.

External type definition:

ANY_TYPE_Type: see 8.3.

GRAPHICS_Type: see 8.2.2.2.

PREFERRED_NAME_Type: see 8.1.2.

SHORT_NAME_Type: see 8.1.2.

SOURCE_DOCUMENT_Type: see 8.2.2.1.

SYNONYMOUS_NAME_Type: see 8.1.2.

TEXT_Type: see 8.1.2.

Constraint specification:

Instance values of i**s_deprecated_interpretation** element shall be defined at the time where deprecation decision was taken.

The length of a **VALUE_FORMAT_TYPE_Type** value shall not exceed 80 characters.

8.3.5 Numeric types

A numeric value may be represented as a general value (**NUMBER_TYPE_Type**), or as a real (**REAL_TYPE_Type**) or as an integer (**INT_TYPE_Type**) value or as a rational value (**RATIONAL_TYPE_Type**).

Representation of numerics is given in Figure 62.

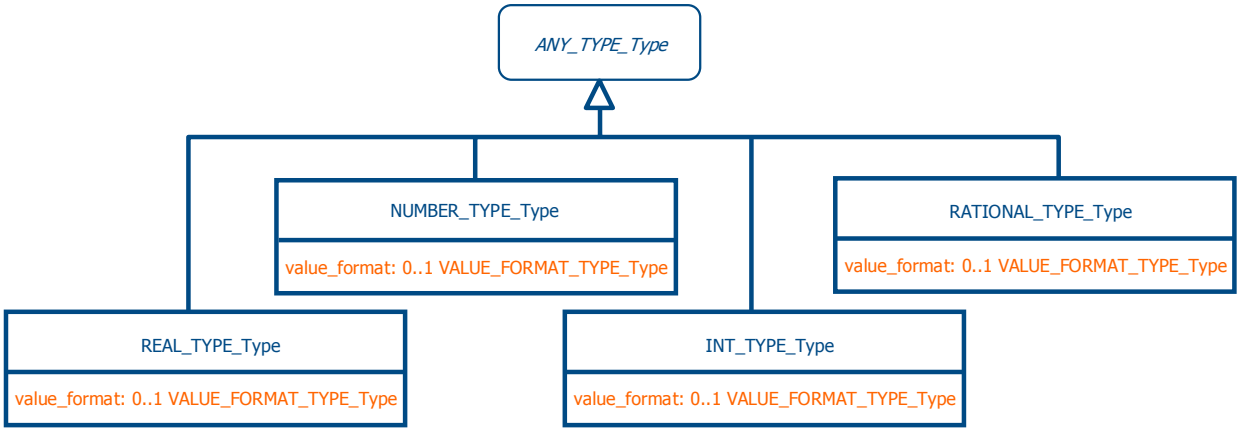

Figure 62 — Numeric types structure

Internal item definition:

value_format (**NUMBER_TYPE_Type**, **REAL_TYPE_Type**, **INT_TYPE_Type, RATIONAL_TYPE_Type**): the specification of the type and length of the recommended presentation for displaying the value of a property. If present, this attribute provides guidance to the system about how the value should be displayed.

NOTE 1 **value_format** must not be used to restrict a numeric type definition.

NOTE 2 If **value_format** is not compatible with the associated numeric type definition, **value_format** will be ignored.

NOTE 3 If any string pattern constraint (see 8.5.3.3.2) applies to the value of a numeric type, then it takes precedence on the **value_format**.

Internal type definition:

VALUE_FORMAT_TYPE_Type: identifies the values allowed for a value format.

NOTE 4 **VALUE_FORMAT_TYPE_Type** values are defined according to Annex H.

Internal subtype definitions:

INT_TYPE_Type: provides for values of properties or user types that are of type integer.

NOTE 5 The lexical representation of a value whose data type is **INT_TYPE_Type** is defined in Annex D, Clause D.1.12.

NUMBER_TYPE_Type: provides for values of properties or user types that are of type number.

NOTE 6 The lexical representation of a value whose data type is **NUMBER_TYPE_Type** is defined in Annex D, Clause D.1.10.

RATIONAL_TYPE_Type: provides for values of properties or user types that are of type rational.

NOTE 7 The lexical representation of a value whose data type is **RATIONAL_TYPE_Type** is defined in Annex D, Clause D.1.13.

REAL_TYPE_Type: provides for values of properties or user types that are of type real.

NOTE 8 The lexical representation of a value whose data type is **REAL_TYPE_Type** is defined in Annex D, Clause D.1.11.

External type definition:

ANY_TYPE_Type: see 8.3.

Constraint specification:

The length of a **VALUE_FORMAT_TYPE_Type** value shall not exceed 80 characters.

8.3.6 Numeric currency types

Both integer and real value domains may represent a currency. The former is represented by the OntoML **INT_CURRENCY_TYPE_Type** XML complex type, the latter by the OntoML **REAL_CURRENCY_TYPE_Type** XML complex type. Figure 63 Illustrates the numeric currency types representation.

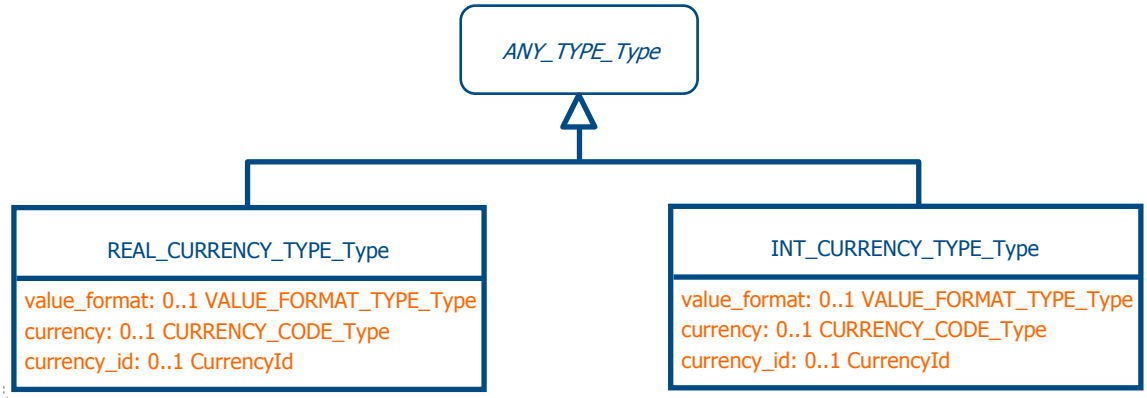

Figure 63 — Numeric currency types structure

Internal item definition:

currency (**REAL_CURRENCY_TYPE_Type**, **INT_CURRENCY_TYPE_Type**): the possible associated code of the described currency.

NOTE 1 **currency** is expressed according ISO 4217.

NOTE 2 When not defined, the **currency** is intended to be explicitly represented at the library level.

currency_id (**REAL_CURRENCY_TYPE_Type**, **INT_CURRENCY_TYPE_Type**): the possible identifier of the currency associated to the described currency.

NOTE 3 When both **currency** and **currency_id** are provided, **currency** takes precedence.

NOTE 4 If the value of a property whose domain is a currency is exchanged as a single number, this means that this value is expressed in the **currency** or **currency_id** currency.

value_format (**REAL_CURRENCY_Type**, **INT_CURRENCY_Type**): the specification of the type and length of the recommended presentation for displaying the value of a property. If present, this attribute provides for guidance to the system about how the value should be displayed.

NOTE 5 **value_format** must not be used to restrict a currency type definition.

NOTE 6 If **value_format** is not compatible with the associated currency type definition, **value_format** will be ignored.

NOTE 7 If any string pattern constraint (see 8.5.3.3.2) applies to the value of a currency type, then it takes precedence on the **value_format**.

Internal type definition:

CURRENCY_CODE_Type: a string (**xs:string** XML Schema datatype) that represents the values allowed for a currency. This string length shall be equal to 3 characters.

EXAMPLE Currency values could be "CHF" for Swiss Francs, "CNY" for Yuan Renminbi (Chinese), "JPY" for Yen (Japanese), "SUR" for SU Rouble, "USD" for US Dollars, "EUR" for Euros etc ...

CurrencyId: see 9.1.4.

VALUE_FORMAT_TYPE_Type: identifies the values allowed for a value format.

NOTE 8 **VALUE_FORMAT_TYPE_Type** values are defined according to Annex H.

Internal subtype definitions:

INT_CURRENCY_TYPE_Type: provides for values of properties or user types that are of type integer currency.

NOTE 9 The lexical representation of a value whose data type is **INT_CURRENCY_TYPE_Type** is defined in Annex D, Clause D.1.15.

REAL_CURRENCY_TYPE_Type: provides for values of properties or user types that are of type real currency.

NOTE 10 The lexical representation of a value whose data type is **REAL_CURRENCY_TYPE_Ttype** is defined in Annex D, Clause D.1.16.

External type definition:

ANY_TYPE_Type: see 8.3.

Constraint specifications:

Either a **currency** is provided, or a **currency_id** or both.

The length of a **VALUE_FORMAT_TYPE_Type** value shall not exceed 80 characters.

8.3.7 Numeric measure types

Integer, real and rational value domains may represent a measure. The first is represented by the OntoML **INT_MEASURE_TYPE_Type** XML complex type, the second by the OntoML **REAL_MEASURE_TYPE_Type** XML complex type, and the third by the **RATIONAL_MEASURE_TYPE_Type** XML complex type. Figure 64 illustrates the numeric measure types representation.

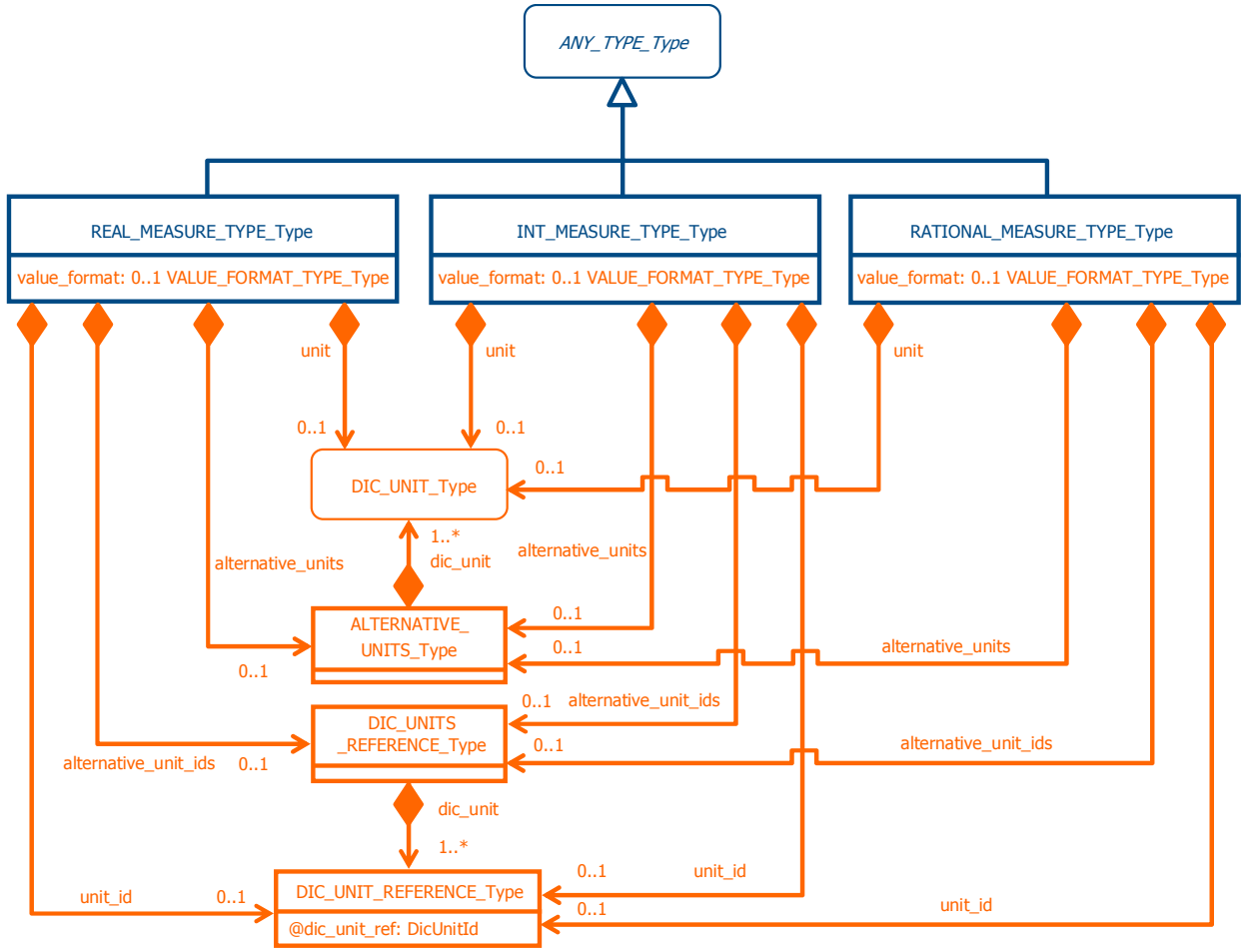

Figure 64 — Numeric measure types structure

Internal item definition:

@dic_unit_ref (**DIC_UNIT_REFERENCE_Type**): the reference to the dictionary unit.

alternative_units (**REAL_MEASURE_TYPE_Type**, **INT_MEASURE_TYPE_Type, RATIONAL_MEASURE_TYPE_Type**): the list of other units that may be used to express the value of the property whose value domain is a measure.

alternative_unit_ids (**REAL_MEASURE_TYPE_Type**, **INT_MEASURE_TYPE_Type, RATIONAL_MEASURE_TYPE_Type**): the list of identifiers of other units that may be used to express the value of the property whose value domain is a measure.

NOTE 1 When the value of a property whose domain is a measure is evaluated in a unit either defined by means of **alternative_units** or identified by means of **alternative_unit_ids**, its value cannot be represented as a single real. It needs to be represented as a pair (value, unit).

NOTE 2 The list order is used to ensure that **alternative_units** and **alternative_unit_ids**, if both exist define the same unit in the same order.

dic_unit (**ALTERNATIVE_UNITS_Type**): the specification of an alternative unit.

dic_unit (**DIC_UNITS_REFERENCE_Type**): the specification of an reference to an alternative unit.

unit (**REAL_MEASURE_TYPE_Type**, **INT_MEASURE_TYPE_Type, RATIONAL_MEASURE_TYPE_Type**): the default unit of reference associated to the described measure.

NOTE 3 If the value of a property, whose value domain is a measure, is exchanged as a single number, this means that this value is expressed in this defined unit.

unit_id (**REAL_MEASURE_TYPE_Type**, **INT_MEASURE_TYPE_Type, RATIONAL_MEASURE_TYPE_Type**): the identifier of the default unit of reference associated to the described measure.

NOTE 4 When both **unit** and **unit_id** are provided, **unit** takes precedence.

NOTE 5 If the value of a property whose domain is a measure is exchanged as a single number, this means that this value is expressed in the **unit** or **unit_id** unit of measure.

value_format (**REAL_MEASURE_TYPE_Type**, **INT_MEASURE_TYPE_Type, RATIONAL_MEASURE_TYPE_Type**): the specification of the type and length of the recommended presentation for displaying the value of a property. If present, this attribute provides guidance to the system about how the value should be displayed.

NOTE 6 **value_format** must not be used to restrict a numeric measure type definition.

NOTE 7 If **value_format** is not compatible with the associated numeric measure type definition, **value_format** will be ignored.

NOTE 8 If any string pattern constraint (see 8.5.3.3.2) applies to the value of a measure type, then it takes precedence on the **value_format**.

Internal type definition:

ALTERNATIVE_UNITS_Type: the set of possible alternative units.

DIC_UNIT_Type: the specification of a dictionary unit, see 8.4.

DIC_UNITS_REFERENCE_Type: the specification of a set of references to some unit identifiers.

DIC_UNIT_REFERENCE_Type: the specification of a reference to a unit identifier.

NOTE 9 The unit identifier is built according to the rules defined in ISO/TS 29002-5.

VALUE_FORMAT_TYPE_Type: identifies the values allowed for a value format.

NOTE 10 **VALUE_FORMAT_TYPE_Type** values are defined according to Annex H.

Internal subtype definitions:

INT_MEASURE_TYPE_Type: provides for values of properties or user types that are of type integer measure.

NOTE 11 The lexical representation of a value whose data type is **INT_MEASURE_TYPE_Type** is defined in Annex D, Clause D.1.17.

RATIONAL_MEASURE_TYPE_Type: provides for values of properties or user types that are of type rational measure.

EXAMPLE Screw diameter: 4 1/8 inches.

NOTE 12 The lexical representation of a value whose data type is **RATIONAL_MEASURE_TYPE_Type** is defined in Annex D, Clause D.1.18.

REAL_MEASURE_TYPE_Type: provides for values of properties or user types that are of type real measure.

NOTE 13 The lexical representation of a value whose data type is **REAL_MEASURE_TYPE_Type** is defined in Annex D, Clause D.1.16.

External type definition:

ANY_TYPE_Type: see 8.3.

DIC_UNIT_Type: see 8.4.

Constraint specifications:

Either a **unit** is provided, or a **unit_id** or both.

If both **alternative_units** and **alternative_unit_ids** collections are provided, they shall have the same length.

Each **dic_unit** described in the **alternative_units** collection shall have a **string_representation**.

The length of a **VALUE_FORMAT_TYPE_Type** value shall not exceed 80 characters.

8.3.8 Enumeration of integer codes type

An integer that shall take its value among a set of enumerated codes is represented by the **NON_QUANTITATIVE_INT_TYPE_Type** XML complex type. Each of those codes is associated to a meaning. It is illustrated in Figure 65.

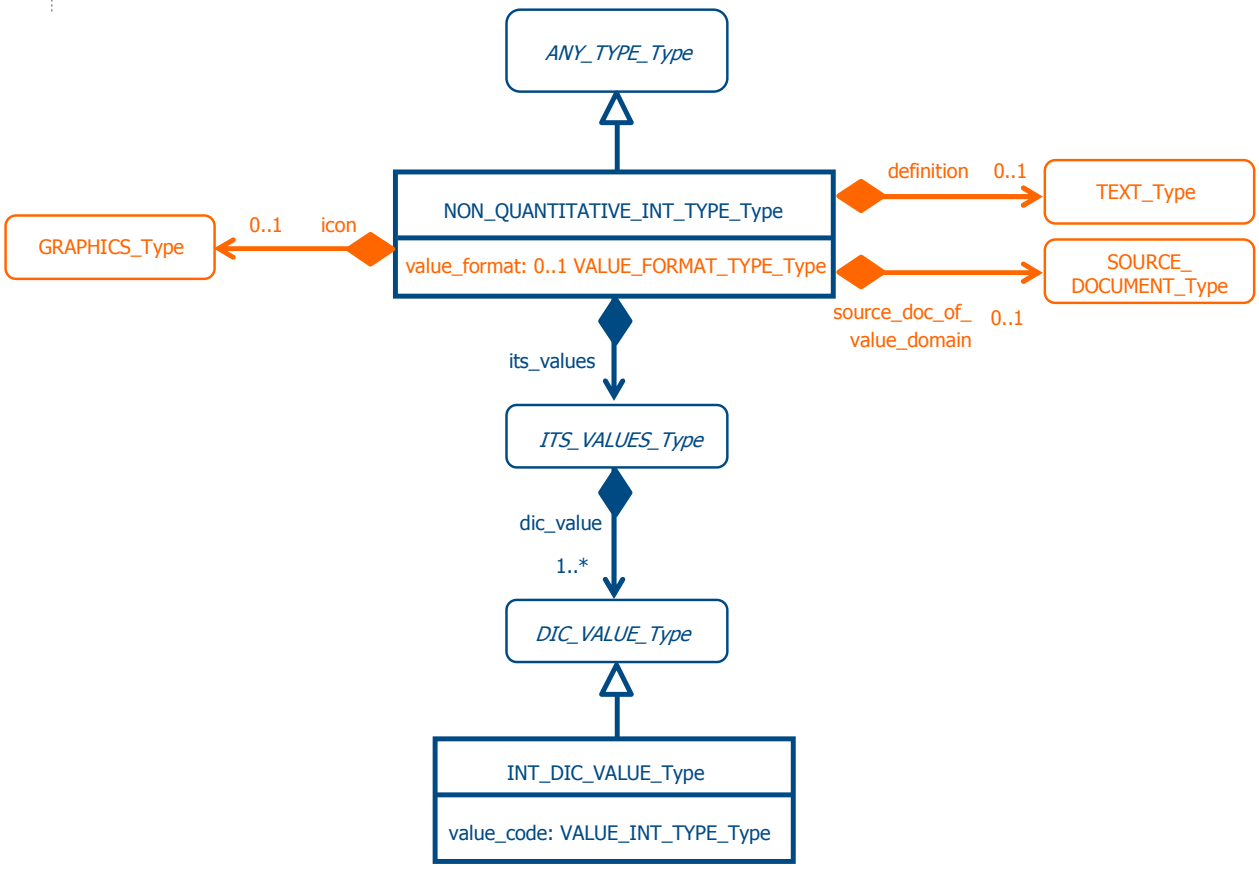

Figure 65 — Enumeration of integer codes type structure

The enumeration items are defined in an XML element container called **its_values** whose content model is defined by is a **ITS_VALUES_Type** XML complex type. Each enumeration item is represented through the **dic_value** XML element. Its own content model is defined as a **DIC_VALUE_Type**, an more precisely, in case of the specification of an enumeration of integer codes, by its specific **INT_DIC_VALUE_Type** XML complex type subtype.

Internal item definition:

definition: the text describing this enumeration, possibly translated.

icon: a graphics representing the description associated with the enumeration.

its_values: enumeration items container.

its_values/dic_value/value_code: the enumeration item integer value.

NOTE 1 Each **dic_value** XML element content model (defined in the **ITS_VALUES_Type** XML complex type) is represented as a **INT_DIC_VALUE_Type** XML complex type.

source_doc_of_value_domain: the possible source document from which the enumeration definition comes.

value_format: the specification of the type and length of the recommended presentation for displaying the value of a property. If present, this attribute provides guidance to the system about how the value should be displayed.

NOTE 2 **value_format** must not be used to restrict a numeric measure type definition.

NOTE 3 If **value_format** is not compatible with the associated numeric measure type definition, **value_format** will be ignored.

NOTE 4 If any string pattern constraint (see 8.5.3.3.2) applies to the value of an enumeration of integer codes type, then it takes precedence on the **value_format**.

Internal type definition:

GRAPHICS_Type: an XML abstract complex type representing an external resource that is a graphics.

VALUE_FORMAT_TYPE_Type: identifies the values allowed for a value format.

NOTE 5 **VALUE_FORMAT_TYPE_Type** values are defined according to Annex H.

Internal subtype definition:

NON_QUANTITATIVE_INT_TYPE_Type: provides for values of properties or user types that are of type enumeration of integer codes.

NOTE 6 The lexical representation of a value whose data type is **NON_QUANTITATIVE_INT_TYPE_Type** is defined in Annex D, Clause D.1.19.

External type definition:

ANY_TYPE_Type: see 8.3.

DIC_VALUE_Type: see 8.3.3.

ITS_VALUES_Type: see 8.3.3.

SOURCE_DOCUMENT_Type: see 8.2.2.1.

TEXT_Type: see 8.1.2.

Constraint specification:

The length of a **VALUE_FORMAT_TYPE_Type** value shall not exceed 80 characters.

8.3.9 Collection types

Collections provide for the definition of data types that may be expressed as lists, sets, bags, arrays or constrained subsets of any kind of values. Five kind of collections are distinguished: bag, set, list, array and set with a subset constraint.

8.3.9.1 Bag type

A bag is an unordered collection of values that may contain duplicates A bag type is represented as a **BAG_TYPE_Type** XML complex type as illustrated in Figure 66.

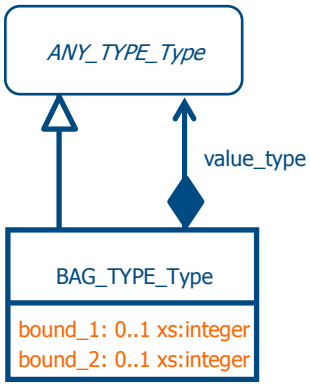

Figure 66 — Bag type structure

Internal item definition:

value_type: type of value (simple or complex) which is used for each element of the bag.

bound_1: the possible minimal cardinality of the bag.

bound_2: the possible maximal cardinality of the bag.

Internal subtype definition:

BAG_TYPE_Type: provides for values of properties or user types that are of type unordered collection of typed values with possible duplicates.

NOTE The lexical representation of a value whose data type is **BAG_TYPE_Type** is defined in Annex D, Clause D.1.20.

External type definition:

ANY_TYPE_Type: see 8.3.

Constraint specifications:

If **bound_1** is defined, it is greater or equal to 0. Otherwise, its default value is equal to 0.

If **bound_2** is defined, it is greater than 0. Otherwise, its default value is equal to unknown (unbounded).

If **bound_2** is defined, **bound_1** is also defined, and **bound_2** is greater than **bound_1**.

8.3.9.2 Set type

A set is an unordered collection of values that does not contain duplicates. A set type is represented as a **SET_TYPE_Type** XML complex type as illustrated in Figure 67.

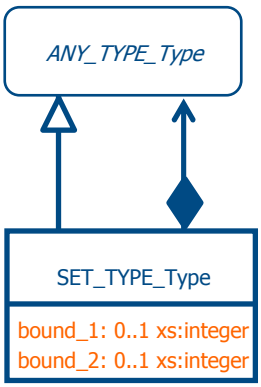

Figure 67 — Set type structure

Internal item definition:

value_type: type of value (simple or complex) which is used for each element of the set.

bound_1: the possible minimal cardinality of the set.

bound_2: the possible maximal cardinality of the set.

Internal subtype definition:

SET_TYPE_Type: provides for values of properties or user types that are of type unordered collection of typed values without duplicates.

NOTE The lexical representation of a value whose data type is **SET_TYPE_Type** is defined in Annex D, Clause D.1.21.

External type definition:

ANY_TYPE_Type: see 8.3.

Constraint specifications:

If **bound_1** is defined, it is greater or equal to 0. Otherwise, its default value is equal to 0.

If **bound_2** is defined, it is greater than 0. Otherwise, its default value is equal to unknown (unbounded).

If **bound_2** is defined, **bound_1** is also defined, and **bound_2** is greater than **bound_1**.

8.3.9.3 List type

A list is an ordered collection of values, possibly unique. A list type is represented as a **LIST_TYPE_Type** XML complex type as illustrated in Figure 68.

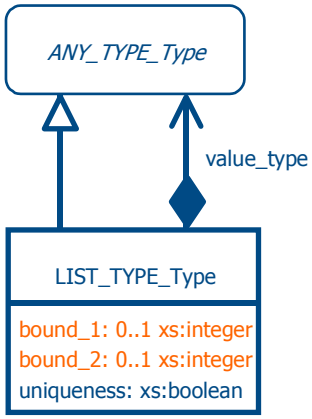

Figure 68 — List type structure

Internal item definition:

value_type: type of value (simple or complex) which is used for each element of the list.

bound_1: the possible minimal cardinality of the set.

bound_2: the possible maximal cardinality of the set.

uniqueness: a Boolean flag to indicate whether all elements of the list shall be unique (true) or whether duplicates are allowed (false).

Internal subtype definition:

LIST_TYPE_Type: provides for values of properties or user types that are of type ordered collection of typed values, possibly unique.

NOTE The lexical representation of a value whose data type is **LIST_TYPE_Type** is defined in Annex D, Clause D.1.22.

External type definition:

ANY_TYPE_Type: see 8.3.

Constraint specifications:

If **bound_1** is defined, it is greater or equal to 0. Otherwise, its default value is equal to 0.

If **bound_2** is defined, it is greater than 0. Otherwise, its default value is equal to unknown (unbounded).

If **bound_2** is defined, **bound_1** is also defined, and **bound_2** is greater or equal than **bound_1**.

8.3.9.4 Array type

An array is an ordered collection of possibly unique and optional values, of fixed length, whose members are indexed by a range of consecutive integers. An array type is represented as an **ARRAY_TYPE_Type** XML complex type as illustrated in Figure 69.

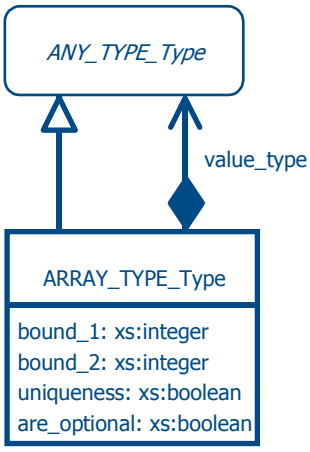

Figure 69 — Array type structure

Internal item definition:

value_type: type of value (simple or complex) which is used for each element of the array.

bound_1: the integer that defines the low index of the defined array type.

bound_2: the integer that defines the upper index of the defined array type.

uniqueness: a Boolean flag that indicates whether all elements of the array shall be present (false) or whether some elements of the array may be missing (true).

are_optional: a Boolean flag that indicates whether all elements of the array shall be present (false) or whether some elements of the array may be missing (true).

Internal subtype definition:

ARRAY_TYPE_Type: provides for values of properties or user types that are of type ordered and indexed collection of typed values, possibly unique and optional.

NOTE The lexical representation of a value whose data type is **ARRAY_TYPE_Type** is defined in Annex D, Clause D.1.23.

External type definition:

ANY_TYPE_Type: see 8.3.

Constraint specifications:

bound_1 shall be less than or equal to **bound_2**.

8.3.9.5 Set with subset constraint type

A set with a subset constraint is an unordered collection of values that does not contain duplicates and for which a subset of a minimal size to a maximal size may be extracted. A set with a subset constraint type is represented as a **SET_WITH_SUBSET_CONSTRAINT_TYPE_Type** XML complex type as illustrated in Figure 70.

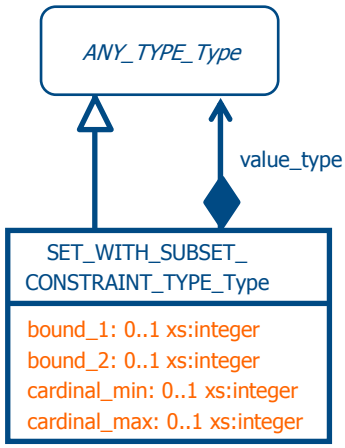

Figure 70 — Set with a subset constraint type structure

Internal item definition:

value_type: type of value (simple or complex) which is used for each element of the array.

bound_1: the integer that defines the low index of the defined set with a subset constraint type.

bound_2: the integer that defines the upper index of the defined set with a subset constraint type.

cardinal_min: the minimal size of the subsets that may be extracted.

cardinal_max: the maximal size of the subsets that may be extracted.

Internal subtype definition:

SET_WITH_SUBSET_CONSTRAINT_TYPE_Type: provides for values of properties or user types that are of type unordered collection of values that does not contain duplicates and for which a subset of a minimal size to a maximal size may be extracted.

NOTE The lexical representation of a value whose data type is **SET_WITH_SUBSET_CONSTRAINT_Type** is defined in Annex D, Clause D.1.24.

External type definition:

ANY_TYPE_Type: see 8.3.

Constraint specifications:

If **bound_1** is defined, it is greater or equal to 0. Otherwise, its default value is equal to 0.

If **bound_2** is defined, it is greater than 0. Otherwise, its default value is equal to unknown (unbounded).

If **bound_2** is defined, **bound_1** is also defined, and **bound_2** is greater than **bound_1**.

cardinal_min is less or equal than **cardinal_max**.

If **bound_1** and **cardinal_min** are defined**, cardinal_min** is not greater than **bound_1**.

If **bound_2** and **cardinal_max** are defined**, cardinal_max** is not greater than **bound_2**.

8.3.10 Class reference type

A class reference type allows to define a property value domain that is a class instance.

NOTE 1 A property whose type is a class reference type establishes a relationship between both classes. This relationship may be, for instance, a composition relationship.

EXAMPLE A *bolted assembly* may be constituted of a *screw* and a *nut*. Assuming that *bolted assembly*, *screw* and *nut* are parts families, the *bolted assembly* constituents may be represented firstly by defining two properties in the *bolted assembly* class (respectively *its_screw* and *its_nut*), and secondly by assigning to these properties a value domain that would be a **CLASS_REFERENCE_TYPE_Type**, referencing respectively the *screw* class and the *nut* class.

A class reference type is represented as a **CLASS_REFERENCE_TYPE_Type** XML complex type as it is illustrated in Figure 71.

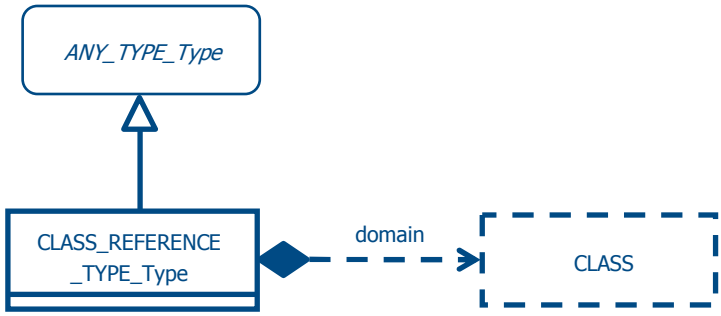

Figure 71 — Instance value domain structure

Internal item definition:

domain: a reference to the class ontology concept that defines the value domain of the data type.

Internal subtype definition:

CLASS_REFERENCE_TYPE_Type: provides for values of properties or user types that are of type instance of a class.

NOTE 2 The lexical representation of a value whose data type is **CLASS_REFERENCE_TYPE_Type** is defined in Annex D, Clause D.1.25.

External type definition:

ANY_TYPE_Type: see 8.3.

8.3.11 Level type

A is a complex type indicating that the value of a property consists of one up to four real measure or integer measure values which define a characteristic of an item in the fixed sequence: min, nom, typ, max.

— **min**: lowest value specified of a quantity, established for a specified set of operating conditions at which a component, device or equipment is operable and performs according to specified requirements;

— **nom**: value of a quantity used to designate and identify a component, device, equipment, or system;

— **type**: commonly encountered value of a quantity used for specification purposes, established for a specified set of operating conditions of a component, device, equipment, or system;

— **max**: highest value specified of a quantity, established for a specified set of operating conditions at which a component device or equipment is operable and performs according to specified requirements.

NOTE 1 The nominal value is generally a rounded value.

EXAMPLE A 12 V (nominal) car battery has 6 cells with a typical voltage of about 2.2 V each, giving a typical battery voltage of about 13.5 V. On charge, the voltage may reach a maximum of about 14.5 V but it is considered fully discharged when the voltage falls below a minimum of 12.5 V.

NOTE 2 It is advised that the use of the level type is restricted to those properties that are applicable in domains where the reporting of multiple values on a single characteristic is recognized as common practice and requested, as is true for the electronic component industry.

It is represented as a **LEVEL_TYPE_Type** XML complex type as it is illustrated in Figure 72.

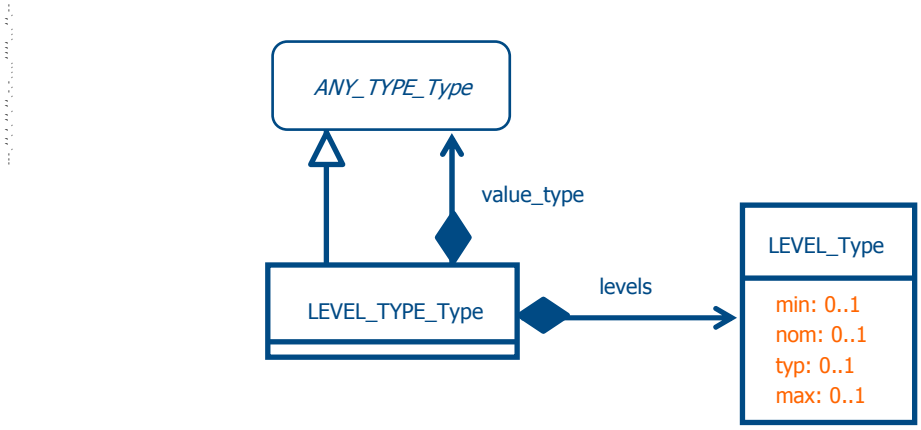

Figure 72 — Levels value domain structure

Internal item definition:

levels: the list of qualifiers that are associated with the property.

max: the maximal qualifier assigned to the property value domain.

min: the minimal qualifier assigned to the property value domain.

nom: the nominal qualifier assigned to the property value domain.

typ: the typical qualifier assigned to the property value domain.

value_type: type of value which is associated to the level specification.

NOTE 3 There isn't any content model associated to the defined qualifiers. Therefore, no datatype is assigned to the XML elements corresponding to each qualifier.

Internal item definition:

LEVEL_Type: the specification of the possible levels.

NOTE 4 Each level is represented as an optional and empty XML element.

Internal subtype definition:

LEVEL_TYPE_Type: provides for values of properties or user types that are of type qualified numeric values.

NOTE 5 The lexical representation of a value whose data type is **LEVEL_TYPE_Type** is defined in Annex D, Clause D.1.26.

External type definition:

ANY_TYPE_Type: see 8.3.

Constraint specifications:

The underlying data type defined by the **value_type** XML element is a numeric measure type as defined in Clause 8.3.7.

At least one qualifier is defined.

8.3.12 Named type

A named type enables to represent a domain of values as a reference to an identified data type ontology concept (see 6.7.6). It is represented by a **NAMED_TYPE_Type** XML complex type as illustrated in Figure 73.

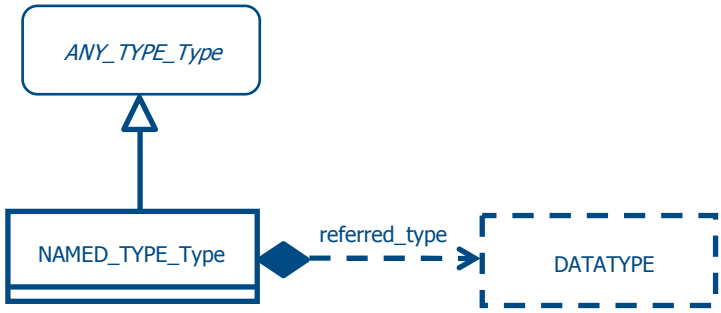

Figure 73 — Named type structure

Internal item definition:

referred_type: a reference to the data type ontology concept that defines the value domain.

Internal subtype definition:

NAMED_TYPE_Type: provides for values of properties or user types that are of any kind of OntoML dataypes, as specified by the referenced datatype.

NOTE The lexical representation of a value whose data type is **NAMED_TYPE_Type** is the lexial representation of its underlying datatype.

External type definition:

ANY_TYPE_Type: see 8.3.

8.3.13 Advanced-level data types

The OntoML type system specifies datatypes that may be used to characterize complex property value domains when describing a product ontology.

These datatypes are defined in an ontology that is part of this standard.

NOTE 1 The ontology of advanced-level types is specified in Annex E.

Thus, each advanced-level data type is defined by a reference to an ontology class that specifies its value structure. Property values may then be represented and exchanged according to ISO/TS 29002-10 constructs.

NOTE 2 ISO/TS 29002 is developed as a joint effort of several standardization committees to promote interoperability between the various standards that require product characterization.

The extended data types that are supported by OntoML are described in Figure 74.

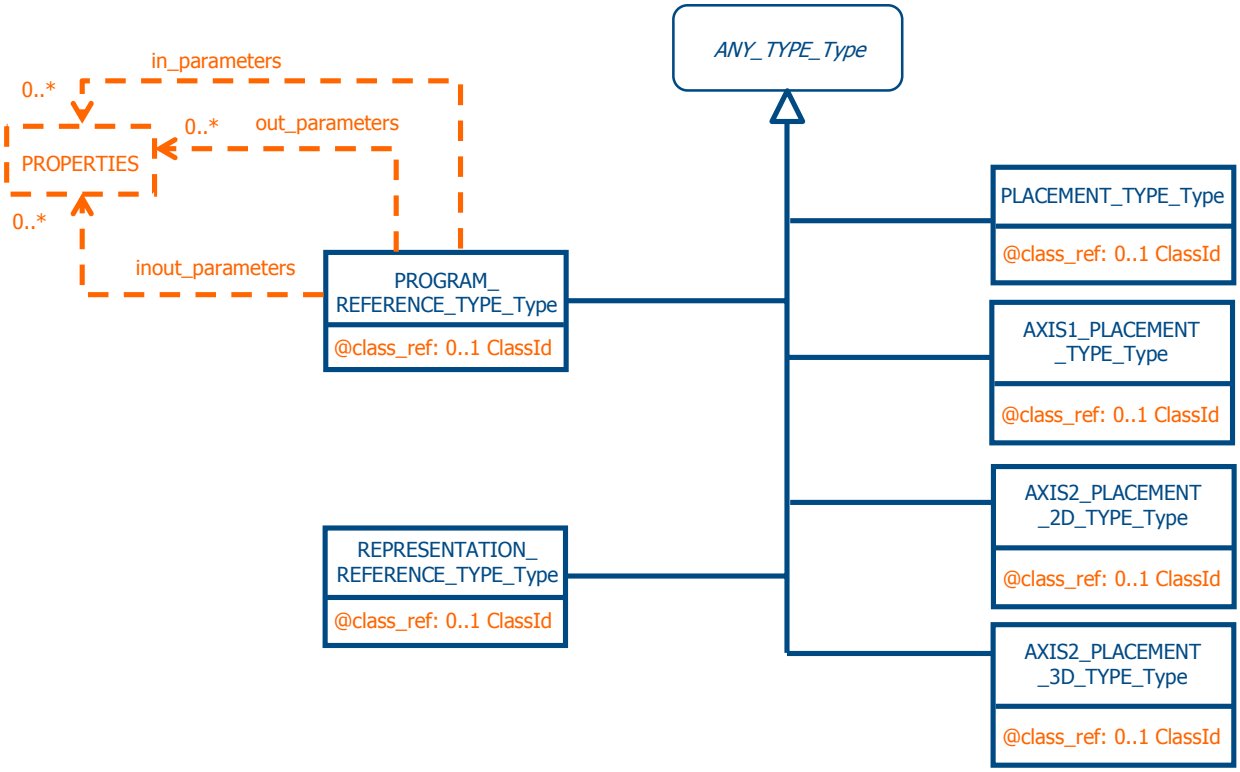

Figure 74 — Advanced-level data types structure

Internal item definition:

@class_ref (**AXIS1_PLACEMENT_TYPE_Type**): the possible reference to the ontology class that defines the structure of the **AXIS1_PLACEMENT_TYPE_Type**.

NOTE 3 When provided, the **@class_ref** XML attribute shall be set to the "0112-1---13584_32_1#01-AXIS1_PLACEMENT#1" IRDI.

@class_ref (**AXIS2_PLACEMENT_2D_TYPE_Type**): the possible reference to the ontology class that defines the structure of the **AXIS2_PLACEMENT_2D_TYPE_Type**.

NOTE 4 When provided, the **@class_ref** XML attribute shall be set to the "0112-1---13584_32_1#01-AXIS2_PLACEMENT_2D#1" IRDI.

@class_ref (**AXIS2_PLACEMENT_3D_TYPE_Type**): the possible reference to the ontology class that defines the structure of the **AXIS2_PLACEMENT_3D_TYPE_Type**.

NOTE 5 When provided, the **@class_ref** XML attribute shall be set to the "0112-1---13584_32_1#01-AXIS2_PLACEMENT_3D#1" IRDI.

@class_ref (**PLACEMENT_TYPE_Type**): the possible reference to the ontology class that defines the structure of the **PLACEMENT_TYPE_Type**.

NOTE 6 When provided, the **@class_ref** XML attribute shall be set to the "0112-1---13584_32_1#01-PLACEMENT#1" IRDI.

@class_ref (**PROGRAM_REFERENCE_TYPE_Type**): the possible reference to the ontology class that defines the structure of the **PROGRAM_REFERENCE_TYPE_type**.

NOTE 7 When provided, the **@class_ref** XML attribute shall be set to the "0112-1---13584_32_1#01-PROGRAM_REFERENCE#1" IRDI.

@class_ref (**REPRESENTATION_REFERENCE_TYPE_Type**): the possible reference to the ontology class that defines the structure of the **REPRESENTATION_REFERENCE_TYPE_Type**.

NOTE 8 When provided, the **@class_ref** XML attribute shall be set to the "0112-1---13584_32_1#01-REPRESENTATION_REFERENCE#1" IRDI.

in_parameters: the list of property references that specifies the type of the input parameters of the referenced program.

inout_parameters: the list of property references that specifies the type of the inout parameters of the referenced program.

out_parameters: the list of property references that specifies the type of the output parameters of the referenced program.

Internal subtype definitions:

AXIS1_PLACEMENT_TYPE_Type: it allows to define a property value domain whose value is an instance of the *axis 1 placement* ontology class that specifies a direction and a location in three-dimensional space of a single axis.

NOTE 9 The lexical representation of a value whose data type is **AXIS1_PLACEMENT_TYPE_Type** is defined in Annex D, Clause D.3.1.2.

NOTE 10 The *axis 1 placement* ontology class, identified by the "0112-1---13584_32_1#01-AXIS1_PLACEMENT#1" IRDI, is specified in the advanced-level data type value structures ontology defined in Annex E.

NOTE 11 The *axis 1 placement* ontology class structure corresponds to the *axis1_placement* EXPRESS entity data type structure specified in ISO 10303-42.

AXIS2_PLACEMENT_2D_TYPE_Type: it allows to define a property value domain whose value is an instance of the *axis 2 placement 2D* ontology class that specifies a location and an orientation in two-dimensional space of two mutually perpendicular axes.

NOTE 12 The lexical representation of a value whose data type is **AXIS2_PLACEMENT_2D_TYPE_Type** is defined in Annex D, Clause D.3.1.3.

NOTE 13 The *axis 2_placement 2D* ontology class, identified by the "0112-1---13584_32_1#01-AXIS2_PLACEMENT_2D#1" IRDI, is specified in the advanced-level data type value structures ontology defined in Annex E.

NOTE 14 The *axis 2 placement 2D* ontology class structure corresponds to the *axis2_placement_2d* EXPRESS entity data type structure specified in ISO 10303-42.

AXIS2_PLACEMENT_3D_TYPE_Type: it allows to define a property value domain whose value is an instance of the *axis 2 placement 3D* ontology class that specifies a location and an orientation in three-dimensional space of two mutually perpendicular axes.

NOTE 15 The lexical representation of a value whose data type is **AXIS2_PLACEMENT_3D_TYPE_Type** is defined in Annex D, Clause D.3.1.4.

NOTE 16 The *axis 2 placement 3D* ontology class, identified by the "0112-1---13584_32_1#01-AXIS2_PLACEMENT_3D#1" IRDI, is specified in the advanced-level data type value structures ontology defined in Annex E.

NOTE 17 The *axis 2 placement 3D* ontology class structure corresponds to the *axis2_placement_3d* EXPRESS entity data type structure specified in ISO 10303-42.

PLACEMENT_TYPE_Type: it allows to define a property value domain whose value is an instance of the *placement* ontology class that specifies a position with respect to the coordinate system of its geometric context.

NOTE 18 The lexical representation of a value whose data type is **PLACEMENT_TYPE_Type** is defined in Annex D, Clause D.3.1.1.

NOTE 19 The *placement* ontology class, identified by the "0112-1---13584_32_1#01-PLACEMENT#1" IRDI, is specified in the advanced-level data type value structures ontology defined in Annex E.

NOTE 20 The *placement* ontology class structure corresponds to the *placement* EXPRESS entity data type structure specified in ISO 10303-42.

PROGRAM_REFERENCE_TYPE_Type: it allows to define a property value domain whose value is an instance of the *program reference* ontology class containing an algorithm that shall be triggered and provided with parameter values to generate each item representation.

NOTE 21 The lexical representation of a value whose data type is **PROGRAM_REFERENCE_TYPE_Type** is defined in Annex D, Clause D.3.2.2.

NOTE 22 The *program reference* ontology class, identified by the "0112-1---13584_32_1#01-PROGRAM_REFERENCE#1" IRDI, is specified in the advanced-level data type value structures ontology defined in Annex E.

NOTE 23 The *program reference* ontology class structure corresponds to the *program_reference* EXPRESS entity data type structure specified in ISO 13584-25.

REPRESENTATION_REFERENCE_TYPE_Type: it allows to define a property value domain whose value is an instance of the r*epresentation reference* ontology class.

NOTE 24 The lexical representation of a value whose data type is **REPRESENTATION_REFERENCE_TYPE_Type** is defined in Annex D, Clause D.3.2.1.

NOTE 25 The *representation reference* ontology class, identified by the "0112-1---13584_32_1#01-REPRESENTATION_REFERENCE#1" IRDI, is specified in the advanced-level data type value structures ontology defined in Annex E.

NOTE 26 The *representation reference* ontology class structure corresponds to the *representation reference* EXPRESS entity data type structure specified in ISO 13584-25.

External type definition:

ANY_TYPE_Type: see 8.3.

8.4 Units

Properties for which the data type represents a measure are associated with units.

NOTE 1 OntoML units are defined according to the ISO 10303-41 information model for representing units.

In OntoML, the unit of a measure property is represented by a **DIC_UNIT_Type** XML complex type as illustrated in Figure 75.

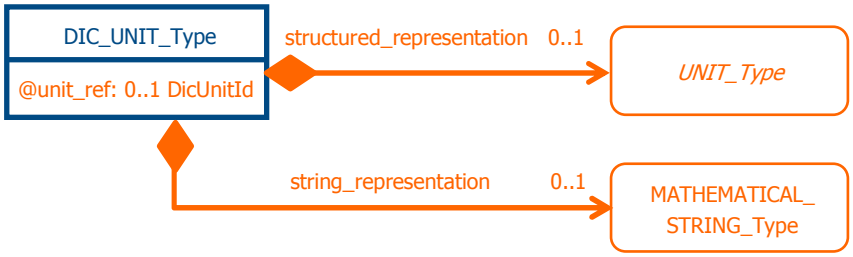

Figure 75 — General measure property unit structure

Internal item definition:

@unit_ref: the possible reference to a unit identifier.

string_representation: string representation of the measure property unit.

structured_representation: the explicit description of the measure property unit.

NOTE 2 If both a reference to a unit and the structured representation of the unit are provided, in case of inconsistencies, the explicit representation takes precedence.

External type definitions:

MATHEMATICAL_STRING_Type: the representation of a mathematical string, see 8.8.2.

UNIT_Type: the unit specification, see 8.4.1.

Constraint specification:

Either a reference to a unit (**unit_ref**) or the explicit representation of the unit (**structured_representation**) or both are provided.

8.4.1 Unit structure

A unit is defined using the **UNIT_Type** abstract XML complex type. It is represented in Figure 76.

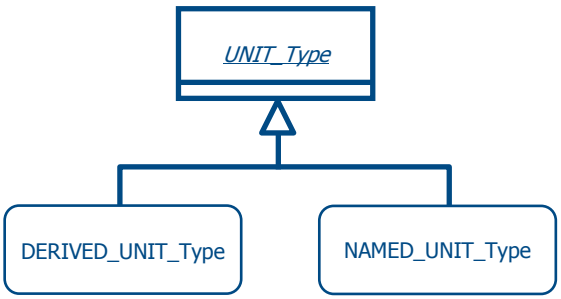

Figure 76 — Basic unit structures

External type definitions:

DERIVED_UNIT_Type: derived unit, see 8.4.3.

EXAMPLE 1 Newton per square millimeter is a derived unit.

NAMED_UNIT_Type: named unit, see 8.4.2.

EXAMPLE 2 Millimeter or Pascal are kinds of named unit.

8.4.2 Named unit

A named unit is a unit associated with the word, or group of words, by which the unit is identified. It is represented by the **NAMED_UNIT_Type** XML complex type as illustrated in Figure 77.

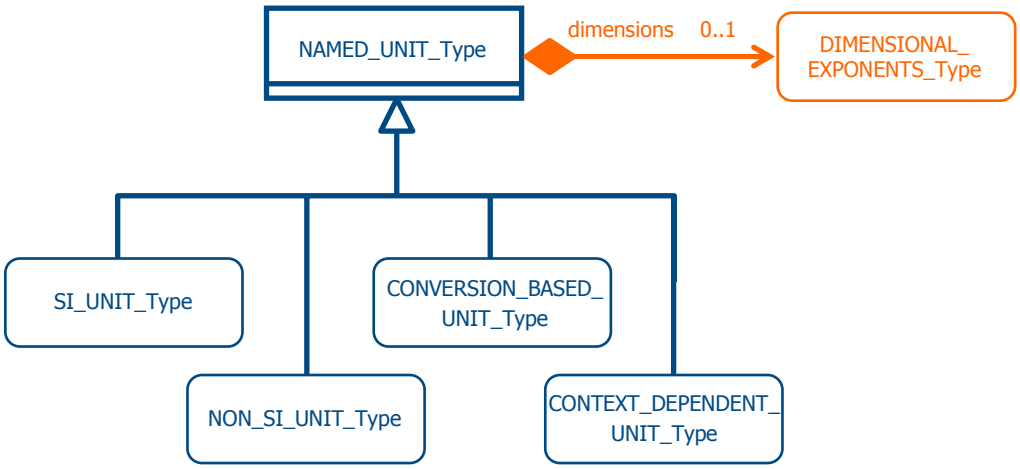

Figure 77 — Named unit general structure

Internal item definition:

dimensions: possible exponents of the base properties by which the named unit is defined.

Internal type definition:

DIMENSIONAL_EXPONENTS_Type: dimensional equation, see 8.4.2.1.

External type definitions:

CONTEXT_DEPENDENT_UNIT_Type: context dependent unit, see 8.4.2.5.

CONVERSION_BASED_UNIT_Type: a conversion based unit, see 8.4.2.4.

NON_SI_UNIT_Type: a non internationally standardized unit, see 8.4.2.3.

SI_UNIT_Type: an internationally standardized unit, see 8.4.2.2.

8.4.2.1 Dimensional exponent

A named unit may be associated to its dimensional equation. It is represented by the **DIMENSIONAL_EXPONENTS_Type** XML complex type as illustrated in Figure 78.

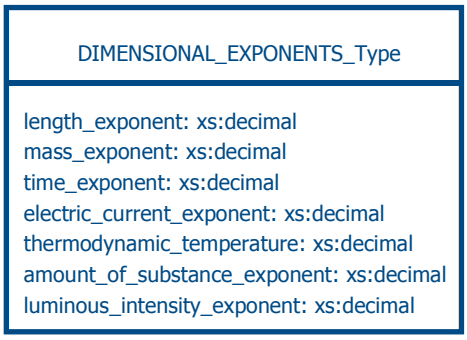

Figure 78 — Dimensional exponent structure

Internal item definitions:

amount_of_substance_exponent: the power of the amount of substance base quantity.

electric_current_exponent: the power of the electric current base quantity.

length_exponent: the power of the length base quantity.

luminous_intensity_exponent: the power of the luminous intensity base quantity.

mass_exponent: the power of the mass base quantity.

thermodynamic_temperature: the power of the thermodynamic temperature base quantity.

time_exponent: the power of the time base quantity.

8.4.2.2 Internationally standardized unit

An internationally standardized unit is the fixed quantity used as a standard in terms whose items are measured as defined by ISO 80000-1.

EXAMPLE 1 Millimeter is an internationally standardized unit.

An internationally standardized unit is specified through an optional prefix and the associated SI unit. It is represented by the **SI_UNIT_Type** XML complex type as illustrated in Figure 79.

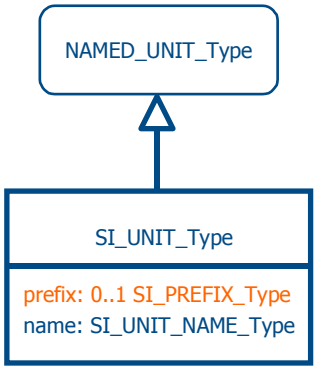

Figure 79 — International standardized unit structure

Internal item definitions:

name: the word or group of words by which the internationally standardized unit is referred to.

EXAMPLE 2 Millimeter is a SI unit: According to the **SI_UNIT_Type** XML complex type specification, the “MILLI” value would be assigned to the optional **prefix** XML element, and the “METRE” value would be assigned to the mandatory **name** XML element.

prefix: the international standardized unit prefix.

Internal type definitions:

SI_PREFIX_Type: the name of a prefix that may be associated with an internationally standardized unit. It is represented by an enumeration of the allowed internationally standardized unit prefix.

NOTE 1 The allowed international standardized unit prefixes are specified in ISO 80000-1.

SI_UNIT_NAME_Type: the name of an internationally standardized unit. It is represented by an enumeration of the allowed internationally standardized unit.

NOTE 2 The allowed internationally standardized unit names are specified in ISO 80000-1.

External type definition:

NAMED_UNIT_Type: see 8.4.2.

Constraint specification:

For internationally standardized unit, the inherited **dimensions** attribute is not set.

8.4.2.3 Non internationally standardized unit

A non internationally standardized unit allows for the representation of units that are not SI units, nor conversion based units, nor length units. It is represented by the **NON_SI_UNIT_Type** XML complex type as illustrated in Figure 80.

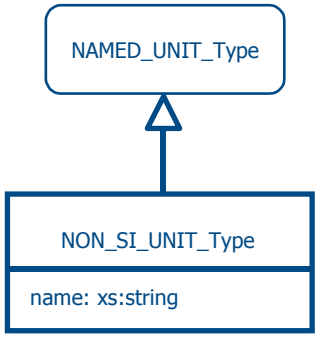

Figure 80 — Non international standardized unit structure

Internal item definitions:

name: the label used to name the described unit.

External type definition:

NAMED_UNIT_Type: see 8.4.2.

8.4.2.4 Conversion based unit

A conversion based unit is a unit defined on the base of another unit.

EXAMPLE 1 An inch is a conversion based unit.

It is represented by the **CONVERSION_BASED_UNIT_Type** XML complex type as illustrated in Figure 81.

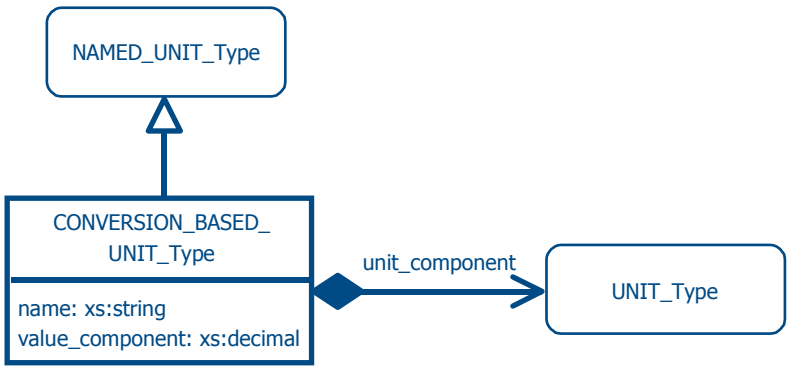

Figure 81 — Conversion based unit structure

Internal item definitions:

name: the word or group of words by which the conversion based unit is referred to.

unit_component: the unit in which the physical quantity is expressed.

value_component: value of the physical quantity that corresponds to one unit of the conversion based unit when expressed in the **unit_component** unit.

EXAMPLE 2 An inch is a conversion based unit. It is from the Imperial system, its **name** XML element value is equal to “inch", and it can be related to the SI unit, millimeter, through a measure value (**value_component** XML element) equal to 25.4 millimeter (**unit_component** XML element).

External type definition:

NAMED_UNIT_Type: see 8.4.2.

UNIT_Type: a general unit, see 8.4.

8.4.2.5 Context dependent unit

A context dependent unit: it is a unit which is not related to the SI system.

It is represented by the **CONTEXT_DEPENDENT_UNIT_Type** XML complex type as illustrated in Figure 82.

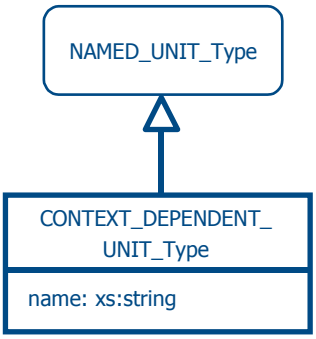

Figure 82 — Context dependent unit structure

Internal item definitions:

name: the word or group of words by which the context dependent unit is referred to.

EXAMPLE The number of parts in an assembly is a physical quantity measured in units that may be called “parts" but which cannot be related to an SI unit. The value “part” would be then assigned to the context dependent unit **name**.

External type definition:

NAMED_UNIT_Type: see 8.4.2.

8.4.3 Derived unit

A derived unit stands for an expression of units.

EXAMPLE 1 Newton per square millimeter is a derived unit.

It is represented by the **DERIVED_UNIT_Type** XML complex type as illustrated in Figure 83.

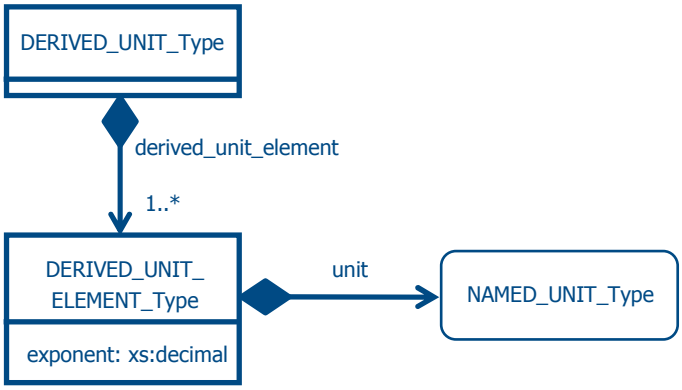

Figure 83 — Derived unit structure

Internal item definitions:

derived_unit_element: the unit quantity which makes up a derived unit.

derived_unit_element/exponent: the power that is applied to the **unit** XML element.

derived_unit_element/unit: the fixed quantity which is used as the mathematical factor.

EXAMPLE 2 *Newton per square millimeter* is a derived unit. It would then be represented by two **derived_unit_element**s: the former for representing the *Newton* SI unit, the latter for representing the *millimeter* SI unit to which a -2 **exponent** would also be assigned.

External type definition:

NAMED_UNIT_Type: see 8.4.2.

8.5 Constraints

OntoML constraints allow to further restrict the co-domain of any property defined in an ontology. Constraints may also be used to restrict co-domains when defined in a given class, and when the property domain is subclass of this class.

NOTE OntoML allows to represent the whole set of constraints specified in the ISO 13584-42:2010 information model.

Every datatype is defined as a subtype of the **CONSTRAINT_Type** XML complex type. The general constraints structure is illustrated in Figure 84.

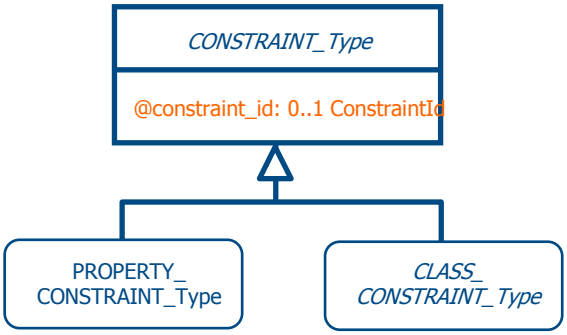

Figure 84 — General constraints structure

Internal item definition:

@constraint_id: the possible **ConstraintId** that identifies the constraint.

External type definitions:

ConstraintId: see 9.1.4.

CLASS_CONSTRAINT_Type: class instances related constraint, see 8.5.1.

PROPERTY_CONSTRAINT_Type: property value related constraint, see 8.5.3.

Constraint specification:

The constraint defined by the **constraint_id** XML element is unique in the OntoML document instance.

8.5.1 Constraint reference

Depending on the use context, a constraint may be either referenced or explicitly defined. The class constraint reference structure is illustrated in Figure 85.

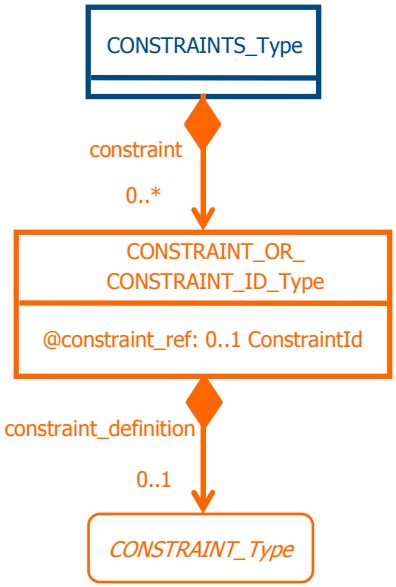

Figure 85 — Constraint reference structure

Internal item definition:

constraint: a constraint that restrict the target domain of values of properties of the class to a subset of its inherited domain of values.

constraint/@constraint_ref: a constraint identifier.

constraint/constraint_definition: the specification of a constraint.

Internal type definitions:

CONSTRAINT_OR_CONSTRAINT_ID_Type: the specification of a constraint defined either explicitly or by reference to a constraint identifier.

CONSTRAINTS_Type: the specification of a set of constraints done either explicitly or by a reference to an identified constraint.

External type definitions:

ConstraintId: see 9.1.

CONSTRAINT_Type: see 8.5.

8.5.2 Class constraint

A class constraint is a constraint that restricts the allowed set of instances of a class by constraining several properties or by defining global constraints. The class constraint structure is illustrated in Figure 86.

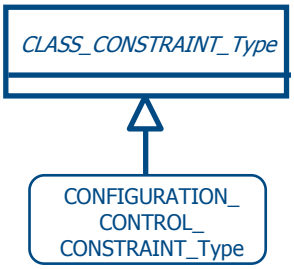

Figure 86 — Class constraint structure

Internal type definition:

CONFIGURATION_CONTROL_CONSTRAINT_Type: referenced instances constraint, see 8.5.2.1.

8.5.2.1 Configuration control constraint

The configuration control constraint allows to restrict the set of instances, called the referenced instances, that a particular instance, called the referencing instance, may reference directly or indirectly by means of a chain of properties. The referencing instance is any instance of a class that references the configuration control constraint by means of the class **constraints** XML element. The configuration control constraint defines an optional precondition that specifies the condition on the referencing instance for the restriction to apply. It defines a postcondition that specifies the allowed sets of values for some properties of the referenced instance class. It is represented by an **CONFIGURATION_CONTROL_CONSTRAINT_Type** XML complex type. It is illustrated in Figure 87.

NOTE 1 Both precondition and postcondition may only restrict properties whose value domain is defined as an enumeration of string codes (see 8.3.4) or an enumeration of integer codes (see 8.3.8). Such properties may be assigned a value either at the instance level, or at the class level if they are declared as class valued properties, i.e., referenced in the **sub_class_properties** XML element in an item class (see 6.7.2.1).

NOTE 2 Properties referenced in the precondition are applicable to the class that references the configuration control constraint.

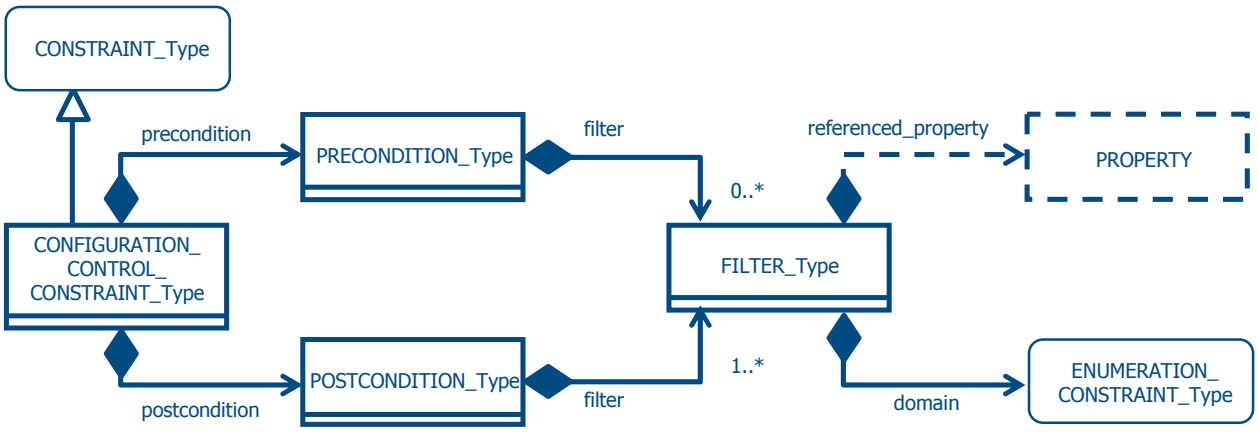

Figure 87 — Configuration control constraint structure

Internal item definition:

domain (**FILTER_Type**): the enumeration constraint that restricts the domain of values of the referenced property.

filter (**PRECONDITION_Type**): the constraints that shall be satisfied for the configuration control constraint to apply.

filter (**POSTCONDITION_Type**): the constraints that shall be satisfied for a referenced instance for being allowed for reference.

postcondition: the specification of the **filters** that shall hold on a referenced instance for being allowed for reference.

precondition: the specification of the filters that shall hold on the referencing instance for the restriction to apply.

NOTE 3 If the set of filters is empty, the restriction applies on any referencing instance.

referenced_property (**FILTER_Type**): the reference to the property whose domain of values is restricted by the associated filter.

Internal type definitions:

FILTER_TYPE: specifies a restriction on the allowed domain of a property whose data type is either an enumeration of string codes (see 8.3.4) or an enumeration of integer codes (see 8.3.8).

PRECONDITION_Type: specifies the conditions on the referencing instance for the restriction to apply.

POSTCONDITION_Type: specifies the allowed sets of values for some properties of the referenced instance class.

External type definitions:

CONSTRAINT_Type: see 8.5.

ENUMERATION_CONSTRAINT_Type: collection constraint, see 8.5.3.3.6.

Constraint specification:

The underlying data type of the property referenced by the **referenced_property** shall be either **NON_QUANTITATIVE_INT_TYPE_Type** or **NON_QUANTITATIVE_CODE_TYPE_Type**.

The **domain** shall define a restriction that is compatible with the initial domain of values of the property referenced by the **referenced_property** XML element.

8.5.3 Property constraint

A property constraint is a constraint that restricts the allowed set of instances of a class by a single restriction of the domain of values of one of its properties. The property constraint structure is illustrated in Figure 88.

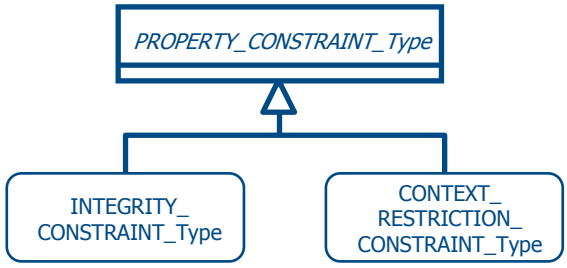

Figure 88 — Property constraint structure

External type definitions:

CONTEXT_RESTRICTION_CONSTRAINT_Type: constraint on property conditions, see 8.5.3.1.

INTEGRITY_CONSTRAINT_Type: collection constraint, see 8.5.3.2.

8.5.3.1 Context restriction constraint

A context restriction constraint applies to properties that play the role of a context dependent property.

NOTE 1 Context dependent properties are defined in Clause 6.7.4.

It specifies that the value domain(s) of a context parameter used for specifying the value of this context dependent property is(are) restricted using any kind of available domain constraint.

NOTE 2 Context parameters are defined in Clause 6.7.4.

It is represented by an **INTEGRITY_CONSTRAINT_Type** XML complex type. It is illustrated in Figure 89.

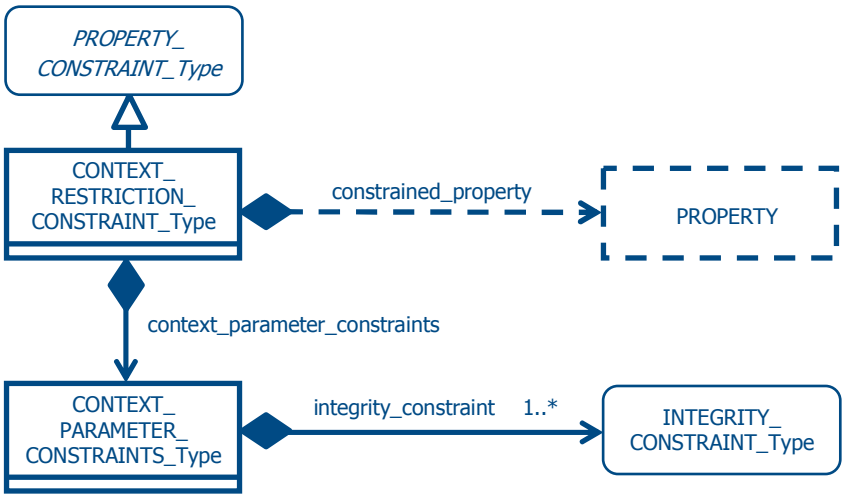

Figure 89 — Context restriction constraint structure

Internal item definition:

constrained_property: the property for which the constraint applies.

NOTE 3 The referenced property is a context dependent property (see 6.7.4).

context_parameter_constraints: the set of constraints that apply on the domain of values of each context parameters of the referenced property.

integrity_constraint (CONTEXT_PARAMETER_CONSTRAINTS_Type): the integrity constraint that reduces the allowed domain of a context parameter of the constrained context dependent property.

EXAMPLE The *resistance* of a *thermistor* depends on (**DEPENDENT_P_DET_Type** XML complex type) the *ambient temperature* (**CONDITION_DET_Type** XML complex type) whose value domain is an **INT_TYPE_Type**. A **CONTEXT_RESTRICTION_CONSTRAINT_Type** constraint allows to require that this *ambient temperature* be 25° by applying a range constraint (**RANGE_CONSTRAINT_Type** XML complex type) on it.

Internal type definitions:

CONTEXT_PARAMETER_CONSTRAINTS_Type: the specification of the set of constraints that apply on the domain of values of the context parameters.

External type definitions:

PROPERTY_CONSTRAINT_Type: see 8.5.3.

INTEGRITY_CONSTRAINT_Type: property domain restriction constraint, see 8.5.3.2.

Constraint specification:

The property referenced by the **constrained_property** XML element shall have a base type that is **DEPENDENT_P_DET_Type**.

The set of properties whose domain is constrained by the **context_parameter_constraints** shall be context parameters on which the property referenced by the **constrained_property** XML element depends.

8.5.3.2 Integrity constraint

An integrity constraint allows to make explicit that for some particular class, as a result of the class definition, and all its subclasses, only a restriction of the domain of values specified by a data type is allowed for a property. It is represented by an **INTEGRITY_CONSTRAINT_Type** XML complex type. It is illustrated in Figure 90.

EXAMPLE In the reference dictionary defined for fasteners in ISO 13584-511, a *metric threaded bolt/screw* has a *head properties* property that may takes, as value, a member of any subclass of the *head* feature class. If the *metric threaded bolt/screw* is also a member of the *hexagon head screw* subclass, the *head properties* may only be a member of the *hexagon head* feature class, else the *metric threaded bolt/screw* cannot be a member of the *hexagon head screw* subclass.

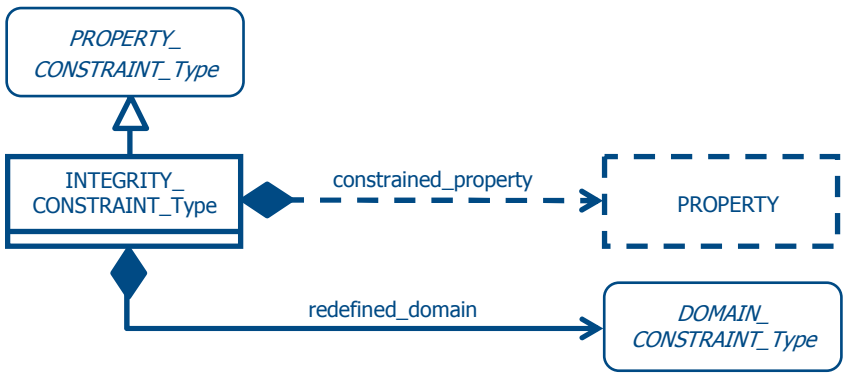

Figure 90 — Integrity constraint structure

Internal item definition:

constrained_property: the property for which the constraint applies.

redefined_domain: the integrity constraint that applies on the domain of values of the constrained property.

External type definition:

DOMAIN_CONSTRAINT_Type: the particular constraint that applies to the reference property, see 8.5.3.3.

PROPERTY_CONSTRAINT_Type: see 8.5.3.

Constraint specification:

The **redefined_domain** shall define a restriction that is compatible with the initial domain of values of the property referenced by the **constrained_property** XML element.

8.5.3.3 Domain constraint

OntoML provides resources for describing different kinds of domain of values constraints. Every specific constraint is defined as a subtype of the **DOMAIN_CONSTRAINT_Type** XML complex type. Figure 91 gives a global overview of the available property value constraints.

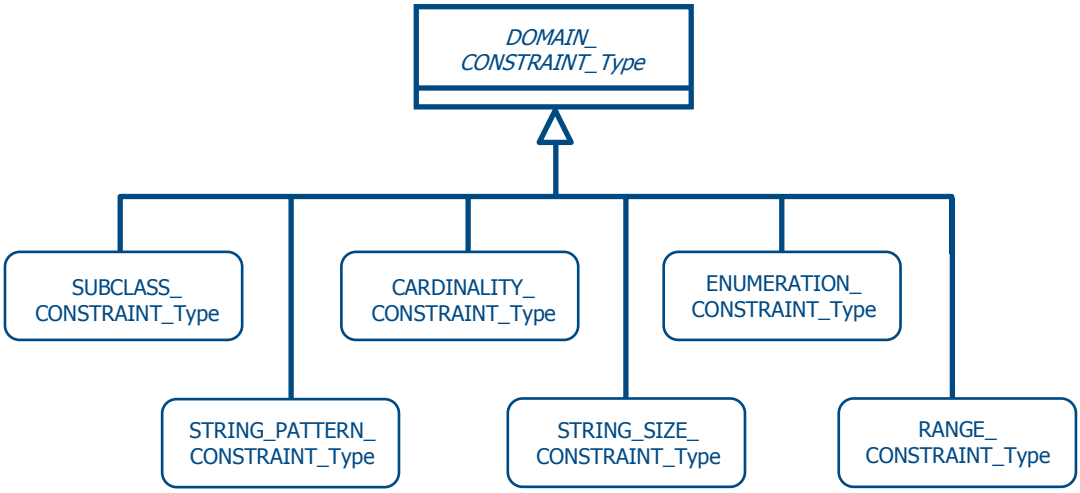

Figure 91 — Domain constraints

External type definitions:

CARDINALITY_CONSTRAINT_Type: cardinality constraint specification, see 8.5.3.3.3.

RANGE_CONSTRAINT_Type: range constraint specification, see 8.5.3.3.5.

STRING_PATTERN_CONSTRAINT_Type: string pattern constraint specification, see 8.5.3.3.2.

STRING_SIZE_CONSTRAINT_Type: string size constraint specification, see 8.5.3.3.4.

SUBCLASS_CONSTRAINT_Type: subclass constraint specification, see 8.5.3.3.1.

ENUMERATION_CONSTRAINT_Type: enumeration constraint specification, see 8.5.3.3.6.

8.5.3.3.1 Subclass constraint

A subclass constraint applies to properties whose value domain is defined by a class instance type (see 8.3.10). It specifies that the property value domain is restricted to a subclass of this class. It is represented by a **SUBCLASS_CONSTRAINT_Type** XML complex type as illustrated in Figure 92.

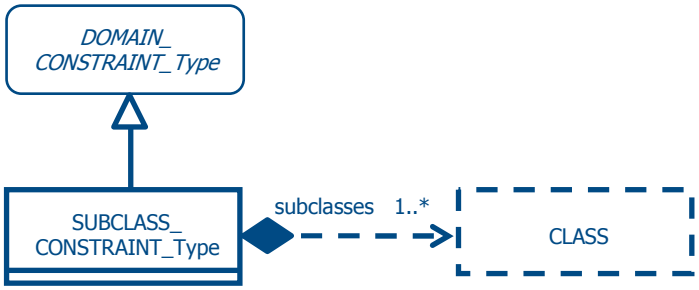

Figure 92 — Subclass constraint representation

Internal item definitions:

subclasses: the references to the class ontology concepts which redefine the new value domain of the constrained entity.

EXAMPLE Let's consider a property (**NON_DEPENDENT_P_DET_Type** XML complex type) called *screw_head* defined in the context of a *screw* class (**ITEM_CLASS_Type** XML complex type) whose value domain (**CLASS_REFERENCE_TYPE_Type** XML complex type) is a *head* class (**ITEM_CLASS_Type** XML complex type) defining the general characteristics of any kind of screw heads. Additionally, let's consider an *hexagonal_screw* class (**ITEM_CLASS_Type** XML complex type), subclass of the *screw* class, and an *hexagonal_head* class (**ITEM_CLASS_Type** XML complex type), subclass of the *head* class. In the *hexagonal_screw* class, the *screw_head* value domain could be restricted as being a subclass of the *head* class, i.e., the *hexagonal_head* class. This restriction would be expressed by defining a specific **SUBCLASS_CONSTRAINT_Type constraint** whose **subclass** XML element value would be a reference to the particular **ITEM_CLASS_Type** representing the *hexagonal_head* class.

External type definitions:

DOMAIN_CONSTRAINT_Type: see 8.5.3.3.

8.5.3.3.2 String pattern constraint

A string pattern constraint applies to properties whose value domain is defined by a string.

NOTE 1 A string property value domain is either a **STRING_TYPE_Type** (see 8.3.2), or a **NON_TRANSLATABLE_STRING_TYPE_Type** (see 8.3.2) or a **TRANSLATABLE_STRING_TYPE_Type** (see 8.3.2) , or a **URI_TYPE_Type** (see 8.3.2), or **NON_QUANTITATIVE_CODE_TYPE_Type** (see 8.3.4) or a **DATE_DATA_TYPE_Type** (see 8.3.3) or a **TIME_DATA_TYPE_Type** (see 8.3.3) or a **DATE_TIME_DATA_TYPE_Type** (see 8.3.3).

A string pattern constraint specifies that the property value domain is restricted according to string values that match a particular pattern. It is represented by a **STRING_PATTERN_CONSTRAINT_Type** XML complex type as illustrated in Figure 93.

For properties whose data type is defined as a **STRING_TYPE_Type**, a **NON_TRANSLATABLE_STRING_TYPE_Type**, a **URI_TYPE_Type**, a **DATE_DATA_TYPE_Type**, a **TIME_DATA_TYPE_Type** or a **DATE_TIME_DATA_TYPE_Type**, the constraint applies to the (unique) string that is the value of the data type.

For properties whose data type is defined as a **TRANSLATED_STRING_TYPE_Type**, the constraint applies to the string that is in the source language into which the property domain was defined. This source language may be defined in the the **source_language** XML element of the **CLASS_TYPE** XML complex type (see 6.7.2.1), or the **PROPERTY_Type** XML complex type (see 6.7.4) or the **DATATYPE_Type** XML complex type (see 6.7.6) or the **DOCUMENT_Type** XML compex type (see 6.7.7). If this attribute does not exist, this source language is supposed known by the dictionary user

For properties whose data type is defined as a **NON_QUANTITATIVE_CODE_TYPE_type**, the constraint applies to the code.

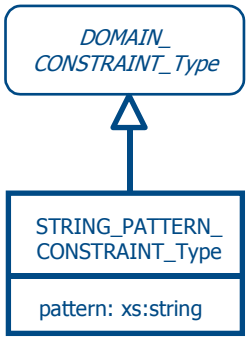

Figure 93 — String pattern constraint representation

Internal item definitions:

pattern: the pattern of string values that are allowed as values for the property identified by the constrained property.

NOTE 2 The **pattern** XML element value syntax is based on the regular expression and the associated matching algorithms that are defined by the XML Schema Part 2: Datatypes recommendation.

EXAMPLE The XML Schema pattern that corresponds to the “[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]” SQL SIMILAR expression is “[0-9]{4}\-[0-9]{2}\-[0-9]{2}”. It allows to match strings as “2010-03-24”.

External type definitions:

DOMAIN_CONSTRAINT_Type: see 8.5.3.3.

8.5.3.3.3 Cardinality constraint

A **cardinality_constraint** restricts the cardinality of a collection data type.

NOTE 1 The resulting cardinality range is the intersection of preexisting cardinality ranges and of the one defined by the cardinality constraint.

NOTE 2 Cardinality constraints are not allowed on arrays (see 8.3.9.4).

It is represented by a **CARDINALITY_CONSTRAINT_Type** XML complex type as illustrated in Figure 94.

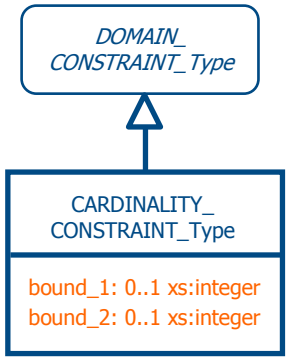

Figure 94 — Cardinality constraint representation

Internal item definitions:

bound_1: the lower bound of the cardinality.

bound_2: the upper bound of the cardinality.

External type definitions:

DOMAIN_CONSTRAINT_Type: see 8.5.3.3.

Constraint specification:

When **bound_1** does not exist, the minimal cardinality is 0.

When **bound_2** does not exist, there is no constraint on the maximal cardinality.

When **bound_1** exists, its value shall be greater or equal than 0.

When both **bound_1** and **bound_2** are provided, **bound_2** shall be greater or equal than **bound_1**.

8.5.3.3.4 String size constraint

A string size constraint applies to properties whose value domain is defined by a string.

NOTE 1 A string property value domain is either a string (see 8.3.2), or a non translatable string (see 8.3.2) or a translatable string (see 8.3.2) , or a remote http address (see 8.3.2), or enumeration of string codes (8.3.3).

A string size constraint specifies that the property value domain is restricted to possibly have a minimum and/or a maximum length. It is represented by a **STRING_SIZE_CONSTRAINT_Type** XML complex type as illustrated in Figure 95.

NOTE 2 For properties whose data type is defined as a **TRANSLATED_STRING_TYPE_Type**, the constraint applies to any language-specific representation of the string.

NOTE 3 For properties whose data type is defined as a **NON_QUANTITATIVE_CODE_TYPE_type**, the constraint applies to the code.

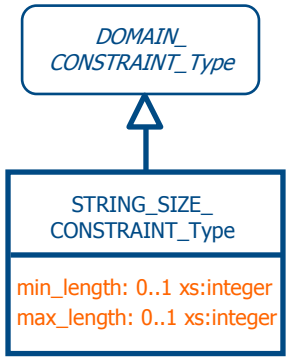

Figure 95 — String size constraint representation

Internal item definitions:

max_length: the maximal length for the strings that is allowed as value for the constrained property.

min_length: the minimal length for the strings that is allowed as value for the constrained property.

NOTE 4 **min_length** is greater than 0 and less or equal to **max_value**.

EXAMPLE A property whose value domain is a string (**STRING_TYPE_Type** XML complex type) and that is defined in a given class, may be restricted in a subclass of this class as having no more than 10 characters by defining a **STRING_SIZE_CONSTRAINT_Type** constraint and assigning 10 to the **max_length** XML element.

External type definitions:

DOMAIN_CONSTRAINT_Type: see 8.5.3.3.

Constraint specification:

When **min_length** does not exist, the minimal length is 0.

When **max_length** does not exist, there is no constraint on the maximal length.

When **min_length** exists, its value shall be greater or equal than 0.

When both **min_length** and **max_length** are provided, **max_length** shall be greater or equal than **min_length**.

8.5.3.3.5 Range constraint

A range constraint applies to properties whose value domain is defined by a number.

EXAMPLE 1 A number property value domain is either a number (see 8.3.5), an integer (see 8.3.5), an integer currency (see 8.3.6), an integer measure (see 8.3.7), a real (see 8.3.5), a real currency (see 8.3.6), a real measure (see 8.3.7) or a enumeration of integer codes (see 8.3.8).

A range constraint specifies that the property value domain is restricted to a subset of its values defined by a range. It is represented by a **RANGE_CONSTRAINT_Type** XML complex type as illustrated in Figure 96.

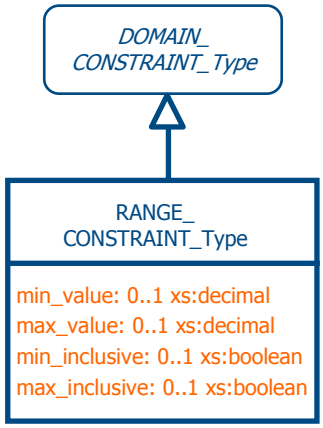

Figure 96 — Range constraint representation

Internal item definitions:

max_value: the number defining the high bound of the range of values.

max_inclusive: if true, specifies that the **max_value** is included in the specified range.

min_value: the number defining the low bound of the range of values.

min_inclusive: if true, specifies that the **min_value** is included in the specified range.

EXAMPLE 2 A property whose value domain is an integer (**INT_TYPE_Type** XML complex type) and that is defined in a given class, may be restricted in a subclass of this class as a [10..50] range by defining a **RANGE_CONSTRAINT_Type** constraint and assigning 10 to the **min_value** and 50 to the **max_value** XML elements.

External type definitions:

DOMAIN_CONSTRAINT_Type: see 8.5.3.3.

Constraint specifications:

min_value shall be less than or equal to **max_value**.

min_value and **max_value** shall be both integers or both reals.

Either **min_value** or **max_value** shall be defined.

If **min_inclusive** is not specified, there isn't any restriction about the lowest bound of the defined range.

If **max_inclusive** is not specified, there isn't any restriction about the highest bound of the defined range.

If **min_value** is not defined, **min_inclusive** shall not be defined.

If **max_value** is not defined, **max_inclusive** shall not be defined.

If **min_value** is defined, **min_inclusive** shall be defined.

If **max_value** is defined, **max_inclusive** shall be defined.

8.5.3.3.6 Enumeration constraint

An enumeration constraint restricts the domain of values of a data type to a list defined in extension. The order defined by the list is the recommended order for presentation purposes. A particular description may optionally be associated with each value of the list by mean of a **NON_QUANTITATIVE_INT_TYPE_Type**, of which the i-th value describes the meaning of the i-th value of the list.

This constraint is represented by an **ENUMERATION_CONSTRAINT_Type** XML complex type as illustrated in Figure 97.

NOTE 1 For a currency (see 8.3.6), or a measure (see 8.3.7) associated with alternative unit, the constraint applies whatever be the currency or the unit.

NOTE 2 For translatable strings (see 8.3.2), the constraint applies to any language-specific representation of the string.

NOTE 3 If another enumeration constraint is applied to a property already associated with an enumeration constraint in some superclass, both constraints apply. Thus the allowed set of values is the intersection of both subsets. Concerning the presentation order, and the possible meaning associated with each value, only those meanings defined in the lower enumeration constraint apply.

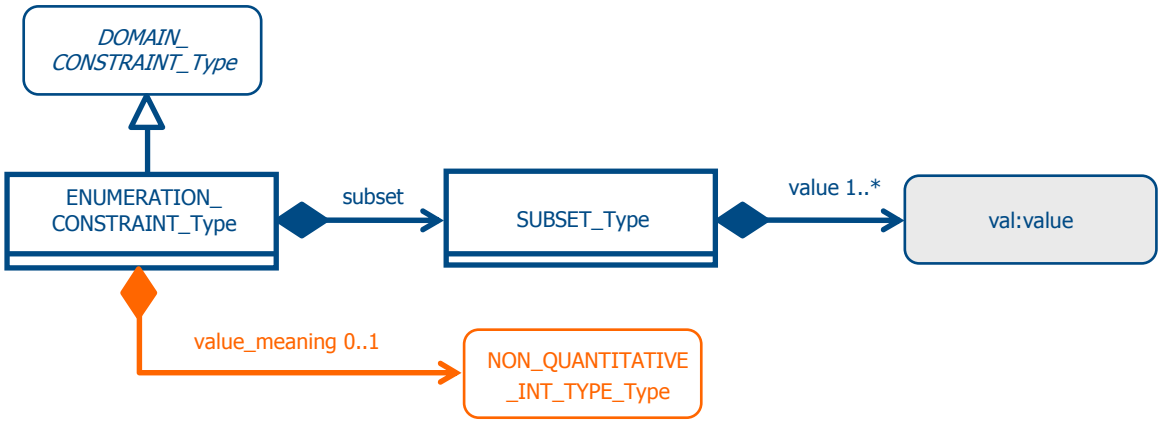

Figure 97 — Enumeration constraint representation

EXAMPLE 1 Assume that a property whose identifier is 0123-ABCD#02-P1#1 and whose value domain is an integer. In the context of a class identified by the 0123-ABCD#01-C1#1 IRDI, this property is associated with an enumeration constraint, where the allowed values are defined by the following integer subset: {1, 3, 5, 7}. Then, in the class identified by the 0123-ABCD#01-C1#1 IRDI, and any of its subclasses, the property identified by the 0123-ABCD#02-P1#1 IRDI may only takes one of the four following values: 1 or 3 or 5 or 7. The OntoML representation of this enumeration constraint would be as follows:

```
<ontoml:class xsi:type="ontoml:ITEM_CLASS_Type" id="0123-ABCD#01-C1#1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ontoml="urn:iso:std:iso:13584:-32:ed-1:tech:xml-schema:ontoml"
    xmlns:val="urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:value">
    …
    <constraints>
        <constraint>
            <constraint_definition xsi:type="ontoml:INTEGRITY_CONSTRAINT_Type">
                <constrained_property property_ref="0123-ABCD#02-P1#1"/>
                <redefined_domain xsi:type="ontoml:ENUMERATION_CONSTRAINT_Type">
                    <subset>
                        <val:integer_value>1</val:integer_value>
                        <val:integer_value>3</val:integer_value>
                        <val:integer_value>5</val:integer_value>
                        <val:integer_value>7</val:integer_value>
                    </subset>
```

```
                </redefined_domain>
            </constraint_definition>
        </constraint>
    </constraints>
</ontoml:class>
```

EXAMPLE 2 Assume that a property whose identifier is 0123-ABCD#02-P1#1 and whose value domain is a list of at teast one and at most 4 integers. In the context of a class identified by the 0123-ABCD#01-C1#1 IRDI, this property is associated with an enumeration constraint where the allowed values are defined by the following list of integers subset: { {1} , {3, 5}, {7}, {1, 3, 7} }. It that context, it means that the is 0123-ABCD#02-P1#1 property may take the following values: {1} or {3, 5} or {7} or {1, 3, 7}. The OntoML representation of this example would be as follows:

```
<ontoml:class xsi:type="ontoml:ITEM_CLASS_Type" id="0123-ABCD#01-C1#1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:ontoml="urn:iso:std:iso:13584:-32:ed-1:tech:xml-schema:ontoml"
    xmlns:val="urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:value">
    …
    <constraints>
        <constraint>
            <constraint_definition xsi:type="ontoml:INTEGRITY_CONSTRAINT_Type">
                <constrained_property property_ref="0123-ABCD#02-P1#1"/>
                <redefined_domain xsi:type="ontoml:ENUMERATION_CONSTRAINT_Type">
                    <subset>
                        <val:sequence_value>
                            <val:integer_value>1</val:integer_value>
                        </val:sequence_value>
                        <val:sequence_value>
                            <val:integer_value>3</val:integer_value>
                            <val:integer_value>5</val:integer_value>
                        </val:sequence_value>
                        <val:sequence_value>
                            <val:integer_value>7</val:integer_value>
                        </val:sequence_value>
                        <val:sequence_value>
                            <val:integer_value>1</val:integer_value>
                            <val:integer_value>3</val:integer_value>
                            <val:integer_value>7</val:integer_value>
                        </val:sequence_value>
                    </subset>
                </redefined_domain>
            </constraint_definition>
        </constraint>
    </constraints>
</ontoml:class>
```

EXAMPLE 3 Assume that a property (defined in a class identified by the 0123-ABCD#01-C1#1 IRDI) whose identifier is 0123-ABCD#02-P1#1 and whose value domain is a real measure expressed in millimetres, restricted to the following value set {10.5, 30}. Moreovoer, each value of the restricted value set is associated to a particular meaning, defined both in French and English. Meanings are the followings: 10.5 stands for “a little value (English) or “une petite valeur” (French); 30 stands for a “big value” (English) or “une grande valeur” (French). In OntoML, this property value domain would be represented as an **REAL_MEASURE_TYPE_Type** XML complex type (see 8.3.7), restricted by defining a **ENUMERATION_CONSTRAINT_Type** constraint. The constraint specifies two possible values: 10.5 and 30. Additionally, both values are associated to a particular meaning (**value_meaning** XML element). This meaning is expressed using an enumeration (**NON_QUANTITATIVE_INT_Type** XML complex type). Each element of the enumeration (**dic_value** XML element):

— is firstly associated to a value code (an integer, represented by the **INT_DIC_VALUE_Type** XML complex type) identifying the position of the value for which the current meaning is associated (10.5 is the first value of the value set, and then is associated with a value code equal to 1; 30 is the second value of the value set, and then is associated with a value code equal to 2);

— Is secondly associated, through the **preferred_name** XML element, to its meaning (a translated label, see 8.1).

The OntoML representation of this example would be as follows:

```
<ontoml:property xsi:type="ontoml:NON_DEPENDENT_P_DET_Type" id="0123-ABCD#02-PROPERTY#1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ontoml="urn:iso:std:iso:13584:-32:ed-1:tech:xml-schema:ontoml"
    xmlns:val="urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:value">
    …
    <domain xsi:type="ontoml:REAL_MEASURE_TYPE_Type">
        <constraints>
            <constraint xsi:type="ontoml:ENUMERATION_CONSTRAINT_Type">
                <subset>
                    <val:real_value>10.5</val:real_value>
                    <val:real_value>30</val:real_value>
                </subset>
                <value_meaning>
                    <its_values>
                        <dic_value xsi:type="ontoml:INT_DIC_VALUE_Type">
                            <preferred_name>
                                <label language_code="en">a little value</label>
                                <label language_code="fr">une petite valeur</label>
                            </preferred_name>
                            <value_code>1</value_code>
                        </dic_value>
                        <dic_value xsi:type="ontoml:INT_DIC_VALUE_Type">
                            <preferred_name>
                                <label language_code="en">a big value</label>
                                <label language_code="fr">une grande valeur</label>
                            </preferred_name>
                            <value_code>2</value_code>
                        </dic_value>
                    </its_values>
                </value_meaning>
            </constraint>
        </constraints>
        <unit>
            <structured_representation xsi:type="ontoml:SI_UNIT_Type">
                <prefix>MILLI</prefix>
                <name>METRE</name>
            </structured_representation>
        </unit>
    </domain>
</ontoml:property>
```

Internal item definitions:

subset: the list describing the subset of values that are allowed as possible values for the constrained property.

NOTE 4 The order defined by the list is the recommended order for presentation purposes.

subset.value: values that are allowed as possible value for the constrained property.

NOTE 5 Representations of values comply with ISO/TS 29002-10.

value_meaning: the optional description that may be associated with each value of the **subset** by means of an enumeration of integer codes, whose the i-th value describes the meaning of the i-th value of the **subset**.

Internal type definition:

SUBSET_Type: a set of values of a type specified by the **DATA_TYPE_Type** XML complex type.

External type definitions:

val:value: specification of a typed value.

NOTE 6 **val:value** is defined in the ISO/TS 29002-10 product exchange format.

DOMAIN_CONSTRAINT_Type: see 8.5.3.3.

NON_QUANTITATIVE_INT_TYPE_Type: see 8.3.8.

Constraint specification:

If **value_meaning** is provided, then the size of the collection representing the associated value codes shall be equal to the size of the **subset** collection.

If **value_meaning** is provided, every value of the **subset** collection shall be associated to a particular meaning.

If **value_meaning** is provided, the the i-th **dic_value** describes the meaning of the i-th value of the subset.

8.6 A posteriori semantics relationship

An *a posteriori* semantic relationship is an oriented relationship between two classes from which property equivalencies are modeled. A properties equivalence, or mapping, describes how properties of one of the two classes may be computed from properties of the other class. All the property mappings have the same orientation which depends upon the semantic relationship.

More than one properties equivalence may be defined in the same *a posteriori* semantic relationship. Classes involved in one or several *a posteriori* semantic relationship may have properties which are not mapped.

NOTE It is not required that all characteristics of instances of each class be represented by properties (in the dictionary) and/or values in the library. Only properties defined in the ontology may be mapped.

The *a posteriori* semantic relationship is represented by an **A_POSTERIORI_SEMANTIC_RELATIONSHIP_Type** abstract XML complex type as illustrated in Figure 98.

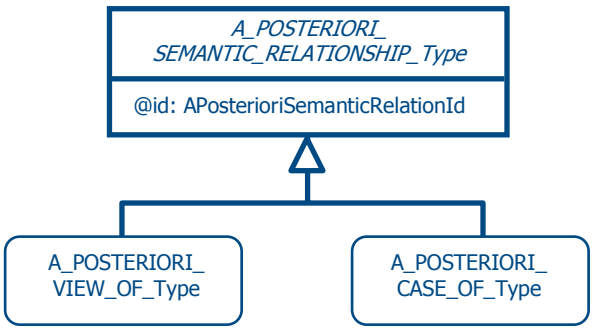

Figure 98 — *A posteriori* relationship general structure representation

A concrete *a posteriori* relationship is defined through one of the **A_POSTERIORI_SEMANTIC_RELATIONSHIP_Type** abstract XML complex type subtypes.

Internal item definitions:

@id: the *a posteriori* semantic relationship identifier.

Internal type definitions:

APosterioriSemanticRelationId: see 9.1.

External type definitions:

A_POSTERIORI_CASE_OF_Type: *a posteriori* a *case-of* relationship, see 8.6.1.

A_POSTERIORI_VIEW_OF_Type: *a posteriori view-of* relationship, see 8.6.2.

8.6.1 *A posteriori* mapping in a *case-of* relationship

An *a posteriori* case-of semantic relationship allows to define an inclusion relationship between classes that may belong to different reference dictionaries. When *A* is case-of *B*, this means that all instances of *A* are also instances of *B*. *A* is designated **case_of_sub** and *B* is designated **case_of_super**. All the property **mapping**s describe how properties of the **case_of_super** class, defined in their **range**, may be computed from properties of the **case_of_sub** class, defined in their **domain** (see Figure 99).

In OntoML, it is represented by an **A_POSTERIORI_CASE_OF_Type** XML complex type as illustrated in Figure 99.

NOTE 1 The *case-of* relationship allows each organization to define its own reference dictionary while providing for data integration and for data exchange with other organizations.

EXAMPLE 1 Assume that the **case_of_super** class belongs to a standard ontology, and that the **case_of_sub** class belongs to a user ontology. An *a posteriori* case-of relationship would allow the export of local data according to the standard ontology.

NOTE 2 When two classes, *A* and *B*, are equivalent, i.e., all instances of *A* are also instances of *B* and all instances of *B* are also instances of *A,* this may be represented using two *a posteriori* case-of semantic relationships with property mapping in the reverse side.

EXAMPLE 2 Assume that the **case_of_sub** class belongs to a standard ontology, and that the **case_of_super** class belongs to a user ontology. An *a posteriori* case-of relationship would allow the import of data described according to the standard ontology into the user data base.

NOTE 3 In this version of OntoML the only mapping function available is equality. But **MAPPING_FUNCTION_Type** is provided for latter standardization.

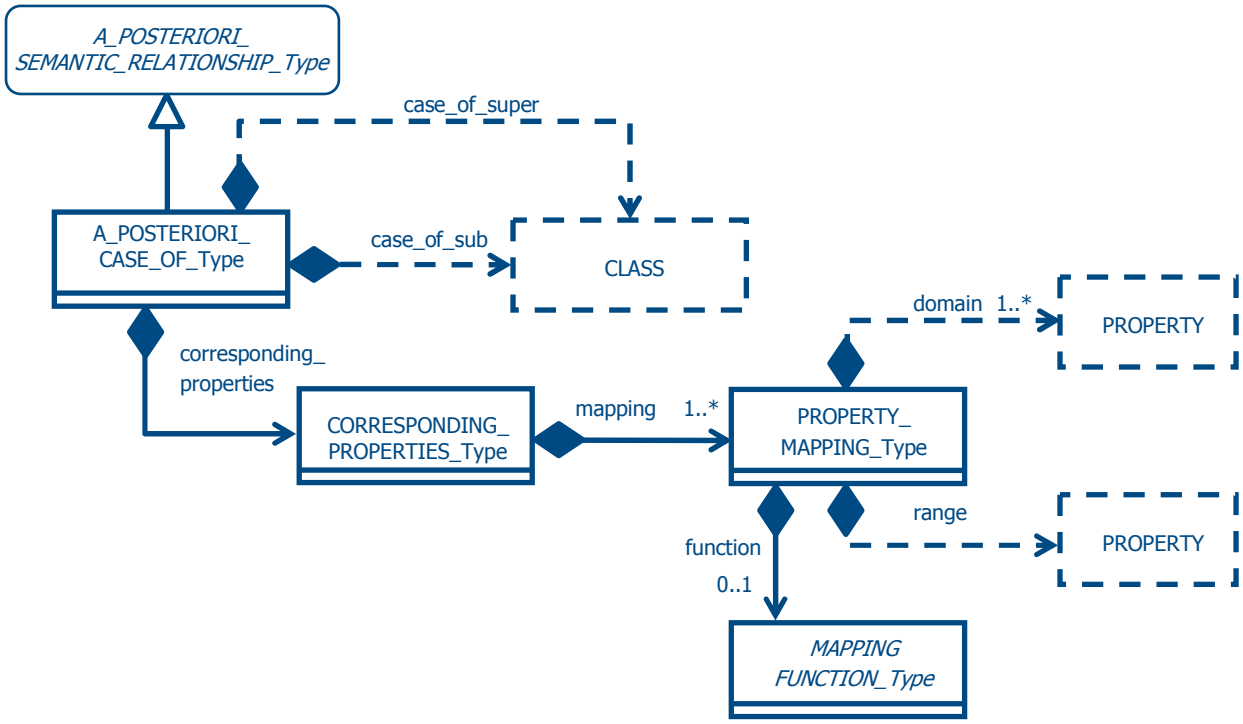

Figure 99 — *A posteriori case-of* relationship representation

Internal item definitions:

corresponding_properties: a set of property mappings.

corresponding_properties/mapping: a mapping that defines how the **range** property may be computed for each instance from the **domain** property (or properties) of the relationship by means of a function.

NOTE 4 When the **function** XML element is not provided, the **domain** XML element contains a single property and the mapping means a property equality of values between the **domain** property and the **range** property.

corresponding_properties/mapping/domain: the reference to the property or properties from which the range property is computed.

corresponding_properties/mapping/function: an optional function that specifies how the **range** property is computed from the **domain** property(ies). When not provided, the default function is that the **range** property is value equal for each instance to the **domain** property that shall be unique.

corresponding_properties/mapping/range: the reference to the property that is computed by the mapping.

case_of_sub: the reference to the class that is case of the **case_of_super** class of the relationship.

NOTE 5 **case_of_super** and **case_of_sub** referenced classes are both item classes or both functional model classes.

case_of_super: the reference to the class of which the **case_of_sub** class is case of.

Internal type definitions:

CORRESPONDING_PROPERTIES_Type: a container for the set of pairs of properties.

PROPERTY_MAPPING_Type: a property mapping specification.

MAPPING_FUNCTION_Type: an abstract XML complex type intended to represent a mapping function, and reserved for latter standardization.

External type definitions:

A_POSTERIORI_SEMANTIC_RELATIONSHIP_Type: see 8.6.

Constraint specifications:

Case_of_super class and **case_of_sub** class referenced by the **A_POSTERIORI_CASE_OF_Type** XML type shall be both item classes or both functional model classes.

Properties referenced by the **PROPERTY_MAPPING_Type** XML complex type by its **range** element shall belong to the **case_of_super** class.

Properties referenced by the **PROPERTY_MAPPING_Type** XML complex type by its **domain** element shall belong to the **case_of_sub** class.

When the **function** referenced by the **PROPERTY_MAPPING_Type** XML complex type does not exist, the **domain** property shall be unique.

8.6.2 *A posteriori* mapping in a *view-of* relationship

An *a posteriori* view-of relationship allows to associate a functional model class, called **model**, to a characterization class of items, called **item**. The **model** class provides additional descriptive properties for describing each item according to the discipline-oriented point of view defined by a functional view class. Each instance of **model** consists of a list of property-value pairs. The subset of these properties that is included in the **instance_identification** XML element constitutes the key properties of these instances.

Mapping some or all of these key properties with item properties allows to identify which model instance or instances may be matched with each **item** instance. The matching criterion is that a **model** instance is matched with an **item** instance if for all **model** properties mapped onto one (or a set of) **item** property, the value of the **model** instance property equals the result of the mapping of this property onto property or properties of this **item** instance.

NOTE 1 When all the key properties of the functional model class may be mapped onto properties of the class of items it is view of, each item is associated with at most one functional model which may be computed for the whole class by a left outer join.

EXAMPLE Assume that a manufacturer of screws decides to design an ontology for describing the manufactured screws. Only one kind of screws is manufactured. They may be described by five properties: *part number*, length, *thread diameter*, *euro price,* and *quantity of order.* Three ontology structures may be considered as described hereafter:

1 - Screw class, an **item_class** is the unique class of the ontology. The properties *part number*, *length*, *thread diameter* are all represented as applicable properties of this class. Technical properties and business properties are gathered within the same class.

2 - *Screw class*, an **item_class**, and *screw business class*¸ a **fm_class_view_of**, are the two classes of the ontology. The properties *part number*, *length* and *thread diameter* are represented as applicable properties of *screw class*. *Screw business class* is a functional model declared as view-of *screw class.* It imports *part number* from *screw class*. It declares *euro price* and *quantity of order* as applicable properties. The common property, *part number*, allows to make a join between the two classes while separating rigid characteristic properties and business oriented properties. Technical properties and business properties are separated into two classes connected by the a priori view-of relationship.

3 - *Screw class*, an **item_class**, and *screw business class*¸ a **functional_model_class**, are two classes that may be defined in the same or in two separate ontologies. The properties *part number*, *length* and *thread diameter* are represented as applicable properties of *screw class*. S*crew business class* declares three applicable properties: *screw_id*, *euro price* and *quantity of order*. *Screw_id* contains the part number of screw corresponding to the *price* and the *quantity of order*. This approach separates completely the technical description of the screw and its business oriented description. Now, if in some context it would be convenient to integrate the two kinds of data, this can be done by creating an *a posteriori* view-of relationship whose *screw business class* would be the **model**, *screw class would be the* ***item****, screw_id* would be the **range** of the unique **PROPERTY_MAPPING_Type**, and *part number* its **target**.

NOTE 2 The key properties of a functional model class may also contain view control variable properties imported from the functional view class referenced by the functional model class. In this case, the value of the view control variables properties must be defined by the user to get each particular model corresponding to the same item.

In OntoML, an *a posteriori* view-of relationship is represented by an **A_POSTERIORI_VIEW_OF_Type** XML complex type as illustrated in Figure 100.

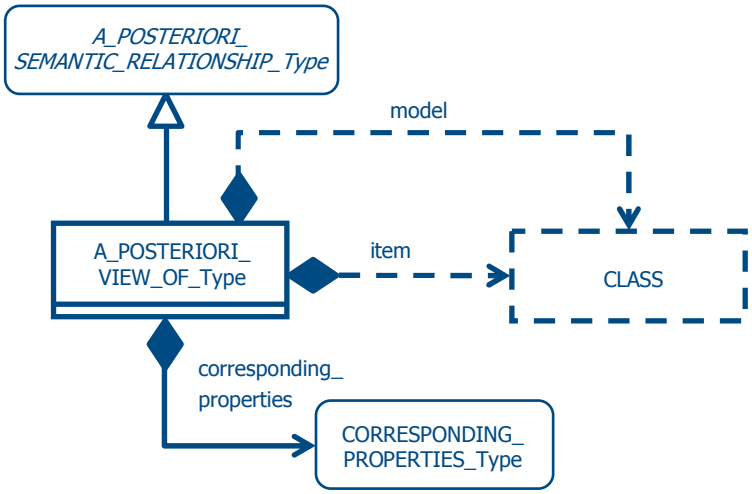

Figure 100 — *A posteriori* semantic relationships structure

Internal item definitions:

corresponding_properties: a set of property mappings.

model: the reference to the functional model class that will provide additional properties for the product of the **item** class.

item: the reference to the item class to which the functional model class is *view-of*.

External type definitions:

A_POSTERIORI_SEMANTIC_RELATIONSHIP_Type: see 8.6.

CORRESPONDING_PROPERTIES_Type: property mappings, see 8.6.1.

Constraint specifications:

The **model** class referenced by the **A_POSTERIORI_VIEW_OF_Type** XML complex type shall be a functional model class. The **item** class referenced by the **A_POSTERIORI_VIEW_OF_Type** XML complex type shall be an item class.

Properties referenced by the **range** element of the **PROPERTY_MAPPING_Type** XML complex type shall belong to **instance_identification** XML element of the **model** class.

Properties referenced by the **domain** element of the **PROPERTY_MAPPING_Type** XML complex type shall belong to the **item** class.

When the **function** referenced by the **PROPERTY_MAPPING_Type** XML complex type does not exist, the **domain** property shall be unique.

8.7 Data exchange specification identification

A data exchange specification allows to specify various characteristics of an OntoML document instance:

— the subset of the OntoML specification used,

— the possible view exchange protocols used when dealing with functional models associated to characterization classes,

— the implementation method namely in the case of OntoML, an OntoML document instance.

Two data exchange specification elements are defined. They allow to specify:

— characteristics of the exchanged ontology and/or library (see 8.7.1).

— characteristics of the possible view exchange protocols used (see 8.7.2).

8.7.1 Simple-level ontology data exchange specification: library integrated information model identification

The library integrated information model identification identifies the model on which the exchange is based and the format used.

NOTE 1 Annex C specifies standard values that are used when exchanging OntoML data. These values are also defined below in notes.

It is represented by a **LIBRARY_IIM_IDENTIFICATION_Type** XML complex type as illustrated in Figure 101.

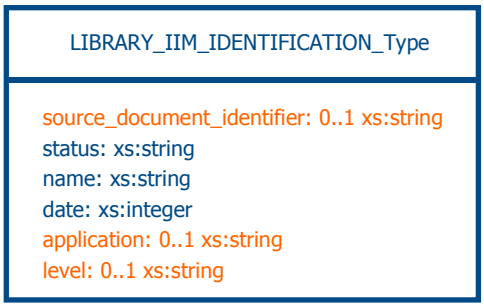

Figure 101 — Library integrated information model identification structure

Internal item definitions:

application: an identifier to characterize an allowed functional subset of the complete data specification.

NOTE 2 Values is 1 if conformance class 1 is used (simple level ontology), 2 if conformance class 2 is used (advanced level ontology), 3 if conformance class 3 is used (simple level library), 4 if conformance class 4 is used (advanced level library).

date: the year when the data specification reached its **status**.

level: an identifier that further characterizes an allowed subset of the **application** subset.

NOTE 3 This XML element is not used for OntoML exchanges.

name: the identifier of the data specification.

NOTE 4 For OntoML, this identifier is “ONTOML” in capital letters.

source_document_identifier: the identifier of the document that contains the data specification.

NOTE 5 For those documents issued by ISO TC184/SC4/WG2 this identifier is the integer part of the N number.

status: classification of the data specification with respect to its acceptance by the approving body of this International Standard, possibly followed by an integer version.

NOTE 6 A **status** may only take the following values: 'WD', 'CD', 'DIS', 'FDIS', 'IS', 'TS', 'PAS', 'ITA'.

8.7.2 Advanced-level ontology data exchange specification: view exchange protocol identification

A view exchange protocol is intended to specify which library external files are used within an OntoML document instance, which dictionary entries shall be recognized by a receiving system that claims conformance to this view exchange protocol, and which additional constraints are fulfilled by a library delivery file.

NOTE 1 An example of view exchange protocol is ISO 13584-102 for exchanging representations of the items described in a library by means of a representation conforming to one application protocol of ISO 10303.

A view exchange protocol identification identifies a particular view exchange protocol. It is represented by a **VIEW_EXCHANGE_PROTOCOL_IDENTIFICATION_Type** XML complex type as illustrated in Figure 102.

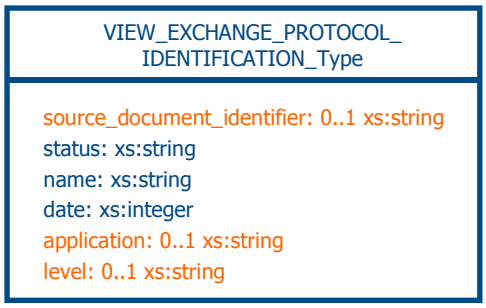

Figure 102 — View exchange protocol identification structure

Internal item definitions:

application: an identifier to characterize an allowed functional subset of the complete data specification as possibly defined in the view exchange protocol standards.

date: the year when the data specification reached its **status**.

level: an identifier that further characterizes an allowed subset of the **application** subset as possibly defined in the view exchange protocol standards.

name: the identifier of the data specification as defined in the view exchange protocol standards.

source_document_identifier: the identifier of the document that contains the data specification.

NOTE 2 For those documents issued by ISO TC184/SC4/WG2 this identifier is the integer part of the N number.

status: classification of the data specification with respect to its acceptance by the approving body of this International Standard, possibly followed by an integer version.

NOTE 3 A **status** may take the values: 'WD', 'CD', 'DIS', 'FDIS', 'IS', 'TS', 'PAS', 'ITA'.

8.8 Other structured information elements

This clause describes structured information elements that are used by the other information elements or CIIM ontology concept presented in the previous clauses.

8.8.1 Organization representation

An organization is an administrative structure. It is represented by the **ORGANIZATION_Type** XML complex type as illustrated in Figure 103.

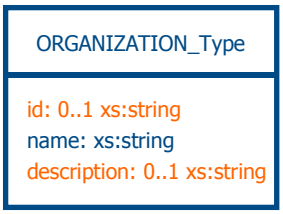

Figure 103 — Organization structure

Internal item definitions:

description: the text that relates the nature of the organization.

id: the identifier that distinguishes the organization.

name: the word or group of words by which the organization is referred to.

8.8.2 Mathematical string

A mathematical string construct provides resources for defining a representation for mathematical strings. It also allows a representation in the MathML format. The mathematical string structure is represented by the **MATHEMATICAL_STRING_Type** XML complex type as illustrated in Figure 104.

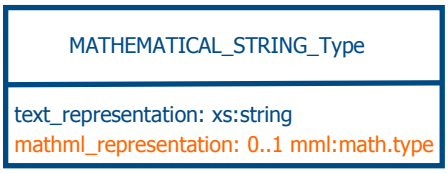

Figure 104 — Mathematical string structure

Internal item definitions:

mathml_representation: marked up according to the Mathematical Markup Language (MathML) Version 2.0 (Second Edition) specification (document Type Definition).

NOTE 1 MathML specification is available at the following URL: www.w3.org/TR/MathML2.

text_representation: "linear" form of a mathematical string.

NOTE 2 Use ISO 843 if necessary.

Internal item definitions:

mml:math.type: the Mathematical Markup Language construct that allows to specify mathematical expressions.

NOTE 3 **mml** specifies the namespace prefix associated to the following MathML namespace: http://www.w3.org/1998/Math/MathML.

8.8.3 Geometric context

A geometric context is a coordinate space. It is represented by the **GEOMETRIC_CONTEXT_Type** XML complex type as illustrated in Figure 105.

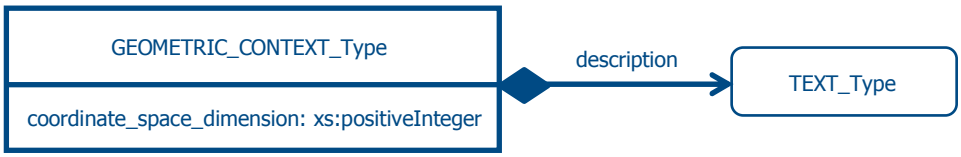

Figure 105 — Geometric context structure

Internal item definitions:

description: a description of the reference coordinate system of the object in which the **geometric_representation_context** is defined.

coordinate_space_dimension: the positive integer dimension count of the coordinate space which is the geometric context.

8.8.4 Geometric unit context

A geometric unit context specifies the units that apply in a geometric context. It is represented by the **GEOMETRIC_UNIT_CONTEXT_Type** XML complex type, as illustrated in Figure 106.

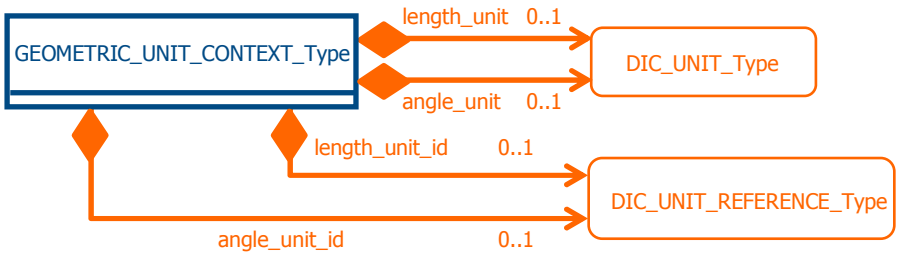

Figure 106 — Geometric unit context structure

Internal item definitions:

angle_unit: the angle unit which applies in a geometric context.

angle_unit_id: the identifier of the angle unit which applies in geometric context.

length_unit: the length unit which applies in a geometric context.

length_unit_id: the identifier of the length unit which applies in geometric context.

NOTE 1 At least the **length_unit** shall be defined. It can be associated to an **angle_unit**.

NOTE 2 When both **angle_unit** and **angle_unit_id** are provided, **angle_unit** takes precedence.

NOTE 3 When both **length_unit** and **length_unit_id** are provided, **length_unit** takes precedence.

External type definitions:

DIC_UNIT_Type: see 8.4.

DIC_UNIT_REFERENCE_Type: see 8.3.7.

Constraint specifications:

Either a **length_unit** is provided, or a **length_unit_id** or both.

9 OntoML exchange structure

OntoML defines a set of ontology-based XML Schemas for exchanging ontology and catalogue data. OntoML is modularized. It is constituted of a set of XML schemas, each of them carrying a part of the CIIM modeling constructs.

Beside the OntoML logical structure presented in the clauses 5, 6 and 7, this clause specifies physical level requirements that shall be fulfilled by any XML document instance that claims to be represented according to OntoML standard.

First, the identifier structure of the OntoML CIIM ontology concept is specified. Second, the specific namespace that allows to unambiguously reference resources coming from the OntoML vocabulary is defined. Third, the modular structure of OntoML is described. Finally, OntoML subsets and their corresponding conformance classes are specified.

9.1 Identifiers of CIIM ontology concepts

In this part of ISO 13584, global identifiers are used to identify and to reference CIIM ontology concepts. Their structure and content are specified in this clause.

OntoML identifiers are built according to the ISO/TS 29002-5 identification scheme structure.

NOTE 1 ISO/TS 29002-5 defines an identifier structure based on the ISO/IEC 11179-5 International Registration Data Identifier (IRDI). In ISO/IEC 11179-5, a global identifier, called an IRDI, is assigned to administered items submitted to registration.

NOTE 2 All CIIM ontology concepts and ontology as a whole are identified by an IRDI.

Such an IRDI is threefold:

— a Registration Authority Identifier (**RAI**) intended to identify unambiguously the organization in charge of delivering data identifiers;

— a Data Identifier (**DI**) uniquely defined for an **RAI**, intended to identify an administered item;

— a Version Identifier (**VI**) intended to be used for change management of administered items.

According to ISO/TS 29002-5, an OntoML ontology entry identifier is defined according to the following structure:

```
ontoml_concept_identifier ::= RAI '#' DI '#' VI
```

In the next clauses, the structure of each part of these IRDI is presented.

9.1.1 Registration Authority Identifier (RAI) structure

An organization wishing to assign identifiers to OntoML ontology entry shall be itself unambiguously identified. This organization identification shall be defined according to the rules prescribed in ISO/TS 29002-5.

ISO/TS 29002-5 rules specify a structure for the identification of organizations. It consists of two mandatory parts followed by three optional parts

The mandatory parts are as follows:

— the International Code Designator (ICD);

— the identification of an organization within an identification scheme: a data element containing an organization identifier (OI);

The optional parts are as follows:

— the identification of an organization part: a data element containing an organization part identifier (OPI);

— the OPI source indicator (OPIS): a data element containing a code value indicating the source of the OPI;

— some additional information (AI): a data element containing information assigned by the issuing organization.

When an ontology is described in a standardized document, the ISO/TS 29002-5 rules are not sufficient for representing accurately the registration authority identifier. Thus, the standard document number shall be supplied as part of the general registration authority identifier in the AI part.

In such a case, the RAI of CIIM ontology concepts and ontologies defined in a standard document shall consist of three parts:

— the International Code Designator (ICD), fixed to "0112", where the standardization organization shall be identified;

NOTE 1 the ICD "0112" is defined in ISO 6523 List B, the numeric list of all ICDs that have been issued.

— the organization identifier (OI)of the standardized organization in the coding scheme "0112";

NOTE 2 In the coding scheme (ICD) "0112", the ISO OI equals to 1, the IEC OI equals to 2, the ISO/IEC OI equals to 3.

— the identification of the standard itself in the AI part:

  — the number of the standard (NB);

  — the part number (PART);

NOTE 3 If the standard part is not part of a multipart series, the part number is represented by the empty string.

  — the edition number (ED);

NOTE 4 The identification of CIIM ontology concepts and ontologies defined in a standard document do not require the specification of both the OPI and the OPIS.

Thus, an OntoML **RAI** is defined according to the following syntactical structure:

```
AI ::= NB '_' PART '_' ED

RAI1 ::= ICD '-' OI ('-' OPI ('-' OPIS)? )?

RAI2 ::= ICD '-' OI '---' AI

RAI ::= RAI1 | RAI2
```

NOTE 5 The above is compatible but more restrictive than ISO/TS 29002-5.

Where

— the “?” character stands for the optional operator;

— the “|” character stands for the logical disjunction operator;

— parenthesis are meta-notations characters for defining groups;

— characters between quotes (‘) are terminal characters that may appear in the RAI;

— identifiers outside quotes shall have the following structure:

  — ICD: a string of exactly 4 numeric characters;

NOTE 6 Leading 0 are concatenated to the ICD in case of an ICD length smaller than 4 characters.

  — OI: a string up to 35 alphanumeric characters;

  — OPI: a string up to 35 alphanumeric characters;

  — OPIS: a single numeric character;

  — NB: a string up to 10 alphanumeric characters;

NOTE 7 For ISO and IEC standards, NB consists only of digits;

  — PART: a string (that may be empty) up to 10 alphanumeric characters;

NOTE 8 For ISO and IEC standards, PART consists only of digits.

  — ED: a string up to 5 numeric characters.

NOTE 9 For ISO and IEC standards, ED consists only of digits.

EXAMPLE 1 The French “FOO” company is identified (ORGID) by the “12345678901234” SIRENE number according to the French codification system for identifying companies (ICD=“0002”). The corresponding organization identifier would then be:

0002-12345678901234

EXAMPLE 2 The identifier (ORGID) allocated to a big device manufacturer is “123456789”. It is allocated as a DUNS number for identifying companies (ICD=”0060”). The particular plant of the manufacturer located in “Town” is also identified (OPI) by the following identifier: “12345”. Because this OPI identifier is allocated by the company itself, the OPIS is set to 1. The corresponding organization identifier would then be:

0060-123456789-12345-1

EXAMPLE 3 IEC 61360-4 is a standard that specifies a data dictionary about electronic components. This standard is edited by IEC. IEC is identified (ORGID="2") identifier according to the ISO register for standards producing organization (ICD="112"). This standard number (NB) is equal to "61360", and the specific part (PART) in which the data dictionary is defined is numbered "4". If we assume that we want to reference the first edition of this document (ED="1"), the corresponding organization identifier would then be:

0112-2---61360_4_1

9.1.2 Version Identifier (VI) structure

Each CIIM ontology concept is versioned for life cycle management purposes. In order to unambiguously identify a version of each concept, its corresponding version shall also be represented.

The OntoML VI is defined as follows:

— VI: a string up to 10 numeric characters.

NOTE The above is compatible but more restrictive than ISO/TS 29002-5.

9.1.3 Data Identifier (DI) structure

Each ontology entry defined in a given ontology shall be assigned an identifier called a data identifier. It is defined under the responsibility of an identified registration authority. Therefore, it is assumed that every data identifier assigned to a CIIM ontology concept instance shall be unique in the context of this registration authority.

In OntoML, the data identifier (DI) is defined as follows:

— the code space identifier (CSI): identification of a domain within which each code denotes a single meaning;

EXAMPLE In OntoML, a class and a property may have the same code. Using two different code space identifiers to define the coding space of the class code and the property code allows to make the resulting identifiers unambiguous.

NOTE 1 IS0/TS 29002-5 specifies a list of registered code space identifiers.

— the item code (IC);

The OntoML data identifier is defined according to the following syntactical structure:

```
DI ::= CSI '-' IC
```

Where:

— CSI: a string of exactly 2 alphanumeric characters:

  — "01": code space identifier of a class;

  — "02": code space identifier of a property;

  — "04": code space identifier of a constraint;

  — "05": code space identifier of a unit;

  — "07": code space identifier of a value code;

  — "08": code space identifier of a currency;

  — "09": code space identifier of a data type;

— "10": code space identifier of a document;

— "11": code space identifier for an ontology or a library;

NOTE 2 CSI are defined according to IS0/TS 29002-5.

— IC: a string up to 71 alphanumeric characters.

9.1.3.1 DI for classes

According to ISO/TS 29002-5, a class is associated to a code space identifier (CSI) equal to '01'.

The DI for a class consists of three parts:

— the code space identifier for the class;

— the hyphen ("-") character;

— the code of the class.

EXAMPLE ISO 13584-511 specifies an ontology dealing with fasteners. In this ontology, a product characterization class called “hexagon head bolt” is defined. The class code space identifier (CSI) is equal to "01". Its code (IC) is equal to “P511AAA156”, and its version (VI) is set to “1”. The corresponding product characterization class identifier would then be:

0112-1---13584_511_1#01-P511AAA156#1

9.1.3.2 DI for properties, data types and documents

The DI of a property, data type or document is defined as follows:

— the code space identifier for the property, data type or document;

— the hyphen ("-") character;

— the code of the property, data type or document.

EXAMPLE ISO 13584-501 specifies an ontology dealing with measuring instruments. In the “laboratory measuring instrument” product characterization class, a property called “accuracy rating" property is defined. According to ISO/TS 29002-5, the property code space identifier (CSI) is equal to "02". This property is identified (IC) by the “P501_P000178” code, and is associated to a version (VI) equal to “000000001”. The corresponding property identifier would then be:

112-1---13584_501_1#02-P501_P000178#000000001

9.1.3.3 DI for units, currencies, constraints, value codes and *a posteriori* semantic relationships

OntoML allows to reference units, currencies, constraints, value codes and *a posteriori* semantic relationships using IRDIs. The unit, currency, constraint or value code is assigned by the RAI. This code constitutes the data identifier (DI) of the unit, currency, constraint or value code.

The DI for a unit, a currency, a constraint, a value code or an *a posteriori* semantic relationship consists of three parts:

— the code space identifier for the unit, currency, constraint, value code or *a posteriori* semantic relationships;

— the hyphen ("-") character;

— the code of the unit, currency, constraint, value code or *a posteriori* semantic relationships.

9.1.3.4 DI for ontologies and libraries

OntoML also allows to associate an IRDI globally to an ontology or a library. The ontology or library code may be assigned by the RAI. The only constraint is that this code is unique over all the classes, ontologies and libraries defined by this RAI. This code constitutes the data identifier (DI) of the ontology or library.

The DI for an ontology or library consists of three parts:

— the code space identifier for the ontology;

— the hyphen ("-") character;

— the code of the ontology or library.

EXAMPLE The first version of the ontology identified by the “99999” code is defined by a big device manufacturer. According to ISO/TS 29002-5, the ontology code space identifier (CSI) is equal to "11". This manufacturer is identified by “123456789”. This manufacturer code is allocated as a DUNS number for identifying companies (ICD=”0060”). The particular plant of the manufacturer located in “Town” is also identified (OPI) by the following identifier: “12345”. Because this OPI identifier is allocated by the company itself, the OPIS is set to 1. The corresponding ontology identifier would then be:

0060-123456789-12345-1#11-99999#1

9.1.4 OntoML identifier representation

OntoML CIIM ontology concepts identifiers are represented using a mandatory XML attribute assigned to the corresponding CIIM ontology concept. The name of this attribute is always **id**. Its specific datatype depends on the kind of CIIM ontology concept that is intended to be identified. These types are as follows:

— **SupplierId** type for a supplier ontology concept,

— **ClassId** type for a class ontology concept,

— **PropertyId** type for a property ontology concept,

— **DatatypeId** type for a datatype ontology concept,

— **DocumentId** type for a document ontology concept.

Moreover, the identifier type of an ontology and / or a library, of a unit, a constraint, a value code or an *a posteriori* semantic relationship is defined as follows:

— **OntologyId** for an ontology or a library,

— **DicUnitId** for a unit,

— **CurrencyId** for a currency,

— **ConstraintId** for a constraint,

— **ValueCodeId** for a value code,

— **APosterioriSemanticRelationId** for an *a posteriori* semantic relationship.

9.2 OntoML namespace

OntoML defines a namespace based on both a URN and a URI specifications.

OntoML processors shall use the XML namespaces mechanism to recognize elements and attributes from this namespace. Vendors shall not extend the OntoML namespace with additional elements or attributes.

This specification does not use any prefix for referring to elements in the OntoML namespace. However, OntoML documents are free to use any prefix, provided that there is a namespace declaration that binds the prefix to the URI of the OntoML namespace.

9.2.1 OntoML URN

The OntoML namespace has the following URN:

urn:iso:std:iso:13584:-32:ed-1:tech:xml-schema:ontoml

NOTE The URN is defined according to the ISO URN scheme for resources defined in ISO TC184/SC4 standards.

9.2.2 OntoML URI

The OntoML namespace has the following URI:

http://www.tc184-sc4.org/2010/OntoML.

NOTE The “2010“ substring in the URI indicates the year in which the URI was allocated. It does not indicate the version of the OntoML module being used, which is specified by attributes.

9.3 Modular structure

OntoML is a set of XML schemas, called modules, that define general resources on which a top level schema is built. The list of modules is given hereafter:

— **ontoml.xsd**: the main XML schema. It defines the upper level of the OntoML ontology exchange format. It is based on the definition of a single **ontoml** XML element whose content is a mandatory header, followed by an optional dictionary specification, followed by an optional content specification.

— **header.xsd**: this module contains management resources for characterizing the content (dates, authors …) of an OntoML XML file.

— **dictionary.xsd**: this module defines the container of any CIIM compliant ontology.

— **supplier.xsd**: this module contains the resources intended to represent an information source according to the structure defined in Clause 6.7.1.

— **class.xsd**: this module contains the resources intended to represent an ontology class according to the structure defined in Clause 6.7.2, whatever be its nature (a general model class, a functional model class or a functional view class).

— **property.xsd**: this module contains the resources intended to represent a property according to the structure defined in Clause 6.7.4, whatever be its nature (a characteristic property, a context property, a context dependent property or a representation property).

— **datatype.xsd**: this module contains the constructs intended to represent data types according to the structure defined in Clause 6.7.6.

— **document.xsd**: this module contains the constructs intended to represent a document according to the structure defined in Clause 6.7.7.

— **type_system.xsd**: this module contains an XML representation of the complete OntoML type system presented in Clause 8.3.

— **translation.xsd**: this module contains resources for defining multilingual representations of clear text as presented in Clause 8.1.

— **external_file.xsd**: this module contains constructs for referencing external resources that may be either exchanged together with the OntoML document instance or referenced in the Internet; they are presented in Clause 8.2.

— **unit.xsd**: this module contains constructs for explicitly describing any kind of units according to the structure defined in Clause 8.4.

— **a_posteriori.xsd**: this module contains constructs for representing *a posteriori* mappings between classes and properties defined in different ontologies according to the structure defined in Clause 8.6.

— **constraint.xsd**: this module contains constructs intended to represent constrained properties according to the structure defined in Clause 8.5.

— **basic.xsd**: this module contains all the OntoML simple an complex type definitions shared by OntoML modules.

— **identifier.xsd**: this module provides the constructs needed for identifying and referencing CIIM ontology concepts as presented in Clause 9.1;

— **content.xsd**: this module defines specifies the structure of families of products belonging to a library;

— **library.xsd**: this module defines structure for representing libraries of families of products, by their product characterizations possibly associated with the ontology where product characterizations classes and properties are defined.

Table 1 illustrates the reference relationships between these modules where gray squares specify the relationship from one module to another module.

Table 1 — OntoML modules cross-references

references →	basic.xsd	identifier.xsd	translation.xsd	unit.xsd	external_file.xsd	header.xsd	supplier.xsd	type_system.xsd	datatype.xsd	constraint.xsd	property.xsd	class.xsd	document.xsd	a_posteriori.xsd	dictionary.xsd	content.xsd	library.xsd	ontoml.xsd
basic.xsd	■																	
identifier.xsd		■																
translation.xsd	▒	▒	■															
unit.xsd	▒			■														
external_file.xsd	▒	▒	▒	▒	■													
header.xsd		▒	▒		▒	■												
supplier.xsd	▒	▒	▒				■											
type_system.xsd	▒	▒	▒	▒	▒			■										
datatype.xsd	▒	▒	▒		▒			▒	■									
constraint.xsd	▒	▒						▒		■								
property.xsd	▒	▒	▒		▒			▒			■							
class.xsd	▒	▒	▒		▒					▒		■						
document.xsd	▒	▒	▒		▒								■					
aPosteriori.xsd		▒												■				
dictionary.xsd	▒	▒					▒		▒		▒	▒	▒	▒	■			
content.xsd	▒	▒		▒	▒											■		
library.xsd		▒														▒	■	
ontoml.xsd						▒									▒		▒	■

9.4 Levels of exchange and conformance classes

OntoML integrates a set of resource constructs into a single XML schema for representing ontologies and possibly supplier libraries for the purpose of exchange. In order to support various levels of exchange requirements, four functional subsets of OntoML have been identified as allowed levels of OntoML exchange. These various functional subsets are called conformance classes.

These conformance classes also specify allowed levels of OntoML implementation for those systems that claim conformance to the OntoML international standard.

The four conformance classes of OntoML are defined as follows:

— simple ontology: an ontology that define hierarchies of classes of items on the base of the common ISO/IEC dictionary schema together with the required external resources, documents, data types and collection data types, but without description of item representations (i.e., functional model classes) and of representation categories of items (i.e., functional view classes), corresponds to OntoML conformance class 1;

— advanced ontology: an ontology that corresponds to a simple ontology with the addition of resources for defining hierarchies of item representations (i.e., functional model classes) and of representation categories of items (i.e., functional view classes), corresponds to conformance class 2;

— simple library: a library content defining products on the basis of simple ontologies, possibly exchanged together with simple ontology definitions correspond to conformance class 3;

— advanced library: a library content defining products and product representations on the basis of advanced ontologies, possibly exchanged together with advanced ontology definitions corresponds to conformance class 4;

Table 2 summarizes the supported capabilities of the different conformance classes of OntoML.

Table 2 — Conformance options of OntoML

	Ontology		**Library**	
Conformance Class	Common ISO/IEC dictionary definitions Documents Collections External resources	Representations Representation categories	Library of products	Library of representations
1	X			
2	X	X		
3	X		X	
4	X	X	X	X

Annex C defines the standard data that allow to precisely identify one conformance class among all the available conformance classes.

9.5 Conformance class requirements

9.5.1 Conformance class 1

Conformance class 1 addresses those implementations that support ontologies that define hierarchies of classes of items on the base of the common ISO/IEC dictionary schema together with the required external resources, document ontology concept, data type ontology concepts and collection data types. It shall support standardized data defined in Annex C. It shall support the following XML complex types:

— From *ontoml.xsd*:

  — **ONTOML_Type**

— From *identifiers.xsd*:

  — **CLASS_REFERENCE_Type**

  — **CLASSES_REFERENCE_Type**

  — **DATATYPE_REFERENCE_Type**

- **DATATYPES_REFERENCE_Type**
- **DIC_UNIT_REFERENCE_Type**
- **DIC_UNITS_REFERENCE_Type**
- **DICTIONARY_REFERENCE_Type**
- **DICTIONARIES_REFERENCE_Type**
- **DOCUMENT_REFERENCE_Type**
- **DOCUMENTS_REFERENCE_Type**
- **PROPERTY_REFERENCE_Type**
- **PROPERTIES_REFERENCE_Type**
- **SUPPLIER_REFERENCE_Type**
- **SUPPLIERS_REFERENCE_Type**

NOTE 1 **idt:IRDI** is defined outside OntoML. The *idt* prefix stands for the namespace identified by the following URN: *urn:iso:std:iso:29002:-5:ed-1:tech:schema:identifier*.

- From *header.xsd*:
  - **HEADER_Type**
  - **INFORMATION_Type**
  - **LIBRARY_IIM_IDENTIFICATION_Type**
- From *dictionary.xsd*:
  - **A_POSTERIORI_SEMANTIC_RELATIONSHIPS_Type**
  - **CONTAINED_CLASSES_Type**
  - **CONTAINED_DOCUMENTS_Type**
  - **CONTAINED_DATATYPES_Type**
  - **CONTAINED_PROPERTIES_Type**
  - **CONTAINED_SUPPLIERS_Type**
  - **DICTIONARY_IN_STANDARD_FORMAT_Type**
  - **DICTIONARY_Type**
- From *supplier.xsd*:
  - **SUPPLIER_Type**

— From *class.xsd*:

  — **ASSIGNED_VALUE_Type**

  — **CLASS_CONSTANT_VALUES_Type**

  — **CLASS_Type**

  — **CLASS_VALUE_ASSIGNMENT_Type**

  — **CATEGORIZATION_CLASS_Type**

  — **ITEM_CLASS_CASE_OF_Type**

  — **ITEM_CLASS_Type**

  — **val:value**

NOTE 2 **val:value** is defined outside OntoML. The *cat* prefix stands for the namespace identified by the following URN: *urn:iso:std:iso:29002:-10:ed-1:tech:schema:catalogue*.

— From *property.xsd*:

  — **CONDITION_DET_Type**

  — **DEPENDENT_P_DET_Type**

  — **NON_DEPENDENT_P_DET_Type**

  — **PROPERTY_Type**

  — **SYNONYMOUS_SYMBOLS_Type**

— From *datatype.xsd*:

  — **DATATYPE_Type**

— From *datatypeSystem.xsd*:

  — **ALTERNATIVE_UNITS_Type**

  — **ANY_TYPE_Type**

  — **ARRAY_TYPE_Type**

  — **BAG_TYPE_Type**

  — **BOOLEAN_TYPE_Type**

  — **CLASS_REFERENCE_TYPE_Type**

  — **DATE_DATA_TYPE_Type**

  — **DATE_TIME_DATA_TYPE_Type**

  — **DIC_VALUE_Type**

- **INT_CURRENCY_TYPE_Type**
- **INT_MEASURE_TYPE_Type**
- **INT_TYPE_Type**
- **INT_DIC_VALUE_Type**
- **ITS_VALUES_Type**
- **LEVEL_Type**
- **LEVEL_TYPE_Type**
- **LIST_TYPE_Type**
- **NAMED_TYPE_Type**
- **NON_QUANTITATIVE_CODE_TYPE_Type**
- **NON_QUANTITATIVE_INT_TYPE_Type**
- **NON_TRANSLATABLE_STRING_TYPE_Type**
- **NUMBER_TYPE_Type**
- **REAL_CURRENCY_TYPE_Type**
- **REAL_MEASURE_TYPE_Type**
- **REAL_TYPE_Type**
- **REMOTE_HTTP_ADDRESS_Type**
- **REPRESENTATION_REFERENCE_TYPE_Type**
- **SET_TYPE_Type**
- **SET_WITH_SUBSET_CONSTRAINT_TYPE_Type**
- **STRING_DIC_VALUE_Type**
- **STRING_TYPE_Type**
- **TIME_DATA_TYPE_Type**
- **TRANSLATABLE_STRING_TYPE_Type**

— From *translations.xsd*:

- **DOCUMENT_IDENTIFIER_NAME_LABEL_Type**
- **DOCUMENT_IDENTIFIER_Type**
- **GENERAL_TEXT_Type**
- **KEYWORD_LABEL_Type**

- **KEYWORD_Type**
- **LANGUAGE_Type**
- **PREFERRED_NAME_LABEL_Type**
- **PREFERRED_NAME_Type**
- **SHORT_NAME_LABEL_Type**
- **SHORT_NAME_Type**
- **SYNONYMOUS_NAME_LABEL_Type**
- **SYNONYMOUS_NAME_TYPE_Type**
- **TEXT_Type**
- **TRANSLATION_DATA_Type**
- **TRANSLATION_Type**

— From *externalFiles.xsd*:

- **EXTERNAL_GRAPHICS_Type**
- **EXTERNAL_RESOURCE_Type**
- **EXTERNAL_FILES_Type**
- **GRAPHICS_Type**
- **HTTP_FILE_Type**
- **IDENTIFIED_DOCUMENT_Type**
- **REFERENCED_DOCUMENT_Type**
- **REFERENCED_GRAPHICS_Type**
- **SOURCE_DOCUMENT_Type**

— From *units.xsd*:

- **CONTEXT_DEPENDENT_UNIT_Type**
- **CONVERSION_BASED_UNIT_Type**
- **DERIVED_UNIT_ELEMENT_Type**
- **DERIVED_UNIT_Type**
- **DIC_UNIT_Type**
- **DIMENSIONAL_EXPONENTS_Type**
- **NAMED_UNIT_Type**

- **NON_SI_UNIT_Type**
- **SI_UNIT_Type**
- **UNIT_Type**

— From *constraints.xsd*:

- **CARDINALITY_CONSTRAINT_Type**
- **CLASS_CONSTRAINT_Type**
- **CONFIGURATION_CONTROL_CONSTRAINT_Type**
- **CONSTRAINT_OR_CONSTRAINT_ID_Type**
- **CONSTRAINT_Type**
- **CONSTRAINTS_Type**
- **CONTEXT_PARAMETER_CONSTRAINTS_Type**
- **CONTEXT_RESTRICTION_CONSTRAINT_Type**
- **DOMAIN_CONSTRAINT_Type**
- **ENUMERATION_CONSTRAINT_Type**
- **FILTER_Type**
- **INTEGRITY_CONSTRAINT_Type**
- **POSTCONDITION_Type**
- **PRECONDITION_Type**
- **PROPERTY_CONSTRAINT_Type**
- **RANGE_CONSTRAINT_Type**
- **STRING_PATTERN_CONSTRAINT_Type**
- **STRING_SIZE_CONSTRAINT_Type**
- **SUBCLASS_CONSTRAINT_Type**
- **SUBSET_Type**

— From *baseTypes.xsd*:

- **MATHEMATICAL_STRING_Type**
- **ORGANIZATION_Type**
- **mml:math.type**

NOTE 3 **mml:math.type** is defined outside OntoML. The *mml* prefix stands for the namespace identified by the following URI: *http://www.w3.org/1998/Math/MathML*.

— From *document.xsd*:

  — **AUTHORS_Type**

  — **DOCUMENT_CONTENT_Type**

  — **DOCUMENT_Type**

  — **PERSON_Type**

  — **REMOTE_LOCATIONS_Type**

  — **STRINGS_Type**

— From *aPosteriori.xsd*:

  — **A_POSTERIORI_CASE_OF_Type**

  — **A_POSTERIORI_SEMANTIC_RELATIONSHIP_Type**

  — **CORRESPONDING_PROPERTIES_type**

  — **MAPPING_FUNCTION_Type**

  — **PROPERTY_MAPPING_Type**

9.5.2 Conformance class 2

Conformance class 2 addresses those implementations that support ontologies that define hierarchies of classes of items on the base of the common ISO/IEC dictionary schema together with the required external resources, document ontology concept, data type ontology concepts collection data types, description of hierarchy of item representations and of representation categories of items. It shall support standardized data defined in Annex C. It shall also supports complex types and related constructs defined for conformance class 1, more the following complex types and related constructs:

— From *header.xsd*:

  — **SUPPORTED_VEP_Type**

  — **VIEW_EXCHANGE_PROTOCOL_IDENTIFICATION_Type**

— From *class.xsd*:

  — **FM_CLASS_VIEW_OF_Type**

  — **FUNCTIONAL_MODEL_CLASS_Type**

  — **GEOMETRIC_CONTEXT_Type**

  — **GEOMETRIC_UNIT_CONTEXT_Type**

  — **NON_INSTANTIABLE_FUNCTIONAL_VIEW_CLASS_Type**

  — **V_C_V_RANGE_Type**

  — **VIEW_CONTROL_VARIABLE_RANGE_Type**

— From *property.xsd*:

  — **REPRESENTATION_P_DET_Type**

— From *datatypeSystem.xsd*:

  — **AXIS1_PLACEMENT_TYPE_Type**

  — **AXIS2_PLACEMENT_3D_TYPE_Type**

  — **AXIS2_PLACEMENT_2D_TYPE_Type**

  — **PLACEMENT_TYPE_Type**

  — **PROGRAM_REFERENCE_TYPE_Type**

  — **REPRESENTATION_REFERENCE_TYPE_Type**

— From *aPosteriori.xsd*:

  — **A_POSTERIORI_VIEW_OF_Type**

9.5.3 Conformance class 3

Conformance class 3 addresses those implementations that support exchange of products with dictionary definitions. It shall support standardized data defined in Annex C. It shall also supports complex types and related constructs defined for conformance class 1, more the following complex types and related constructs:

— From *externalFiles.xsd*:

  — **ILLUSTRATION_Type**

  — **MESSAGE_Type**

— From *library.xsd*:

  — **CONTAINED_CLASS_EXTENSIONS_Type**

  — **LIBRARY_IN_STANDARD_FORMAT_Type**

  — **LIBRARY_Type**

— From *content.xsd*:

  — **CLASS_EXTENSION_Type**

  — **CLASS_PRESENTATION_ON_PAPER_Type**

  — **CLASS_PRESENTATION_ON_SCREEN_Type**

  — **CLASSIFICATION_Type**

  — **CREATE_ICON_Type**

  — **EXPLICIT_ITEM_CLASS_EXTENSION_Type**

  — **PROPERTY_VALUE_RECOMMENDED_PRESENTATION_Type**

— **PROPERTY_CLASSIFICATION_Type**

— **RECOMMENDED_PRESENTATION_Type**

— **cat:catalogue_Type**

NOTE **cat:catalogue_Type** is defined outside OntoML. The *cat* prefix stands for the namespace identified by the following URN: *urn:iso:std:iso:29002:-10:ed-1:tech:schema:catalogue*.

9.5.4 Conformance class 4

Conformance class 4 addresses those implementations that support exchange of products and engineering models with dictionary definitions. It shall support standardized data defined in Annex C. It shall also supports complex types and related constructs defined for conformance class 3.

— From *content.xsd*:

— **CONTEXT_PARAM_ICON_Type**

— **EXPLICIT_FUNCTIONAL_MODEL_CLASS_EXTENSION_Type**

10 Dictionary Change Management Rules

This clause defines the rules for organizing, controlling and tracking changes on reference dictionaries.

Given a particular version O_t of a reference dictionary, and a set of product descriptions based on version $O_{t'}$ of this dictionary, the goal of these rules are (1) to permit to decide whether the available O_t dictionary allows to interpret correctly the available product descriptions, based on $O_{t'}$, (2) when it is not the case, to decide which part of the O_t reference dictionary should be updated to allows to interpret correctly the available product descriptions.

NOTE The whole discussion on the dictionary management rules presented in this clause is based on product characterization. Categorization classes being not used in product characterization, they are not concerned in the discussion. Thus in this clause "class" means "characterization class", and "ontology concepts" means "all ontology concepts but categorization classes". The rule for categorization class are defined in Rule 8.

10.1 Principle of ontological continuity

The role of a domain ontology in the ISO 13584 series, called a reference dictionary for this domain, is to allow:

— exchange of unambiguous information about products between business partners, and

— storage of stable product characterizations in various persistent repositories.

The method used is to encode each product by a characterization that consists of:

— the characterization class to which the product belongs, and

— a set of property-value pairs, where the properties are selected among the properties that are applicable to this characterization class.

EXAMPLE When a product is represented, using the ISO 13584-511 reference dictionary for fasteners, as a cap nut with a nominal diameter of 5 and a height of nut of 4; for short:

cap nut (nominal diameter = 5; height of nut = 4),

this characterization represents a product that is an "hexagon nut closed at one side by a flat cap" whose "nominal thread diameter" is 5 mm and whose "overall height of nut" is 4 mm.

NOTE 1 In the computer-to-computer exchange format, both the characterization class and the properties appearing in a characterization are represented using codes that include the version number of these ontology concepts. In the above example, characterization is represented by class and property preferred names to make them understandable.

NOTE 2 In this ontology-based approach, each product is represented as an ontology instance.

The fundamental assumption on which this encoding is based is that:

— in an exchange, both the sender and the receiver must associate the same meaning to the same characterization, and

— a characterization recorded at time = t must be interpreted with the same meaning at time = t+1, even if the reference dictionary evolves between time t and t+1.

In general there are two solutions which allow respecting this fundamental assumption:

— In the case where the reference dictionary has changed between t and t+1 and where there are no restrictions on the allowed changes, a reference dictionary user needs to be able to access both the reference dictionary available at time t and the reference dictionary available at t+1, and thus all the various versions of the reference dictionary.

— Another solution is retained by the dictionary change management rules defined in this part of ISO 13584, which allows to use only one reference dictionary version, namely the more recent version of all its ontology concepts

This solution, called the *principle of ontological continuity*, restricts the allowed changes to those which ensure that any product characterization defined at time = t with the reference dictionary existing at this time must still keep the same meaning when interpreted with the reference dictionary existing at time = t+1. Consequently the meaning of a ontology concept introduced at some time will be retained in the future.

NOTE 3 Over its lifetime, a reference dictionary description may contain small errors, like typos. It may also need to be refined, for instance to take into account technology improvements. Finally, it also arrives, in some cases, that reference dictionary definitions contain conceptual errors where the meaning of classes and/or properties need to be changed.

The dictionary change management rules documented in this clause classifies the various changes that may be needed during the lifetime of reference dictionaries. It also specifies how each change should be represented to ensure that the same meaning is always associated with the same existing characterization.

10.2 Revisions and Versions

The impact of a change depends upon its impact on existing and future characterizations. We first define what means the fact that a characterization conforms to a reference dictionary.

Let:

— O_t be the reference dictionary O at time t;

— C_t be the classes of the reference dictionary O at time t;

— P_t be the properties of reference dictionary O at time t;

— applicable_properties$_t$ be the function that associates to each class of C_t its applicable properties in P_t at time t

NOTE 1 applicable_properties$_t$(Class_P_t) represents all properties declared by the **described_by** XML element of the **CLASS_Type** XML complex type that defines Class_P_t, more, if Class_P_t is a case-of class, the properties imported by the **imported_properties** of Class_P_t, more all the applicable properties of the possible superclasses of Class_P_t.

— $domain_t$ be the function that associates to each property Pi in P_t its domain of values at time t

NOTE 2 $domain_t(Pi_t)$ represents the domain of values declared by the **domain** XML element of the **PROPERTY_DET_Type** XML compex type that defines Pi at time t.

A characterization x_t conforms to the reference dictionary O_t if and only if x_t may be represented as an instance of O_t. This means that:

— x_t belongs to one class of C_t, say $Class\_P_t$;

— x_t is characterized by values of some properties, say $P1_t$, $P2_t$,..., Pn_t;

— $P1_t$, $P2_t$,..., Pn_t belongs to P_t;

— $P1_t$, $P2_t$,..., Pn_t are properties that are applicable to $Class\_P_t$;

NOTE 3 $P1_t$, $P2_t$,..., Pn_t may be the set of all properties applicable to $Class\_P_t$, it may also be any subset of this set.

— for each property: $P1_t$, $P2_t$,..., Pn_t, the value assigned to this property belongs to the domain of values of that property at time t, as defined by the function: $domain_t(Pi_t)$, i = 1..n..

Formally, x_t conforms to the reference dictionary O_t if a dictionary user may encode it:

$$x_t = Class\_P_t\ (P1_t = v1,\ P2_t = v2,...,\ Pn_t = vn)$$

With:

$$Class\_P_t \in O_t,\ P1_t \in O_t\ ,\ P2_t \in O_t,\ ...,\ Pn_t \in O_t$$

$$\wedge\ P1_t \in applicable\_properties_t(Class\_P_t) \wedge P2_t \in applicable\_properties_t(Class\_P_t)$$

$$\wedge\ \ldots \wedge Pn_t \in applicable\_properties_t(Class\_P_t)$$

$$\wedge\ v1 \in domain_t(P1_t) \wedge v2 \in domain_t(P2_t) \wedge \ldots \wedge vn \in domain_t(Pn_t)$$

The set of all the characterizations x_t that conform to O_t, called the *population* Pop_t of O_t, is defined as follows:

$$Pop_t = \text{all } x_t \text{ such that } x_t \text{ conforms to } O_t$$

We said that:

— Pop_t and x_t *conform* to O_t, and

— O_t *interprets* Pop_t and x_t.

These definitions allow to classify the various changes of reference dictionaries.

The first kind of changes in a reference dictionary are those changes that do not modify at all the set of characterizations that may be defined by this reference dictionary, i.e., the population of the reference dictionary. It is the case, for instance, when a typo is corrected, when new translations are added or when the definition of a class is redrafted to make its content clearer without changing its meaning. In this case, Pop_t, the population of O_t at time = t, is identical to Pop_{t+1}, the population of O_{t+1} at time = t+1. This means that:

— any characterization x_t defined from O_t also *conforms* to O_{t+1}. Thus, O_{t+1} is *backward compatible* with O_t since it allows to interpret all its instances, moreover

— any characterization x_{t+1} defined from O_{t+1} also *conforms* to O_t. Thus, O_{t+1} is also *upward compatible* with O_t since O_t allows to interpret all O_{t+1} instances.

In case of a change for which backward and upward compatibility are existing, it is not necessary to record if the characterization x was built at time = t or at time = t+1. Thus this change would not require to change the version numbers that were already assigned to the various ontology concepts at time = t. This change is called a *revision change*, and it will be traced by increasing either the **revision** XML element of the corresponding ontology concept that was modified if the change affects the description in the source language in which the ontology concept was defined, and/or one or several **translation_revision** XML elments corresponding to the other languages in which the ontology concept is translated if the change affects the description in the corresponding language.

NOTE 4 Revision numbers are not recorded in the identifiers of ontology concepts. The characterization using the ontology concepts will allow to use O_t as well as O_{t+1} for interpretation.

NOTE 5 Each ontology concept has a **revision** XML element. Each ontology concept that is translated has some administrative data that specify both the **source_language** language in which the ontology concept was initially defined, and, for each translation, a **translation_revision** information

The second kind of changes in a reference dictionary are those changes that refine the reference dictionary and allow to define new characterizations. New classes are introduced, new properties are introduced, and new property values are added to their domain of values. To ensure the ontological continuity principle, no class, property or value should be removed. The reference dictionary O_{t+1} defined after the change shall remain able to interpret Pop_t. O_{t+1} is still *backward compatible* with O_t and it allows to interpret all its instances. But it is no longer *upward compatible* because some characterizations that conform to O_{t+1} do not conform to O_t.

In case of changes for which only backward compatibility is existing, a characterization x that was built at time = t+1 and depends upon the modified ontology concepts should express this clearly in its representation. This change is called a *version change* and it will be traced by increasing the **version** of the ontology concept that was modified, and of all the other ontology concepts that were also modified as a consequence.

NOTE 6 Version numbers are recorded in the identifiers (IRDIs) of ontology concepts, the version of each element used in a characterization will prevent to use O_t for trying to interpret a characterization based on O_{t+1} specific versions.

Table 3 summarizes the differences between version and revision.

Table 3 — Revision and version

	Backward compatibility Pop_t *conforms to* O_{t+1}	**Upward compatibility** Pop_{t+1} *conforms to* O_t
Revision	Yes	Yes
Version	Yes	No

10.3 Correction of errors

In case of errors in a reference dictionary which is already used to define product characterizations, the need is to correct the reference dictionary errors but also to provide a mechanism that allows reference dictionary users to understand and process the error correction. For each data set that contains product characterization, processing errors means (1) recognizing which characterizations are in errors and (2) defining how erroneous characterizations should be corrected to be in line with the corrected reference dictionary.

When the ontology concepts which are erroneous have not yet been used for creating product characterizations or if in a closed user environment, the characterizations can be corrected concurrently with the reference dictionary. It is the responsibility of the reference dictionary supplier to decide how to remove erroneous elements from the current reference dictionary, and how to perform the reference dictionary correction.

In the dictionary change management rules defined in this part of ISO 13584, an open user environment is assumed, where all possible characterizations are not accessible to the dictionary supplier, and correction cannot be performed together with the reference dictionary. In such an environment, a mechanism called "deprecation" has to be used.

Deprecation means that:

— to ensure backward compatibility, the erroneous ontology concepts and/or the erroneous property values remain in the reference dictionary to ensure backward compatibility, but

— all the erroneous ontology concepts are associated with a **is_deprecated** XML element with a true value, the meaning of which being: "this ontology concept or value shall no longer be used for new characterizations", and

— an XML element associated with each **is_deprecated** XML element, called **is_deprecated_interpretation**, is used to specify how a characterization that references deprecated ontology concepts should be changed to be in line with the up-dated reference dictionary.

NOTE 1 The specification in **is_deprecated_interpretation** may be either informal, to explain to a reference dictionary user how the corresponding data should be processed, or formal to direct a computer how to correct automatically the data.

NOTE 2 In the current specification of the dictionary change management rules, no formal language is defined for representing the content of **is_deprecated_interpretation**. It is the intent of the team that developed the CIIM to consider the development of such a language.

EXAMPLE 1 If in a class C1 an applicable property P1 whose value was supposed to be expressed in meters, is replaced by a property P2 which has the same meaning but whose value shall be expressed in microns, (1) the P1 **is_deprecated** XML element is set to true, and (2) its **is_deprecated_interpretation** XML element could be set to: "the value of this property shall now be expressed in microns and recorded in property P2".

EXAMPLE 2 In example 1 above, the value of the **is_deprecated_interpretation** XML element of P1 could be represented, if this approach has been agreed by the community that uses the reference dictionary, as an expression using a given syntax, and representing the values of properties by the property identifiers. In this case the content could be set to: "P2 := P1 * 1 000".

10.4 Rules for change management

This subclause provides rules for managing changes in reference dictionaries.

10.4.1 Criteria for classifying a change

The impact of a change in an ontology concept onto the populations of characterizations that are interpretable by O_t and/or O_{t+1} provides a criteria for classifying the change impact as a revision change, a version change or an error correction. This clause describes the how each change should, at least, be recorded according to its impact to ensure that the receiver of an exchange file that contains item characterizations will understand whether its current dictionary allows to interpret the exchange file or not.

These rules define the minimal requirements. But a reference dictionary supplier may always decide to update the version of an ontology concept when the rules request only the updating of the revision, or to deprecate a modified element when the rules request only the updating of its revision or of its version.

Rule 1: Revision change

If after changing a concept (class, datatype, properties, …) Ent_t of the reference dictionary O_t into Ent_{t+1}, (1) the new reference dictionary O_{t+1} may interpret all the characterizations that might be defined by O_t, and (2) it does not allow to define any new characterization, the change is a revision change of the ontology concept changed by the change of the Ent_t entity.

If the description of the dictionary element changed is only defined in a single language, or if it is translated and the change affects the description in the source language in which it was defined, the change should increase the value of the **revision** attribute of the dictionary element modified by the change. If the change of Ent also affects the translation in any other languages in which the dictionary element is translated, the corresponding translations should be changed, and the change should increase the values of the **translation_revision** attribute of the corresponding translations.

EXAMPLE 1 In a dictionary that is available only in a single language, if one changes the definition of a class without changing the characterizations it can interpret, the **revision** XML element of the class must be increased.

EXAMPLE 2 In a dictionary whose source language is English and that is translated in German and French, if one changes the French definition of a class without changing the characterizations it can interpret, the **translation_revision** XML element of the French translation should be increased.

EXAMPLE 3 In a dictionary whose source language is English and that is translated in German and French, if one changes the value of the **figure** XML element of a class without changing the characterizations the class can interpret, then the **revision** XML element of the class should be increased. If the **figure** value is a **graphic_files** that is not language-dependent, and thus applies to all language descriptions, the German and French **translation_revision** should not be increased because the translated part was not changed.

EXAMPLE 4 If one adds a visible property to a class without making it applicable in the class or any of its subclass, then no characterization may be described by this new property. No direct attribute of the class being modified, neither the revision nor the version of the class needs to be updated.

Rule 2: Version change

If after changing a concept (class, datatype, properties, …) Ent_t of the reference dictionary O_t into Ent_{t+1}, (1) the new reference dictionary O_{t+1} may interpret all the characterizations that might be defined by O_t, but (2) it also provides new characterizations that cannot be interpreted by O_t, the change should increase the version of Ent_t.

NOTE 1 Constraints have an impact on those item characterizations that fulfill the constraints. Thus, change of constraints should be represented by an increase of version of the class that contains these constraints. But change of constraint does not change the set of characterization that may be interpreted by a dictionary. Thus, when constraints are modified in a class, the set of item characterizations that fulfill the constraints may become broader or narrower without violating the ontological continuity principle.

EXAMPLE 5 If one adds an applicable property to a class, thus (1) all characterizations defined by the previous reference dictionary may still be interpreted (without using the new applicable property), but (2) some characterizations may also be defined by the new reference dictionary that could not be interpreted by the previous one (those that use the new applicable property). Thus the version of the class must be increased.

EXAMPLE 6 If one adds a new alternative unit to the real measure type of some property, thus all characterizations defined by the previous reference dictionary may still be interpreted (without using the new alternative unit), but some characterizations may also be defined by the new reference dictionary that could not be interpreted by the previous one (by using the new alternative unit). Thus the version of the class must be increased.

Rule 3: Error correction

If after changing an ontology concept or an information element (class, datatype, property, name, definition …), Ent_t of the reference dictionary into Ent_{t+1}, the new reference dictionary O_{t+1} is not able to interpret all the characterizations that might be defined by O_t, the change would not be backward compatible and is violating the principle of ontological continuity.

For allowing error correction, it is needed to (1) identify those ontology concepts that should be modified and assign the value true to their **is_deprecated** XML element, (2) define new entities which would correct the errors, and (3) describe in the **is_deprecated_interpretation** XML element of the deprecated elements, why the element was deprecated and how a characterization that references deprecated elements should be changed to be in line with the up-dated reference dictionary.

EXAMPLE 7 If one corrects the unit of the real measure type of some property, then all products characterized by this property that are already recorded somewhere should be described differently. The new reference dictionary would not be backward compatible and this change could not be done that way. Thus (1) the corresponding property should be deprecated, (2) a new property (with a different identifier) should be created and made visible and applicable where the previous one was so, and (3) in the **is_deprecated_interpretation** of the previous property it should be specified that its value should be e.g., "divided by 1000, and then put as value to the new property".

EXAMPLE 8 If one adds some new context parameters to those from which a context dependent property depends on, thus all characterizations that involve this property should be described differently after the change. The new reference dictionary would no longer be able to interpret some previous characterizations. Thus (1) deprecate the old context dependent property, (2) create and introduce a new one, and (3) explain the deprecation rational, and possibly, if the previous context dependent property was supposed to be measured at a fixed value of the new context parameter, how values of the deprecated context dependent property could be converted into value of the new property.

NOTE 2 In this example, it would also be possible to keep in the reference dictionary both the previous and the new context dependent property.

Figure 107 summarizes how to classify a change:

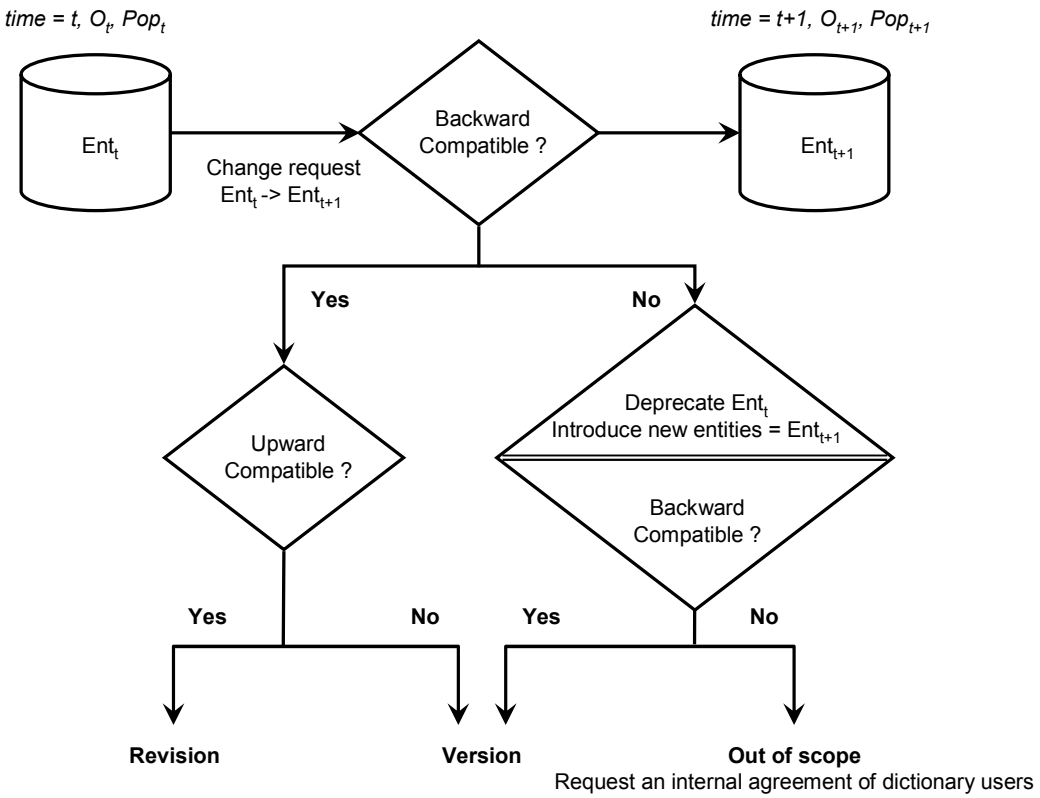

Figure 107 — Classifying a dictionary change

10.4.2 Dependency and the propagation of changes

In a reference dictionary, each ontology concept may exist only in a single version. Thus, when the version number of a ontology concept of a reference dictionary is increased, all the ontology concepts that references this ontology concept should be changed to reference its new version. Indeed, such a change should be traced at the level of the identifiers of all the referencing ontology concepts in order to be sure that, when a ontology concept identifier is replaced by its ontology concept description, it contains the correct internal references. Thus every change in the version of an ontology concept referenced within another ontology concept must be represented by a new version of the latter.

The oneness of each ontology concept applies within a reference dictionary, but not between several dictionaries. When class C1 of reference dictionary D1 references class C2 of reference dictionary D2 by the case-of relationship, it is the responsibility of the D1 reference dictionary supplier to decide which classes are referenced, in which version, and possibly which properties are imported. Thus, if the dictionary supplier of D2 changes the version of C2, it is the responsibility of the D1 dictionary supplier to decide if and when the new version of C2 will be referenced in C1. The old reference may be kept. But if the version of the reference is increased, then the version of C1 shall be increased.

This is summarized in the four rules below.

Rule 4: No propagation between reference dictionaries

When a class C1 of the reference dictionary D1 references a class C2 of the reference dictionary D2 by the case-of relationship, and when the version of class C2 is upgraded, it is the responsibility of the D1 dictionary supplier to decide if and when C1 will reference the new version of C2. If this is done, the version of C1 shall be increased.

EXAMPLE 1 C1 imports, through case-of, properties P1 and P2. A new applicable property P3 is added to C2. The supplier of D1 is not interested in P3. C1 may continue referencing the previous version of C2.

Rule 5: Version propagation

Increasing the version number of any ontology concepts that is referenced by other ontology concepts of the same reference dictionary must be propagated to them.

NOTE 1 Same reference dictionary means identified by the same information supplier.

NOTE 2 When an ontology concept references another ontology concept, this reference is done through a IRDI that includes the version of the target ontology concept. Thus, if the version of the target ontology concept is changed, the content of the source ontology concept should also be changed to record the correct (new) reference. This change induces that new characterization could be described (indirectly) by the source ontology concept and that its version should be changed.

EXAMPLE 2 Changing the version of a **DATATYPE_Type**, for instance to extend its domain of values, change also the domain of values of any property that references it as its domain. Thus the version of these properties should also be updated. This would also change the set of characterizations that may be described by the classes were these properties become applicable by the **described_by** XML element.

EXAMPLE 3 Change of the version number of a class leads to a change of the version number of its sub-classes and of the subclasses of this class, and so on.

EXAMPLE 4 If the version of the definition class of a property is increased, the version of this property must also be increased.

NOTE 3 This rule ensures that when the version number of the characterization class used for characterizing an item in an exchange file is smaller than or equal to its version in the local dictionary of the receiving system, This system is able to interpret correctly this characterization, whatever be its complexity.

Rule 6: Computation of new version values

For each particular change, all the propagated changes shall be computed together and the version number of each entity shall be increased at most once. It is also allowed to gather a number of different changes to compute new versions of a set of ontology concepts.

Rule 7: Circulation of new version

It is the responsibility of the dictionary supplier that provides reference dictionaries to decide how and when updates should be distributed

EXAMPLE 5 A reference dictionary may be associated with a server compliant with ISO/TS 29002-20 that makes available each update as soon as it has been validated

EXAMPLE 6 A reference dictionary may be distributed by releases. Every year a new release integrates all the updates elaborated during the year. In this case, all modified ontology concepts may have only one version increased.

10.4.3 Management of categorization classes

Categorization classes having no impact on item characterization, the above rules cannot apply to them.

Rule 8: Versioning of categorization classes

Increasing versions of categorization classes are requested when one or several of their superclasses, referenced by the **categorization_class_superclasses** XML element, are changed. All other changes may be recorded by revision increasing.

Change of versions of categorization classes are not propagated to characterization classes that reference them. Such changes are only recorded as revision changes.

10.4.4 Management of dictionary version and revision

During a file exchange of item characterizations based on a dictionary, the following rule ensures that in a file exchange, just by checking the dictionary version, the file receiver may know whether his/her available version of the dictionary allows to interpret the file.

Rule 9: Versions and revisions of a dictionary

When an updated dictionary is distributed according to rule 6:

— if the version of any ontology concept defined in this dictionary has been incremented, and/or if a new ontology concept has been introduced, the version of the dictionary shall be incremented,

EXAMPLE A new ontology concept is introduced in the dictionary when a new subclass of an existing class is introduced, or when a new visible type is defined.

— if the revision of any ontology concept defined in this dictionary has been incremented, but the version of the dictionary is not changed, then the revision of the dictionary shall be incremented,

— if the version of the dictionary is incremented, its revision shall be reset to '0'.

10.5 Dictionary Changes and Attributes

10.5.1 System maintained attributes

Dictionary change management rules are restricted to attributes that are available for change triggered by user change requests. System maintained attributes are therefore out of scope for reference dictionary changes, since they are modified automatically as consequence of another change:

— Revision change: if the change affects the description in the source language in which the ontology concept was described, **revision** is incremented and **date_of_current_revision** is updated with the current time. If the change affects the description in one, or several, of other languages in which the description of the ontology concept is translated, **translation_revision** corresponding to this language is incremented and **date_of_current_translation_revision** corresponding to this language is updated with the current time.

— Version change: **version** is incremented and **date_of_current_version** is updated with the current time. **revision** is set to the value defined as the minimum of **revision** attribute and **date_of_current_revision** is updated with the modification time.

— Creation of a new ontology concept: a new ontology concept is created with a new **code** and **date_of_original_definition** is set to the current time. **Version** is set to the value defined as the minimum of **version** attribute and **date_of_current_version** is updated with the current time. **Revision** is set to the value defined as the minimum of revision number and **date_of_current_revision** is updated with the current time.

10.5.2 Attributes available for textual change

The role of the terminological attributes of ontology concepts, such that names, definition, note, icon, is to explain which kind of products are characterized by a particular reference dictionary class, and which kind of product characteristic is represented by each particular property.

To ensure backward compatibility for textual changes of terminological attributes, such changes should not reduce the meaning of the class or of the property, even if it may precise its meaning. But a textual change may enlarge the definition of the class, for instance to take into account the development of new products.

Thus, a textual change requires, at least, a new revision number. But it is the responsibility of the dictionary supplier to decide whether this change should also change the version of the ontology concept to ensure that dictionary users will access to the terminological attributes that were used when some characterization was defined. It is also the responsibility of the dictionary supplier to decide whether a new ontology concept should be associated with a new code because the new definition seems to be not backward compatible with the previous one.

10.6 Constraints on the evolution of reference dictionaries

In this clause, we summarize the constraints for each kind of ontology concept (classes, relation between classes, properties and characterizations) during reference dictionary evolution.

Permanence of the classes

Existence of a characterization class could not be denied across evolution. Because each characterization class allows to define some characterizations, any class existing at time t, shall still exist at time t', with t' > t.

NOTE 1 To make the change management model more flexible, a class may become obsolete. It will then be marked as ”deprecated”, and possibly replaced by another class. But it will continue belonging to the newer versions of the reference dictionary.

This principle allows that the most recent reference dictionary will be able to interpret all the earlier-defined characterizations. It is the responsibility of each reference dictionary user to decide if, and until when, deprecated elements will be kept in each user dictionary.

The problem of permanence is different for categorization classes. Since categorization classes are not used for defining characterizations, these classes may be suppressed or modified without creating a backward compatibility problem.

Permanence of properties

Similarly, all properties existing at time t shall still exist at t', t' > t. A property may also become obsolete but neither its existence, nor its value for a particular item may be modified. The value domain of a property may evolve. Taking into account the backward compatibility requirement, a value domain can only increase, certain values being eventually marked as deprecated.

Permanence of the class-subclass relationship

The class-subclass relationship is the relation between a class, and all its subclasses, direct or obtained by transitivity. The class-subclass relationship supports inheritance between the superclass and the subclasses. Requirement for permanence of a particular class-subclass relationship between two classes C1, as superclass, and C2, as subclass, depends upon the consequences of this relation for the characterizations defined by the subclass:

— if C2 does not inherit from C1 any element (property, type, value,...) that may be used in a characterization, then the C1-C2 relationship may be suppressed. Consequences in version and revision number are defined by the dictionary change management rules.

— if C2 inherits from C1 some elements (property, type, value,...) that may be used in a characterization of a C2 instance, then the C1-C2 relationship shall not be suppressed.

Note that this constraint allows large evolution of the class-subclass relationship hierarchy, for example by intercalating intermediate classes between two classes linked by a class-subclass relation.

Permanence of Characterizations

The fact that a property P is applicable to a class C at time t requests that P remains applicable to C at t', t' > t.

NOTE 2 This does not require at all that the same applicable properties are always used to describe the instances of the same class. Properties used to characterize an item do not depend on reference dictionary evolutions. It depends mainly of the requirements of the application that uses the reference dictionary.

NOTE 3 If a property P1 is declared as applicable in class C2 which is a subclass of C1, P1 may become applicable in class C1 without any backward compatibility problem because applicability is inherited.

Annex A
(normative)

Information object registration

In order to provide for unambiguous identification of an information object in an open system, the object identifier

{ iso standard 13584 part (32) version (1) }

is assigned to this part of ISO 13584. The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

Annex B
(normative)

Computer interpretable listings

This annex contains the complete OntoML XML Schema in accordance with the various UML diagram defined in this part of ISO 13584. These listings are available in computer-interpretable form in Table B.1

The main XML schema is the "ontoml" schema.

The following notice applies to the computer-interpretable files in this annex.

Table B.1 — XML schema defined in this part of ISO 13584

Description	ASCII file	URI	Source document
OntoML XML Schema	ontoml.xsd	urn:iso:std:iso:13584:-32:ed-1:tech:xml-schema:ontoml	ISO 13584-32

The schemas in Table B.1 reference, directly or indirectly, the externally-defined schemas listed in Table B.2.

Table B.2 — XML schemas defined outside of this part of ISO 13584

Description	ASCII file	URI	Source document
Identifier XML Schema	identifier.xsd	urn:iso:std:iso:ts:29002:-5:ed-1:tech:xml-schema:identifier	ISO/TS 29002-5
Catalogue XML Schema	catalogue.xsd	urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:catalogue	ISO/TS 29002-10
Value XML Schema	value.xsd	urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:value	ISO/TS 29002-10
Basic XML Schema	basic.xsd	urn:iso:std:iso:ts:29002:-4:ed-1:tech:xml-schema:basic	ISO/TS 29002-4
MathML XML Schema	mathml2.xsd	http://www.w3.org/1998/Math/MathML	www.w3.org/TR/MathML2

Annex C
(normative)

Standard data requirements for OntoML

Standard data are the XML elements that shall be recognized by any implementation that claims conformance to some conformance class of OntoML.

Standard data shall be specified for this library integrated information model for each conformance class.

C.1 Conformance class specification table

Table C.1 specifies the values of the **name** and **application** XML elements defined in the **LIBRARY_IIM_IDENTIFICATION_Type** XML complex type that are allowed for use in a **LIBRARY_IIM_IDENTIFICATION_Type** to reference OntoML in either of its conformance classes.

Table C.1 — ISO 13584 LIIM 32 conformance class specification

Conformance Class	LIBRARY_IIM_IDENTIFICATION_Type name XML element mandatory value	LIBRARY_IIM_IDENTIFICATION_Type application XML element mandatory value
1	'ONTOML'	'1'
2	'ONTOML'	'2'
3	'ONTOML'	'3'
4	'ONTOML'	'4'

C.2 Standard data for conformance classes 1 to 4

The **LIBRARY_IIM_IDENTIFICATION_Type** XML complex type values allowed for use in a library delivery file conform to OntoML defined in this part of ISO 13584 shall obey the constraints defined in this clause.

An informal constraint is defined on the **LIBRARY_IIM_IDENTIFICATION_Type** XML complex type to be allowed for use to reference conformance class 1 to 4 of OntoML defined in this part of ISO 13584.

A **LIBRARY_IIM_IDENTIFICATION_Type** XML complex type is allowed for use to reference conformance class 1 to 4 of OntoML if the following conditions hold:

— the **name** XML element of the **LIBRARY_IIM_IDENTIFICATION_Type** XML complex type that references library integrated model LIIM 32 shall be equal to 'ONTOML', and

— the **status** XML element of the **LIBRARY_IIM_IDENTIFICATION_Type** XML complex type shall be equal to 'WD', 'CD' or 'DIS', 'FDIS', 'IS', 'TS', 'PAS' or 'ITA'-and

— the **application** XML element of the **LIBRARY_IIM_IDENTIFICATION_Type** XML complex type shall be equal to ‘1', '2', '3', or '4'.

Annex D
(normative)

Value representation of ISO 13584 / IEC 61360 entities and data types on ISO/TS 29002-10 shared XML schemas

To ensure interoperability with other standards representing product or object characterization by means of class belonging, and set of property-value pairs, OntoML uses the interoperability resources defined by the ISO Technical Specification series 29002, Exchange of Characteristic Data.

More precisely OntoML uses:

— the XML schema “urn:iso:std:iso:ts:29002:-5:ed-1:tech:xml-schema:identifier", defined in ISO/TS 29002-5, for representing global identifiers;

— the XML schema “urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:value", defined in ISO/TS 29002-10, for representing property values;

— the XML schema “urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:catalogue”, defined in ISO/TS 29002-10, for representing high level elements.

In this annex we specify how each OntoML data type or entity instance is represented using an XML element from ISO/TS 29002-10. Thus, this annex represents all the mandatory resources provided by ISO/TS 29002-10 for their particular use in the context of OntoML.

As a rule, the same ISO/TS 29002-10 elements may be used for representing different kinds of OntoML instances, and for each kind of instance, only some of the subelements or attributes of each ISO/TS 29002-10 elements are allowed for use. Thus, the following rules are used:

— RULE 1: for each OntoML data type or entity instance we present its representation using the relevant ISO/TS 29002-10 element(s);

EXAMPLE 1 The OntoML representation of an integer currency value is based on the following ISO/TS 29002-10 XML element:

```
<val:currency_value …>
    <
</val:currency_value>
```

— RULE 2: in this representation, only the subelements and attributes that are allowed for use for this particular representation are represented;

EXAMPLE 2 The subelements and attributes that are allowed for use for the OntoML representation of an integer currency value are:

```
<val:currency_value currency_code="…">
    <val:integer_value>…</val:integer_value>
</val:currency_value>
```

— RULE 3: the value of each subelement and attribute allowed for use is associated with a content name, the content of this name is described in terms of OntoML information;

EXAMPLE 3 The value of each subement and attribute that are allowed for use for the OntoML representation of an integer currency value are:

```
<val:currency_value currency_code="CurrencyCodeValue">
    <val:integer_value>IntegerValue</val:integer_value>
</val:currency_value>
```

IntegerValue is a mandatory integer value defined according to Clause 3.3.17 of XML Schema Part 2: Datatypes. ***CurencyCodeValue*** is defined according ISO 4217 (see 7.3.6)

— RULE 4: the fact that this value is optional or mandatory for the particular OntoML data type or entity instance considered is precised by a note;

EXAMPLE 4 The **currency_code** XML attribute assignment is mandatory for the OntoML representation of an integer currency value.

— RULE 5: when the same content name is used for several attributes or subelements, we qualify the content name by the attribute or subelement name.

This annex specifies the various value representations, according to ISO/TS 29002, of the corresponding OntoML data types.

In the next subclauses, according to ISO/TS 29002, the following namespace prefix are used:

— **id** stands for “urn:iso:std:iso:ts:29002:-5:ed-1:tech:xml-schema:identifier";

— **val** stands for “urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:value”;

— **cat** stands for “urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:catalogue”.

D.1 Value representation of ISO 13584 / IEC 61360 data types

This clause specifies the value representations corresponding to the various ISO 13584 / IEC 61360 data types that are defined in the CIIM. These values are represented according to the **val:schema**.

For each OntoML datatype, the representation of OntoML value using ISO/TS 29002-10 resource is specified.

D.1.1 Boolean type

The OntoML representation of the value of a property whose value domain is a Boolean type is represented as follows:

<val:boolean_value>***BooleanValue***</val:boolean_value>

BooleanValue is a Boolean value defined according to Clause 3.2.2 of XML Schema Part 2*:* Datatypes.

NOTE 1 Boolean type is defined in Clause 8.3.1.

NOTE 2 **val:boolean_value** is defined in ISO/TS 29002-10:2009, Clause 6.3.3.

NOTE 3 The **val:boolean_value** XML element content assignment is mandatory.

EXAMPLE The following are valid Boolean type values:

```
<val:boolean_value>true</val:boolean_value>
<val:boolean_value>1</val:boolean_value>
```

D.1.2 String type

The OntoML representation of the value of a property whose value domain is a string type is represented as follows:

<val:string_value>***StringValue***</val:string_value>

StringValue is a string value defined according to Clause 3.2.1 of XML Schema Part 2*:* Datatypes.

NOTE 1 String type is defined in Clause 8.3.2.

NOTE 2 **val:string_value** is defined in ISO/TS 29002-10:2009, Clause 6.3.2.

NOTE 3 The **val:string_value** XML element content assignment is mandatory.

EXAMPLE The following are valid string type values:

```
<val:string_value>it is a string value</val:string_value>
<val:string_value>&#x0064;</val:string_value>
```

D.1.3 Enumeration of string codes type

The OntoML representation of the value of a property whose value domain is an enumeration of string codes type is represented as follows:

```
<val:controlled_value value_code="StringValue"/>
```

StringValue is a mandatory string value defined according to Clause 3.2.1 of XML Schema Part 2: Datatypes.

NOTE 1 Enumeration of string codes type is defined in Clause 8.3.4.

NOTE 2 **val:controlled_value** is defined in ISO/TS 29002-10:2009, Clause 6.8.2.

NOTE 3 The **value_code** XML attribute assignment is mandatory.

EXAMPLE The following is a valid enumeration of string codes type value:

```
<val:controlled_value value_code="AAA012"/>
```

D.1.4 Translatable string type

The OntoML representation of the value of a property whose value domain is a translatable string type is represented as follows:

```
<val:localized_text_value>
    <val:content>
        <val:local_string>
            <val:content>StringValue</val:content>
            <val:language_code>LanguageCode</val:language_code>
            <val:country_code>PossibleCountryCode</val:country_code>
        </val:local_string>
        OtherPossibleLocalStrings
    </val:content>
</val:localized_text_value>
```

StringValue is string value defined according to Clause 3.2.1 of XML Schema Part 2: Datatypes.

LanguageCode and ***PossibleCountryCode*** are strings defined according to Clause 8.1.1.

OtherPossibleLocalStrings defines the localtion where other string localizations (**val:local_string**) may be represented.

NOTE 1 Translatable string type is defined in Clause 8.3.2.

NOTE 2 **val:localized_text_value** is defined in ISO/TS 29002-10:2009, Clause 6.4.4.

NOTE 3 **val:language_code** is defined according to ISO 3166-1 (see 8.1.1 of this part of ISO 13584).

NOTE 4 **val:country_code** is defined according to ISO 639-1 or ISO 639-2 (see 8.1.1 of this part of ISO 13584).

NOTE 5 For each **val:local_string**, a **val:content** and a **val:language_code** shall be specified (mandatory).

NOTE 6 If required, **val:country_code** may be specified.

NOTE 7 Only one translation must be provided for each couple (**val:language_code**, **val:country_code**).

EXAMPLE The following is a valid translatable string type value:

```
<val:localized_text_value>
    <val:content>
        <val:local_string>
            <val:content>screw</val:content>
            <val:language_code>en</val:language_code>
            <val:country_code>us</val:country_code>
        </val:local_string>
        <val:local_string>
            <val:content>vis</val:content>
            <val:language_code>fr</val:language_code>
        </val:local_string>
    </val:content>
</val:localized_text_value>
```

D.1.5 URI type

The OntoML representation of the value of a property whose value domain is a URI type is represented as follows:

```
<val:file_value>
    <val:content>
        <val:element>
            <val:URI>URIValue</val:URI>
        </val:element>
    </val:content>
</val:file_value>
```

URIValue is a URI value defined according to Clause 3.2.17 of XML Schema Part 2*:* Datatypes.

NOTE 1 URI type is defined in Clause 8.3.2.

NOTE 2 **val:file_value** is defined in ISO/TS 29002-10:2009, Clause 6.9.3.

NOTE 3 No translation shall be provided.

NOTE 4 The **val:URI** XML element content assignment is mandatory.

EXAMPLE The following is a valid remote http address type value:

```
<val:file_value>
  <val:content>
    <val:element>
      <val:URI>http://www.tc184-sc4.org/2010/OntoML</val:URI>
    </val:element>
  </val:content>
</val:file_value>
```

D.1.6 Non translatable string type

The OntoML representation of the value of a property whose value domain is a non translatable string type is represented as follows:

```
<val:string_value>StringValue</val:string_value>
```

StringValue is a string value defined according to Clause 3.2.1 of XML Schema Part 2*:* Datatypes.

NOTE 1 Non translatable string type is defined in Clause 8.3.2.

NOTE 2 The **val:string_value** XML element content assignment is mandatory.

EXAMPLE The following is a valid non translatable string type value:

<val:string_value>OntoML</val:string_value>

D.1.7 Date time data type

The OntoML representation of the value of a property whose value domain is a date time data type is represented as follows:

<val:date_time_value>***DateTimeValue***</val:date_time_value>

DateTimeValue is a value defined according to Clause 3.2.7 of XML Schema Part 2*:* Datatypes.

NOTE 1 Date time data type is defined in Clause 8.3.3.

NOTE 2 **val:date_time_value** is defined in ISO/TS 29002-10:2009, Clause 6.3.7.

NOTE 3 The **val:boolean_value** XML element content assignment is mandatory.

NOTE 4 The **val:date_time_value** XML element content assignment is mandatory.

EXAMPLE The following is a valid date time data type value:

<val:date_time_value> 2010-03-24T20:45:00+01:00</val:date_time_value>

D.1.8 Date type

The OntoML representation of the value of a property whose value domain is a date data type is represented as follows:

<val:date_value>***DateValue***</val:date_value>

DateValue is a date value defined according to Clause 3.2.9 of XML Schema Part 2*:* Datatypes.

NOTE 1 Date data type is defined in Clause 8.3.3.

NOTE 2 **val:date_value** is defined in ISO/TS 29002-10:2009, Clause 6.3.6.

NOTE 3 The **val:date_value** XML element content assignment is mandatory.

EXAMPLE The following is a valid date data type value:

<val:date_value>2008-09-30</val:date_value>

D.1.9 Time data type

The OntoML representation of the value of a property whose value domain is a time data type is represented as follows:

<val:time_value>***TimeValue***</val:time_value>

TimeValue is a time value is defined according to Clause 3.2.9 of XML Schema Part 2*:* Datatypes.

NOTE 1 Time data type is defined in Clause 8.3.3.

NOTE 2 **val:time_value** is defined in ISO/TS 29002-10:2009, Clause 6.3.8.

NOTE 3 The **val:time_value** XML element content assignment is mandatory.

EXAMPLE The following is a valid time data type value:

```
<val:time_value>20:45:00+01:00</val:time_value>
```

D.1.10 Number type

The OntoML representation of the value of a property whose value domain is a number type is represented as follows:

<val:real_value>***RealValue***</val:real_value>

RealValue is a real value defined according to Clause 3.2.5 of XML Schema Part 2*:* Datatypes.

NOTE 1 Number data type is defined in Clause 8.3.5.

NOTE 2 The **val:real_value** XML element content assignment is mandatory.

EXAMPLE The following are valid number type values:

```
<val:real_value>3.21</val:real_value>
<val:real_value>12</val:real_value>
<val:real_value> 12.78e-2</val:real_value>
```

D.1.11 Real type

The OntoML representation of the value of a property whose value domain is a real type is represented as follows:

<val:real_value>***RealValue***</val:real_value>

RealValue is a real value defined according to Clause 3.2.5 of XML Schema Part 2*:* Datatypes.

NOTE 1 Real data type is defined in Clause 8.3.5.

NOTE 2 **val:real_value** is defined in ISO/TS 29002-10:2009, Clause 6.3.10.

NOTE 3 The **val:real_value** XML element content assignment is mandatory.

EXAMPLE The following are valid real type values:

```
<val:real_value>3.21</val:real_value>
<val:real_value>12</val:real_value>
<val:real_value> 12.78e-2</val:real_value>
```

D.1.12 Integer type

The OntoML representation of the value of a property whose value domain is an integer type is represented as follows:

<val:integer_value>***IntegerValue***</val:integer_value>

IntegerValue is an integer value defined according to Clause 3.3.17 of XML Schema Part 2*:* Datatypes.

NOTE 1 Integer data type is defined in Clause 8.3.5.

NOTE 2 **val:integer_value** is defined in ISO/TS 29002-10:2009, Clause 6.3.13.

NOTE 3 The **val:integer_value** XML element content assignment is mandatory.

EXAMPLE The following is a valid integer type value:

```
<val:integer_value>10</val:integer_value>
```

D.1.13 Rational type

The OntoML representation of the value of a property whose value domain is an integer type is represented as follows:

```
<val:rational_value>
    <val:whole_part>WholePartIntegerValue</val:whole_part>
    <val:numerator>NumeratorIntegerValue</val:numerator>
    <val:denominator>DenominatorIntegerValue</val:denominator>
</val:rational_value>
```

WholePartIntegerValue, ***NumeratorIntegerValue*** and ***DenominatorIntegerValue*** is an integer value defined according to Clause 3.3.17 of XML Schema Part 2*:* Datatypes.

NOTE 1 ***DenominatorIntegerValue*** must be a non null integer value.

NOTE 2 Rational data type is defined in Clause 8.3.5.

NOTE 3 **val:rational_value** is defined in ISO/TS 29002-10:2009, Clause 6.3.13.

NOTE 4 The **val:whole_part** XML element content assignment is optional.

NOTE 5 The **val:numerator** and **val:denominator** XML element content assignments are mandatory.

EXAMPLE The following is a valid rational type value:

```
<val:rational_value>
    <val:whole_part>5</val:whole_part>
    <val:numerator>2</val:numerator>
    <val:denominator>3</val:denominator>
</val:rational_value>
```

D.1.14 Real currency type

The OntoML representation of the value of a property whose value domain is a real currency type is represented as follows:

```
<val:currency_value currency_code="CurrencyCodeValue" currency_ref="CurrencyIRDI">
    <val:real_value>RealValue</val:real_value>
</val:currency_value>
```

RealValue is a real value defined according to Clause 3.2.5 of XML Schema Part 2*:* Datatypes.

CurencyCodeValue is defined according ISO 4217 (see 8.3.6 of this part of ISO 13584).

CurrencyIRDI a possible reference to a currency defined in a currency dictionary.

NOTE 1 Real currency type is defined in Clause 8.3.6.

NOTE 2 **val:currency_value** is defined in ISO/TS 29002-10:2009, Clause 6.6.3.

NOTE 3 The **currency_code** XML attribute assignment is optional. It must be provided only if it is not defined in the associated real currrency type.

NOTE 4 **currency_ref** is a valid unit identifier (IRDI) as defined in Clause 9.1.3.3.

NOTE 5 The **currency_ref** XML attribute assignment is optional.

NOTE 6 If both **currency_code** and **currency_ref** XML attributes are provided, they shall denote the same currency concept.

NOTE 7 The **val:real_value** XML element content assignment is mandatory.

EXAMPLE The following is a valid real currency type value:

```
<val:currency_value currency_code="EUR">
        <val:real_value>10.53</val:real_value>
</val:currency_value>
```

D.1.15 Integer currency type

The OntoML representation of the value of a property whose value domain is an integer currency type is represented as follows:

```
<val:currency_value currency_code="CurrencyCodeValue" currency_ref="CurrencyIRDI">
    <val:integer_value>IntegerValue</val:integer_value>
</val:currency_value>
```

IntegerValue is a mandatory integer value defined according to Clause 3.3.17 of XML Schema Part 2*:* Datatypes.

CurencyCodeValue is defined according ISO 4217 (see 8.3.6).

CurrencyIRDI a possible reference to a currency defined in a currency dictionary.

NOTE 1 Integer currency type is defined in Clause 8.3.6.

NOTE 2 **val:currency_value** is defined in ISO/TS 29002-10:2009, Clause 6.6.3.

NOTE 3 The **currency_code** XML attribute assignment is optional. It must be provided only if it is not defined in the associated integer currrency type.

NOTE 4 **currency_ref** is a valid unit identifier (IRDI) as defined in Clause 9.1.3.3.

NOTE 5 The **currency_ref** XML attribute assignment is optional.

NOTE 6 If both **currency_code** and **currency_ref** XML attributes are provided, they shall denote the same currency concept.

NOTE 7 The **val:integer_value** XML element content assignment is mandatory.

EXAMPLE The following is a valid integer currency type value:

```
<val:currency_value currency_code="EUR">
    <val:integer_value>10</val:integer_value>
</val:currency_value>
```

D.1.16 Real measure type

The OntoML representation of the value of a property whose value domain is a real measure type is represented as follows:

```
<val:measure_single_number_value UOM_code="UnitValue" UOM_ref="UnitIRDI">
    <val:real_value>RealValue</val:real_value>
</val:measure_single_number_value>
```

RealValue is a mandatory real value defined according to Clause 3.2.5 of XML Schema Part 2*:* Datatypes.

UnitValue is a possible unit belonging to the list of units specified in the corresponding real measure type specification (mathematical string representation of the main unit or of one of its alternative units, see 8.3.7).

NOTE 1 Use ISO 843 if necessary.

UnitIRDI a possible reference to a unit defined in a unit dictionary.

NOTE 2 **UOM_ref** is a valid unit identifier (IRDI) as defined in Clause 9.1.3.3.

NOTE 3 Real measure type is defined in Clause 8.3.7.

NOTE 4 **val:measure_single_number_value** is defined in ISO/TS 29002-10:2009, Clause 6.5.5.

NOTE 5 **UOM_code** or **UOM_ref** XML attribute assignment shall be provided. If both **UOM_ref** and **UOM_code** XML attributes are provided, they shall denote the same unit concept.

NOTE 6 The **val:real_value** XML element content assignment is mandatory.

EXAMPLE The following is a valid real measure type value:

```
<val:measure_single_number_value UOM_code="mm">
    <val:real_value>10.53</val:real_value>
</val:measure_single_number_value>
```

D.1.17 Integer measure type

The OntoML representation of the value of a property whose value domain is an integer measure type is represented as follows:

```
<val:measure_single_number_value UOM_code="UnitValue" UOM_ref="UnitIRDI">
    <val:integer_value>IntegerValue</val:integer_value>
</val:measure_single_number_value>
```

IntegerValue is a mandatory integer value defined according to Clause 3.3.17 of XML Schema Part 2*:* Datatypes.

UnitValue is a possible unit belonging to the list of units specified in the corresponding integer measure type specification (mathematical string representation of the main unit or of one of its alternative units, see 8.3.7).

UnitIRDI a possible reference to a unit defined in a unit dictionary.

NOTE 1 Use ISO 843 if necessary.

NOTE 2 **UOM_ref** is a valid unit identifier (IRDI) as defined in Clause 9.1.3.3.

NOTE 3 Integer measure type is defined in Clause 8.3.7.

NOTE 4 **val:measure_single_number_value** is defined in ISO/TS 29002-10:2009, Clause 6.5.5.

NOTE 5 **UOM_code** or **UOM_ref** XML attribute assignment shall be provided. If both **UOM_ref** and **UOM_code** XML attributes are provided, they shall denote the same unit concept.

NOTE 6 The **val:integer_value** XML element content assignment is mandatory.

EXAMPLE The following is a valid integer measure type value:

```
<val:measure_single_number_value UOM_code="mm">
    <val:integer_value>5</val:integer_value>
</val:measure_single_number_value>
```

D.1.18 Rational measure type

The OntoML representation of the value of a property whose value domain is an rational measure type is represented as follows:

```
<val:measure_single_number_value UOM_code="UnitValue" UOM_ref="UnitIRDI">
        <val:rational_value>
                <val:whole_part>WholePartIntegerValue</val:whole_part>
                <val:numerator>NumeratorIntegerValue</val:numerator>
                <val:denominator>DenominatorIntegerValue</val:denominator>
        </val:rational_value>
</val:measure_single_number_value>
```

WholePartIntegerValue, ***NumeratorIntegerValue*** and ***DenominatorIntegerValue*** is an integer value defined according to Clause 3.3.17 of XML Schema Part 2*:* Datatypes.

UnitValue is a possible unit belonging to the list of units specified in the corresponding integer measure type specification (mathematical string representation of the main unit or of one of its alternative units, see 8.3.7).

UnitIRDI a possible reference to a unit defined in a unit dictionary.

NOTE 1 Use ISO 843 if necessary.

NOTE 2 ***DenominatorIntegerValue*** must be a non null integer value.

NOTE 3 **UOM_ref** is a valid unit identifier (IRDI) as defined in Clause 9.1.3.3.

NOTE 4 Rational measure type is defined in Clause 8.3.7.

NOTE 5 **val:measure_single_number_value** is defined in ISO/TS 29002-10:2009, Clause 6.5.5.

NOTE 6 **UOM_code** or **UOM_ref** XML attribute assignment shall be provided. If both **UOM_ref** and **UOM_code** XML attributes are provided, they shall denote the same unit concept.

NOTE 7 The **val:whole_part** XML element content assignment is optional.

NOTE 8 The **val:numerator** and **val:denominator** XML element content assignments are mandatory.

EXAMPLE The following is a valid rational measure type value:

```
<val:measure_single_number_value UOM_code="mm">
        <val:rational_value>
                <val:whole_part>5</val:whole_part>
                <val:numerator>2</val:numerator>
                <val:denominator>3</val:denominator>
        </val:rational_value>
</val:measure_single_number_value>
```

D.1.19 Enumeration of integer codes

The OntoML representation of the value of a property whose value domain is an enumeration of integer codes type is represented as follows:

```
<val:controlled_value value_code="StringValue"/>
```

String value shall denote an integer value.

NOTE 1 Enumeration of integer codes type is defined in Clause 8.3.8.

NOTE 2 **val:controlled_value** is defined in ISO/TS 29002-10:2009, Clause 6.8.2.

NOTE 3 The **value_code** XML attribute assignment is mandatory.

EXAMPLE The following is a valid enumeration of integer codes type value:

```
<val:controlled_value value_code=”38”/>
```

D.1.20 Bag type

The OntoML representation of the value of a property whose value domain is a bag type is represented as follows:

```
<val:bag_value>
    ABagOfBaseTypeValue
</val:bag_value>
```

ABagOfBaseTypeValue is any kind of value defined in this normative annex.

NOTE 1 Any value defined in a bag shall have the same base type.

NOTE 2 Duplicated values are allowed.

NOTE 3 Bag type is defined in Clause 8.3.9.1.

NOTE 4 **val:bag_value** is defined in ISO/TS 29002-10:2009, Clause 6.7.5.

NOTE 5 **val:bag_value** content (***ABagOfBaseTypeValue***) is mandatory.

EXAMPLE The following is a valid bag type value, for which the constituent elements are integers:

```
<val:bag_value>
    <val:integer_value>10</val:integer_value>
    <val:integer_value>10</val:integer_value>
    <val:integer_value>30</val:integer_value>
</val:bag_value>
```

D.1.21 Set type

The OntoML representation of the value of a property whose value domain is a set type is represented as follows:

```
<val:set_value>
    ASetOfBaseTypeValue
</val:set_value>
```

ASetOfBaseTypeValue is any kind of value defined in this normative annex.

NOTE 1 Any value defined in a set shall have the same base type.

NOTE 2 Duplicated values are not allowed.

NOTE 3 Set type is defined in Clause 8.3.9.2.

NOTE 4 **val:set_value** is defined in ISO/TS 29002-10:2009, Clause 6.7.4.

NOTE 5 **val:set_value** content (***ASetOfBaseTypeValue***) is mandatory.

EXAMPLE The following is a valid set type value, for which the constituent elements are integers:

```
<val:set_value>
    <val:integer_value>10</val:integer_value>
    <val:integer_value>20</val:integer_value>
    <val:integer_value>30</val:integer_value>
</val:set_value>
```

D.1.22 List type

The OntoML representation of the value of a property whose value domain is a list type is represented as follows:

```
<val:sequence_value>
        AListOfBaseTypeValue
</val: sequence _value>
```

AListOfBaseTypeValue is any kind of value defined in this normative annex.

NOTE 1 Any value defined in a list shall have the same base type.

NOTE 2 Depending on the list type specification, duplicated values may or may not be allowed.

NOTE 3 List type is defined in Clause 8.3.9.3.

NOTE 4 **val:sequence_value** is defined in ISO/TS 29002-10:2009, Clause 6.7.6.

NOTE 5 **val:sequence _value** content (***AListOfBaseTypeValue***) is mandatory.

EXAMPLE The following is a valid list type value, for which the constituent elements are integers:

```
<val:sequence_value>
    <val:integer_value>10</val:integer_value>
    <val:integer_value>20</val:integer_value>
    <val:integer_value>30</val:integer_value>
</val: sequence _value>
```

D.1.23 Array type

The OntoML representation of the value of a property whose value domain is an array type is represented as follows:

```
<val:sequence_value>
        AnArrayOfBaseTypeValue
</val: sequence_value>
```

AnArrayOfBaseTypeValue is any kind of value defined in this normative annex.

NOTE 1 Any value defined in an array shall have the same base type, or may be a null value.

NOTE 2 Depending on the array type specification, duplicated values may or may not be allowed.

NOTE 3 Array type is defined in Clause 8.3.9.4.

NOTE 4 **val:sequence_value** is defined in ISO/TS 29002-10:2009, Clause 6.7.6.

NOTE 5 **val:sequence _value** content (***AnArrayOfBaseTypeValue***) is mandatory.

EXAMPLE The following is a valid array type value, for which the constituent elements are integers, and for which the second element is not defined (null value):

```
<val:sequence_value>
    <val:integer_value>10</val:integer_value>
    <val:null_value/>
    <val:integer_value>30</val:integer_value>
</val: sequence_value>
```

D.1.24 Set with subset constraint type

The OntoML representation of the value of a property whose value domain is a set with subset constraint type is represented as follows:

```
<val:set_value>
        ASetOfBaseTypeValue
</val:set_value>
```

ASetOfBaseTypeValue *value* is any kind of value defined in this normative annex.

NOTE 1 Any value defined in an array shall have the same base type.

NOTE 2 Set with subset constraint type is defined in Clause 8.3.9.5.

NOTE 3 **val:set_value** is defined in ISO/TS 29002-10:2009, Clause 6.7.4.

NOTE 4 Dynamic bounds of a set with subset constraint value used to overload the static bounds defined at the type level cannot be mapped.

NOTE 5 **val:set _value** content (***ASetOfBaseTypeValue***) is mandatory.

EXAMPLE The following is a valid set with subset constraint type value, for which the constituent elements are integers:

```
<val:set_value>
    <val:integer_value>10</val:integer_value>
    <val:integer_value>20</val:integer_value>
    <val:integer_value>30</val:integer_value>
</val:set_value>
```

D.1.25 Class reference type

The OntoML representation of the value of a property whose value domain is a class reference type, based on an explicit reference mechanism, is represented as follows:

```
<val:item_reference_value item_local_ref="LocalIdRef"/>

<cat:item class_ref="ClassIRDI" local_id="LocalId">
        <cat:property_value property_ref="PropertyIRDI">
        …
        </cat:property_value>
…
</cat:item>
```

LocalId stands for a local XML identifier. It is intended to be used as a reference target for a property value whose value domain is a class reference type. The reference is performed by setting the ***LocalIdRef*** value of the **item_local_ref** XML attribut. ***LocalId*** is defined according to Clause 3.3.8 of XML Schema Part 2*:* Datatypes.

NOTE 1 ***LocalId***/***LocalIdRef*** is similar to the the ID/IDREF XML reference mechanism.

LocalIdRef is defined according to Clause 3.3.6 of XML Schema Part 2*:* Datatypes.

ClassIRDI is a valid class concept identifier (IRDI) defined according to Clause 9.1.3.1.

PropertyIRDI is a valid property concept identifier (IRDI) defined according to Clause 9.1.3.2.

NOTE 2 **cat:property_value** represents a property-value pair constructor, see D.2.1.4.

NOTE 3 Class reference type is defined in Clause 8.3.10.

NOTE 4 The **item_local_ref** and **local_id** XML attribute assignments are mandatory.

NOTE 5 The **class_ref** XML element assignment is mandatory.

NOTE 6 **val:item_reference_value** is defined in ISO/TS 29002-10:2009, Clause 6.9.2.

NOTE 7 **cat:item** is defined in Clause D.2.2.1 of this normative annex.

EXAMPLE The following is a valid class reference type value: the referenced item is specified according to the class identified by the "0123-ABC#01-SCREW#1" IRDI, and is locally identified (**local_id** XML attribute) by the "ITEM_ID_1" local identifier. The reference (**item_local_ref** XML attribute) is performed through this local identifier.

```
<val:item_reference_value item_local_ref="ITEM_ID_1"/>
…
<cat:item class_ref="0123-ABC#01-SCREW#1" local_id="ITEM_ID_1">
    <cat:property_value property_ref="0123-ABC#02-DIAMETER#1">
    …
    </cat:property_value>
…
</cat:item>
```

D.1.26 Level type

The OntoML representation of the value of a property whose value domain is a level type is represented as follows:

```
<val:measure_qualified_number_value UOM_code="UnitValue" UOM_ref="UnitIRDI">
    <val:qualified_value qualifier_code="Qualifier">
        NumericValueOrNullValue
    </val:qualified_value>
    OtherPossibleQualifiedValues
</val:measure_qualified_number_value>
```

UnitValue is a possible unit belonging to the list of units specified in the corresponding real measure type specification (mathematical string representation of the main unit or of one of its alternative units, see 8.3.7).

UnitIRDI a possible reference to an IRDI of a unit defined in a unit dictionary.

Qualifier stands for an OntoML valid qualifier. It may take the following values: **min**, **nom**, **max** or **typ**.

NumericValueOrNullValue is either a real value (see D.1.11), an integer value (see D.1.12) or a null value (see 7.3.11).

OtherPossibleQualifiedValues stands for other possible qualified values (**val:qualified_value** XML construct)defined in the associated level type specification.

NOTE 1 The **qualifier_code** XML attribute assignment is mandatory.

NOTE 2 **UOM_code** or **UOM_ref** XML attribute assignment shall be provided. If both **UOM_ref** and **UOM_code** XML attributes are provided, they shall denote the same unit concept.

NOTE 3 **UOM_ref** is a valid unit identifier (IRDI) as defined in Clause 9.1.3.3.

NOTE 4 Any qualified value shall be typed using the same base type, or, when the value is not available, the qualified value shall be represented as a null value.

NOTE 5 Level type is defined in Clause 8.3.11.

NOTE 6 **val:measure_qualified_number_value** is defined in ISO/TS 29002-10:2009, Clause 6.5.3.

EXAMPLE The following is a valid qualified value: it is a physical measure, whose unit (**UOM_code** XML attribute) is millimeter (*mm*). this physical measure is associated with two qualifiers (**qualifier_code** XML attribute): a minimum integer value (*min*), and a typical value (typ) that is not available and then, that is represented by an ISO/TS 29002 null value (**null_value** XML element).

```
<val:measure_qualified_number_value UOM_code="mm">
    <val:qualified_value qualifier_code="min">
        <val:integer_value>10</val:integer_value>
    </val:qualified_value>
    <val:qualified_value qualifier_code="typ">
        <val:null_value/>
    </val:qualified_value>
</val:measure_qualified_number_value>
```

D.1.27 Named type

The OntoML representation of the value of a property whose value domain is a named type is defined according to the underlying base type of the named type.

D.2 Representation of items

This Clause specifies the value representation of items (i.e., instances) in OntoML. According to ISO/TS 29002, instances are represented using a reference to the class where their specification is defined, and by a set of property values, each of them being defined by a reference to a property and a typed value.

References to classes and properties are represented by a global identifier as respectively defined in Clauses 9.1.3.1 and 9.1.3.2. Typed values are specified in Clause D.1 of this normative annex.

This clause introduces firstly the ISO/TS 29002-10 representation of OntoML property values, depending on the kind of the property (non dependent property, condition property, dependent property and representation property). Then, the representation of OntoML instances (item class or functional model class instances) according to ISO/TS 29002-10 is defined.

D.2.1 Representation of property-value pairs

This subclause specifies the OntoML representation of:

— non dependent property-value pairs;

— condition property-value pairs;

— dependent property-value pairs;

— representation property-value pairs.

D.2.1.1 Non dependent property

The OntoML representation of the value of a property that is defined as a non dependent property is defined as follows:

```
<cat:property_value property_ref="PropertyIRDI" subitem_path_property_ref="PropertyIRDIs">
        ValueRepresentation
</cat:property_alue>
```

PropertyIRDI is a valid property concept identifier (IRDI) defined according to Clause 9.1.3.2.

NOTE 1 The **property_ref** XML attribute assignment is mandatory.

PropertyIRDIs is a list of property concept identifiers (IRDIs) defined according to Clause 9.1.3.2, that allows to describe the path from a source property (specified by the **property_ref** XML attribute) of which data type is a class reference type, to a target property that participates to the description of an (sub-)embedded instance of the instance for which the source property is defined.

NOTE 2 The **subitem_path_property_ref** XML attribute assignment is optional. It may only be used for an identified property (**property_ref**) for which the underlying data type is a class reference type (see 8.3.10).

NOTE 3 The **subitem_path_property_ref** XML attribute may be used as a simple way for representing embedded instance property-value pairs.

ValueRepresentation (mandatory) stands for any kind of typed value defined in this normative annex.

NOTE 4 Non dependent property is defined in Clause 6.7.4.

NOTE 5 **cat:property_value** is defined in ISO/TS 29002-10:2009, Clause 5.2.4.

NOTE 6 The **cat:property_value** XML element content assignment is mandatory.

EXAMPLE 1 The following is a valid representation of a non dependent property value: the property is identified by the “0123-ABC#02-DIAMETER#1” IRDI, and the associated value is an integer measure value (see D.1.17).

```
<cat:property_value property_ref="0123-ABC#02-DIAMETER#1">
        <val:measure_single_number_value UOM_code="mm">
                <val:integer_value>5</val:integer_value>
        </val:measure_single_number_value>
</cat:property_value>
```

EXAMPLE 2 Assume that a notebook is described by a non dependent property called *its mother board* whose identifier is the “0123-ABC#02-ITS_MOTHERBOARD#1” IRDI, and whose data type is the *mother board* class, itself identified by the “0123-ABC#01-MOTHERBOARD#1” IRDI. Assume that the *mother board* class is described by a non dependent property called *its processor unit* whose identifier is the “0123-ABC#02-ITS_PROCESSOR_UNIT#1” IRDI, and whose data type is the *processor unit* class, itself identified by the “0123-ABC#01-PROCESSORUNIT#1” IRDI. Assume that the *processor unit* class is described by a non dependent property called *its processor* whose identifier is the “0123-ABC#02-ITS_PROCESSOR#1” IRDI, and whose data type is the *processor* class, itself identified by the “0123-ABC#01-PROCESSOR#1” IRDI. Finally, assume that the *processor* class is described by a non dependent property called *frequency* whose identifier is the “0123-ABC#02-FREQUENCY#1” IRDI, and whose data type is a real measure type. The following is a valid representation of such a composition of non dependent properties:

```
<cat:property_value property_ref="0123-ABC#02-ITS_MOTHERBOARD#1" subitem_path_property_ref="0123-ABC#02-
ITS_PROCESSOR_UNIT#1 0123-ABC#02-ITS_PROCESSOR#1 0123-ABC#02-FREQUENCY#1">
        <val:measure_single_number_value UOM_code="GHz">
                <val:real_value>4.2</val:real_value>
        </val:measure_single_number_value>
</cat:property_value>
```

Alternatively, and according to the OntoML representation of property whose value domain is a class reference type (see D.1.25), this same example could have been represented as follows:

```
<cat:property_value property_ref="0123-ABC#02-ITS_MOTHERBOARD#1">
        <item_reference_value item_local_ref="MOTHERBOARD_ID"/>
</cat:property_value>

<cat:item class_ref="0123-ABC#01-MOTHERBOARD#1" local_id="MOTHERBOARD_ID">
        <cat:property_value property_ref="0123-ABC#02-ITS_PROCESSOR_UNIT#1">
                <item_reference_value item_local_ref="PROCESSOR_UNIT_ID"/>
        </cat:property_value>
</cat:item>

<cat:item class_ref="0123-ABC#01-PROCESSOR_UNIT#1" local_id="PROCESSOR_UNIT_ID">
        <cat:property_value property_ref="0123-ABC#02-ITS_PROCESSOR #">
                <item_reference_value item_local_ref="PROCESSOR_ID"/>
        </cat:property_value>
</cat:item>
```

```
<cat:item class_ref="10123-ABC#01-PROCESSOR#1" local_id="PROCESSOR_ID">
        <cat:property_value property_ref="0123-ABC#02-FREQUENCY#1">
                <val:measure_single_number_value UOM_code="GHz">
                        <val:real_value>4.2</val:real_value>
                </val:measure_single_number_value>
        </cat:property_value>
</cat:item>
```

D.2.1.2 Condition property

The OntoML representation of the value of a property that is defined as a condition property is defined as follows:

```
<cat:property_value property_ref="PropertyIRDI" subitem_path_property_ref="PropertyIRDIs">
        ValueRepresentation
</cat:property_alue>
```

PropertyIRDI is a valid property concept identifier (IRDI) defined according to Clause 9.1.3.2.

NOTE 1 The **property_ref** XML attribute assignment is mandatory.

PropertyIRDIs is a list of property concept identifiers (IRDIs) defined according to Clause 9.1.3.2, that allows to describe the path from a source property (specified by the **property_ref** XML attribute) of which data type is a class reference type, to a target property that participates to the description of an (sub-)embedded instance of the instance for which the source property is defined.

NOTE 2 The **subitem_path_property_ref** XML attribute assignment is optional. It may only be used for an identified property (**property_ref**) for which the underlying data type is a class reference type (see 8.3.10).

NOTE 3 The **subitem_path_property_ref** XML attribute may be used as a simple way for representing embedded instance property-value pairs (see example in Clause D.2.1.1).

ValueRepresentation (mandatory)stands for any kind of typed value defined in this normative annex.

NOTE 4 Condition property is defined in Clause 6.7.4.

NOTE 5 **cat:property_value** is defined in ISO/TS 29002-10:2009, Clause 5.2.4.

NOTE 6 The **cat:property_value** XML element content assignment is mandatory.

EXAMPLE The following is a valid representation of a condition property value: the property is identified by the “0123-ABC#02-LOAD#1” IRDI, and the associated value is a real measure value (see D.1.16).

```
<cat:property_value property_ref="0123-ABC#02-LOAD#1">
        <val:measure_single_number_value UOM_code="N">
                <val:real_value>1512.37</val:real_value>
        </val:measure_single_number_value>
</cat:property_value>
```

D.2.1.3 Dependent property

The OntoML representation of the value of a property that is defined as a dependent property is defined as follows:

```
<cat:property_value property_ref="PropertyIRDI" subitem_path_property_ref="PropertyIRDIs">
        DependentValueRepresentation
        <val:environment>
                <val:property_value property_ref= PropertyIRDI">
                        ConditionValueRepresentation
                </val:property_value>
                OtherPossibleConditions
        </val:environment>
</cat:property_value>
```

PropertyIRDI is a valid property concept identifier (IRDI) defined according to Clause 9.1.3.2.

NOTE 1 The **property_ref** XML attributes assignment is mandatory.

PropertyIRDIs is a list of property concept identifiers (IRDIs) defined according to Clause 9.1.3.2, that allows to describe the path from a source property (specified by the **property_ref** XML attribute) of which data type is a class reference type, to a target property that participates to the description of an (sub-)embedded instance of the instance for which the source property is defined.

NOTE 2 The **subitem_path_property_ref** XML attribute assignment is optional. It may only be used for an identified property (**property_ref**) for which the underlying data type is a class reference type (see 8.3.10).

NOTE 3 The **subitem_path_property_ref** XML attribute may be used as a simple way for representing embedded instance property-value pairs (see example in Clause D.2.1.1 of this normative annex)

DependentValueRepresentation and ***ConditionValueRepresentation*** stand for any kind of typed value defined in this normative annex.

NOTE 4 ***DependentValueRepresentation*** and ***ConditionValueRepresentation*** are mandatory.

The dependence of the property to some other properties is characterized by the **val:environment** XML element that contains at least one **val:property_value**. Other conditions may be defined in the ***OtherPossibleConditions*** area, using the same **val:property_value** structure.

NOTE 5 A condition is represented as a common property value couple: a reference to a property identifier (IRDI), and a typed value.

NOTE 6 Condition property is defined in Clause 6.7.4.

NOTE 7 The **cat:property_value**, **val:property_value** and **val:environment** XML element content assignments are mandatory.

NOTE 8 **val:property_value** and **val:environment** are defined in ISO/TS 29002-10:2009, Clause 5.2.4.

EXAMPLE The following is a valid representation of a dependent property value: the dependent property is identified by the “0123-ABC#02-LIFETIME#1” IRDI, and the associated value is an integer measure value (see D.1.17) expressed in second (**UOM_code**). This value depends on a single condition. The condition property is identified by the “0123-ABC#02-LOAD#3” IRDI, and its associated value is a real measure value (see D.1.16).

```
<cat:property_value property_ref="0123-ABC#02-LIFETIME#1">
        <val:measure_single_number_value UOM_code="s">
                <val:integer_value>1000000</val:integer_value>
        </val:measure_single_number_value>
        <val:environment>
                <val:property_value property_ref="0123-ABC#02-LOAD#3">
                        <val:measure_single_number_value UOM_code="N">
                                <val:real_value>1512.37</val:real_value>
                        </val:measure_single_number_value>
                </val:property_value>
        </val:data_environment>
</cat:property_value>
```

D.2.1.4 Representation property

The OntoML representation of the value of a property that is defined as a representation property is defined as follows:

```
<cat:property_value property_ref="PropertyIRDI" subitem_path_property_ref="PropertyIRDIs">
        ValueRepresentation
</cat:property_value>
```

PropertyIRDI is a valid property concept identifier (IRDI) defined according to Clause 9.1.3.2.

NOTE 1 The **property_ref** XML attributes assignment is mandatory.

PropertyIRDIs is a list of property concept identifiers (IRDIs) defined according to Clause 9.1.3.2, that allows to describe the path from a source property (specified by the **property_ref** XML attribute) of which data type is a class reference type, to a target property that participates to the description of an (sub-)embedded instance of the instance for which the source property is defined.

NOTE 2 The **subitem_path_property_ref** XML attribute assignment is optional. It may only be used for an identified property (**property_ref**) for which the underlying data type is a class reference type (see 8.3.10).

NOTE 3 The **subitem_path_property_ref** XML attribute may be used as a simple way for representing embedded instance property-value pairs (see example in Clause D.2.1.1 of this normative annex)

ValueRepresentation (mandatory) stands for any kind of typed value defined in this normative annex.

NOTE 4 Representation property is defined in Clause 6.7.5.

NOTE 5 **cat:property_value** is defined in ISO/TS 29002-10:2009, Clause 5.2.4.

EXAMPLE The following is a valid representation of a representation property value: the property is identified by the "0123-ABC#02-PRICE#1" IRDI, and the associated value is a real currency value (see D.1.13).

```
<cat:property_value property_ref="0123-ABC#02-PRICE#1">
    <val:currency_value currency_code="EUR">
        <val:real_value>10.53</val:real_value>
    </val:currency_value>
</cat:property_value>
```

D.2.2 Representation of class instances

This subclause specifies the OntoML representation of:

— item class and item case-of class instances;

— functional model class and functional model class view-of instances.

D.2.2.1 Representation of item class and item class case-of instances

The OntoML representation of an instance of an item class or an item class case-of is defined as follows:

```
<cat:item       class_ref="ClassIRDI"
                local_id="LocalId"
                data_specification_ref="DataSpecificationIRDI"
                is_global_id="GloballyIdentifiedBoolean">
    <cat:classification_ref>ClassIRDI</cat:classification_ref>
    …
    OtherPossibleClassificationReferences
    …
    <cat:reference      reference_number="StringReferenceNumber"
                        organization_code="StringOrganizationCode"
                        organization_ref="SupplierIRDI">
        <cat:designation>
            <val:local_string>
                <val:content>DesignationString</val:content>
            </val:local_string>
        </cat:designation>
    </cat:reference>
    …
    PropertyValue(s)Representation
    …
</cat:item>
```

NOTE 1 Item class and item class case-of are defined in Clause 6.7.2.1.

NOTE 2 **cat:item** is defined in ISO/TS 29002-10:2009, Clause 5.2.3.

Class_ref:ClassIRDI and ***classification_ref:ClassIRDI*** stand for an item class or an item class case-of valid class concept identifier (IRDI) defined according to Clause 9.1.3.1.

NOTE 3 The **class_ref** XML attribute assignment is mandatory.

LocalId stands for a local identifier: it is defined for a given **item** (**local_id** XML attribute), and referenced from a property (**item_local_ref** XML attribute) whose value domain is a class reference type.

NOTE 4 The **local_id** XML attribute is used if the class is intended to be referenced as a property value (see D.1.25). Therefore, its assignment is optional.

DataSpecificationIRDI stands for a valid class data specification identifier (IRDI) defined according to ISO/TS 29002-5.

NOTE 5 The **data_specification_ref** XML attribute assignment is optional.

GloballyIdentifiedBoolean is a possible Boolean value that specifies whether an item is identified globally, or if it is identified through its constituent components. When this value is equal to false, the item is an assembly. Its human-readable identification shall consists of its associated reference numbers (see **reference_number** XML attribute), if any, followed by the set of reference numbers of its constituent components computed recursively until those components for which the **is_global_id** XML attribute is set to true. When this value is set to true, the item is identified on its own. If present, the associated reference number contains enough information for identifying unambiguously the item, whether it is a component or a system.

NOTE 6 The **global_id** XML attribute assignment is optional.

cat:classification_ref (optional) specifies a possible reference (performed using a valid categorization class concept identifier (IRDI) defined according to 9.1.3.1 of this part of ISO 13584) to a categorization class defined in a given classification. If several references are required, they must be added using the same structure in the ***OtherPossibleClassificationReferences*** area.

NOTE 7 Categorization class is defined in Clause 6.7.2.1.

The **cat:reference** (optional) XML element provides for the specification of a reference number assigned by the information supplier.

NOTE 8 The specification of an item reference using the **cat:reference** XML element is optional.

StringReferenceNumber is a string assigned by the information supplier to unambiguously identify an item.

NOTE 9 When the **cat_reference** XML element is provided, the **reference_number** XML attribute is mandatory.

StringOrganizationCode is a code assigned to the organization that assigned the reference number. The format for such codes, and the system for assigning such codes, is not specified within this document.

NOTE 10 The **organization_code** XML attribute assignment is optional.

SupplierIRDI stands for a valid supplier data specification identifier (IRDI) defined according to Clause 9.1.1. It corresponds to the IRDI of the organization that assigned the reference number.

NOTE 11 The **organization_ref** XML attribute assignment is optional.

DesignationString gives a human readable designation of the item. This designation may be translated in various languages (see D.1.4 for localizable string representation).

NOTE 12 The **cat:designation** XML element use is optional.

PropertyValue(s)Representation is the location where property value representations (see D.2.1.1 to D.2.1.3) are defined.

EXAMPLE The following example is a valid representation of a simple instance. It is an instance of a class identified by the "0123-ABC#01-HEXASCREW#1" IRDI. A local identifier ("ITEM_1") is also assigned to this instance. This instance relates to two classes (identified by the "4444-XYZ#01-SCREW#1" and "5555-UVW#01-BOLT#1" IRDIs) belonging to two different classification scheme. Finally, this instance is described using two property values.

```
<cat:item class_ref="0123-ABC#01-HEXASCREW#1" local_id="ITEM_1">
        <cat:classification_ref>4444-XYZ#01-SCREW#1</cat:classification_ref>
        <cat:classification_ref>5555-UVW#01-BOLT#1</cat:classification_ref>
        <cat:property_value property_ref="0123-ABC#02-DIAMETER#1">
                <val:measure_single_number_value UOM_code="mm">
                        <val:integer_value>5</val:integer_value>
                </val:measure_single_number_value>
        </cat:property_value>
        <cat:property_value property_ref="0123-ABC#02-LENGTH#1">
                <val:measure_single_number_value UOM_code="mm">
                        <val:integer_value>20</val:integer_value>
                </val:measure_single_number_value>
        </cat:property_value>
</cat:item>
```

D.2.2.2 Representation of functional model and functional model view-of instances

The OntoML representation of an instance of a functional model and functional model view-of class is defined as follows:

```
<cat:item       class_ref="ClassIRDI"
                local_id="LocalId"
                data_specification_ref="DataSpecificationIRDI"
                is_model="IsModelBoolean"
                created_view="FunctionalViewIRDI"
                view_of="LocalItemRef">
        <cat:classification_ref>ClassIRDI</cat:classification_ref>
        OtherPossibleClassificationReferences
        …
        PropertyValue(s)Representation
        …
</cat:item>
```

NOTE 1 Functional model class is defined in Clause 6.7.3.2.

NOTE 2 **cat:item** is defined in ISO/TS 29002-10:2009, Clause 5.2.3.

Class_ref:ClassIRDI and ***classification_ref:ClassIRDI*** stand for an item class or an item class case-of valid class concept identifier (IRDI) defined according to Clause 9.1.3.1.

NOTE 3 The **class_ref** XML attribute assignment is mandatory.

LocalId stands for a local identifier: it is defined for a given **item** (**local_id** XML attribute), and referenced from a property (**item_local_ref** XML attribute) whose value domain is a class reference type.

NOTE 4 The **local_id** XML attribute is used if the class is intended to be referenced as a property value (see D.1.25). Therefore, its assignment is optional.

DataSpecificationIRDI stands for a valid class data specification identifier (IRDI) defined according to ISO/TS 29002-5.

NOTE 5 The **data_specification_ref** XML attribute assignment is optional.

IsModelBoolean allows to specify that the item is an instance of a functional model (i.e., a representation of another item). It shall be set to true.

NOTE 6 The **is_model** XML attribute assignment is mandatory.

FunctionalViewIRDI stands for a valid functional view class identifier (IRDI) defined according to Clause 9.1.3.1 of this part of ISO 13584, such a functional view specifying the point of view described by the functional model instance

NOTE 7 The **created_view** XML attribute assignment is mandatory.

LocalItemRef stands for a local identifier: it specifies, in case of the representation of a functional model class view-of instance, a local reference to an item for which the current functional model view-of instance is a view.

NOTE 8 The **view_of** XML attribute assignment is mandatory for functional model view-of class instances. It is not used for functional model class instances.

cat:classification_ref (optional) specifies a possible reference (performed using a valid categorization class concept identifier (IRDI) defined according to Clause 9.1.3.1 of this part of ISO 13584) to a categorization class defined in a given classification. If several references are required, they must be added using the same structure in the ***OthePossibleClassificationReferences*** area.

NOTE 9 Categorization class is defined in Clause 6.7.2.1.

PropertyValue(s)Representation is the location where representation property values (see D.2.1.4) are defined.

EXAMPLE The following example is a valid representation of a simple functional model instance. It is an instance of a functional model class identified by the “0123-ABC#01-HEXASCREWPRICE#1” IRDI. This instance describes the referenced item (identified by the “ITEM_1” local reference) in a point of view specified by the referenced functional view class (“0123-ABC#01-PRICEVIEW#1” IRDI). Finally, this instance is described using a single representation property value (“0123-ABC#02-PRICE#1” IRDI).

```
<cat:item class_ref="0123-ABC#01-HEXASCREWPRICE#1" is_model="true"
      created_view="0123-ABC#01-PRICEVIEW#1" view_of="ITEM_1">
      <cat:property_value property_ref="0123-ABC#02-PRICE#1">
            <val:currency_value currency_code="EUR">
                  <val:real_value>10.53</val:real_value>
            </val:currency_value>
      </cat:property_value>
</cat:item>
```

D.3 Value representation of extended types defined in OntoML

This clause specifies the OntoML representation of the value of a property whose value domain is an ISO 13584 / IEC 61360 extended data types.

OntoML specifies an ontology of extended types.

NOTE 1 The ontology of extended values is defined in Annex E.

In this ontology, the value structure of each OntoML extended data type is defined by an item class ontology concept (an item class) to which describing properties are associated. Consequently, the value representation of an extended data type is defined by an instance that refers to the item class (defined in the ontology) that specifies the associated value structure.

NOTE 2 Representation of item class instances (**cat:item** XML element) is defined in Clause D.2.2.1 of this normative annex.

The general structure of a reference to a value representation of an extended data type is defined as follows:

```
<val:item_reference_value item_local_ref="LocalIdRef"/>

<cat:item class_ref="ExtendedDataTypeValueStructureClassIRDI" local_id="LocalId">
…
</cat:item>
```

LocalId stands for a local XML identifier. It is intended to be used as a reference target for a property value whose value domain is a class reference type. The reference is performed by setting the ***LocalIdRef*** value of the **item_local_ref** XML attribut.

NOTE 3 LocalId/LocalIdRef is similar to the the ID/IDREF XML reference mechanism.

LocalId is defined according to Clause 3.3.8 of XML Schema Part 2*:* Datatypes.

LocalIdRef is defined according to Clause 3.3.6 of XML Schema Part 2*:* Datatypes.

NOTE 4 Class reference type value (**val:item_reference_value** XML element) is defined in Clause D.1.25 of this normative annex.

ExtendedDataTypeValueStructureClassIRDI is the extended data value structure class identifier (a valid class concept identifier (IRDI) defined according to Annex E of this part of ISO 13584).

The following subclauses specify, for each available data type, the corresponding value representation (instance) structure.

D.3.1 Value representation of ISO 13584 / IEC 61360 extended data types

This clause specifies the value representation of extended data types (see 8.3.13) according to the value structures defined in the ontology of OntoML extended values.

D.3.1.1 Placement type

The OntoML representation of the value of a property whose value domain is a placement type is represented as follows:

```
<cat:item       class_ref="0112-1---13584_32_1#01-PLACEMENT#1"
                local_id="LocalId">
        <cat:property_value property_ref="0112-1---13584_32_1#02-PLACEMENT:LOCATION#1">
                <item_reference_value item_local_ref="CART_P"/>
        </cat:property_value>
</cat:item>

<cat:item class_ref="0112-1---13584_32_1#01-CARTESIAN_POINT#1" local_id="CART_P">
        <cat:property_value property_ref="0112-1---13584_32_1#02-CARTESIAN_POINT:COORDINATES#1">
                <val:sequence_value>
                        <val:real_value>0</val:real_value>
                        <val:real_value>0</val:real_value>
                </val:sequence_value>
        </cat:property_value>
</cat:item>
```

NOTE The representation given above is complete according to the ontology of extended data type value structures. In the next clauses, only the general structure of each extended data type value is represented.

The IRDI of the class specifying the placement type value structure shall be set to "**0112-1---13584_32_1#01-PLACEMENT#1**", as defined in the ontology of external values specified in Annex E.

D.3.1.2 Axis 1 placement type

The OntoML representation of the value of a property whose value domain is an axis 1 placement type is represented as follows:

```
<cat:item      class_ref="0112-1---13584_32_1#01-AXIS1_PLACEMENT#1"
               local_id="LocalId">
       …
       PropertyValue(s)Representation
       …
</cat:item>
```

The IRDI of the class specifying the axis 1 placement type value structure shall be set to "**0112-1---13584_32_1#01-AXIS1_PLACEMENT#1**", as defined in the ontology of extended values specified in Annex E.

PropertyValue(s)Representation is the representation of the property(ies) of the item class that represents the extended type *axis1 placement type* according to Annex E. These properties are represented as specified in Clause D.1.

D.3.1.3 Axis 2 placement 2D type

The OntoML representation of the value of a property whose value domain is an axis 2 placement 2D type is represented as follows:

```
<cat:item      class_ref="0112-1---13584_32_1#01-AXIS2_PLACEMENT_2D#1"
               local_id="LocalId">
       …
       PropertyValue(s)Representation
       …
</cat:item>
```

The IRDI of the class specifying the axis 2 placement 2D type value structure shall be set to "**0112-1---13584_32_1#01-AXIS2_PLACEMENT_2D#1**", as defined in the ontology of extended values specified in Annex E.

PropertyValue(s)Representation is the representation of the property(ies) of the item class that represents the extended type *axis2 placement 2D type* according to Annex E. These properties are represented as specified in Clause D.1.

D.3.1.4 Axis 2 placement 3D type

The OntoML representation of the value of a property whose value domain is an axis 2 placement 3D type is represented as follows:

```
<cat:item      class_ref="0112-1---13584_32_1#01-AXIS2_PLACEMENT_3D#1"
               local_id="LocalId">
       …
       PropertyValue(s)Representation
       …
</cat:item>
```

The IRDI of the class specifying the axis 2 placement 3D type value structure shall be set to "**0112-1---13584_32_1#01-AXIS2_PLACEMENT_3D#1**", as defined in the ontology of extended values specified in Annex E.

PropertyValue(s)Representation is the representation of the property(ies) of the item class that represents the extended type *axis2 placement 3D type* according to Annex E. These properties are represented as specified in Clause D.1.

D.3.2 Reference to external representation of ISO 13584 / IEC 61360 items

This clause specifies the value representation of reference to external representation types defined in the ontology of extended data types.

D.3.2.1 Representation reference type

The OntoML representation of the value of a property whose value domain is a representation reference type is represented as follows:

```
<cat:item      class_ref="0112-1---13584_32_1#01-REPRESENTATION_REFERENCE#1"
               local_id="LocalId">
    …
    PropertyValue(s)Representation
    …
</cat:item>
```

The IRDI of the class specifying the representation reference type value structure shall be set to "**0112-1---13584_32_1#01-REPRESENTATION_REFERENCE#1**", as defined in the ontology of extended values specified in Annex E.

PropertyValue(s)Representation is the representation of the property(ies) of the item class that represents the extended type *representation type* according to Annex E. These properties are represented as specified in Clause D.1.

D.3.2.2 Program reference type

The OntoML representation of the value of a property whose value domain is a program reference type is represented as follows:

```
<cat:item      class_ref="0112-1---13584_32_1#01-PROGRAM_REFERENCE#1"
               local_id="LocalId">
    …
    PropertyValue(s)Representation
    …
</cat:item>
```

The IRDI of the class specifying the program reference type value structure shall be set to "**0112-1---13584_32_1#01-PROGRAM_REFERENCE#1**", as defined in the ontology of extended values specified in Annex E.

PropertyValue(s)Representation is the representation of the property(ies) of the item class that represents the extended type *program reference type* according to Annex E. These properties are represented as specified in Clause D.1.

Annex E
(normative)

Ontology specification of extended values used in OntoML

The OntoML model contains some application oriented data types whose values cannot be directly represented using ISO/TS 29002-10 elements. These data types are called extended data types (see 8.3.13).

To provide for their representation, this normative annex specifies an ontology that models these data types. In this ontology each extended data type value structure is represented as an item class, described by properties. Thus, values of extended data types are represented as instances of these item classes.

This annex defines the structure of this ontology, together with IRDIs allowing to reference these item classes and their properties. Annex D.3 specifies how instances of these item classes are represented using ISO/TS 29002-10 schemas.

The ontology of OntoML extended data type values distinguishes two kinds of classes:

— first level classes that corresponds to the actual OntoML extended data type values;

— second level classes that allows to describe first level classes in an accurate and consistent way.

EXAMPLE The *axis 1 placement* class is defined as a first level class, and consequently, it may be used to characterize the value domain of a property defined in a product ontology and whose data type would be an *axis 1 placement type*. *axis 1 placement* is itself described using various properties, as for instance, its associated *reference axis*. The *reference axis* property underlying data type is a complex structure (a class) representing the axis *direction*. This latter class is considered as a second level class, because only used in the context of the *axis 1 placement* first level class specification.

This ontology of OntoML extended data type value structures distinguishes the following categories of first level classes:

— STEP spatial positioning;

— PLIB external representation.

The next clauses describe the general structure of the ontology of OntoML extended data type value structures.

E.1 Structure of the ontology of extended values

A planning model of the ontology of OntoML extended values is represented in the UML diagram defined in Figure E.1.

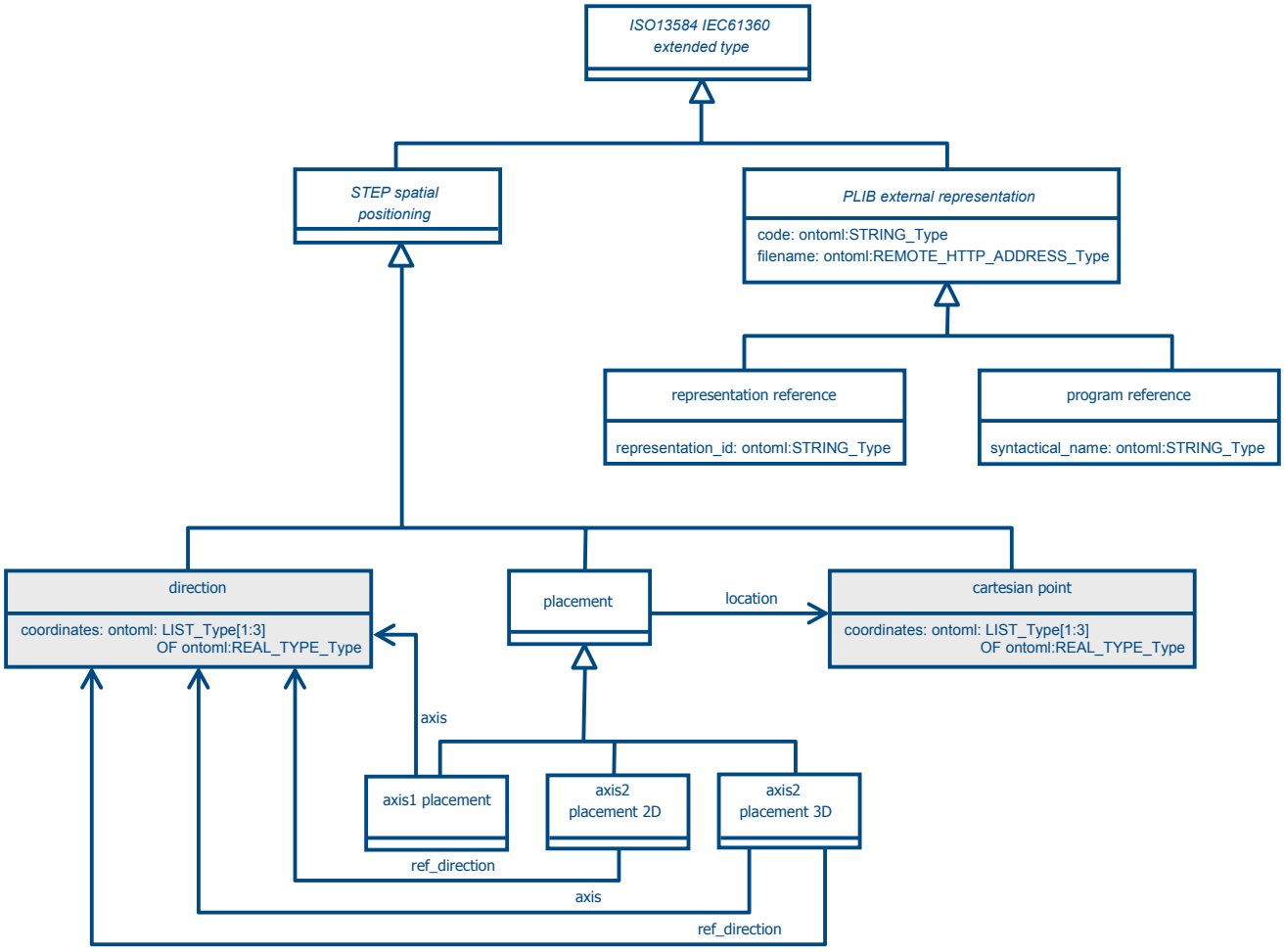

Figure E.1 — Planning model of the ontology of extended values

In this planning model, the following graphical notations are used:

— plain boxes represent ontology classes;

— plain boxes with a white colored background represent first level ontology classes, i.e., actual extended data types intended to be referenced from properties in product ontology definitions;

— classes in italic face represent non instanciable classes;

— plain boxes with a gray colored background represent second level ontology classes, i.e., classes that are intended to be referenced by first level ontology classes or by some other second level ontology classes;

— triangles specify the is-a semantic relationship between ontology classes;

— arrows specify the association between ontology classes, the associated label being the name of the property that represents a given association.

— simple type properties are typed using the OntoML basic type system where the *ontoml* prefix stands for the OntoML Schema URI.

First level classes may be seen as split in the following two categories:

— STEP spatial positioning category:

  — axis 1 placement;

  — axis2 placement 2D;

  — axis2 placement 3D;

  — placement;

— PLIB external representation category;

  — program reference;

  — representation reference;

E.2 Definition of the OntoML extended values

This clause defines the structure of the various OntoML extended values specified in the ontology of OntoML extended values.

E.2.1 First level classes: extended value structures

This clause gives the structure of the first level classes specified in the ontology of OntoML extended values.

E.2.1.1 OntoML extended value

The **OntoML extended value** class is the root class of the ontology of extended values.

NOTE This class is not instanciable.

The **OntoML extended value** class is identified by the following IRDI: “0112-1---13584_32_1#01-ONTOML_EXTENDED_VALUE#1”.

E.2.1.2 STEP spatial positioning

The **STEP spatial positioning** class is intended to factorize the various STEP postioning constructs (see next clauses).

NOTE This class is not instanciable.

The **STEP spatial positioning** class is identified by the following IRDI: “0112-1---13584_32_1#01-STEP_SPATIAL_POSITIONING#1”.

E.2.1.3 Placement

The **placement** class specifies a position with respect to the coordinate system of its geometric context.

NOTE **Placement** represents the *placement* entity from ISO 10303-42 in OntoML.

The **placement** class is identified by the following IRDI: “0112-1---13584_32_1#01-PLACEMENT#1”.

Defined properties:

placement location (IRDI: “0112-1---13584_32_1#02-PLACEMENT:LOCATION#1”): the geometric position of a reference point, such as the centre of a circle, of the item.

E.2.1.4 Axis 1 placement

The **axis 1 placement** class specifies a direction and a location in three-dimensional space of a single axis.

NOTE **Axis 1 placement** represents the *axis1_placement* entity from ISO 10303-42 in OntoML.

The **axis 1 placement** class is identified by the following IRDI: “0112-1---13584_32_1#01-AXIS1_PLACEMENT#1”.

Defined properties:

axis (IRDI: “0112-1---13584_32_1#02-AXIS1_PLACEMENT:AXIS#1”): the direction (see E.2.2.2) of the local Z axis.

E.2.1.5 Axis 2 placement 2D

The **axis 2 placement 2D** class specifies a location and an orientation in two-dimensional space of two mutually perpendicular axes.

NOTE **Axis 2 placement 2D** represents the *axis2_placement_2D* entity from ISO 10303-42 in OntoML.

The **axis 2 placement 2D** class is identified by the following IRDI: “0112-1---13584_32_1#01-AXIS2_PLACEMENT_2D#1”.

Defined properties:

ref direction (IRDI: “0112-1---13584_32_1#02-AXIS2_PLACEMENT_2D:REF_DIRECTION#1”): the direction (see E.2.2.2) used to determine the direction of the local X axis.

E.2.1.6 Axis 2 placement 3D

The **axis 2 placement 3D** class specifies a location and an orientation in three-dimensional space of two mutually perpendicular axes.

NOTE **Axis 2 placement 3D** represents the *axis2_placement_3D* entity from ISO 10303-42 in OntoML.

The **axis 2 placement 3D** class is identified by the following IRDI: “0112-1---13584_32_1#01-AXIS2_PLACEMENT_3D#1”.

Defined properties:

axis (IRDI: “0112-1---13584_32_1#02-AXIS2_PLACEMENT_3D:AXIS#1”): the direction (see E.2.2.2) of the local Z axis.

ref direction (IRDI: “0112-1---13584_32_1#02-AXIS2_PLACEMENT_3D:REF_DIRECTION#1”): the direction (see E.2.2.2) used to determine the direction of the local X axis.

E.2.1.7 PLIB external representation

The **PLIB external representation** class is intended to factorize the various PLIB representation constructs (see next clauses).

NOTE 1 This class is not instanciable.

The **PLIB external representation** class is identified by the following IRDI: “0112-1---13584_32_1#01-PLIB_EXTERNAL_REPRESENTATION#1”.

Defined properties:

code (IRDI: “0112-1---13584_32_1#02-PLIB_EXTERNAL_REP:CODE#1”): identifies the external representation in a class.

file name (IRDI: “0112-1---13584_32_1#02-PLIB_EXTERNAL_REP:FILE_NAME#1”): the URI of the file that contains the external representation.

NOTE 2 See ISO 13584-24 for details.

E.2.1.8 Representation reference

The **representation reference** class specifies the kind of representation for an item that is provided as an external file, and the name of the external file that contains this representation.

NOTE 1 In **DICTIONARY_IN_STANDARD_FORMAT_Type** and in **LIBRARY_IN_STANDARD_FORMAT_Type**, only external file protocols that are allowed either by the library integrated information model indicated by the **ontoml_structure** XML element or the view exchange protocols referenced in the **supported_vep** XML element, both defined in the **HEADER_Type** XML complex type..

EXAMPLE ISO 13584-102 specifies a representation category that captures the generic concepts used to describe the representation of a product in ISO 10303 application protocols. Such a representation is intended to be referenced using a **representation reference** instance.

The **representation reference** class is identified by the following IRDI: “0112-1---13584_32_1#01-REPRESENTATION_REFERENCE#1”.

Defined properties:

representation id (IRDI: “0112-1---13584_32_1# 02-REP_REF:REPRESENTATION_ID#1”):a label that corresponds to the referenced representation.

NOTE 2 See ISO 13584-24 for details.

E.2.1.9 Program reference

The **program reference** class specifies a reference to an algorithm intended to generate a representation of an item.

NOTE 1 See ISO 13584-24 for details.

The **program reference** class is identified by the following IRDI: “0112-1---13584_32_1#01-PROGRAM_REFERENCE#1”.

Defined properties:

syntactical name (IRDI: “0112-1---13584_32_1# 02-PROG_REF:SYNTACTICAL_NAME#1”): the name by which the program shall be triggered.

NOTE 2 See ISO 13584-24 for details.

E.2.2 Second level of OntoML extended values

This clause gives the structure of the second level classes that are used for defining the various extended data types.

E.2.2.1 Cartesian point

The **cartesian point** class is specified by a point defined by its coordinates in a rectangular Cartesian coordinate system, or in a parameter space. The cartesian point is defined in a one, two or three-dimensional space as determined by the number of coordinates in the list.

NOTE 1 See ISO 10303-42 for details.

The **cartesian point** class is identified by the following IRDI: “0112-1---13584_32_1#01-CARTESIAN_POINT#1”.

Defined properties:

coordinates (IRDI: “0112-1---13584_32_1#02-CARTESIAN_POINT:COORDINATES#1”): list of at least one and at most three real values that specifies coordinates in a rectangular Cartesian coordinate system, or in a parameter space.

NOTE 2 See ISO 10303-42 for details.

E.2.2.2 Direction

The **direction** class defines a general direction vector in two or three dimensional space.

NOTE 1 See ISO 10303-42 for details.

The **direction** class is identified by the following IRDI: “0112-1---13584_32_1#01-DIRECTION#1”.

Defined properties:

direction ratios (IRDI: “0112-1---13584_32_1#02-DIRECTION:DIRECTION_RATIOS#1”): list of at least two and at most three real values that a general direction vector in two or three dimensional space. The actual magnitudes of the components have no effect upon the direction being defined, only the ratios x:y:z or x:y are significant.

NOTE 2 See ISO 10303-42 for details.

E.3 Synthesis of OntoML extended values identifiers (IRDIs)

Table E.1 gives the list of the IRDI defined for the classes belonging to the ontology of OntoML extended values.

Table E.1 — OntoML extendedvalues: class identifiers

OntoML class name	OntoML class identifier (IRDI)
axis 1 placement	0112-1---13584_32_1#01-AXIS1_PLACEMENT#1
axis 2 placement 2D	0112-1---13584_32_1#01-AXIS2_PLACEMENT_2D#1
axis 2 placement 3D	0112-1---13584_32_1#01-AXIS2_PLACEMENT_3D#1
OntoML extended value (abstract)	0112-1---13584_32_1#01-ONTOML_EXTENDED_VALUE#1
placement	0112-1---13584_32_1#01-PLACEMENT#1
PLIB external representation (abstract)	0112-1---13584_32_1#01-PLIB_EXTERNAL_REPRESENTATION#1
program reference	0112-1---13584_32_1#01-PROGRAM_REFERENCE#1
representation reference	0112-1---13584_32_1#01-REPRESENTATION_REFERENCE#1
STEP spatial positioning (abstract)	0112-1---13584_32_1#01-STEP_SPATIAL_POSITIONING#1

Table E.2 gives the list of the IRDI defined for the properties belonging to the ontology of OntoML extended values.

Table E.2 — OntoML extendedvalues: property identifiers

OntoML class name	OntoML class identifier (IRDI)
axis 1 placement: axis	0112-1---13584_32_1#02-AXIS1_PLACEMENT:AXIS#1
axis 2 placement 2D: ref direction	0112-1---13584_32_1#02-AXIS2_PLACEMENT_2D:REF_DIRECTION#1
axis 2 placement 3D: axis	0112-1---13584_32_1#02-AXIS2_PLACEMENT_3D:AXIS#1
axis 2 placement 3D: ref direction	0112-1---13584_32_1#02-AXIS2_PLACEMENT_3D:REF_DIRECTION#1
cartesian point: coordinates	0112-1---13584_32_1#02-CARTESIAN_POINT_COORDINATES#1
direction: direction ratio	0112-1---13584_32_1#02-DIRECTION:DIRECTION_RATIO#1
placement: location	0112-1---13584_32_1#02-PLACEMENT:LOCATION#1
PLIB external representation: code	0112-1---13584_32_1#02-PLIB_EXTERNAL_REP:CODE#1
PLIB external representation: file name	0112-1---13584_32_1#02-PLIB_EXTERNAL_REP:FILE_NAME#1
program reference: syntactical name	0112-1---13584_32_1#02-PROG_REF:SYNTACTICAL_NAME#1
representation reference: representation id	0112-1---13584_32_1#02-REP_REF:REPRESENTATION_ID#1

E.4 Formal model of the ontology of OntoML extended values

This formal model of the ontology of OntoML extended values may be downloaded at the following URL:

http://www.tc184-sc4.org/implementation_information/13584/00032/

Annex F
(normative)

Structural transformation of the CIIM model from OntoML XML Schema to EXPRESS

This normative annex specifies how the OntoML constructs shall be converted into EXPRESS data. Such a conversion allows to build XML tools that generate EXPRESS representations of OntoML document instances in order to check the semantic consistency of its content with respect to the integrity constraints defined in the CIIM.

F.1 Difference between OntoML and CIIM information elements

OntoML is an XML Schema allowing to represent, within an XML document instance, all the information elements represented in a CIIM EXPRESS physical file.

As a rule, all the information elements of a CIIM EXPRESS physical file are explicitly represented in an OntoML document instance, and all the constraints that these information elements are supposed to fulfill shall also hold for OntoML document instances content.

Nevertheless, four differences were decided during the OntoML design:

1. sharing *Vs* duplication of some information elements

Both in OntoML and in CIIM EXPRESS physical files, CIIM ontology concepts are defined only once and referenced several times. Concerning the other pieces of information of the CIIM:

— in EXPRESS several CIIM ontology concepts may share some EXPRESS entity.

EXAMPLE 1 The **item_names** entity data type is shared some ontology concepts.

NOTE 1 **item_names** is defined in ISO 13584-42:2010, Clause F.3.9.2.7.

— in XML, it was decided that each CIIM ontology concept will only reference other CIIM ontology concepts. All the others pieces of information that are referenced by a CIIM ontology concept in a CIIM EXPRESS physical file are embedded in XML. Thus, their content is possibly duplicated if the same piece of information is referenced by several CIIM ontology concept.

NOTE 2 The duplication of pieces of information does not change the semantics of the underlying CIIM EXPRESS data model.

2. Use of pre-existing XML capabilities to characterize internet resources

In the CIIM, some powerful but complex EXPRESS mechanisms were defined in order to be able to represent both the external files information and the way to process them. In OntoML, this representation has been replaced by the use of the MIME protocol mechanism that is sufficient to interpret external files content.

3. Removing some constraints that cannot be checked in XML

In the CIIM model, the **prefix_ordered_class_list** constraint stipulates that classes contained in an OntoML document instance are sorted in such a way that no forward reference from one class to some other classes appear. Due to the fact that such a constraint cannot be checked in XML, it is removed from the OntoML specification and OntoML document instances are not supposed to fulfill this constraint. If needed, this order may be compiled and ensured, when the OntoML content is translated into EXPRESS for constraint checking.

4. Simplification of the CIIM model

The following simplifications were assumed.

— we assume that when a document is visible, it is also applicable;

— translations representations have been simplified;

— resources for the representation of documentation that can be associated to any CIIM ontology concept has been simplified;

— removing of CIIM constructs that are not pertinent in the OntoML context;

EXAMPLE 2 The entity instance type CIIM data type has been omitted in the OntoML type system.

— simplification of the representation of CIIM global identifiers (known as basic semantic units) of CIIM ontology concepts.

F.2 Introductory example

This annex of OntoML specifies a formal mapping from the XML pieces of information, included in an OntoML document instance, to those EXPRESS pieces of information that would be included in a CIIM EXPRESS physical file used to exchange the same information.

This clause identifies, on a very simple example, the various components of such a mapping.

Let's consider the following UML model used to illustrate a simple information model (Figure F.1):

NOTE 1 Such an information model could be represented in EXPRESS.

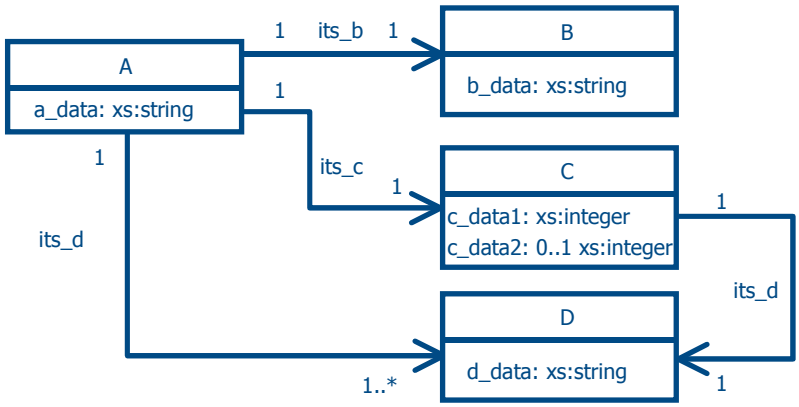

Figure F.1 — A UML information model example

A composite class *A* is defined through a *a_data* string attribute, and an *its_b*, an *its_c* and an *its_d* attributes whose the data types are respectively classes *B*, *C* and *D*. Class *B* is described through a *b_data* string attribute. Class *C* is described through a *c_data1 and* an optional *c_data2* integer attributes, and an *its_d* attributes whose the data type is class *D*. Class *D* is described through a *d_data* string attribute.

The OntoML-like representation of this information model, using the same transformation principles than those used to design OntoML from the CIIM EXPRESS model, could be as illustrated in the following UML-like model (Figure F.2).

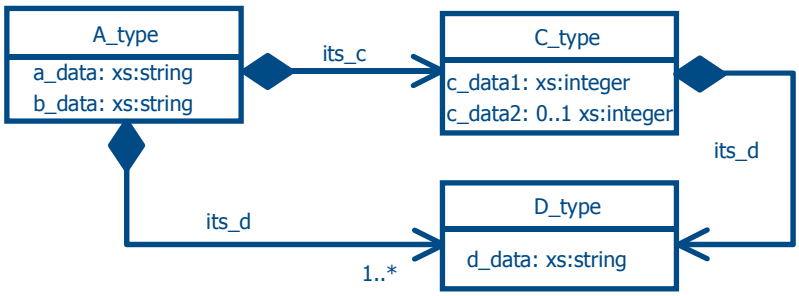

Figure F.2 — An UML-like representation of the information model

NOTE 2 Notations used in this UML-like model are those defined in Clause 6.3.1.

A corresponding XML Schema representation, could be the following (Figure F.3):

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="a" type="A_type"/>
    <xs:complexType name="A_type">
        <xs:sequence>
            <xs:element name="a_data" type="xs:string"/>
            <xs:element name="b_data" type="xs:string"/>
            <xs:element name="its_c" type="C_type"/>
            <xs:element name="its_d" type="its_D_type">
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="C_type">
        <xs:sequence>
            <xs:element name="c_data1" type="xs:int"/>
            <xs:element name="c_data2" type="xs:int" minOccurs="0"/>
            <xs:element name="its_d" type="D_type">
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="its_D_type">
        <xs:sequence>
            <xs:element name="d" type="D_type" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="D_type">
        <xs:sequence>
            <xs:element name="d_data" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Figure F.3 — An XML Schema example

The XML element named *a* has been defined for playing both the role of a data container and the role of the root of the XML document: its type is the *A_type* XML complex type.

Thus it is possible to represent the same information using two representation formats: the ISO 10303 21 EXPRESS instance syntax on the base of the UML model defined in Figure F.1, and the XML document syntax on the base of the XML Schema defined in Figure F.4. Table F.1 represents an example of these two representation formats.

Table F.1 — XML and corresponding ISO 10303-21 instances

XML document	ISO 10303-21 instances
<a> <a_data>string 1</a_data> <b_data>string 2</b_data> <its_c> <c_data1>10</c_data1> <c_data2>20</c_data2> <its_d> <d_data>string 3</d_data> </its_d> </its_c> <its_d> <d> <d_data>string 4</d_data> </d> <d> <d_data>string 5</d_data> </d> </its_d> 	#1=A('string 1', #2, #6, (#4, #5)); #2=D('string 2'); #3=D('string 3'); #4=D('string 4'); #5=D('string 5'); #6=C(10, 20, #3);

We note that defining mapping rules from each XML piece of information into EXPRESS pieces of information needs the following:

— capability to instantiate the EXPRESS representation of the container XML element, i.e., the XML element named *a*, and to reference it;

— capability, in the XML document, to identify any piece of information embedded, directly or indirectly, within the container named *a* and, in the EXPRESS file, to instantiate and to identify any piece of information referenced directly or indirectly from the entity instance named *a*;

— capability to express how each particular value represented in XML shall be converted to be represented in EXPRESS.

In the mapping defined in this annex:

— all the mappings start from an embedding XML element that is either a CIIM ontology concept or the dictionary root element. The EXPRESS image of this embedding element is denoted **SELF**.

— the identification of XML embedded element uses the location where the mapping rules are represented in OntoML (Clause F.3) and, when need XPath / XSLT notations.

— the identification of EXPRESS items referenced by the EXPRESS image of the embedding XML element use the EXPRESS path syntax, starting from **SELF** (see F.4).

— instance creation and value representation use a set of specific functions defined in Clause F.4.3.4.2.

Moreover, Clause F.3 defines the overall structure of OntoML embedding elements.

F.3 Mapping rules location in OntoML

Every OntoML element definition is associated to a (some) mapping rule(s). This mapping rule is expressed using the *annotation* element proposed by the XML Schema specification. Figure F.4 illustrates the mapping location that would be defined in the *A_Type* XML complex type of the XML Schema example proposed in Figure F.3.

```
<xs:complexType name="A_type">
    <xs:sequence>
        <xs:element name="a_data" type="xs:string">
            <xs:annotation>
                <xs:appinfo> mapping rule(s)</xs:appinfo>
            </xs:annotation>
        </xs:element>
        <xs:element name="b_data" type="xs:string">
            <xs:annotation>
                <xs:appinfo> mapping rule(s)</xs:appinfo>
            </xs:annotation>
        </xs:element>
        <xs:element name="its_c" type="C_type">
            <xs:annotation>
                <xs:appinfo> mapping rule(s)</xs:appinfo>
            </xs:annotation>
        </xs:element>
        <xs:element name="its_d" type="its_D_type">
            <xs:annotation>
                <xs:appinfo> mapping rule(s)</xs:appinfo>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

Figure F.4 — Mapping representation in OntoML

The *appinfo* sub-element of the *annotation* element is intended to contain all the mapping rules.

Mapping rules involve two aspects:

— the link between an XML element and the corresponding EXPRESS attribute in the ISO 13584 data model: it is specified by EXPRESS target paths (using the group reference and attribute reference mechanism defined in ISO 10303-11:1994); it is the *localization part* of the mapping;

— the assignment of the XML element value(s) to the corresponding EXPRESS item(s) in the CIIM; it is the *value-conversion part* of the mapping.

F.4 Link between an OntoML element and a CIIM attribute

The OntoML to CIIM mapping is based on the definition of EXPRESS-based CIIM target paths and XPath-based OntoML source paths. These paths are used for expressing the correspondence between an XML element and an EXPRESS item.

F.4.1 OntoML source path

The XML source path is defined by the location of the annotation. Figure F.5 outlines the XML source path concept:

```
<xs:element name="b_data" type="xs:string">
        <xs:annotation>
            <xs:appinfo>mapping rule(s)</xs:appinfo>
        </xs:annotation>
</xs:element>
```

Figure F.5 — XML source Path

This annotation being assigned to the *b_data* element, the source path is the *B_data* element.

F.4.2 EXRESS target path

The role of the target path is to localize the target EXPRESS construct that corresponds to the source XML construct defined by the location of the annotation. The target EXPRESS construct is defined by a path that is built using EXPRESS attribute reference mechanism that use the classical dot notation (".") used to defined the entity datatype of the entity instance to be created.

NOTE Attribute reference is defined in ISO 10303-11:1994, Clause 12.7.3.

OntoML defines some XML elements as global elements and the other as local elements.

Only global elements are referenced by means of identifiers. All the other elements are embedded within global elements.

Thus, in the EXPRESS representation of an OntoML document instance:

— each OntoML global element will be represented by instantiating a corresponding EXPRESS entity; this instance will be denoted by and will define the context in which the embedded XML element of the OntoML global element is mapped.

— each piece of information that is not a global element is embedded within an XML global element and will be mapped onto an EXPRESS image of its embedding XML global element.

This global structure is shown in Figure F.6.

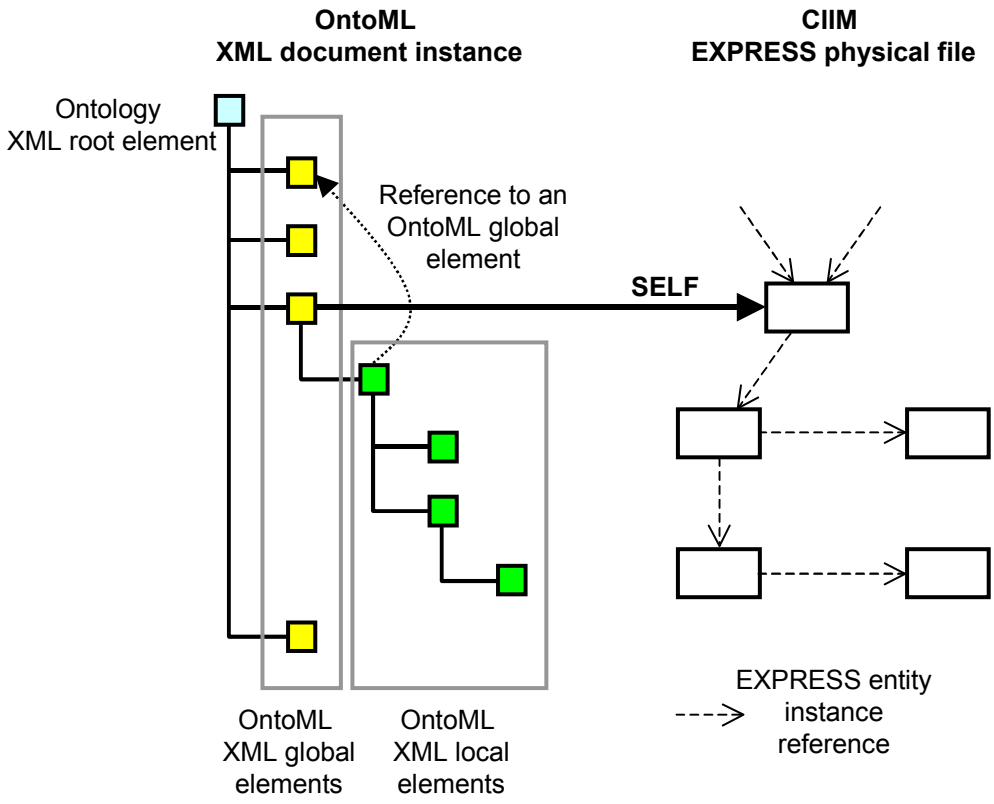

Figure F.6 — Global Vs local XML elements

OntoML specifies seven global elements:

— **supplier**, representing the supplier ontology concept;

— **class**, representing the class ontology concept;

— **property**, representing the property ontology concept;

— **data_type**, representing the datatype ontology concept;

— **document**, representing the document ontology concept;

— **ontoml**, representing the root element of an ontology and / or library.

The EXPRESS path is defined in a modular way: each embedded element of a local element contains a local path that starts by "." and that contains the EXPRESS path from the EXPRESS image of the XML local element to the EXPRESS attribute corresponding to the embedded element.

The complete path from the EXPRESS image of an XML global element instance to the EXPRESS image of an XML element which is indirectly embedded in the XML global element is built by concatenation of the global path, and of all the local paths encountered when moving in the XML tree structure from the initial XML global element to the final embedded XML element.

The complete path structure is built from a CIIM ontology concept instance as defined in Clause F.4.2.1, and from local paths as defined in Clause F.4.2.2. The particular part structure for embedded element belonging to a collection of element is defined in Clause F.4.2.3. The compete path structure built by aggregation of these sub paths is defined in Clause F.4.2.4.

F.4.2.1 CIIM ontology concept instance

In OntoML, XML global elements represent CIIM ontology concepts. The mapping of an XML global element to its corresponding EXPRESS image is done through the **SELF** keyword. It represents the instance of an EXPRESS entity, image of the XML global element.

The table below gives the meaning of the CIIM SELF instance depending on the OntoML context where it is defined. The context is defined by a pair: an XML complex type and an XML local element. It is defined in Table F.2.

Table F.2 — SELF meaning in its use context

OntoML context	"SELF" CIIM EXPRESS entity type
XML complex type: *CONTAINED_SUPPLIERS_Type* XML element: *supplier*	**supplier_element**
XML complex type: *CONTAINED_CLASSES_Type* XML element: *class*	**class** or one of its subtype
XML complex type: *CONTAINED_PROPERTIES_Type* XML element: *property*	**property_det** or one of its subtype

OntoML context	"SELF" CIIM EXPRESS entity type
XML complex type: *CONTAINED_DATATYPES_Type* XML element: *datatype*	**data_type_element**
XML complex type: *CONTAINED_DOUMENTS_Type* XML element: *document*	**document_element**
XML root element: *ontoml*	**dictionary** or one of its subtype

NOTE 1 If the OntoML document instance contains only a dictionary specification, the **SELF** instance is an instance of the **dictionary** or of the **dictionary_in_standard_format** CIIM EXPRESS entity data type.

NOTE 2 If the OntoML document instance contains only a library specification, the **SELF** instance is an instance of the **library** or of the **library_in_standard_format** CIIM EXPRESS entity data type.

NOTE 3 If the OntoML document instance contains both a dictionary and a library specification, the **SELF** instance is an instance of the **library** or of the **library_in_standard_format** CIIM EXPRESS entity data type.

EXAMPLE 1 In OntoML, class ontology concept is represented by the **class** XML global element. In the CIIM EXPRESS information model, this class ontology concept is represented by one instance of the **class** entity data type, or one of its subtypes. This EXPRESS instance is said the ***SELF*** instance. It represents the **class** XML element in an EXPRESS based universe.

XML global elements support polymorphism through the possibly associated **xsi:type** XML attribute/

NOTE 4 *xsi* stands for the prefix associated to the XML Schema:Structure specification that defines several attributes for direct use in XML document. Its namespace is: <http://www.w3.org/2001/XMLSchema-instance>.

It means that the EXPRESS mapping of an XML global element shall take into account this datatype information. Consequently, the following applies:

— when no **xsi:type** XML attribute is used to specify an XML global element, the **SELF** instance corresponds to an instance of the EXPRESS entity image of the XML global element complex type specification;

— when an **xsi:type** XML attribute is used to specify an XML global element, the **SELF** instance corresponds to an instance of the EXPRESS entity image of the referenced XML complex type;

EXAMPLE 2 The property ontology concept is partially defined as follows:

```
<xs:element name="property" type="PROPERTY_Type" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>A reference to a property</xs:documentation>
        <xs:appinfo>SELF</xs:appinfo>
    </xs:annotation>
</xs:element>
<xs:complexType name="PROPERTY_Type" abstract="true">
…
</xs:complexType>

<xs:complexType name="CONDITION_DET_Type">
    <xs:complexContent>
        <xs:extension base="PROPERTY_Type"/>
```

```
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="NON_DEPENDENT_P_DET_Type">
        <xs:complexContent>
            <xs:extension base="PROPERTY_Type"/>
        </xs:complexContent>
    </xs:complexType>
```

Let's now consider the following OntoML fragment:

```
<property xsi:type="NON_DEPENDENT_P_DET_Type" …>
```

In such a case, the **SELF** instance is an instance of the EXPRESS image associated to the OntoML **NON_DEPENDENT_P_DET_Type** XML complex type, i.e., an instance of the CIIM **non_dependent_p_det** EXPRESS entity data type.

F.4.2.2 Information elements of an ontology: local EXPRESS target path

A local CIIM target path is assigned to every local XML element defined in the XML complex type assigned either to a global XML element or to another local XML element.

It specifies a mapping link between the EXPRESS image of a local XML element and an EXPRESS attribute.

NOTE 1 This EXPRESS target path is said local, because it defines only locally the partial mapping.

The local CIIM target path has the structure defined in Figure F.7:

.local_path.attribute

Figure F.7 — Local EXPRESS target path structure

Where:

— *"."*: the instance of the EXPRESS entity image of the XML complex type used to represent the local XML element;

NOTE 2 If the complex type represents the specification of a global XML element, "." Represents the SELF instance.

— *sub_path*: the path defined from the EXPRESS entity instance to the EXPRESS entity instance where the target attribute is defined;

— *attribute*: the name of the EXPRESS attribute to which the global CIIM path defines a mapping.

EXAMPLE Real measures may be used to represent either a property value domain. In OntoML, real measures are represented on the base of the **REAL_MEASURE_TYPE_Type** XML complex type. This complex type defines a **unit** local XML element representing the specific unit of the real measure. The mapping of the unit XML element to the corresponding EXPRESS attribute is specified as follows:

.UNIT

The **"."** instance is an instance of the EXPRESS image associated to the OntoML **REAL_MEASURE_TYPE_Type** XML complex type, i.e., an instance of the CIIM **real_measure_type** EXPRESS entity data type.

F.4.2.3 Local EXPRESS target path for indexing a collection of XML elements

Some XML elements are defined as collections of other embedded XML elements. They correspond to EXPRESS attributes whose types are collections. Each of their embedded element corresponds to one element of the corresponding EXPRESS collection. Therefore, it is needed to specify a mapping between each embedded XML element and the corresponding EXPRESS collection element.

For that purpose, EXPRESS paths are completed using the EXPRESS aggregate indexing operator.

NOTE 1 Aggregate indexing operator is defined in ISO 10303-11:1994, Clause 12.6.1.

It consists of the collection value being indexed (the target EXPRESS attribute) and the index specification. The index specification is either set to a “i” integer variable or to an integer constant.

In case of an integer variable, its range is implicitly defined as follows:

— its minimum value is equal to the minimum bound of the target collection structure if it is an array, else is it is equal to 1;

— its maximum value is equal to the minimum value, minus one, plus the number of embedded XML elements that appear in the XML element for which the mapping is defined.

EXAMPLE This example illustrates an EXPRESS path for indexing collection of XML elements.

```
<xs:element name="translation" type="TRANSLATION_Type" minOccurs="0">
    <xs:annotation>
        <xs:appinfo>.ADMINISTRATION.TRANSLATION[i]</xs:appinfo>
    </xs:annotation>
</xs:element>
<xs:complexType name="TRANSLATION_Type">
    <xs:sequence>
        <xs:element name="translation_data" type="TRANSLATION_DATA_Type"
            maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="TRANSLATION_DATA_Type" abstract="false">
    <xs:sequence>
        …
        <xs:element name="translation_revision" type="REVISION_TYPE_Type">
            <xs:annotation>
                <xs:appinfo>.TRANSLATION_REVISION</xs:appinfo>
            </xs:annotation>
        </xs:element>
        …
    </xs:sequence>
</xs:complexType>
```

The OntoML **translation** element corresponds (**.ADMINISTRATION.TRANSLATION[i]**) to the EXPRESS **translation** attribute defined in the **administrative_data** EXPRESS entity data type and referenced by the **administration** EXPRESS attribute. The EXPRESS **translation** attribute datatype is a collection of **translation_data** EXPRESS entity instances. Each EXPRESS entity instance is described by a set of attributes. Among them, there is the **translation_revision** EXPRESS attribute. In OntoML, each item of the **translation** XML element collection is represented by an embedded **translation_data** XML element. The **translation_data** XML element is defined according to the **TRANSLATION_DATA_Type** XML complex type specification that describes a **translation_revision** XML element whose EXPRESS image is the previously described **translation_revision** EXPRESS attribute (**.TRANSLATION_REVISION**).

NOTE 2 Indexing a 2D aggregate structure would be expressed as follows: *attribute_name*[i][j].

F.4.2.4 Complete EXPRESS target path structure

A complete EXPRESS target path specifies the EXPRESS path from the **SELF** instance to the EXPRESS attribute that is the image of a final XML local element.

The complete path from the EXPRESS image of an XML global element instance to the EXPRESS image of an XML element (which is indirectly embedded in the XML global element) is built by concatenation of the **SELF** instance, and of all the local paths encountered when moving in the XML tree structure from the initial XML global element to the final embedded XML element.

The complete CIIM target path has the structure defined in Figure F.8:

SELF.sub_path.attribute

Figure F.8 — Complete EXPRESS target path structure

Where:

— *SELF*: the EXPRESS instance corresponding to the global OntoML element;

— *sub_path*: the concatenation of all the local paths encountered when moving in the XML tree structure from the initial XML global element to the final embedded XML element;

— *attribute*: the name of the target EXPRESS attribute corresponding to the final embedded XML element.

EXAMPLE 1 A class is a CIIM ontology concept that is associated with a name. In the CIIM EXPRESS information model, this name is represented in the **ITEM_NAMES** entity datatype by a **preferred_name** attribute. In OntoML, this CIIM ontology concept is represented by a **class** global XML element whose content model is defined by a **CLASS_Type** complex type. This complex type specifies an XML element called **preferred_name** that represents this class name. The complete EXPRESS target path would be implicitly defined as follows:

SELF.NAMES.PREFERRED_NAME

This complete CIIM target path specifies the link between the **preferred_name** element of an OntoML class ontology concept and the **preferred_name** attribute of the EXPRESS **item_names** entity. **SELF** represents a class entity instance (or an instance of one of its subtypes).

EXAMPLE 2 Assuming that the translation XML local element presented in the example defined in Clause F.4.2.3, is directly embedded in an XML global element representing for instance a property ontology concept, the complete path that leads to the translation_revision XML local element would be:

SELF.ADMINISTRATION.TRANSLATION[i].TRANSLATION_REVISION

F.4.3 Assignment of an OntoML element value to an EXPRESS attribute

Assigning an OntoML element value to an EXPRESS attribute requires to specify:

OntoML source and an EXPRESS target attribute paths defining the information elements to be mapped;

— an assignment operator;

— a syntax for accessing information units in the OntoML compliant XML instance document;

— specific EXPRESS constructors for those information elements that are not represented in a straightforward manner in OntoML according to the CIIM.

This clause specified all these assignment aspects.

F.4.3.1 Assignment operator

The assuagement of a value represented in an OntoML document to an EXPRESS target path is performed using the ":=" assignment operator.

F.4.3.2 Assignment operation

The assignment of an EXPRESS value to an EXPRESS target path is only defined for those XML element that define the final target of the corresponding complete EXPRESS target path. The assignment is performed using the following syntax:

EXPRESS target path := EXPRESS value

Where:

— *EXPRESS target path*: a local target path assigned to an XML element;

NOTE No other mapping is defined for the embedded XML element of this XML element

— *EXPRESS value*: a value being either referenced (simple value) or computed (complex value) by a specific mapping function.

EXAMPLE 1 The revision number of a class ontology concept is represented by an EXPRESS attribute called **revision** whose data type is a simple string. It is represented in OntoML by an XML element (**revision**) whose content model is also a simple string. The mapping between the OntoML representation of a revision consists of assigning the OntoML **revision** string value to the EXPRESS **revision** attribute.

EXAMPLE 2 The preferred name of a class ontology concept is represented by an EXPRESS attribute called preferred_name whose data type is the **translatable_label** entity. It is represented in OntoML by an XML element (**preferred_name**) whose content model is not represented using the same constructs for simplification purposes. Consequently, the mapping can not be represented simply using an EXPRESS target path. A specific mapping function shall be used.

F.4.3.3 Retrieving OntoML information

Defining a mapping between an OntoML document instance and EXPRESS instances requires to retrieve the XML document data, and then to assign them to EXPRESS entity attributes.

For that purpose, OntoML mapping rules use the XPath syntax. Every XPath is defined locally to the XML element for which the mapping rule is defined. The XPath syntax is restricted to the following constructs:

— **.**: returns the current node;

— ***@attribute_name****:* returns the <*attribute_name*> attribute value of the current node*;*

— ***:** returns all the children elements of the current element, whatever be their names.

— */*: separator used to specify the XPath localization steps;

— ***element_name***: returns all the <*element_name*> children nodes of the contextual node.

F.4.3.4 Assigning OntoML information to EXPRESS target paths

This clause defines the different means used to assigned values referenced from an XML document to EXPRESS target paths.

F.4.3.4.1 Assigning a simple OntoML value to a simple EXPRESS attribute

An XML simple value to an EXPRESS attribute assignment is implicitly done for those XML element whose content model is defined as simple. For simplification purposes, such a simple assignment does not require to use the assignment operator.

EXAMPLE The date of original definition of a CIIM ontology concept is mapped as follows:

```
<xs:element name="class" type="CLASS_Type" maxOccurs="unbounded">
    <xs:annotation>
        <xs:appinfo>SELF</xs:appinfo>
    </xs:annotation>
</xs:element>
<xs:complexType name="CLASS_Type" abstract="true">
    <xs:sequence>
    …
```

```
<xs:element name="date_of_original_definition" type="DATE_TYPE_Type" minOccurs="0">
    <xs:annotation>
<xs:appinfo>.TIME_STAMPS.DATE_OF_ORIGINAL_DEFINITION</xs:appinfo>
    </xs:annotation>
</xs:element>
    …
</xs:complexType>
```

Let's now consider this OntoML fragment:

```
<class xsi:type="…" id="…">
…
    <revision>1</revision>
…
</class>
```

The mapping of the OntoML revision value to the corresponding complete EXPRESS target path is implicitly defined as follows:

SELF.TIME_STAMPS.DATE_OF_ORIGINAL_DEFINITION := 1

F.4.3.4.2 Assigning an OntoML value to a complex EXPRESS attribute

Some information elements are not represented in a straightforward manner in OntoML according to the CIIM. The mapping can not be expressed defining only simple EXPRESS target paths and assigning simple XML element values. It requires to specify mapping functions. Their role is to retrieve information from the OntoML document, to process it, and to assign an instance value to the attribute referenced by the complete EXPRESS target path. Their corresponding algorithm may be more or less complex.

NOTE The mapping does not define the algorithm of each of these mapping functions, but only their signature and their behavior.

A complex EXPRESS attribute is an attribute whose data type is an entity datatype. The assignment of an OntoML value to such an attribute requires to build a type-compatible instance. The general structure of such an assignment is defined as follows:

EXPRESS target path := <function_name>({parameters})

Where

— *EXPRESS target path*: a local target path assigned to an XML element, the targeted attribute datatype being an EXPRESS entity;

— <*function_name*>: a mapping function whose role is to create a set of EXPRESS instances and to assign one of them to the attribute referenced in the EXPRESS target path;

— {*parameters*}: the set of effective parameters corresponding to XML items (element or attribute) value retrieved (using XPath operator) from the OntoML document instance.

EXAMPLE The preferred name of a class ontology concept is represented by an EXPRESS attribute called preferred_name whose data type is the **translatable_label** entity. It is represented in OntoML by an XML element (**preferred_name**) whose content model is not represented using the same constructs for simplification purposes. Consequently, the mapping can not be represented simply using an EXPRESS target path. A specific mapping function is used:

```
<xs:element name="class" type="CLASS_Type" maxOccurs="unbounded">
    <xs:annotation>
        <xs:appinfo>SELF</xs:appinfo>
    </xs:annotation>
</xs:element>
<xs:complexType name="CLASS_Type" abstract="true">
    <xs:sequence>
    …
```

```
                <xs:element name="preferred_name" type="PREFERRED_NAME_Type">
                    <xs:annotation>
                        <xs:appinfo>.NAMES.PREFERRED_NAME :=
                            createLabel(*)</xs:appinfo>
                    </xs:annotation>
                </xs:element>
            …
        </xs:complexType>
```

Let's now consider this OntoML fragment:

```
<class xsi:type="…" id="…">
…
    <preferred_name>
        <label language="en">bearing</label>
    </preferred_name>
…
</class>
```

The mapping of the OntoML preferred name complex value to the corresponding complete EXPRESS target path is implicitly defined as follows:

SELF.NAMES.PREFERRED_NAME := createLabel(*)

The EXPRESS **preferred_name** attribute value is intended to be built using the **createLabel** mapping function. This function takes as an effective parameter a set of nodes referenced by the "*" XPath (the set of children nodes of the contextual XML element, i.e., the nodes children of the **preferred_name** XML element), and process (create required EXPRESS instances) it according to the CIIM. It returns a type compatible EXPRESS **preferred_name** attribute value (a **translated_label** entity instance) that is assigned to the specified complete EXPRESS target path.

The following subclauses specify these mapping functions.

F.4.3.4.2.1 BSU from a CIIM ontology concept identifier mapping

Every CIIM ontology concept is associated to an unambiguous identifier whose structure is defined in this part of ISO 13584.

In OntoML, these identifiers are represented by a string, whereas they are structurally and descriptively specified in the CIIM. Thus, we define a function that is intended to build the EXPRESS entity data type instance resources for representing a CIIM ontology concept identifier from an identifier represented by a string. Its signature is the following:

<ontologyConcept>BSUFromId(ontoMLId: string): basic_semantic_unit

where:

— *<ontologyConcept>*: the specific CIIM ontology concept for which the CIIM identifier is built. It may take the following values:

  — supplier ontology concept: "supplier";

  — class ontology concept: "class";

  — property ontology concept: "property";

  — datatype ontology concept: "datatype";

  — document ontology concept: "document";

— *ontoMLId*: a string representing an OntoML concept identifier;

NOTE 1 The **ontoMLId** effective value is intended to be retrieved from an OntoML document instance using XPath localization path.

— *basic_semantic_unit*: the instance type returned by this function call.

NOTE 2 **basic_semantic_unit** is defined in ISO 13584-42:2010, Clause F.3.4.2.1.

Depending on the <ontologyconcept>, the **<ontologyConcept>BSUFromId** function returns an instance of:

— the **supplier_BSU** entity data type for the supplier ontology concept;

— the **class_BSU** entity data type for the class ontology concept;

— the **property_BSU** entity data type for the property ontology concept;

— the **datatype_BSU** entity data type for the datatype ontology concept;

— the **document_BSU** entity data type for the document ontology concept.

If the OntoML identifier has already been mapped in an instance of one of these CIIM entity datatype, it shall not be created twice, but its corresponding CIIM entity instance shall be retrieved and returned by the function.

Additionally, depending on the CIIM ontology concept identified, the following apply:

— class ontology concept: if the identified supplier in the OntoML class identifier has not already been mapped into an EXPRESS entity instance, it shall be created and then referenced, otherwise, the existing EXPRESS entity instance shall be only referenced;

— property, datatype or document ontology concept: if the identified supplier and the identified class in the OntoML property, datatype or document identifier have not already been mapped into EXPRESS entity instances, they shall be created and then referenced, otherwise, the existing EXPRESS entity instances shall be only referenced.

Table F.3 lists OntoML CIIM ontology concept identifiers (see 9.1) and their corresponding CIIM EXPRESS instances.

Table F.3 — OntoML identifiers mapping

OntoML identifiers	EXPRESS instances
`SupplierId ::= icd oi [opi [opis]] [std]`	`#supp=SUPPLIER_BSU(CIIMrai, *);` *CIIMrai* is built from *supplierId* according to ISO 13584-26 rules.
`classId ::= rai # di #vi`	`#cl=CLASS_BSU(di, vi, #supp);` *#supp* is a reference to an instance of a **supplier_BSU** identified in the *rai* part of the *dictionaryId* identifier
`propertyId ::= rai # di # vi`	`#prop=PROPERTY_BSU(diProp, vi, #cl);` *diProp* is the CIIM property code identified in the *di* part of the OntoML *propertyId*. *#cl* is a reference to an instance of a **class_BSU** identified in the *rai* and *di* part of the OntoML *propertyId*.

OntoML identifiers	EXPRESS instances
`documentId ::= rai # di # vi`	`#doc=DOCUMENT_BSU(diDoc, vi, #cl);` *diDoc* is the CIIM document code identified in the *di* part of the OntoML *documentId*. *#cl* is a reference to an instance of a **class_BSU** identified in the *rai* and *di* part of the OntoML *documentId*.
`datatypeId ::= rai # di # vi`	`#type=DATA_TYPE_BSU(diType, vi, #cl);` *diType* is the CIIM data type code identified in the *di* part of the OntoML *datatypeId*. *#cl* is a reference to an instance of a **class_BSU** identified in the *rai* and *di* part of the OntoML *datatypeId*.

EXAMPLE The class ontology concept identifier is represented as follows:

```
<xs:element name="class" type="CLASS_Type" maxOccurs="unbounded">
    <xs:annotation>
        <xs:appinfo>SELF</xs:appinfo>
    </xs:annotation>
</xs:element>
<xs:complexType name="CLASS_Type" abstract="true">
    …
    <xs:attribute name="id" type="ClassId" use="required">
        <xs:annotation>
            <xs:appinfo>.IDENTIFIED_BY := classBSUFromId(string(@id))</xs:appinfo>
        </xs:annotation>
    </xs:attribute>
</xs:complexType>
```

The complete EXPRESS target path defined for the **id** (the class ontology concept identifier) XML attribute and its corresponding assignment is therefore:

SELF.IDENTIFIED_BY := classBSUFromId(string(@id))

It means that the **identified_by** EXPRESS attribute of the **SELF** instance (an instance representing the class ontology concept) is set to the value returned by the **classBSUFromId** function. This function takes as an argument the result of the defined XPath (“string(@id)”), i.e., the OntoML **id** attribute value.

Let’s now consider this OntoML fragment:

```
<class … id="0002-38491502100024#BEARING#001">
…
</class>
```

If we assume that a **supplier_BSU** entity instance has already been created (*#supp*), the mapping would look like:

```
#cl = CLASS_BSU('BEARING', '001', #supp);
SELF.IDENTIFIED_BY := #cl
```

Where

— *#cl*: an EXPRESS **class_BSU** instance identifier;

— *#supp*: an EXPRESS **supplier_BSU** instance identifier (assumed to exist).

F.4.3.4.2.2 Class BSUs from class identifiers mapping

The CIIM EXPRESS model requires to reference every defined or referenced classes from the **dictionary** EXPRESS entity, through its **contained_classes** attribute. The **classBSUsFromIds** is used for that purpose. It retrieves all the OntoML class identifiers and returns their corresponding CIIM representation (**class_BSU** instances). Its signature is the following:

classBSUsFromIds(ontoMLIds: XPath): LIST OF UNIQUE class_BSU

where:

— *ontoMLIds*: an XPath allowing to retrieve all the OntoML class identifiers;

NOTE 1 In OntoML, such an identifier is represented by an XML attribute.

— *class_BSU*: the instance type returned by this function call.

NOTE 2 **Class_BSU** is defined in ISO 13584-42:2010, Clause F.3.6.1.1.

NOTE 3 The **classBSUsFromIds** is used only once, in the mapping specification of the **contained_classes** XML element defined in the **DICTIONARY_Type** XML complex type.

Table F.4 presents an application example of the **classBSUsFromIds** function, assuming that this mapping function is defined in the context of an OntoML **contained_classes** XML element, as follows: `.CONTAINED_CLASSES := classBSUsFromIds(*/@id).`

Table F.4 — OntoML list of class identifiers mapping

OntoML	EXPRESS instances

`<contained_classes>`
`<class id=”rai#di1#vi” …>`
`… </class>`
`<class id=” rai#di2#vi” …>`
`… </class>`
`…`
`<class id=” rai#din#vi” …>`
`… </class>`
`</contained_classes>`
	`#cl1=CLASS_BSU(di1, vi, #supp);`
	`#cl2=CLASS_BSU(di2, vi, #supp);`
	`…`
	`#cln=CLASS_BSU(din, vi, #supp);`
	#supp is a reference to an instance of a **supplier_BSU** identified in the *rai* part of the *dictionaryId* identifier
	
	classBSUsFromIds function result:
	`[#cl1, #cl2, …, #cln]`

F.4.3.4.2.3 Dictionary and library identification mapping

The **dictionaryCodeFromId** function allows to build an EXPRESS **dictionary_identification** entity instance form an OntoML dictionary identifier. Its signature is the following:

DictionaryCodeFromId(ontoMLDicLibId: string): dictionary_identification

Where:

— *ontoMLDicLibId*: an XPath allowing to access the OntoML dictionary and / or library identifier;

NOTE 1 In OntoML, such an identifier is represented by an XML attribute.

— *dictionary_identifier*: the instance type returned by this function call.

NOTE 2 **dictionary_identification** is defined in ISO 13584-24:2003, Clause 11.5.

Table F.5 presents the dictionary and library identifier (see 9.1) and its corresponding CIIM EXPRESS representation.

Table F.5 — OntoML ontology identifier mapping

OntoML	EXPRESS instances
`ontologyId ::= rai # di #vi`	`#dic=DICTIONARY_IDENTIFICATION(di, vi, revision, #supp);` *#supp* is a reference to an instance of a **supplier_BSU** identified in the *rai* part of the *ontologyId* identifier. The *revision* attribute is not mapped by this function.

F.4.3.4.2.4 Label and translated label mapping

The **createLabel** function allows to build the CIIM EXPRESS resources corresponding to some clear label information, possibly translated. Its signature is the following:

createLabel(ontoMLLabel: XPath): translatable_label

Where:

— *ontoMLLabel*: an XPath that references a set of OntoML **label** XML element (possibly associated to a **language** XML attribute) intended to be processed.

— *translatable_label*: the general data type returned by this function call. In case of not translated text, it returns a CIIM **label_type** value. In case of a translated text, it returns a CIIM **translated_label** entity instance value.

NOTE **translatable_label** is defined in ISO 13584-42:2010, Clause F.4.1.4.

Table F.6 presents labels and translated labels and their corresponding CIIM EXPRESS representation.

Table F.6 — OntoML label and translated label mapping

OntoML	EXPRESS instances

`<…>`
`  <label> a label </label>`
`</…>`
	`LABEL('a label')`
	The CIIM representation of a non translated label is a string whose specific data type is **LABEL**.

`<…>`
`  <label language="en"> a label </label>`
`  <label language="fr"> un label </label>`
`</…>`
	`#tlabel=TRANSLATED_LABEL(('a label', 'un label'), #pt);`
	`#pt=PRESENT_TRANSLATIONS((#lc1, #lc2));`
	`#lc1=LANGUAGE_CODE('en', $);`
	`#lc2=LANGUAGE_CODE('fr', $);`
	The CIIM representation of a translated label consists of instances of the following three CIIM EXPRESS entity data type: **translated_label**, **present_translations** and **language_code**.

EXAMPLE The class ontology concept preferred name is represented as follows:

```
<xs:element name="class" type="CLASS_Type" maxOccurs="unbounded">
        <xs:annotation>
                <xs:appinfo>SELF</xs:appinfo>
        </xs:annotation>
</xs:element>
<xs:complexType name="CLASS_Type" abstract="true">
        …
        <xs:element name="preferred_name" type="PREFERRED_NAME_Type">
                <xs:annotation>
                        <xs:appinfo>.NAMES.PREFERRED_NAME := createLabel(*)</xs:appinfo>
                </xs:annotation>
        </xs:element>
        …
</xs:complexType>
<xs:complexType name="PREFERRED_NAME_Type">
        <xs:sequence>
                <xs:element name="label" type="PREFERRED_NAME_LABEL_Type"
                        maxOccurs="unbounded"/>
        </xs:sequence>
</xs:complexType>
<xs:complexType name="PREFERRED_NAME_LABEL_Type">
        <xs:simpleContent>
                <xs:extension base="PREFERRED_NAME_TYPE_Type">
                        <xs:attribute name="language" type="LANGUAGE_CODE_Type"
                                use="optional"/>
                </xs:extension>
        </xs:simpleContent>
</xs:complexType>
```

The complete EXPRESS target path defined for the **preferred_name** (the class preferred name) XML element and its corresponding assignment is therefore:

SELF.NAMES.PREFERRED_NAME := createLabel(*)

Let's now consider this OntoML fragment:

```
<class …>
…
        <preferred_name>
                <label language="en">roller bearing</label>
        </preferred_name>
…
</class>
```

The result of the expressed mapping could be as follows:

```
#tlabel = TRANSLATED_LABEL(('roller bearing'), #pt);
#pt = PRESENT_TRANSLATIONS((#lc));
#lc = LANGUAGE_CODE('en', $);
SELF.NAMES.PREFERRED_NAME := #tlabel
```

F.4.3.4.2.5 Text and translated text mapping

The **createText** function allows to build the CIIM EXPRESS resources corresponding to some clear text information, possibly translated. Its signature is the following:

createText(ontoMLText: XPath): translatable_text

Where:

— *ontoMLText*: an XPath that references a set of OntoML **text** XML element (possibly associated to a **language** XML attribute) intended to be processed.

— *translatable_Text*: the general data type returned by this function call. In case of not translated text, it returns a CIIM **text_type** value. In case of a translated text, it returns a CIIM **translated_text** entity instance value.

NOTE **translatable_text** is defined in ISO 13584-42:2010, Clause F.4.1.6.

Table F.7 presents texts and translated texts and their corresponding CIIM EXPRESS representation.

Table F.7 — OntoML text and translated text mapping

OntoML	EXPRESS instances
<…> <text> a text </text> </…>	TEXT('a label') The CIIM representation of a non translated label is a string whose specific data type is **LABEL**.
<…> <text language="en"> a text </label> <text language="fr"> un texte </text> </…>	#tlabel=TRANSLATED_TEXT(('a text, 'un texte'), #pt); #pt=PRESENT_TRANSLATIONS((#lc1, #lc2)); #lc1=LANGUAGE_CODE('en', $); #lc2=LANGUAGE_CODE('fr', $); The CIIM representation of a translated text consists of instances of the following three CIIM EXPRESS entity data type: **translated_text**, **present_translations** and **language_code**.

F.4.3.4.2.6 Synonymous names mapping

The **createSynonymous** function allows to build the CIIM EXPRESS resources corresponding to some synonymous labels information, possibly translated. Its signature is the following:

createSynonymous(ontoMLLabel: XPath): SET OF syn_name_type

Where:

— *ontoMLLabel*: an XPath that references a set of OntoML **label** XML element (possibly associated to a **language** XML attribute) intended to be processed.

— *syn_name_type*: the general data type returned by this function call. In case of not translated label, the function returns a set of CIIM **label** type value. In case of a translated label, it returns a set of CIIM **label_with_language** entity instance value.

NOTE **syn_name_type** is defined in ISO 13584-42:2010, Clause F.3.9.1.16.

Table F.8 presents synonymous names and translated synonymous names and their corresponding CIIM EXPRESS representation.

Table F.8 — OntoML synonymous and translated synonymous mapping

OntoML	EXPRESS instances
<…> <label> a synonymous </label> <label> another synonymous </label> </…>	[LABEL('a synonymous'), LABEL('another synonymous')] The CIIM representation of a non translated synonymous names is a set of strings whose specific data type is **LABEL**.
<…> <label language="en"> a synonymous </label> <label language="fr"> un synonyme </label> <label language="fr"> un autre synonyme </label> </…>	#syn1=LABEL_WITH_LANGUAGE('a synonymous', #lc1); #syn2=LABEL_WITH_LANGUAGE('un synonyme', #lc2); #syn3=LABEL_WITH_LANGUAGE('un autre synonyme', #lc2); #lc1=LANGUAGE_CODE('en', $); #lc2=LANGUAGE_CODE('fr', $); The CIIM representation of a translated synonymous label consists of instances of the following two CIIM EXPRESS entity data: **label_with_language** and **language_code**.

F.4.3.4.2.7 Keywords mapping

The **createKeywords** function allows to build the CIIM EXPRESS resources corresponding to some keyword labels information, possibly translated. Its signature is the following:

createKeywords(ontoMLLabel: XPath): SET OF keyword_type

Where:

— *ontoMLLabel*: an XPath that references a set of OntoML **label** XML element (possibly associated to a **language** XML attribute) intended to be processed.

— *keyword_type*: the general data type returned by this function call. In case of not translated label, the function returns a set of CIIM **label** type value. In case of a translated label, it returns a set of CIIM **label_with_language** entity instance value.

NOTE **keyword_type** is defined in ISO 13584-42:2010, Clause F.3.9.1.17.

Table F.9 presents keywords and translated keywords and their corresponding CIIM EXPRESS representation.

Table F.9 — OntoML keywords and translated keywords mapping

OntoML	EXPRESS instances
<…> <label> a keyword </label> <label> another keyword </label> </…>	[LABEL('a keyword'), LABEL('another keyword')] The CIIM representation of a non translated keywords is a set of strings whose specific data type is **LABEL**.
<…> <label language="en"> a keyword </label> <label language="fr"> un mot clé </label> <label language="fr"> un autre mot clé </label> </…>	#syn1=LABEL_WITH_LANGUAGE('a keyword', 'en'); #syn2=LABEL_WITH_LANGUAGE('un mot clé', 'fr'); #syn3=LABEL_WITH_LANGUAGE('un autre autre mot clé', 'fr'); #lc1=LANGUAGE_CODE('en', $); #lc2=LANGUAGE_CODE('fr', $); The CIIM representation of a translated keyword consists consists of instances of the following two CIIM EXPRESS entity data: **label_with_language** and **language_code**.

F.4.3.4.2.8 Documentation mapping

The association of documentation to CIIM ontology concepts may be done in three different ways.

— either by referencing a document represented as an instance of a CIIM ontology concept: for that purpose, the **referenced_graphics** and **referenced_document** CIIM EXPRESS entities are used;

— or by referencing an well-identified document: for that purpose, the **identified_document** CIIM EXPRESS entity is used;

— or by referencing an http resource: for that purpose, the **external_graphics**, **document_content**, **message**, **illustration**, **a6_illustration** and the **a9_illustration** CIIM EXPRESS entities are used;

For this latter functions group, protocol and translation information shall be provided according to the CIIM.

These CIIM EXPRESS resources may be classified, according to the CIIM, as follows:

— graphics: the EXPRESS resources that represent a CIMM **graphics**: **referenced_graphics** and **external_graphics**;

— documents: the EXPRESS resources that represent a CIMM **document**: **referenced_document** and **identified_document**;

— document ontology concept content specification: the EXPRESS resource that represents a CIIM **document_content**;

— class extension external resources: the EXPRESS resources that represent a CIMM **class_extension_external_item**: **message**, **illustration**, **a6_uillustration** and **a9_illustration**.

Mapping functions are defined according to this classification:

— a mapping function intended to create a graphics:

createGraphics(ontoMLGraphicsType: XPath): graphics

where:

— *ontoMLGraphicsType*: an XPath that references the *xsi:type* attribute of the element to which this function is applied; it allows to specify what kind of graphics is intended to be built;

— *graphics*: the entity data type of the instance returned by this function call; if the function applies to an XML element whose content model is defined as a referenced graphics (**xsi:type = ontoml:REFERENCED_GRAPHICS_Type**), it returns a CIIM **referenced_graphics** entity instance value; if it applies to an XML element whose content model is defined as a an external graphics (**xsi:type = ontoml:EXTERNAL_GRAPHICS_Type**), it returns a CIIM **external_graphics** entity instance value;

— a mapping function intended to create a document:

createDocument(ontoMLDocumentType: XPath): document

where:

— *ontoMLDocumentType*: an XPath that references the *xsi:type* attribute of the element to which this function is applied; it allows to specify what kind of document is intended to be built;

— *document*: the entity data type of the instance returned by this function call; if the function applies to an XML element whose content model is defined as a referenced document (**xsi:type = ontoml:REFERENCED_DOCUMENT_Type**), it returns a CIIM **referenced_document** entity instance value; if it applies to an XML element whose content model is defined as a identified document (**xsi:type = ontoml:IDENTIFIED_DOCUMENT_Type**), it returns a CIIM **identified_document** entity instance value;

— a mapping function intended to create a document ontology concept content specification:

createDocumentContent(doc: string): document_content

where:

— *doc*: an XPath retrieving the OntoML document identifier for which the document content is specified;

— *document_content*: the entity data type of the instance returned by this function call;

— a mapping function intended to create class extension external resources:

createExtResource(ontoMLExtResourceType: XPath): class_extension_external_item

where:

— *ontoMLExtResourceType*: an XPath that references the *xsi:type* attribute of the element to which this function is applied; it allows to specify what kind of document is intended to be built;

— *class_extension_external_item*: the entity data type of the instance returned by this function call; if the function applies to an XML element whose content model is defined as a message (**xsi:type = MESSAGE_Type**), it returns a CIIM **message** entity instance value; if it applies to an XML element whose content model is defined as an illustration (**xsi:type = ILLUSTRATION_Type**) where no **standard_size** XML attribute is defined, it returns a CIIM **illustration** entity instance value; if it applies to an XML element whose content model is defined as an illustration (**xsi:type = ILLUSTRATION_Type**) where the **standard_size** XML attribute is defined, it returns a CIIM **a6_illustration** entity instance value if this attribute value is equal to "**a6_illustration**", or a CIIM **a9_illustration** entity instance value if this attribute value is equal to "**a9_illustration**".

These four functions are intended to process the OntoML sub document part that consists of the sub XML elements tree whose root is the OntoML element where one of those mapping functions is called.

Mapping functions building **external_graphics**, **document_content**, **message**, **illustration**, **a6_illustration** and the **a9_illustration** CIIM entity data type instances deal with reference to http resources. According to the CIIM, it is required for each of these functions to build the following entity instances corresponding to the representation of the HTTP protocol. It is presented in Table F.10.

Table F.10 — OntoML HTTP protocol mapping

OntoML	EXPRESS instances

No explicit information in OntoML.	`#http_prot_name=ITEM_NAMES(LABEL('Hypertext Transfer Protocol'), (), LABEL('HTTP/1.1'), $, $);`
	`#org=ORGANIZATION('', 'World Wide Web Consortium', 'W3C');`
	`#http_protocol=HTTP_PROTOCOL(#org, 'USA', 'NONE', '001', $, #http_prot_name, $, 2621);`
	
	The **http_protocol** instance is intended to be referenced.

Mapping functions building **external_graphics**, **document_content**, **message**, **illustration**, **a6_illustration** and the **a9_illustration** CIIM entity data type instances deal also with translations. According to the CIIM, it is required for each of these functions to build the following entity instances corresponding to the representation of external resource translations. It is represented in Table F.11.

Table F.11 — OntoML translated and not translated files mapping

OntoML	EXPRESS instances

`<file>`
`  <file_name>filename.ext</file_name>`
`  <dir_name>directory name</dir_name>`
`</file>`
 	
dir_name is an optional OntoML element.
	`#ext_cont=NOT_TRANSLATED_EXTERNAL_CONTENT((#lang_spec_cont));`
	`#lang_spec_cont=LANGUAGE_SPECIFIC_CONTENT((#http_file), #http_file, 'utf-8');`
	`#http_file=HTTP_FILE('filename.ext', $, mime_type, mime_subtype, $, 'filename.ext', # http_class_dir, $);`
	
	mime_type and *mime_subtype* are intended to be computed from the file name.
	
	`#http_class_dir=HTTP_CLASS_DIRECTORY('directory_name', #class);`
	
	If the directory name is not provided, it shall be created and managed by the mapping program.

OntoML	EXPRESS instances
<file language="en"> <file_name>filename_en.ext </file_name> <dir_name>directory_name_1 </dir_name> </file> <file language="fr"> <file_name>filename_fr.ext </file_name> <dir_name>directory_name_2 </dir_name> </file> **dir_name** is an optional OntoML element.	#pt=PRESENT_TRANSLATIONS((#lc1, #lc2)); #lc1=LANGUAGE_CODE('en', $); #lc2=LANGUAGE_CODE('fr', $); #lang_spec_cont_1=LANGUAGE_SPECIFIC_CONTENT((#http_file_1), #http_file_1, "utf-8"); #lang_spec_cont_2=LANGUAGE_SPECIFIC_CONTENT((#http_file_2), #http_file_2, 'utf-8'); #http_file_1=HTTP_FILE('***filename_en.ext***', $, mime_type, mime_subtype, $, 'filename_en.ext', # http_class_dir_1, $); #http_file_2=HTTP_FILE('***filename_fr.ext***', $, mime_type, mime_subtype, $, 'filename_fr.ext', # http_class_dir_2, $); *mime_type* and *mime_subtype* are intended to be computed from the file name. #http_class_dir_1=HTTP_CLASS_DIRECTORY("directory_name_1", #class); #http_class_dir_2=HTTP_CLASS_DIRECTORY("directory_name_2", #class); If the directory name is not provided, it shall be created and managed by the mapping program.

Table F.12 presents for each OntoML documentation constructs the corresponding CIIM EXPRESS representation. When required, they reference CIIM EXPRESS entity instances previously introduced.

Table F.12 — OntoML external resource mapping

OntoML	EXPRESS instances

< … xsi:type=
  "**REFERENCED_GRAPHICS_Type**">
  <graphics_reference
  document_ref="documentId"/>
<…/>
	#ref_graph=REFERENCED_GRAPHICS(#doc);
	
	#doc is built from the OntoML *documentId*, according to F.4.3.4.2.1.
	
	The **createGraphics** function would return the *#ref_graph* entity instance.

< … xsi:type=
  "**REFERENCED_DOCUMENT_Type**">
  <document_reference
  document_ref="documentId"/>
<…/>
	#ref_doc=REFERENCED_DOCUMENT(#doc);
	
	#doc is built from the OntoML *documentId*, according to F.4.3.4.2.1.
	
	The **createDocument** function would return the *#ref_doc* entity instance.

OntoML	EXPRESS instances
< … xsi:type= "**IDENTIFIED_DOCUMENT_Type**"> <document_identifier> anIdentifier </document_identifier> <…/>	#doc_id=IDENTIFIED_DOCUMENT(<anIdentifier>) The **createDocument** function would return the *#doc_id* entity instance.
< … xsi:type= "**EXTERNAL_GRAPHICS_Type**"> <file> ... </file> ... <file> ... </file> <…/>	#ext_graph=EXTERNAL_GRAPHICS(#graph_files); #graph_files=GRAPHIC_FILES(#http_protocol, #ext_cont); *#http_protocol* specifies the HTTP protocol *#ext_cont* specifies the possible translations. The **createGraphics** function would return the *#ext_graph* entity instance.
< … xsi:type= "**DOCUMENT_CONTENT_Type**"> <file> ... </file> ... <file> ... </file> <revision>*rev*</revision> <…/>	#doc_cont=DOCUMENT_CONTENT(#doc, #http_protocol, #ext_cont, rev); *#doc represents an instance of the document ontology concept identifier (see F.4.3.4.2.1)* *#http_protocol* specifies the HTTP protocol *#ext_cont* specifies the possible translations. The **createDocumentContent** function would return the *#doc_cont* entity instance.
< … xsi:type= "**MESSAGE_Type**"> <file> ... </file> ... <file> ... </file> <code>a code</code> <…/>	#mess=MESSAGE(#http_protocol, #ext_cont, <aCode>); *#http_protocol* specifies the HTTP protocol *#ext_content* specifies the possible translations. The **createExtResource** function would return the *#mess* entity instance.

OntoML	EXPRESS instances
< … xsi:type= "**ILLUSTRATION_Type**"> <file> ... </file> ... <code>aCode</code> <kind_of_content> *aKind*</kind_of_content> <width>*aWidth*</width> <height>aHeight</height> <…/>	#ill=ILLUSTRATION(#http_protocol, #ext_cont, <aCode>, <aKind>, #width, #height); #width=LENGTH_MEASURE_WITH_UNIT(LENGTH_MEASURE(<aWidth>), #lu_width); #lu_width=(LENGTH_UNIT()SI_UNIT(.MILLI., .METRE.)); #height=LENGTH_MEASURE_WITH_UNIT(LENGTH_MEASURE(<aHeighth>), #lu_height); #lu_height=(LENGTH_UNIT()SI_UNIT(.MILLI., .METRE.)); *#http_protocol* specifies the HTTP protocol *#ext_content* specifies the possible translations. The **createExtResource** function would return the *#ill* entity instance.
< … xsi:type= "**ILLUSTRATION_Type**" standard_size="**a6_illustration**"> <file> ... </file> ... <code>aCode</code> <kind_of_content> *aKind*</kind_of_content> <width>*aWidth*</width> <height>aHeight</height> <…/>	#a6ill=A6_ILLUSTRATION(#http_protocol, #ext_cont, <aCode>, <aKind>, #width, #height); *#http_protocol* specifies the HTTP protocol *#ext_content* specifies the possible translations. *#width* and *#height* are defined above. The **createExtResource** function would return the *#a6ill* entity instance.
< … xsi:type= "**ILLUSTRATION_Type**" standard_size="**a9_illustration**"> <file> ... </file> ... <code>aCode</code> <kind_of_content> *aKind*</kind_of_content> <width>*aWidth*</width> <height>aHeight</height> <…/>	#a9ill=A9_ILLUSTRATION(#http_protocol, #ext_cont, <aCode>, <aKind>, #width, #height); *#http_protocol* specifies the HTTP protocol *#ext_content* specifies the possible translations. *#width* and *#height* are defined above. The **createExtResource** function would return the *#a9ill* entity instance.

F.4.3.4.2.9 *A posteriori* case of relationship mapping

The OntoML representation of *a posteriori* semantic relationship is not the same than in the CIIM EXPRESS model. Consequently, the mapping may only be expressed using a mapping function. The signature of this mapping function is given below:

createAPosteriori(ontoMLAposteriori: XPath): a_posteriori_semantic_relationship

where:

— *ontoMLAposteriori*: an XPath that references the **xsi:type** attribute of the element to which this function is applied; it allows to specify what kind of *a posteriori* semantic relationships is intended to be built;

— a_*posteriori_semantic_relationship*: the entity data type of the instance returned by this function call; if the function applies to an XML element whose content model is defined as an *a posteriori* case of semantic relationship (**xsi:type = ontoml:A_POSTERIORI_CASE_OF_Type**), it returns a CIIM **a_posteriori_case_of** entity instance value; if it applies to an XML element whose content model is defined as an *a posteriori* case of semantic relationship (**xsi:type = ontoml:A_POSTERIORI_VIEW_OF_Type**) it returns a CIIM **a_posteriori_view_of** entity instance value.

Table F.13 presents *a posteriori* case of OntoML representations and its corresponding CIIM EXPRESS representation.

Table F.13 — OntoML *a posteriori* case-of relationship mapping

OntoML	EXPRESS instances
< … xsi:type= "**A_POSTERIORI_CASE_OF_Type**"> <source class_ref="classId1"/> <is_case_of class_ref="classId2"/> <corresponding_properties> <mapping> <domain> <property property_ref="propId1"> </domain> <range property_ref="propId2"/> </mapping> </corresponding_properties> <…/>	#apcof=A_POSTERIORI_CASE_OF(#cl1, #cl2, ((#prop1, #prop2))); *#cl1* and *#cl2* are respectively built from the OntoML *classId1* and *classId2* identifiers, according to *F.4.3.4.2.1*. *#prop1* and *#prop2* are respectively built from the OntoML *propId1* and *propId2* identifiers, according to *F.4.3.4.2.1*. The **createAPosteriori** function would return the *#apcof* entity instance.

Table F.14 presents *a posteriori* view of OntoML representations and its corresponding CIIM EXPRESS representation.

Table F.14 — OntoML *a posteriori* view-of relationship mapping

OntoML	EXPRESS instances
< … xsi:type= "**A_POSTERIORI_VIEW_OF_Type**"> <functional_model class_ref="classId1"/> <is_view_of class_ref="classId2"/> <corresponding_properties> <mapping> <domain> <property property_ref="propId1"> </domain> <range property_ref="propId2"/> </mapping> </corresponding_properties> <…/>	#apcof=A_POSTERIORI_VIEW_OF(#cl1, #cl2, ((#prop1, #prop2))); *#cl1* and *#cl2* are respectively built from the OntoML *classId1* and *classId2* identifiers, according to *F.4.3.4.2.1*. *#prop1* and *#prop2* are respectively built from the OntoML *propId1* and *propId2* identifiers, according to *F.4.3.4.2.1*. The **createAPosteriori** function would return the *#apcof* entity instance.

F.4.3.4.2.10 Instances mapping

OntoML does not define any structure for representing values and instances, but it uses the resources defined in the ISO/TS 29002-10 product exchange format.

The property values and class instances mapping is outside the scope of OntoML Nevertheless, two abstract mapping functions are provided for consistency purposes.

The **createValue** function allows to map values defined in the common product exchange format to the corresponding CIIM entity instance(s). Its signature is the following:

createValue(OntoMLValues: XPath): LIST OF primitive_value

Where:

— *ontoMLValues*: an XPath that references a set of **value** XML elements intended to be processed.

NOTE 1 The **value** XML element is defined in the ISO/TS 29002-10 product exchange format.

— *property_value*: the general data type returned by this function call that represents a CIIM compliant representation of the value.

NOTE 2 **primitive_value** is defined in ISO 13584-24:2003, Clause 6.3.2.

The **createPopulation** function allows to map instances defined in this common format to the corresponding CIIM entity instance(s). Its signature is the following:

createPopulation(OntoMLInstances: XPath): LIST OF UNIQUE dic_class_intance

— *ontoMLInstances*: an XPath that references a set of **item** XML elements intended to be processed.

NOTE 3 The **item** XML element is defined in the ISO/TS 29002-10 product exchange format.

— *dic_class_instance*: the general data type returned by this function call that represents a CIIM compliant representation of the instance.

NOTE 4 **dic_class_instance** is defined in ISO 13584-24:2003, Clause 6.4.7.1.

F.4.4 OntoML resource mapping for un-referenced CIIM EXPRESS items

In OntoML, some un-referenced CIIM EXPRESS resources are represented. The mapping can therefore not be expressed in a common way, on the base of some CIIM ontology concepts or on the base of the general dictionary structure.

For that purpose, in place of defining functions whose result is intended to be assigned to an EXPRESS target path, procedures are specified.

F.4.4.1 Global Ontology language

The **create_global_language_assignment** procedure is used to create a CIIM **GLOBAL_LANGUAGE_ASSIGNMENT** EXPRESS entity data type instance from a specified language. Its signature is the following:

create_global_language_assignment(language_id : XPath, country_id : XPath)

where:

— *language_id*: an XPath retrieving the OntoML general language attribute value.

— *country_id*: an XPath retrieving the OntoML general country attribute value.

Table F.15 gives an example of applying this procedure (it is assumed that this mapping procedure is assigned to the XML language attribute of the ontoml XML element: **create_global_language_assignment(@language_code, @country_code)**).

Table F.15 — OntoML global language mapping

OntoML	EXPRESS instances
<ontoml> <global_language language_code="en"/> … </ontoml>	#gla=GLOBAL_LANGUAGE_ASSIGNMENT('en', $);

F.5 OntoML complex types and CIIM entity datatype correspondence table

Table F.16 lists the correspondence between the whole set of concrete OntoML complex types and the CIIM EXPRESS constructs (entities or data types).

Table F.16 — OntoML complex types / CIIM entity datatypes correspondence

OntoML complex type	CIIM EXPRESS entity datatype
A_POSTERIORI_CASE_OF_Type	a_posteriori_case_of
A_POSTERIORI_SEMANTIC_RELATIONSHIP_Type	a_posteriori_semantic_relationship
A_POSTERIORI_SEMANTIC_RELATIONSHIPS_Type	LIST OF a_posteriori_semantic_relationship
A_POSTERIORI_VIEW_OF_Type	a_posteriori_view_of
ALTERNATIVE_UNIT_IDS_Type	LIST[1:?] OF dic_unit_identifier
ALTERNATIVE_UNITS_Type	LIST[1:?] OF dic_unit
ANY_TYPE_Type	data_type
ARRAY_TYPE_Type	array_type
AUTHORS_Type	LIST[1:?] OF person
AXIS1_PLACEMENT_TYPE_Type	axis1_placement_type
AXIS2_PLACEMENT_2D_TYPE_Type	axis2_placement_2d_type
AXIS2_PLACEMENT_3D_TYPE_Type	axis2_placement_3d_type
BAG_TYPE_Type	bag_type
BOOLEAN_TYPE_Type	boolean_type
CARDINALITY_CONSTRAINT_Type	cardinality_constraint
CATEGORIZATION_CLASS_Type	categorization_class
CLASS_CONSTANT_VALUES_Type	SET OF class_value_assigment
CLASS_CONSTRAINT_Type	class_constraint
CLASS_EXTENSION_Type	class_extension
CLASS_REFERENCE_TYPE_Type	class_reference_type
CLASS_PRESENTATION_ON_PAPER_Type	LIST OF illustration
CLASS_PRESENTATION_ON_SCREEN_Type	LIST OF illustration
CLASS_Type	class

OntoML complex type	CIIM EXPRESS entity datatype
CLASS_VALUE_ASSIGNMENT_Type	class_value_assignment
CLASSIFICATION_Type	SET OF property_classification
CONDITION_DET_Type	condition_DET
CONFIGURATION_CONTROL_CONSTRAINT_Type	configuration_control_constraint
CONSTRAINT_OR_CONSTRAINT_ID_Type	constraint_or_constraint_id
CONSTRAINT_Type	constraint
CONSTRAINTS_Type	SET OF constraint_or_constraint_id
CONTAINED_CLASS_EXTENSIONS_Type	SET OF class_extension
CONTAINED_CLASSES_Type	SET OF class
CONTAINED_DATATYPES_Type	SET OF data_type_element
CONTAINED_DOCUMENTS_Type	SET OF document_element
CONTAINED_PROPERTIES_Type	SET OF property
CONTAINED_SUPPLIERS_Type	SET OF supllier_element
CONTEXT_DEPENDENT_UNIT_Type	context_dependent_unit
CONTEXT_PARAM_ICON_Type	LIST OF a6_illustration
CONTEXT_PARAMETER_CONSTRAINTS_Type	SET OF property_constraint
CONTEXT_RESTRICTION_CONSTRAINT_Type	context_restriction_constraint
CONVERSION_BASED_UNIT_Type	conversion_based_unit
CORRESPONDING_PROPERTIES_type	SET OF LIST[2:2] OF property_BSU
CREATE_ICON_Type	LIST OF a6_illustration
DATATYPE_Type	data_type_element
DATE_DATA_TYPE_Type	date_data_type
DATE_TIME_DATA_TYPE_Type	date_time_data_type
DEPENDENT_P_DET_Type	dependent_P_DET
DERIVED_UNIT_ELEMENT_Type	derived_unit_element
DERIVED_UNIT_Type	derived_unit
DIC_UNIT_REFERENCE_Type	dic_unit_identifier
DIC_UNITS_REFERENCE_Type	SET[1:?] OF dic_unit_identifier
DIC_UNIT_Type	dic_unit
DIC_VALUE_Type	dic_value
DICTIONARY_IN_STANDARD_FORMAT_Type	dictionary_in_standard_format
DICTIONARY_Type	dictionary
DIMENSIONAL_EXPONENTS_Type	dimensional_exponents
DOCUMENT_CONTENT_Type	document_content
DOCUMENT_IDENTIFIER_NAME_LABEL_Type	source_doc_type
DOCUMENT_IDENTIFIER_Type	document_identifier
DOCUMENT_Type	document
DOMAIN_CONSTRAINT_Type	domain_constraint

OntoML complex type	CIIM EXPRESS entity datatype
ENUMERATION_CONSTRAINT_Type	enumeration_constraint
EXPLICIT_FUNCTIONAL_MODEL_CLASS_EXTENSION_Type	explicit_functional_model_class_extension
EXPLICIT_ITEM_CLASS_EXTENSION_Type	explicit_item_class_extension
EXTERNAL_GRAPHICS_Type	external_graphics
EXTERNAL_RESOURCE_Type	external_content
FILTER_Type	filter
FM_CLASS_VIEW_OF_Type	fm_class_view_of
FUNCTIONAL_MODEL_CLASS_Type	functional_model_class
GENERAL_TEXT_Type	text or translated_text
EXTERNAL_FILES_Type	external_item
GRAPHICS_Type	graphics
HEADER_Type	
HTTP_FILE_Type	http_file
IDENTIFIED_DOCUMENT_Type	identified_document
ILLUSTRATION_Type	illustration
INFORMATION_Type	
INT_CURRENCY_TYPE_Type	int_currency_type
INT_DIC_VALUE_Type	dic_value
INT_MEASURE_TYPE_Type	int_measure_type
INT_TYPE_Type	int_type
INTEGRITY_CONSTRAINT_Type	integrity_constraint
ITEM_CLASS_CASE_OF_Type	item_class_case_of
ITEM_CLASS_Type	item_class
ITS_VALUES_Type	LIST OF dic_value
LANGUAGE_Type	language_code
LEVEL_Type	level
LEVEL_TYPE_Type	level_type
LIBRARY_IIM_IDENTIFICATION_Type	library_iim_identification
LIBRARY_IN_STANDARD_FORMAT_Type	library_in_standard_format
LIBRARY_Type	library
LIST_TYPE_Type	list_type
MAPPING_FUNCTION_Type	
MATHEMATICAL_STRING_Type	mathematical_string
MESSAGE_Type	message
NAMED_TYPE_Type	named_type
NAMED_UNIT_Type	named_unit
NON_DEPENDENT_P_DET_Type	non_dependent_P_DET

OntoML complex type	CIIM EXPRESS entity datatype
NON_INSTANTIABLE_FUNCTIONAL_VIEW_CLASS_Type	non_instanciable_functional_view_class
NON_QUANTITATIVE_CODE_TYPE_Type	non_quantitative_code_type
NON_QUANTITATIVE_INT_TYPE_Type	non_quantitative_int_type
NON_SI_UNIT_Type	non_si_unit
NON_TRANSLATABLE_STRING_TYPE_Type	non_translatable_string_type
NUMBER_TYPE_Type	number_type
ONTOML_Type	
ORGANIZATION_Type	organization
PERSON_Type	person
PLACEMENT_TYPE_Type	placement_type
POSTCONDITION_Type	SET[1:?] OF filter
PRECONDITION_Type	SET OF filter
PREFERRED_NAME_LABEL_Type	label or translated_label
PREFERRED_NAME_Type	translatable_label
PROGRAM_REFERENCE_TYPE_Type	program_reference_type
PROPERTY_CLASSIFICATION_Type	property_classification
PROPERTY_CONSTRAINT_Type	property_constraint
PROPERTY_MAPPING_Type	
PROPERTY_Type	property_DET
PROPERTY_VALUE_RECOMMENDED_PRESENTATION_Type	property_value_recommended_representation
RANGE_CONSTRAINT_Type	range_constraint
REAL_CURRENCY_TYPE_Type	real_currency_type
REAL_MEASURE_TYPE_Type	real_measure_type
REAL_TYPE_Type	real_type
RECOMMENDED_PRESENTATION_Type	SET OF property_value_recommended_representation
REFERENCED_DOCUMENT_Type	referenced_document
REFERENCED_GRAPHICS_Type	referenced_graphics
REMOTE_HTTP_ADDRESS_Type	remote_http_address
REMOTE_LOCATIONS_Type	LIST of absolute_URL_type
REPRESENTATION_CONTEXT_Type	representation_context
REPRESENTATION_P_DET_Type	representation_P_DET
REPRESENTATION_REFERENCE_TYPE_Type	representation_reference_type
SET_TYPE_Type	set_type
SET_WITH_SUBSET_CONSTRAINT_TYPE_Type	set_with_subset_constraint
SHORT_NAME_LABEL_Type	label or translated_label
SHORT_NAME_Type	translatable_label

OntoML complex type	CIIM EXPRESS entity datatype
SI_UNIT_Type	SI_unit
SOURCE_DOCUMENT_Type	document
STRING_DIC_VALUE_Type	dic_value
STRING_PATTERN_CONSTRAINT_Type	string_pattern_constraint
STRING_SIZE_CONSTRAINT_Type	string_size_constraint
STRING_TYPE_Type	string_type
STRINGS_Type	SET OF STRING
SUBCLASS_CONSTRAINT_Type	subclass_constraint
SUBSET_Type	LIST[1:?] OF UNIQUE primitive_value
SUPPLIER_Type	supplier_element
SUPPORTED_VEP_Type	SET OF view_exchange_protocol_identification
SYNONYMOUS_NAME_LABEL_Type	syn_name_type
SYNONYMOUS_NAME_TYPE_Type	SET OF syn_name_type
SYNONYMOUS_SYMBOLS_Type	SET[1:?] OF mathematical_string
TEXT_Type	translatable_text
TIME_DATA_TYPE_Type	time_data_type
TRANSLATABLE_STRING_TYPE_Type	translatable_string_type
TRANSLATION_DATA_Type	translation_data
TRANSLATION_Type	LIST OF translation_data
UNIT_ID_Type	dic_unit_identifier
UNIT_Type	unit
V_C_V_RANGE_Type	SET OF view_control_variable_range
VIEW_CONTROL_VARIABLE_RANGE_Type	view_control_variable_range
VIEW_EXCHANGE_PROTOCOL_IDENTIFICATION_Type	view_exchange_protocol_identification

Annex G
(normative)

OntoML exchange levels

OntoML provides XML markup declarations that enable both simple ontologies and advanced ontologies compliant with the common ISO 13584/IEC 61360 model to be exchanged using XML.

Additionally, OntoML may be used as an exchange format for responses to queries performed using the ISO/TS 29002-20 concept dictionary resolution mechanism. For that purpose, OntoML defines a set of XML global element that allow to exchange OntoML based document instances whose root element is one ot these defined XML global elements.

These XML global elements are the following:

— **ontoml**: the most general XML global element that allows to represent OntoML-based document instances for exchanging either a single ontology, or a single library, or an ontology with its associated library.

NOTE 1 The OntoML general structure is defined in Clause 6.4.

— **supplier**: it allows to represent OntoML-based document instances for exchanging information elements describing a supplier identified by an IRDI.

NOTE 2 The supplier ontology concept is defined in Clause 6.7.1.

NOTE 3 The identification of a supplier ontology concept is defined in Clause 9.1.1.

— **class**: it allows to represent OntoML-based document instances for exchanging information elements describing a class identified by an IRDI.

NOTE 4 The class ontology concept is defined in clauses 6.7.2 and 6.7.3.

NOTE 5 The identification of a class ontology concept is defined in Clause 9.1.3.1.

— **property**: it allows to represent OntoML-based document instances for exchanging information elements describing a property identified by an IRDI.

NOTE 6 The property ontology concept is defined in clauses 6.7.4 and 6.7.5.

NOTE 7 The identification of a property ontology concept is defined in Clause 9.1.3.2.

— **datatype**: it allows to represent OntoML-based document instances for exchanging information elements describing a datatype identified by an IRDI.

NOTE 8 The datatype ontology concept is defined in Clause 6.7.6.

NOTE 9 The identification of a datatype ontology concept is defined in Clause 9.1.3.2.

— **document**: it allows to represent OntoML-based document instances for exchanging information elements describing a document identified by an IRDI.

NOTE 10 The document ontology concept is defined in Clause 6.7.7.

NOTE 11 The identification of a document ontology concept is defined in Clause 9.1.3.2.

— **dic_unit**: it allows to represent OntoML-based document instances for exchanging information elements describing a dictionary unit identified by an IRDI.

NOTE 12 Dictionary unit is defined in Clause 8.4.

NOTE 13 The identification of a dictionary unit is defined in Clause 9.1.3.3.

— **constraint**: it allows to represent OntoML-based document instances for exchanging information elements describing a constraint identified by an IRDI.

NOTE 14 Constraint is defined in Clause 8.5.

NOTE 15 The identification of a constraint is defined in Clause 9.1.3.3.

— **dic_value**: it allows to represent OntoML-based document instances for exchanging information elements describing a dictionary value identified by an IRDI.

NOTE 16 Dictionary value is defined in Clause 8.3.4.

NOTE 17 The identification of a dictionary value is defined in Clause 9.1.3.3.

— **a_posteriori_semantic_relationship**: it allows to represent OntoML-based document instances for exchanging information elements describing an *a posteriori* semantic relationship identified by an IRDI.

NOTE 18 *A posteriori* semantic relationship is defined in Clause 8.6..

NOTE 19 The identification of an *a posteriori* semantic relationship is defined in Clause 9.1.3.3.

Annex H
(normative)

Value format specification

ISO 13584-32 provides a particular syntax to specify the allowed formats for the string and numeric values that may be associated with a property.

EXAMPLE 1 The format NR1 3 allows to specify that only integer values consisting of exactly three digits are allowed.

NOTE 1 No value format is defined for any other **ANY_TYPE_Type**, including **BOOLEAN_TYPE_Type**.

NOTE 2 In ISO 13584-32, to define the format of property values is not mandatory.

The syntax of the allowed formats is defined in this Annex using a subset of the Extended Backus-Naur Form (EBNF) defined in ISO/IEC 14977.

EXAMPLE 2 The syntax of the format NR1 3 are the letters 'NR1' '' '3'.

The meaning of each syntax, that is the characters that may be used to represent a value, cannot be defined using the EBNF. Thus the meaning of each part of the format concerning the characters allowed to represent the value is specified separately for each part of the format.

EXAMPLE 3 The syntax of the format NR1 3 has the following meaning: *NR1* means that only an integer value may be represented. Space means that a fixed number of characters is specified by the format. *3* means that exactly three digits are required.

H.1 Notation

Table H.1 summarizes the subset of the ISO/IEC 14977 EBNF syntactic metalanguage used by ISO 13584-32 to specify value format of properties.

Using these notations, the syntax of the subset of the EBNF metalanguage used by ISO 13584-32 to specify value format of properties is summarized by the following grammar (the meta-identifier character, letter and digit are not detailed):

```
syntax = syntaxrule ,{ syntaxrule } ;
syntaxrule = metaidentifier , '=' , definitionslist , ';' ;
definitionslist = singledefinition , { '|' , singledefinition } ;
singledefinition = term , { ',', term } ;
term = primary, [ '-', primary } ;
primary = optionalsequence | repeatedsequence | groupedsequence |
          metaidentifier | terminal | empty ;
optionalsequence = '[' definitionslist ']' ;
repeatedsequence = '{' definitionslist '}' ;
groupedsequence = '(' definitionslist ')' ;
metaidentifier = letter , { letter ] ;
terminal = "'", (character - "'"),{ character - "'" }, "'"
           | '"', (character - '"'),{ character - '"' }, '"' ;
empty = ;
```

The equal sign '=' indicates a syntax rule. The meta-identifier on the left may be re-written by the combination of the elements on the right. Any spaces appearing between the elements are meaningless unless they appear within a `terminal`. A syntax rule is terminated by a semicolon ';'.

Table H.1 — ISO/IEC 14977 EBNF syntactic metalanguage

Representation	ISO/IEC 10646-1 Character names	Metalanguage symbol and role
' '	apostrophe	First quote symbol: represents language terminals. Terminal shall not contain apostrophe. Example: 'Hello'
" "	quotation mark	Second quote symbol: represents language terminals. Terminal shall not contain quotation mark. Example: "John's car"
( )	left parenthesis, right parenthesis	Start/ end group symbols. The content is considered as a single symbol.
[ ]	left square bracket, right square bracket	Start/ end option symbols. The content may or not be present.
{ }	left curly bracket, right curly bracket	Start/ end repeat symbols. The content may be present 0 to n times.
-	hyphen-minus	Except symbol.
,	comma	concatenate symbol.
=	equals sign	Defining symbol. Syntax rule: defines the symbol of the left by the formula on the right.
|	vertical line	Alternative separator symbol.
;	semicolon	Terminator symbol. End of a syntax rule.

The use of a meta-identifier within a definition-list denotes a non-terminal symbol which appears on the left side of another syntax rule. A meta-identifier is composed of letters or digits, the first character being a letter. If a term contains both a `primary` preceding a minus sign, and a `primary` that follows the minus sign, only the sequence of symbols that are represented by the first `primary` and that are not represented by the second `primary` are represented by the term.

EXAMPLE 1 Notation:

```
"'", character - "'", "'"
```

means any character but the apostrophe character, inserted between two apostrophe characters.

The `terminal` denotes a symbol which cannot be expanded further by a syntax rule, and which will appear in the final result. Two ways are allowed to represent a `terminal`: either a set of characters without apostrophe, inserted between two apostrophes, or a set of characters without quotation marks, inserted between two quotation marks.

EXAMPLE 2 Assume that we want to describe, by such a grammar, the price of a product in €. Such a price is a positive number with no more than 2 digits in the cents part. We introduce three meta-identifiers associated with three syntax rules:

```
digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
cents = [ '.' , digit [ , digit ] ];
euros = digit { , digit } cents;
```

With these syntax rules: 012, 4323.3, 3.56 are examples of licit representations of Euros. 12., .10 are examples of non licit representation of Euros.

H.2 Data value format types

The grammar defined in this annex defines eight different types of value formats: four quantitative and five non-quantitative value formats.

In the next clause, we define the meta-identifiers that are used to specify these formats. In Clause H.4 we define the syntax rule for the four meta-identifiers that represent the four quantitative value formats, together with their meaning at the value level. In Clause H.5 we define the meta-identifiers for the five non-quantitative value formats, together with their meaning at the value level.

H.3 Meta-identifier used to define the formats

The meta-identifiers used in the grammar that define the various value formats are the following:

```
dot = '.';
decimalMark = '.';
exponentIndicator = 'E';
numeratorIndicator = 'N';
denominatorIndicator = 'D';
leadingDigit = '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
lengthOfExponent = leadingDigit, {trailingDigit};
lengthOfIntegerPart = (leadingDigit, {trailingDigit});
lengthOfNumerator = leadingDigit, {trailingDigit};
lengthOfDenominator = leadingDigit, {trailingDigit};
lengthOfFractionalPart = (leadingDigit, {trailingDigit}) | '0' ;
lengthOfIntegralPart = (leadingDigit, {trailingDigit})| '0' ;
lengthOfNumber = leadingDigit, {trailingDigit};
trailingDigit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
signedExponent = 'S';
signedNumber = space,'S';
space = ' ';
variableLengthIndicator = '..';
```

`decimalMark`: separator between integral and fractional part of numbers of format NR2 or NR3.
`leadingDigit`: first cipher of a number comprising one or more ciphers.
`trailingDigit`: one of the ciphers that combine to form numbers, except the first one.

NOTE If a number comprises only one digit, no `trailingDigit` is present.

H.4 Quantitative value formats

The four quantitative value format syntax rules and their meanings for value representation are defined in the following four subclauses. They are allowed for use for properties having the following data types:

— **NUMBER_TYPE_Type**, **INT_TYPE_Type**, **INT_MEASURE_TYPE_Type**, **INT_CURRENCY_TYPE_Type**, **NON_QUANTITATIVE_INT_TYPE_Type**, **REAL_TYPE_Type**,

REAL_MEASURE_TYPE_Type, **REAL_CURRENCY_TYPE_Type**, **RATIONAL_TYPE_Type** or **RATIONAL_MEASURE_TYPE_Type**;

— **LEVEL_TYPE_Type** whose **value_type** is **NUMBER_TYPE_Type**, **INT_TYPE_Type**, **INT_MEASURE_TYPE_Type**, **INT_CURRENCY_TYPE_Type**, **NON_QUANTITATIVE_INT_TYPE_Type**, **REAL_TYPE_Type**, **REAL_MEASURE_TYPE_Type**, **REAL_CURRENCY_TYPE_Type**, **RATIONAL_TYPE_Type** or **RATIONAL_MEASURE_TYPE_Type**;

— **LIST_TYPE_Type**, **SET_TYPE_Type**, **BAG_TYPE_Type**, **ARRAY_TYPE_Type** or **SET_WITH_SUBSET_CONSTRAINT_TYPE_Type** whose **value_type** is **NUMBER_TYPE_Type**, **INT_TYPE_Type**, **INT_MEASURE_TYPE_Type**, **INT_CURRENCY_TYPE_Type**, **NON_QUANTITATIVE_INT_TYPE_Type**, **REAL_TYPE_Type**, **REAL_MEASURE_TYPE_Type**, **REAL_CURRENCY_TYPE_Type**, **RATIONAL_TYPE_Type** or **RATIONAL_MEASURE_TYPE_Type**.

NOTE 1 For **NON_QUANTITATIVE_INT_TYPE_Type**, the value format applies to the code.

NOTE 2 The value of this attribute should be compatible with the data type of the property: it should not change this data type, else it should be ignored.

EXAMPLE The value format NR2 is not compatible with **INT_TYPE_Type**, since integer values shall not have a fractional part.

H.4.1 NR1-value format

The NR1-value syntax specifies the format of an integer property value.

Syntax rule:

```
NR1Value = 'NR1', ((signedNumber, variableLengthIndicator) | (signedNumber, space) | variableLengthIndicator | space), lengthOfNumber;
```

The meaning of NR1-value format components for value representation is as follows:

— `'NR1'`: the value shall be an integer.

NOTE 1 NR1 number values shall not contain any spaces.

— `lengthOfNumber`: number of digits of the value.

NOTE 2 If preceded by a `variableLengthIndicator` the actual number of digits may be less.

— `signedNumber`: if `signedNumber` is present, the related number shall have either a positive, negative, or zero value. In case of positive values a '+' sign may be present. Negative values shall be preceded by a '-' sign. The value zero shall not be preceded by a '-' sign.

— `variableLengthIndicator`: if `variableLengthIndicator` is present, the related number shall contain a number of digits that is less or equal to its length specification, i.e., to `lengthOfNumber`.

H.4.2 NR2-value format

The NR2-value syntax specifies the format of real property value that does not need an exponent.

Syntax rule:

```
NR2Value = 'NR2', ((signedNumber,variableLengthIndicator) | (signedNumber, space) | variableLengthIndicator | space), lengthOfIntegralPart, decimalMark, lengthOfFractionalPart;
```

The meaning of NR2-value format components for value representation is as follows:

— `'NR2'`: the value shall be a real.

NOTE 1 NR2 number values shall not contain any spaces.

— `lengthOfFractionalPart`: number of digits of the fractional part of the number.

NOTE 2 If preceded by a `variableLengthIndicator` the actual number of digits of the fractional part may be less.

NOTE 3 `lengthOfFractionalPart` implicitly specifies the recommended accuracy of the value. The actual accuracy of the number from which this value was derived may have been greater than the value expressed here.

— `lengthOfIntegralPart`: number of digits of the integral part of the number.

NOTE 4 If preceded by a `variableLengthIndicator` the actual number of digits of the integral part may be less.

— `signedNumber`: if `signedNumber` is present, the related number shall have either a positive, negative, or zero value. In case of positive values a '+' sign may be present. Negative values shall be preceded by a '-' sign. The value zero shall not be preceded by a '-' sign.

— `variableLengthIndicator`: if `variableLengthIndicator` is present, either integral part or fractional part of the number or both parts shall contain a number of digits that is less or equal to its related length specification, i.e., to `lengthOfIntegralPart` or `lengthOfFractionalPart`. At least one cipher shall be present in the number.

H.4.3 NR3-value format

The NR3-value syntax specifies the format of a real property value that is represented with an exponent.

Syntax rule:

```
NR3Value = 'NR3', ((signedNumber, variableLengthIndicator) | (signedNumber, space) | variableLengthIndicator | space), lengthOfIntegralPart, decimalMark, lengthOfFractionalPart, exponentIndicator, [signedExponent], lengthOfExponent;
```

The meaning of NR3-value format components for value representation is as follows:

— `'NR3'`: the value shall be a real with an exponent of base 10.

NOTE 1 There shall be at least one digit and the decimal mark in the mantissa. The exponent shall contain at least one digit, too.

NOTE 2 NR3 number values shall not contain any spaces.

— `exponentIndicator`: separator between mantissa and exponent in numbers of format NR3.

— `lengthOfExponent`: number of digits of the exponent.

NOTE 3 If preceded by a `variableLengthIndicator` the actual number of digits of the exponent may be less.

NOTE 4 Eventually existing signs or a decimal mark are not counted by `lengthOfNumber`, `lengthOfIntegralPart`, `lengthOfFractionalPart` or `lengthOfExponent`.

— `lengthOfFractionalPart`: number of digits of the fractional part of the mantissa.

NOTE 5 If preceded by a `variableLengthIndicator` the actual number of digits of the fractional part may be less.

NOTE 6 `lengthOfFractionalPart` implicitly specifies the recommended accuracy of the value. The actual accuracy of the number from which this value was derived may have been greater than the value expressed here.

— `lengthOfIntegralPart`: number of digits of the integral part of the mantissa.

NOTE 7 If preceded by a `variableLengthIndicator` the actual number of digits of the integral part may be less.

— `signedExponent`: if `signedExponent` is present, the related exponent shall have either a positive, negative, or zero value. In case of positive values a '+' sign may be present. Negative values shall be preceded by a '-' sign. The value zero shall not be preceded by a '-' sign.

— `variableLengthIndicator`: if `variableLengthIndicator` is present, the related number or exponent shall contain a number of digits that is less or equal to its related length specification, i.e., to `lengthOfIntegralPart`, `lengthOfFractionalPart`, or `lengthOfExponent`. At least one cipher shall be present in the mantissa and in the exponent.

H.4.4 NR4-value format

The NR4-value syntax specifies the format of a rational property value that is represented with an integer part, and possibly a fraction part with a denominator and a numerator.

Syntax rule:

```
NR4Value = 'NR4', ((signedNumber,variableLengthIndicator) | (signedNumber, space)
| variableLengthIndicator | space), lengthOfIntegerPart, numeratorIndicator,
lengthOfNumerator, denominatorIndicator, lengthOfDenominator;
```

The meaning of NR4-value format components for value representation is as follows:

— `'NR4'`: the value shall be a rational number represented either as an integer, or as a fraction consisting of a numerator and a denominator, or as an integer and a fraction.

EXAMPLE 12 ½ and 12 ¾ are values that may be represented in the NR4 format.

NOTE 1 There shall be at least one digit either in the integer part, or both in the numerator and in the denominator part. If one part of the fraction contains a digit, the other part shall also contain some digits. All three parts may also contain digits.

NOTE 2 NR4 number values shall not contain any spaces.

— `numeratorIndicator`: separator between the integer part description and the fraction part description in formats NR4.

— `lengthOfNumerator`: number of digits of the numerator.

NOTE 3 If preceded by a `variableLengthIndicator` the actual number of digits of the numerator may be less.

NOTE 4 If the value of the rational number is completely represented by its integer part, neither the numerator of the fraction nor its denominator shall be represented.

— `denominatorIndicator`: separator between the numerator part description and the denominator part description in formats NR4.

— `lengthOfDenominator`: number of digits of the denominator.

NOTE 5 If preceded by a `variableLengthIndicator` the actual number of digits of the denominator may be less.

NOTE 6 If the value of the rational number is completely represented by its integer part, neither the numerator of the fraction nor its denominator shall be represented.

— `lengthOfIntegerPart`: number of digits of the integer part of the rational number.

NOTE 7 If preceded by a `variableLengthIndicator` the actual number of digits of the integer part may be less.

— `variableLengthIndicator`: if `variableLengthIndicator` is present, the three parts of the rational number shall contain a number of digits that is less or equal to its related length specification, i.e., to `lengthOfIntegralPart`, `lengthOfNumerator`, or `lengthOfDenominator`. At least one cipher shall be present either in the integral part, or in the two parts of the fraction.

H.5 Non-quantitative value formats

The five non-quantitative value format syntax rules and their meanings are defined in the following five sub-clauses. They are allowed for use for properties having the following data types:

— **STRING_TYPE_Type**, **TRANSLATABLE_STRING_TYPE_Type**, **NON_TRANSLATABLE_STRING_TYPE_Type**, **URI_TYPE_Type**, **NON_QUANTITATIVE_CODE_TYPE_Type**, **DATE_DATA_TYPE_Type**, **DATE_TIME_TYPE_Type** or **TIME_DATA_TYPE_Type**;

— **LEVEL_TYPE_Type** whose **value_type** is **STRING_TYPE_Type**, **TRANSLATABLE_STRING_TYPE_Type**, **NON_TRANSLATABLE_STRING_TYPE_Type**, **URI_TYPE_Type**, **NON_QUANTITATIVE_CODE_TYPE_Type**, **DATE_DATA_TYPE_Type**, **DATE_TIME_TYPE_Type** or **TIME_DATA_TYPE_Type**;;

— **LIST_TYPE_Type**, **SET_TYPE_Type**, **BAG_TYPE_Type**, **ARRAY_TYPE_Type** or **SET_WITH_SUBSET_CONSTRAINT_TYPE_Type** whose **value_type** is **STRING_TYPE_Type**, **TRANSLATABLE_STRING_TYPE_Type**, **NON_TRANSLATABLE_STRING_TYPE_Type**, **URI_TYPE_Type**, **NON_QUANTITATIVE_CODE_TYPE_Type**, **DATE_DATA_TYPE_Type**, **DATE_TIME_TYPE_Type** or **TIME_DATA_TYPE_Type**;.

NOTE 1 The value of this attribute should be compatible with the data type of the property: it should no change this data type, else it should be ignored.

NOTE 2 For **NON_QUANTITATIVE_CODE_TYPE_Type** the value format applies to the code.

Non-quantitative values are represented by strings which comprise characters. The length of a string may be either specified by directly specifying the upper limit of the number of contained characters or by specifying that the total number of characters may be any integral multiple of the length specified.

Syntax rule:

```
factor = leadingDigit, {trailingDigit};

numberOfCharacters = (leadingDigit, {trailingDigit})|( '(nx', factor,')');
```

The meaning of the factor components is as follows

— `factor`: when factor is present, then `numberOfCharacters` shall be any integral multiple of the value given in `factor`. `factor` shall not contain the value zero.

— `numberOfCharacters`: determines the maximum amount of characters contained in the string.

H.5.1 Alphabetic Value Format

An “Alphabetic Value Format (A)” defines the value format of a string that contains alphabetic letters. Thus, the content shall be taken from the characters of row 00, cell 20, cell 40 to 7E, or cell C0 to FF, of the Basic Multilingual Plane (BMP) (Plane 00 of Group 00) of ISO/IEC 10646-1.

NOTE 1 Due to potential interpretation problems of value content within components of one system or of multiple systems, it is recommended that, where possible, the characters used should be restricted to the G0 set of ISO/IEC 10646-1 and/or row 00 columns 002 to 007 of ISO/IEC 10646-1.

NOTE 2 For alternative languages, as supported by translated data, the relevant characters or ideographs of the related language specific character set are available as defined by the Unicode Standard. In most cases, there will be no 1:1 relation between the characters of the source language to the characters of the target language.

NOTE 3 In most cases, there will be no 1:1 relation between the characters of the source language to the characters of the target language.

EXAMPLE CJK (Chinese-Japanese-Korean) ideographs.

Syntax rule:

```
AValue = 'A', (space | variableLengthIndicator), numberOfCharacters;
```

The meaning of A-value format components for value representation is as follows:

— `'A'`: the value shall be a string, or several substrings, of alphabetic letters.

— `variableLengthIndicator`: If `variableLengthIndicator` is present, the string may contain fewer characters than indicated by `numberOfCharacters`. The string shall contain at least one character.

H.5.2 Mixed Characters Value Format

A “Mixed Value Format (M)” format defines the value format of a string that may contain any character specified in Clause H.7.

NOTE For alternative languages as supported by translated data, the relevant characters or ideographs of the related language specific character set are available as defined by the Unicode Standard.

EXAMPLE CJK (Chinese-Japanese-Korean) characters.

Syntax rule:

```
MValue = 'M', (space | variableLengthIndicator), numberOfCharacters;
```

The meaning of M-value format components for value representation is as follows:

— `'M'`: the value shall be a string, or several substrings.

— `variableLengthIndicator`: if `variableLengthIndicator` is present, the string may contain fewer characters than indicated by `numberOfCharacters`. The string shall contain at least one character.

H.5.3 Number Value Format

A “Number Value Format (N)”defines the value format of a string that contains numeric digits only. Thus, the content shall be taken from the characters of row 00, cell 2B, cell 2D, cell 30 to 39, or cell 45 of the Basic Multilingual Plane (BMP) (Plane 00 of Group 00) of ISO/IEC 10646-1.

NOTE For alternative languages as supported by translated data, the relevant characters or ideographs of the related language specific character set are available as defined by the Unicode Standard.

EXAMPLE Table H.2 shows the transposition of the European digits “0” to “9” into Arabic digits.

Table H.2 — Transposing European style digits into Arabic digits

European digits	9	8	7	6	5	4	3	2	1	0
Arabic digits	٩	٨	٧	٦	٥	٤	٣	٢	١	٠

Syntax rule:

```
NValue = 'N', (space | variableLengthIndicator), numberOfCharacters;
```

The meaning of N-value format components for value representation is as follows:

— `'N'`: the value shall be a string, or several substrings, of numeric digits;

— `variableLengthIndicator`: if `variableLengthIndicator` is present, the string may contain fewer characters than indicated by `numberOfCharacters`. The string shall contain at least one character.

H.5.4 Mixed Alphabetic or Numeric Characters Value Format

A “Mixed Alphabetic or Numeric Characters Value Format (X)” defines the value format of a string that contains alphanumeric characters, i.e., any combination of characters from A-value format or N-value format.

NOTE For alternative languages as supported by translated data, the relevant characters or ideographs of the related language specific character set are available as defined by the Unicode Standard.

Syntax rule:

```
XValue = 'X', (space | variableLengthIndicator), numberOfCharacters;
```

The meaning of X-value format components for value representation is as follows:

— `'X'`: the value shall be a string, or several substrings, of alphanumeric, i.e., any combination of alphabetic and numeric characters;

— `variableLengthIndicator`: if `variableLengthIndicator` is present, the string may contain fewer characters than indicated by `numberOfCharacters`. The string shall contain at least one character.

H.5.5 Binary Value Format

A “Binary Value Format (B)”defines the value format of a string that contains binary characters, i.e., “0” or “1”. Thus the content shall be taken from the characters of row 00, cell 30 or 31, of the Basic Multilingual Plane (BMP) (Plane 00 of Group 00) of ISO/IEC 10646-1.

NOTE For alternative languages as supported by translated data, the relevant characters or ideographs of the related language specific character set are available as defined by the Unicode Standard.

Syntax rule:

```
BValue = 'B', (space | variableLengthIndicator), numberOfCharacters;
```

The meaning of B-value format components for value representation is as follows:

— `'B'`: the value shall be a string, or several substrings, consisting of binary values, i.e., the characters “0” or ”1” or sequences thereof.

— `variableLengthIndicator`: if `variableLengthIndicator` is present, the string may contain fewer characters than indicated by `numberOfCharacters`. The string shall contain at least one character.

H.6 Value examples

Table H.3 below contains some examples for values that may be contained in numbers and strings characterized by the above value format scheme.

Table H.3 — Number value examples

Value format	Examples of possible values
NR1 3	123; 001; 000;
NR1..3	123; 87; 5
NR1S 3	+123; +000;
NR1S..3	-123; +1; 0; -12;
NR2 3.3	123.300; 000.400 ; 000.420;
NR2..3.3	321.233; 1.234; 23.56; 9.783; .72; 324.
NR2S 3.3	-123.123; +123.300;
NR2S..3.3	-123.123; +12.3; 0.1; +.4; -3. ; 0. ; .0;
NR3 3.3E4	123.123E0004; 003.000E1000;
NR3 3.3ES4	123.123E+0004; 123.123E0004; 123.000E-0001
NR3..3.3E4	123.123E0004; .123E0001; 5.E1234
NR3S 3.3ES4	+123.123E+0004; 123.000E-0001;
NR3S..3.3ES4	-123.123E+0004; +1.00E-01; .0E0; +3.E-1; -.2E-1000;
NR4 3N2D2	001 02/03; 012 00/01; 123 03/04; 000 01/04
NR4..3N2D2	1 ½ ; 12; 123 ¾; ¼ ;
NR4S 3N2D2	+001 02/03; -012 00/01; 123 03/04; -000 01/04
NR4S..3N2D2	-1 ½ ; 12; +123 ¾; ¼ ;
A 19	My name is Reinhard, abcdefghijklmnopqrs
A..3	Abc; de; G
X..5	A23RN1; B1; ca
M..10	A23RN1; B1; ca. 256 µm;
N (nx5)	12345; 1234512345; 222223333344444;
N..(nx5)	1234; 12345; 34512345; 1234512345; 23333344444; 222223333344444; -3; 5E2;
B 1	0; 1;
B 3	011; 101;

H.7 Characters from ISO/IEC 10646-1

The following characters shall be used for the purpose of Mixed Value Format (M) (see H.5.2):

— all characters from row 00 of the Basic Multilingual Plane (BMP) (Plane 00 of Group 00) of ISO/IEC 10646-1;

— characters from other rows of the Basic Multilingual Plane of ISO/IEC 10646-1 as listed in Table H.4;

— for alternative languages as supported by translated data, the full character set is available as defined by the Unicode Standard.

NOTE 1 Due to potential interpretation problems of value content within components of one system or of multiple systems it is recommended that, where possible, the characters used should be restricted to the G0 set of ISO/IEC 10646-1 and/or row 00 columns 002 to 007 of ISO/IEC 10646-1.

NOTE 2 The Unicode Standard is published by The Unicode Consortium, P.O. Box 391476, Mountain View, CA 94039-1476, U.S.A., www.unicode.org.

Table H.4 — Characters from other rows of the Basic Multilingual Plane of ISO/IEC 10646-1

Character	Name	Row	Cell
ˇ	CARON	02	C7
≡	IDENTICAL TO	22	61
∧	LOGICAL AND	22	27
∨	LOGICAL OR	22	28
∩	INTERSECTION	22	29
∪	UNION	22	2A
⊂	SUBSET OF (IS CONTAINED)	22	82
⊃	SUBSET OF (CONTAINS)	22	83
⇐	LEFTWARDS DOUBLE ARROW (IS IMPLIED BY)	21	D0
⇒	RIGHTWARDS DOUBLE ARROW (IMPLIES)	21	D2
∴	THEREFORE	22	34
∵	BECAUSE	22	35
∈	ELEMENT OF	22	08
∋	CONTAINS AS MEMBER (HAS AS AN ELEMENT)	22	0B
⊆	SUBSET OR EQUAL TO (CONTAINED AS SUB-CLASS)	22	86
⊇	SUPERSET OR EQUAL TO (CONTAINS AS SUB-CLASS)	22	87

Character	Name	Row	Cell
∫	INTEGRAL	22	2B
∮	CONTOUR INTEGRAL	22	2E
∞	INFINITY	22	1E
∇	NABLA	22	07
∂	PARTIAL DIFFERENTIAL	22	02
~	TILDE OPERATOR (DIFFERENCE BETWEEN)	22	3C
≈	ALMOST EQUAL TO	22	48
≃	ASYMPTOTICALLY EQUAL TO	22	43
≅	APPROXIMATELY EQUAL TO (SIMILAR TO)	22	45
≤	LESS THAN OR EQUAL TO	22	64
≠	NOT EQUAL TO	22	60
≥	GREATER THAN OR EQUAL TO	22	65
⇔	LEFT RIGHT DOUBLE ARROW (IF AND ONLY IF)	21	D4
¬	NOT SIGN	00	AC
∀	FOR ALL	22	00
∃	THERE EXISTS	22	03
א	HEBREW LETTER ALEF	05	D0
□	WHITE SQUARE (D'ALEMBERTIAN OPERATOR)	25	A1
‖	PARALLEL TO	22	25
Γ	GREEK CAPITAL LETTER GAMMA	03	93
Δ	GREEK CAPITAL LETTER DELTA	03	94
⊥	UPTACK (ORTHOGONAL TO)	22	A5
∠	ANGLE	22	20
⊾	RIGHT ANGLE WITH ARC	22	BE
Θ	GREEK CAPITAL LETTER THETA	03	98
〈	LEFT POINTING ANGLE BRACKET (BRA)	23	29
〉	RIGHT POINTING ANGLE BRACKET (KET)	23	2A
Λ	GREEK CAPITAL LETTER LAMBDA	03	9B

Character	Name	Row	Cell
′	PRIME	20	32
″	DOUBLE PRIME	20	33
Ξ	GREEK CAPITAL LETTER XI	03	9E
∓	MINUS –OR– PLUS SIGN	22	13
Π	GREEK CAPITAL LETTER PI	03	A0
²	SUPERSCRIPT TWO	00	B2
Σ	GREEK CAPITAL LETTER SIGMA	03	A3
×	MULTIPLICATION SIGN	00	D7
³	SUPERSCRIPT THREE	00	B3
Υ	GREEK CAPITAL LETTER UPSILON	03	A5
Φ	GREEK CAPITAL LETTER PHI	03	A6
·	MIDDLE DOT	00	B7
Ψ	GREEK CAPITAL LETTER PSI	03	A8
Ω	GREEK CAPITAL LETTER OMEGA	03	A9
∅	EMPTY SET	22	05
⇀	RIGHTWARDS HARPOON WITH BARB UPWARDS (VECTOR OVERBAR)	21	C0
√	SQUARE ROOT (RADICAL)	22	1A
ƒ	LATIN SMALL LETTER F WITH HOOK (FUNCTION OF)	01	92
∝	PROPORTIONAL TO	22	1D
±	PLUS – MINUS SIGN	00	B1
°	DEGREE SIGN	00	B0
α	GREEK SMALL LETTER ALPHA	03	B1
β	GREEK SMALL LETTER BETA	03	B2
γ	GREEK SMALL LETTER GAMMA	03	B3
δ	GREEK SMALL LETTER DELTA	03	B4
ε	GREEK SMALL LETTER EPSILON	03	B5
ζ	GREEK SMALL LETTER ZETA	03	B6

Character	Name	Row	Cell
η	GREEK SMALL LETTER ETA	03	B7
θ	GREEK SMALL LETTER THETA	03	B8
ι	GREEK SMALL LETTER IOTA	03	B9
κ	GREEK SMALL LETTER KAPPA	03	BA
λ	GREEK SMALL LETTER LAMBDA	03	BB
μ	GREEK SMALL LETTER MU	03	BC
ν	GREEK SMALL LETTER NU	03	BD
ξ	GREEK SMALL LETTER XI	03	BE
‰	PER MILLE SIGN	20	30
π	GREEK SMALL LETTER PI	03	C0
ρ	GREEK SMALL LETTER RHO	03	C1
σ	GREEK SMALL LETTER SIGMA	03	C3
÷	DIVISION SIGN	03	F7
τ	GREEK SMALL LETTER TAU	03	C4
υ	GREEK SMALL LETTER UPSILON	03	C5
φ	GREEK SMALL LETTER PHI	03	C6
χ	GREEK SMALL LETTER CHI	03	C7
ψ	GREEK SMALL LETTER PSI	03	C8
ω	GREEK SMALL LETTER OMEGA	03	C9
†	DAGGER	20	20
←	LEFTWARDS ARROW	21	90
↑	UPWARDS ARROW	21	91
→	RIGHTWARDS ARROW	21	92
↓	DOWNWARDS ARROW	21	93
‾	OVERLINE	20	3E

Annex I
(informative)

XML file example

This annex presents, on an example, an overview of the different resources and steps involved in the description of a parts library according to the ISO 13584 series.

The content of this clause is as follows:

— example introduction;

— OntoML representation of this example.

I.1 Example introduction

Figure I.1 intended to be described. It is a characterization class of washers, denoted *paw*, that is sold by a bearing supplier and that is used as a bearing in some mechanical contexts. Every underlying product ontology concept is represented in a single language: English.

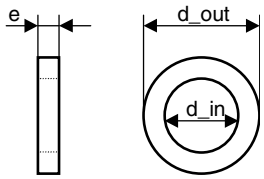

Figure I.1 — General model example: ontology definition

This product characterization class is described by three characteristic properties:

— the inner diameter (*d_in* symbol), a real measure whose unit is expressed in millimeter;

— the outer diameter (*d_out* symbol), a real measure whose unit is expressed in millimeter;

— the thickness (*e* symbol), a real measure whose unit is expressed in millimeter.

This product characterization class is also associated to a drawing (the one presented in Figure I.1), called *paw.jpg*.

For the purpose of this example, we propose to define a very simple product ontology as follow:

— a product characterization class that plays the role of the product ontology root, and whose name is *bearing*; it defines and it is described by two properties:

  — inner diameter (*d_in*);

  — outer diameter (*d_out*).

— a product characterization class corresponding to the *paw* characterization class, subclass of the *bearing* class; it defines and it is described by a single property:

  — thickness (*e*).

The CIIM ontology concepts that need to be used in this example are: dictionary, supplier, class and property. For each of them, we define (according to the identifiers structure defined in Clause 9.1) the following identifiers:

— parts supplier: “0002-38491502100024”;

— dictionary: “0002-38491502100024#11-DICTIONARY#1”;

— *bearing* characterization class: “0002-38491502100024#01-BEARING#1”;

— *paw* characterization class: “0002-38491502100024#01-PAW#1”;

— *inner diameter* property: “0002-38491502100024#02-INNER_DIAMETER#1”;

— *outer diameter* property: “0002-38491502100024#02-OUTER_DIAMTER#1”;

— *thickness* property: “0002-38491502100024#02-THICKNESS#1”.

According to this ontology structure, we propose to describe the following five *paw* products (Figure I.2):

d_in	e	d_out
10	1	15
11	1	16.5
13	2	19.5
17	3	25.5
19	4	28.5

Figure I.2 — General model example: product specification

NOTE OntoML does only provide resources for the representation of information about the container structure of products (product characterization class extension). A product description as property reference and typed value pairs is defined outside the scope.

I.2 OntoML representation

In this clause, the complete OntoML representation of the product ontology example introduced in Clause I.1 is given. This XML file is conformant with conformance class 3 defined in this part of ISO 13584.

```
<?xml version="1.0" encoding="UTF-8"?>
<ontoml:ontoml xmlns:ontoml="urn:iso:std:iso:13584:-32:ed-1:tech:xml-schema:ontoml"
xmlns:cat="urn:iso:std:iso:ts:29002:-10:ed-1:tech:xml-schema:catalogue" xmlns:val="urn:iso:std:iso:ts:29002:-10:ed-
1:tech:xml-schema:value" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:iso:std:iso:13584:-32:ed-1:tech:xml-schema:ontoml
D:\plib\plibDoc\P32\IS\xml_schema\ontoml.xsd">
    <header id="0002-38491502100024#11-DICTIONARY#1">
        <global_language language_code="en"/>
        <description>This file illustrates the use of OntoML</description>
        <version>1</version>
        <name>generalModel.xml</name>
        <date_time_stamp>2010-04-07T08:11:49+02:00</date_time_stamp>
        <author>Eric SARDET</author>
```

```
        <organisation>My Company</organisation>
        <pre_processor_version/>
        <originating_system>XMLSpy</originating_system>
        <authorisation>Eric Sardet - sardet@ensma.fr</authorisation>
        <!-- The XML file is compliant with ISO13584-32, conformance class 3. The library_structure XML element
specifies this particular conformance class. -->
        <ontoml_information>
            <!--We assume that the dictionary/library revision is equal to 1-->
            <revision>1</revision>
            <preferred_name>
                <label>OntoML dictionary + library basic example</label>
            </preferred_name>
        </ontoml_information>
        <ontoml_structure>
            <status>IS</status>
            <name>ONTOML</name>
            <date>2010</date>
            <application>3</application>
        </ontoml_structure>
    </header>
    <!-- The ontology is defined using the dictionary XML element. Because it represents both the ontology itself and
instances compliant to this ontology, the specific content model used is based on the
LIBRARY_IN_STANDARD_FORMAT_Type complex type definition. An identifier is assigned to the dictionary using the id
XML attribute. -->
    <dictionary xsi:type="ontoml:DICTIONARY_IN_STANDARD_FORMAT_Type">
        <!--We assume that the information described is a complete library-->
        <is_complete>true</is_complete>
        <!-- The supplier responsible for the description of the ontology data is identified using the supplier_ref XML
attribute -->
        <responsible_supplier supplier_ref="0002-38491502100024"/>
        <!-- All the characterization classes described in an ontology are gathered in the "contained_classes" XML
element.-->
        <contained_classes>
            <!-- This class XML element defines the "bearing" class. It is defined as an item class ("ITEM_CLASS_Type"
complex type). It is identified using the "id" XML attribute.-->
            <ontoml:class xsi:type="ontoml:ITEM_CLASS_Type" id="0002-38491502100024#01-BEARING#1">
                <!-- We assume that the provided characterizationclass revision is equal to 1 -->
                <revision>1</revision>
                <!-- This characterization class preferred name is not translated and set to "bearing" -->
                <preferred_name>
                    <label>bearing</label>
                </preferred_name>
                <!-- The "bearing" characterization class definition is not translated. -->
                <definition>
                    <text>general bearing parts family</text>
                </definition>
                <!-- The "bearing" characterization class is described by two properties, each of them being identified
unambiguously ("property_ref" XMLattribute). In this example, the properties definitions are provided, but it is not
mandatory. The listed properties are: inner diameter and outer diameter.-->
                <described_by>
                    <property property_ref="0002-38491502100024#02-INNER_DIAMETER#1"/>
                    <property property_ref="0002-38491502100024#02-OUTER_DIAMETER#1"/>
                </described_by>
            </ontoml:class>
            <!-- This class XML element defines the "paw" class. It is defined as an item class ("ITEM_CLASS_Type"
complex type). It is identified using the id XML attribute.-->
            <ontoml:class xsi:type="ontoml:ITEM_CLASS_Type" id="0002-38491502100024#01-PAW#1">
                <!-- We assume that the provided characterization class revision is equal to "1" -->
                <revision>1</revision>
                <!-- This characterization class preferred name is not translated and set to "paw" -->
                <preferred_name>
                    <label>paw</label>
                </preferred_name>
                <!-- The "paw" characterization class definition is not translated. -->
                <definition>
                    <text>paw parts family</text>
                </definition>
```

```
            <!-- The "paw" characterization class is a subclass of the "bearing" characterization class: it is specified using the "its_superclass" element whose "class_ref" attibute references the "bearing" characterization class-->
            <its_superclass class_ref="0002-38491502100024#01-BEARING#1"/>
            <!-- The "paw" characterization class is described by a single property being identified unambiguously ("property_ref" XMLattribute). In this example, the properties definitions are provided, but it is not mandatory. The listed properties are: inner diameter and outer diameter.-->
            <described_by>
                <property property_ref="0002-38491502100024#02-THICKNESS#001"/>
            </described_by>
            <!-- A drawing, stored in a file called "paw.jpeg" located in the same directory that the current XML file, is assigned to the "paw" characterization class.-->
            <simplified_drawing xsi:type="ontoml:EXTERNAL_GRAPHICS_Type">
                <!-- The drawing consists of a single external resource called "paw.jpg"-->
                <representation>
                    <file>
                        <http_file>paw.jpeg</http_file>
                    </file>
                </representation>
            </simplified_drawing>
        </ontoml:class>
    </contained_classes>
    <!-- A name, that is only defined in english, is provided to the dictionary -->
    <!--Every supplier referenced in this ontology shall be described in the "contained_suppliers" XML element. -->
    <contained_suppliers>
        <ontoml:supplier id="0002-38491502100024">
            <revision>1</revision>
            <!-- The supplier is only described through its name. -->
            <org>
                <name>My Company</name>
            </org>
        </ontoml:supplier>
    </contained_suppliers>
    <!-- All the properties described in an ontology are gathered in the "contained_properties" XML element.-->
    <contained_properties>
        <ontoml:property xsi:type="ontoml:NON_DEPENDENT_P_DET_Type" id="0002-38491502100024#02-INNER_DIAMETER#001">
            <!-- Domain definition of the "inner diameter" property -->
            <name_scope class_ref="0002-38491502100024#01-BEARING#1"/>
            <!-- We assume that the provided property revision is equal to "1" -->
            <revision>1</revision>
            <!-- This property preferred name is not translated and set to "inner diameter" -->
            <preferred_name>
                <label>inner diameter</label>
            </preferred_name>
            <!-- The "inner diameter" property definition is not translated. -->
            <definition>
                <text>the bearing inner diameter</text>
            </definition>
            <!-- The "d_in" preferred symbol is assigned to this property -->
            <preferred_symbol>
                <text_representation>d_in</text_representation>
            </preferred_symbol>
            <!-- A real measure value domain (REAL_MEASURE_TYPE_Type XML complex type), expressed in millimetre, is assigned to this property. -->
            <domain xsi:type="ontoml:REAL_MEASURE_TYPE_Type">
                <unit>
                    <structured_representation xsi:type="ontoml:SI_UNIT_Type">
                        <!-- The particular named unit used is an SI Unit, as defined by the SI_UNIT_Type XML complex type-->
                        <prefix>MILLI</prefix>
                        <name>METRE</name>
                    </structured_representation>
                </unit>
            </domain>
        </ontoml:property>
        <ontoml:property xsi:type="ontoml:NON_DEPENDENT_P_DET_Type" id="0002-38491502100024#02-OUTER_DIAMETER#001">
```

```
                <!-- Domain definition of the "outer diameter" property -->
                <name_scope class_ref="0002-38491502100024#01-BEARING#1"/>
                <!-- We assume that the provided property revision is equal to "1" -->
                <revision>1</revision>
                <!-- This property preferred name is not translated and set to "outer diameter" -->
                <preferred_name>
                    <label>outer diameter</label>
                </preferred_name>
                <!-- The "outer diameter" property definition is not translated. -->
                <definition>
                    <text>the bearing outer diameter</text>
                </definition>
                <!-- The "d_in" preferred symbol is assigned to this property -->
                <preferred_symbol>
                    <text_representation>d_out</text_representation>
                </preferred_symbol>
                <!-- A real measure value domain (REAL_MEASURE_TYPE_Type XML complex type), expressed in
millimetre, is assigned to this property. -->
                <domain xsi:type="ontoml:REAL_MEASURE_TYPE_Type">
                    <unit>
                        <structured_representation xsi:type="ontoml:SI_UNIT_Type">
                            <!-- The particular named unit used is an SI Unit, as defined by the SI_UNIT_Type XML complex
type-->
                            <prefix>MILLI</prefix>
                            <name>METRE</name>
                        </structured_representation>
                    </unit>
                </domain>
            </ontoml:property>
            <ontoml:property xsi:type="ontoml:NON_DEPENDENT_P_DET_Type" id="0002-38491502100024#02-
THICKNESS#001">
                <!-- Domain definition of the "thickness" property -->
                <name_scope class_ref="0002-38491502100024#01-BEARING#1"/>
                <!-- We assume that the provided property revision is equal to "1" -->
                <revision>1</revision>
                <!-- This property preferred name is not translated and set to "thickness" -->
                <preferred_name>
                    <label>thickness</label>
                </preferred_name>
                <!-- The "thickness" property definition is not translated. -->
                <definition>
                    <text>the paw thickness</text>
                </definition>
                <!-- The "e" preferred symbol is assigned to this property -->
                <preferred_symbol>
                    <text_representation>e</text_representation>
                </preferred_symbol>
                <!-- A real measure value domain, expressed in millimetre (REAL_MEASURE_TYPE_Type XML complex
type), is assigned to this property. -->
                <domain xsi:type="ontoml:REAL_MEASURE_TYPE_Type">
                    <unit>
                        <structured_representation xsi:type="ontoml:SI_UNIT_Type">
                            <!-- The particular named unit used is an SI Unit, as defined by the SI_UNIT_Type XML complex
type-->
                            <prefix>MILLI</prefix>
                            <name>METRE</name>
                        </structured_representation>
                    </unit>
                </domain>
            </ontoml:property>
        </contained_properties>
        <!-- The ontology is provided in a single language: english.-->
    </dictionary>
    <!-- The "content" XML element allows to represent products according to some product charaterization classes
described in a given ontology-->
    <library xsi:type="ontoml:LIBRARY_IN_STANDARD_FORMAT_Type">
        <contained_class_extensions>
```

```
        <!-- We describe the products that relate to a product characterization class. Every product is described as a set of (property reference, type value) pairs. -->
        <class_extension xsi:type="ontoml:EXPLICIT_ITEM_CLASS_EXTENSION_Type">
            <!-- The particular product characterization class for which the products are specified is referenced using the "class_ref" XML attribute.-->
            <dictionary_definition class_ref="0002-38491502100024#01-PAW#1"/>
            <!-- We assume that the provided content revision is equal to "1" -->
            <content_revision>1</content_revision>
            <!-- the "instance_identification" defines which properties of the describes products play the role of a primary key. In our example, the "inner diameter" property plays such a role. This property is therefore referenced unambiguously using the "property_ref" XML attribute.-->
            <instance_identification>
                <property property_ref="0002-38491502100024#02-INNER_DIAMETER#1"/>
            </instance_identification>
            <!-- A product charaterization class consists of a set of product. they are all gathered in the "population" XML element. -->
            <population>
                <!-- Every product is defines using external resources for describing a product as a set of (property reference, type value) pairs.Moreover, every product references the class to which it belongs using the "class_ref" attribute.-->
                <cat:item class_ref="0002-38491502100024#01-PAW#1">
                    <!-- The first value is about the "inner diameter" property that is referenced using the "property_ref' attribute -->
                    <cat:property_value property_ref="0002-38491502100024#02-INNER_DIAMETER#1">
                        <!-- The valiue is a real measure, whose unit is "mm". This value is equal to "10". -->
                        <val:measure_single_number_value UOM_code="mm">
                            <val:real_value>10</val:real_value>
                        </val:measure_single_number_value>
                    </cat:property_value>
                    <!-- The second value is about the "thickness" property that is referenced using the "property_ref' attribute -->
                    <cat:property_value property_ref="0002-38491502100024#02-THICKNESS#1">
                        <!-- The valiue is a real measure, whose unit is "mm". This value is equal to "10". -->
                        <val:measure_single_number_value UOM_code="mm">
                            <val:real_value>1</val:real_value>
                        </val:measure_single_number_value>
                    </cat:property_value>
                    <!-- The third value is about the "outer diameter" property that is referenced using the "property_ref' attribute -->
                    <cat:property_value property_ref="0002-38491502100024#02-OUTER_DIAMETER#1">
                        <!-- The valiue is a real measure, whose unit is "mm". This value is equal to "10". -->
                        <val:measure_single_number_value UOM_code="mm">
                            <val:real_value>15</val:real_value>
                        </val:measure_single_number_value>
                    </cat:property_value>
                </cat:item>
                <!-- Second product description -->
                <cat:item class_ref="0002-38491502100024#01-PAW#1">
                    <cat:property_value property_ref="0002-38491502100024#02-INNER_DIAMETER#1">
                        <val:measure_single_number_value UOM_code="mm">
                            <val:real_value>11</val:real_value>
                        </val:measure_single_number_value>
                    </cat:property_value>
                    <cat:property_value property_ref="0002-38491502100024#02-THICKNESS#1">
                        <val:measure_single_number_value UOM_code="mm">
                            <val:real_value>1</val:real_value>
                        </val:measure_single_number_value>
                    </cat:property_value>
                    <cat:property_value property_ref="0002-38491502100024#02-OUTER_DIAMETER#1">
                        <val:measure_single_number_value UOM_code="mm">
                            <val:real_value>16.5</val:real_value>
                        </val:measure_single_number_value>
                    </cat:property_value>
                </cat:item>
                <!-- Third product description -->
                <cat:item class_ref="0002-38491502100024#01-PAW#1">
                    <cat:property_value property_ref="0002-38491502100024#02-INNER_DIAMETER#1">
```

```
                        <val:measure_single_number_value UOM_code="mm">
                            <val:real_value>13</val:real_value>
                        </val:measure_single_number_value>
                    </cat:property_value>
                    <cat:property_value property_ref="0002-38491502100024#02-THICKNESS#1">
                        <val:measure_single_number_value UOM_code="mm">
                            <val:real_value>2</val:real_value>
                        </val:measure_single_number_value>
                    </cat:property_value>
                    <cat:property_value property_ref="0002-38491502100024#02-OUTER_DIAMETER#1">
                        <val:measure_single_number_value UOM_code="mm">
                            <val:real_value>19.5</val:real_value>
                        </val:measure_single_number_value>
                    </cat:property_value>
                </cat:item>
                <!-- Fourth product description -->
                <cat:item class_ref="0002-38491502100024#01-PAW#001">
                    <cat:property_value property_ref="0002-38491502100024#02-INNER_DIAMETER#001">
                        <val:measure_single_number_value UOM_code="mm">
                            <val:real_value>17</val:real_value>
                        </val:measure_single_number_value>
                    </cat:property_value>
                    <cat:property_value property_ref="0002-38491502100024#02-THICKNESS#001">
                        <val:measure_single_number_value UOM_code="mm">
                            <val:real_value>3</val:real_value>
                        </val:measure_single_number_value>
                    </cat:property_value>
                    <cat:property_value property_ref="0002-38491502100024#02-OUTER_DIAMETER#001">
                        <val:measure_single_number_value UOM_code="mm">
                            <val:real_value>25.5</val:real_value>
                        </val:measure_single_number_value>
                    </cat:property_value>
                </cat:item>
                <!-- Fifth product description -->
                <cat:item class_ref="0002-38491502100024#01-PAW#001">
                    <cat:property_value property_ref="0002-38491502100024#02-INNER_DIAMETER#001">
                        <val:measure_single_number_value UOM_code="mm">
                            <val:real_value>19</val:real_value>
                        </val:measure_single_number_value>
                    </cat:property_value>
                    <cat:property_value property_ref="0002-38491502100024#02-THICKNESS#001">
                        <val:measure_single_number_value UOM_code="mm">
                            <val:real_value>4</val:real_value>
                        </val:measure_single_number_value>
                    </cat:property_value>
                    <cat:property_value property_ref="0002-38491502100024#02-OUTER_DIAMETER#001">
                        <val:measure_single_number_value UOM_code="mm">
                            <val:real_value>28.5</val:real_value>
                        </val:measure_single_number_value>
                    </cat:property_value>
                </cat:item>
            </population>
            <table_like>true</table_like>
        </class_extension>
    </contained_class_extensions>
    <responsible_supplier supplier_ref="0002-38491502100024"/>
  </library>
</ontoml:ontoml>
```

Annex J
(informative)

Information to support implementations

Additional information may be provided to support implementations. If the information is provided it can be found at the following URL:

http://www.tc184-sc4.org/implementation_information/13584/00032

Bibliography

[1] ISO 639-1, *Codes for the representation of names of languages — Part 1: Alpha-2 code*

[2] ISO 639-2, *Codes for the representation of names of languages — Part 2: Alpha-3 code*

[3] ISO 843, *Information and documentation — Conversion of Greek characters into Latin characters*

[4] ISO 3166-1, *Codes for the representation of names of countries and their subdivisions — Part 1: Country codes*

[5] ISO 10303-1, *Industrial automation systems and integration — Product data representation and exchange — Part 1: Overview and fundamental principles*

[6] ISO 10303-21, *Industrial automation systems and integration — Product data representation and exchange — Part 21: Implementation methods: Clear text encoding of the exchange structure*

[7] ISO 10303-41, *Industrial automation systems and integration — Product data representation and exchange — Part 41: Integrated generic resource: Fundamentals of product description and support*

[8] ISO 10303-42, *Industrial automation systems and integration — Product data representation and exchange — Part 42: Integrated generic resource: Geometric and topological representation*

[9] ISO 13584-1*, Industrial automation systems and integration — Parts library — Part 1: Overview and fundamental principles*

[10] ISO 13584-24:2003*, Industrial automation systems and integration — Parts library — Part 24: Logical resource: Logical model of supplier library*

[11] ISO 13584-25:2004*, Industrial automation systems and integration — Parts library — Part 25: Logical resource: Logical model of supplier library with aggregate values and explicit content*

[12] ISO 13584-26*, Industrial automation systems and integration — Parts library — Part 26: Logical resource: Information supplier identification*

[13] ISO 13584-42:2010*, Industrial automation systems and integration — Parts library — Part 42: Description methodology: Methodology for structuring parts families*

[14] ISO 13584-102, *Industrial automation systems and integration — Parts library — Part 102: View exchange protocol by ISO 10303 conforming specification*

[15] ISO 13584-501, *Industrial automation systems and integration — Parts library — Part 501: Reference dictionary for measuring instruments — Registration procedure*

[16] ISO 13584-511, *Industrial automation systems and integration — Parts library — Part 511: Mechanical systems and components for general use — Reference dictionary for fasteners*

[17] ISO/TS 23768-1[1]*, Rolling bearings — Parts library — Part 1: Reference dictionary*

[18] ISO/TS 29002-20, *Industrial automation systems and integration — Exchange of characteristic data — Part 20: Concept dictionary resolution services*

[1] Under preparation.

[19] ISO 80000-1, *Quantities and units — Part 1: General*

[20] IEC 61360 (all parts), *Standard data element types with associated classification scheme for electronic components*

[21] ISO/IEC Guide 77-2:2008, *Guide for specification of product properties and classes — Part 2: Technical principles and guidance*

[22] ISO/IEC 8824-1, *Information technology — Abstract Syntax Notation One (ASN.1) — Part 1: Specification of basic notation*

Index

ICS 25.040.40

Price based on 259 pages