BS EN 61360-2:2013

BSI Standards Publication

Standard data element types with associated classification scheme for electric components

Part 2: EXPRESS dictionary schema

...making excellence a habit.™

National foreword

This British Standard is the UK implementation of EN 61360-2:2013. It is identical to IEC 61360-2:2012. It supersedes BS EN 61360-2:2002 which is withdrawn.

The UK participation in its preparation was entrusted to Technical Committee GEL/3, Documentation and graphical symbols.

A list of organizations represented on this committee can be obtained on request to its secretary.

Published by BSI Standards Limited 2013.

ISBN 978 0 580 61285 5

ICS 31.020

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 28 February 2013.

Amendments issued since publication

Date	Text affected

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN 61360-2

January 2013

ICS 31.020

Supersedes EN 61360-2:2002 + A1:2004

English version

Standard data element types with associated classification scheme for electric components - Part 2: EXPRESS dictionary schema (IEC 61360-2:2012)

Types normalisés d'éléments de données avec plan de classifcation pour composants électriques -
Partie 2: Schéma d'un dictionnaire EXPRESS
(CEI 61360-2:2012)

Genormte Datenelementtypen mit Klassifikationsschema für elektrische Bauteile -
Teil 2: EXPRESS-Datenmodell
(IEC 61360-2:2012)

CENELEC

European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

Management Centre: Avenue Marnix 17, B - 1000 Brussels

Foreword

The text of document 3D/196/FDIS, future edition 3 of IEC 61360-2, prepared by IEC/SC 3D "Product properties and classes and their identification", of IEC/TC 3 "Information structures, documentation and graphical symbols" was submitted to the IEC-CENELEC parallel vote and approved by CENELEC as EN 61360-2:2013.

The following dates are fixed:

• latest date by which the document has to be implemented at national level by publication of an identical national standard or by endorsement	(dop)	2013-08-06
• latest date by which the national standards conflicting with the document have to be withdrawn	(dow)	2015-11-06

This document supersedes EN 61360-2:2002 + A1:2004.

EN 61360-2:2012 includes the following significant technical changes with respect to EN 61360-2:2002:

– separation of concepts between characterization class and categorization class;

– introduction of value constraints on classes and properties;

– addition of various new subtypes for data types, including rational_type;

– improvement on the representation of unit of measurement.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CENELEC [and/or CEN] shall not be held responsible for identifying any or all such patent rights.

Endorsement notice

The text of the International Standard IEC 61360-2:2012 was approved by CENELEC as a European Standard without any modification.

In the official version, for Bibliography, the following notes have to be added for the standards indicated:

IEC 60027 series	NOTE	Harmonized in EN 60027 series.
IEC 61360-5	NOTE	Harmonized as EN 61360-5.
ISO 80000 series	NOTE	Harmonized in EN ISO 80000 series.

Annex ZA
(normative)

Normative references to international publications with their corresponding European publications

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

NOTE When an international publication has been modified by common modifications, indicated by (mod), the relevant EN/HD applies.

Publication	Year	Title	EN/HD	Year
IEC 61360-1	2009	Standard data elements types with associated classification scheme for electric items - Part 1: Definitions - Principles and methods	EN 61360-1	2010
IEC 61360-4	Data-base	Standard data element types with associated classification scheme for electric components - Part 4: IEC reference collection of standard data element types and component classes	-	-
ISO/IEC 8859-1	1998	Information technology - 8-bit single-byte coded graphic character sets - Part 1: Latin alphabet No.1	-	-
ISO/IEC 10646-1	-	Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane	-	-
ISO/IEC 14977	-	Information technology - Syntactic metalanguage - Extended BNF	-	-
ISO 639	Series	Codes for the representation of names of languages	-	-
ISO 843	1997	Information and documentation - Conversion of Greek characters into Latin characters	-	-
ISO 3166-1	-	Codes for the representation of names of countries and their subdivisions - Part 1: Country codes	EN ISO 3166-1	-
ISO 4217	2008	Codes for the representation of currencies and funds	-	-
ISO 8601	2004	Data elements and interchange formats - Information interchange - Representation of dates and times	-	-
ISO 10303-11	2004	Industrial automation systems and integration - Product data representation and exchange - Part 11: Description methods: The EXPRESS language reference manual	-	-
ISO 10303-21	2002	Industrial automation systems and integration - Product data representation and exchange - Part 21: Implementation methods: Clear text encoding of the exchange structure	-	-

Publication	Year	Title	EN/HD	Year
ISO 10303-41	2000	Industrial automation systems and integration - Product data representation and exchange - Part 41: Integrated generic resource: Fundamentals of product description and support	-	-
ISO 13584-26	2000	Industrial automation systems and integration - Parts library - Part 26: Logical resource: Information supplier identification	-	-
ISO 13584-42	2010	Industrial automation systems and integration - Parts library - Part 42: Description methodology: Methodology for structuring parts families	-	-

CONTENTS

INTRODUCTION

The common ISO/IEC dictionary schema presented here is based on the intersection of the scopes of the following standards:

– IEC 61360-1;
– ISO 13584-42.

Relevant parts of the scope clauses of these standards include the following:

IEC 61360-1:2009

“This part of IEC 61360 provides a firm basis for the clear and unambiguous definition of characteristic properties (data element types) of all elements of electrotechnical systems from basic components to subassemblies and full systems. Although originally conceived in the context of providing a basis for the exchange of information on electric/electronic components, the principles and methods of this standard may be used in areas outside the originalconception such as assemblies of components and electrotechnical systems and subsystems.”

ISO 13584-42:2010

“This part of ISO 13584 specifies the principles to be used for defining characterization classes of parts and properties of parts which provide for characterizing a part independently of any particular supplier-defined identification.

The rules and guidelines provided in this part of ISO 13584 are mandatory for the standardization committees responsible for creating standardized characterization hierarchies.

The use of these rules by suppliers and users is recommended as a methodology for building their own hierarchies.”

IEC SC3D and ISO TC184/SC4 agreed NOT to change and/or modify the presented EXPRESS model independent of each other in order to guarantee the harmonization and the reusability of the work from both committees. Requests for amendments should therefore be sent to both committees. These requests should be adopted by both committees before modifying the EXPRESS information model

STANDARD DATA ELEMENT TYPES WITH ASSOCIATED CLASSIFICATION SCHEME FOR ELECTRIC COMPONENTS –

Part 2: EXPRESS dictionary schema

1 Scope

This part of IEC 61360 series provides a formal model for data according to the scope as given in IEC 61360-1 and ISO 13584-42, and thus provides a means for the computer-sensible representation and exchange of such data.

The intention is to provide a common information model for the work of IEC SC3D and ISO TC184/SC4, thus allowing for the implementation of dictionary systems dealing with data delivered according to either of the standards elaborated by both committees.

The scope of this part of IEC 61360 is the common ISO/IEC dictionary schema based on the intersection of the scopes of the two base standards IEC 61360-1 and ISO 13584-42.

The presented EXPRESS model represents a common formal model for the two standards and facilitates a harmonization of both.

The IEC 61360-2 forms the master document. ISO 13584-42 contains a copy of the IEC 61360-2 EXPRESS model in an informative annex

In a number of clauses, where the common EXPRESS model allows more freedom, IEC has defined more restrictions which are found in the methodology part of IEC 61360-1.

Two schemas are provided in this part of IEC 61360 defining the two options that may be selected for an implementation. Each of these options is referred to as a conformance class.

– The ISO13584_IEC61360_dictionary_schema2 provides for modelling and exchanging technical data element types with associated classification scheme used in the data element type definitions. It constitutes conformance class 1 of this part of IEC 61360.
– The ISO13584_IEC61360_language_resource_schema provides resources for permitting strings in various languages. It has been extracted from the dictionary schema, since it could be used in other schemata. It is largely based on the support_resource_schema from ISO 10303-41:2000, and can be seen as an extension to that. It allows for the usage of one specific language throughout an exchange context (physical file) without the overhead introduced when multiple languages are used.

When used together with ISO 10303-21, each schema defines one single exchange format. The exchange format defined by conformance class 1 is fully compatible with the ISO 13584 series.

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.

IEC 61360-1:2009, *Standard data elements types with associated classification scheme for electric items – Part 1: Definitions – Principles and methods*

IEC 61360-DB, *Standard data element types with associated classification scheme for electric components – Part 4: IEC reference collection of standard data element types and component classes*

ISO/IEC 8859-1:1998, *Information techonology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1*

ISO/IEC 10646-1, *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane*

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

ISO 639 (all parts), *Codes for the representation of names of languages*

ISO 843:1997*, Information and documentation – Conversion of Greek characters into Latin characters*

ISO 3166-1, *Codes for the representation of names of countries and their subdivisions – Part 1: Country codes*

ISO 4217:2008, *Codes for the representation of currencies and funds*

ISO 8601:2004, *Data elements and interchange formats – Information interchange – Representation of dates and times*

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

ISO 10303-21:2002, *Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure*

ISO 10303-41:2000, *Industrial automation systems and integration – Product data representation and exchange – Part 41: Integrated generic resources: Fundamentals of product description and support*[1]

ISO 13584-26:2000, *Industrial automation systems and integration – Parts library – Part 26: Logical resource: Information supplier identification*

ISO 13584-42:2010, *Industrial automation systems and integration – Parts library – Part 42: Description methodology: Methodology for structuring parts families*

3 Terms and definitions

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

3.1
abstract class
class of which all members are also members of one of its subclasses

[1] A new edition of ISO 10303-41 was published in 2005.

Note 1 to entry: Abstract classes are used when it is needed to group different kinds of objects in a class of a class inclusion hierarchy.

Note 2 to entry: In the common ISO13584/IEC61360 dictionary model, both abstract categorization classes and abstract characterization classes can be defined. The fact of being abstract is only a conceptual characteristic of a class. This characteristic is not explicitly represented in the model.

Note 3 to entry: Through inheritance, abstract characterization class allows to share, for example, some visible properties between different subclasses that correspond to different kinds of items.

3.2
applicable property of a class
applicable property necessarily possessed by each part that is member of a characterization class

Note 1 to entry: Each part that is member of a characterization class possesses an aspect corresponding to each applicable property of this characterization class.

Note 2 to entry: The above definition is conceptual, there is no requirement that all the applicable properties of a class should be used for describing each part of this class at the data model level.

Note 3 to entry: All the applicable properties of a superclass are also applicable properties for the subclasses of this superclass.

Note 4 to entry: Only properties defined or inherited as visible and imported properties of a class may be applicable properties.

Note 5 to entry: To facilitate integration of component libraries and electronic catalogues based on ISO 13584-24:2003 and ISO 13584-25, these parts of ISO 13584 request that only properties that are applicable to a class be used to characterize their instances in component libraries and electronic catalogues.

3.3
attribute
data element for the computer-sensible description of a property, a relation or a class

Note 1 to entry: An attribute describes only one 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.

3.4
basic semantic unit
entity that provides an absolute and universally unique identification of a certain object of the application domain that is represented as a dictionary element

EXAMPLE 1 A dictionary compliant with this part of IEC 61360 provides for the identification of classes, properties, information sources and datatypes.

EXAMPLE 2 A dictionary compliant with ISO 13584-24:2003 provides for the identification of classes, properties, information sources, datatypes, tables, documents and program libraries.

EXAMPLE 3 In ISO 13584-511, the class of the hexagon head bolts is identified by a BSU, the property thread tolerance grade is also identified by a BSU.

Note 1 to entry: The content of a basic semantic unit may also be represented as an IRDI.

3.5
characteristic of a product
product characteristic
invariable property, characteristic of a product, whose value is fixed once the product is defined

Note 1 to entry: Changing the value of a characteristic of a product would mean changing the product.

EXAMPLE For a ball bearing, the inner diameter and the outer diameter are product characteristics.

Note 2 to entry: Adapted from ISO 13584-24:2003, definition 3.12.

3.6
class
abstraction of a set of similar products

Note 1 to entry: A product that complies with the abstraction defined by a class is called a class member.

Note 2 to entry: A class is an intentional concept that can take different extensional meanings in different contexts.

EXAMPLE The set of products used by a particular enterprise and the set of all ISO-standardized products are two examples of contexts. In these two contexts (the particular enterprise and ISO), the set of products that are considered as members of the *single ball bearing* class can be different, in particular because employees of each enterprise ignore a number of existing single ball bearing products.

Note 3 to entry: Classes are structured by class inclusion relationships.

Note 4 to entry: A class of products is a general concept as defined in ISO 1087-1. Thus, it is advisable that the rules defined in ISO 704 be used for defining the designation and definition attributes of classes of products.

Note 5 to entry: In the context of the ISO 13584 series, a class is either a characterization class, associated with properties and usable for characterizing products, or a categorization class, not associated with properties and not usable for characterizing products.

3.7
class inclusion relationship
relationship between classes that means inclusion of class members: if A is a superclass of A1 this means that, in any context, any member of A1 is also member of A

EXAMPLE 1 The set of products used by a particular enterprise and the set of all ISO-standardized products are two examples of contexts.

EXAMPLE 2 In any context, the class *capacitor* includes the class *electrolytic capacitor*.

Note 1 to entry: Class inclusion defines a hierarchical structure between classes.

Note 2 to entry: Class inclusion is a conceptual relationship that does not prescribe anything at the data representation level. Consequently, it does not prescribe any particular database schema or data model.

Note 3 to entry: In the model defined in this part of IEC 61360, the “is-a” relationship ensures class inclusion. This part of IEC 61360 recommends that the “case-of” relationship also ensure class inclusion.

Note 4 to entry: The class inclusion relationship is also called subsumption.

3.8
class member
product that complies with the abstraction defined by a class

3.9
class valued property
property that has one single value for a whole characterization class of products

Note 1 to entry: The value of a class valued property is not defined individually for every single product of a characterization class, but globally for the class itself.

Note 2 to entry: When all products from a characterization class of products have the same value for a particular property, defining this property as a class valued property permits to avoid duplication of the value for each instance.

Note 3 to entry: Class valued properties can also be used to capture some commonality between different characterization classes when such a commonality is not captured by the hierarchy structure.

3.10
common ISO13584/IEC61360 dictionary model
data model for product ontology, using the information modeling language EXPRESS, resulting from a joint effort between ISO/TC 184/SC 4/WG 2 and IEC SC3D

Note 1 to entry: Several levels of allowed implementations, known as conformance classes, are defined for the common ISO13584/IEC61360 dictionary model. Conformance class 1 consists of the various schemes documented in this part of IEC 61360 (that duplicate information contained in this standard), more the ISO13584_IEC61360_dictionary_aggregate_extension_schema documented in ISO 13584-25 (duplicated in IEC 61360-5). Other conformance classes are documented in ISO 13584-25 (conformance classes 2, 3 and 4).

Note 2 to entry: In the ISO 13584 standard series, each particular product ontology addressing a particular product domain and based on the common ISO13584/IEC61360 dictionary model is called a reference dictionary for that domain.

3.11
context dependent characteristic of product
property of a *product* whose value depends on some *context parameter*s

Note 1 to entry: For a given product, a context dependent characteristic is mathematically defined as a function whose domain is defined by some context parameters that define the product environment.

EXAMPLE For a *ball bearing*, the *life-tim*e is a context dependent characteristic that depends on the *radial load*, the *axial load* and the *rotational speed*.

Note 2 to entry: Adapted from ISO 13584-24:2003, definition 3.22.

3.12
context parameter
variable whose value characterizes the context in which a *product* is inserted

EXAMPLE 1 The *dynamic-load* applied to a *bearing* is a context parameter for this *bearing*.

EXAMPLE 2 The *ambient temperature* in which the *resistance* of a *resistor* is measured is a context parameter for this *resistor*.

Note 1 to entry: This definition supersedes the definition given in ISO 13584-24:2003, that was the following: “a variable of which the value characterizes the context in which it is intended to insert a *product”*.

Note 2 to entry: In the ISO 13584 standard series, a property value is represented as a data element type.

3.13
data element type
unit of data for which the identification, description and value representation have been specified

Note 1 to entry: In the ISO 13584 standard series, a property value is represented as a data element type.

3.14
dictionary data
set of data that represents product ontologies possibly associated with product categorizations

Note 1 to entry: It is advisable that dictionary data be exchanged using some conformance class of the common ISO/IEC dictionary model.

Note 2 to entry: This definition of dictionary data supersedes the previous definition from the first edition of IEC 61360-2 that was the following: “the set of data that describes hierarchies of characterization classes of products and properties of these products”.

3.15
dictionary element
set of attributes that constitutes the dictionary description of certain objects of the application domain

EXAMPLE 1 A dictionary compliant with this part of IEC 61360 provides for the description of classes, properties, information sources and datatypes.

EXAMPLE 2 A dictionary compliant with ISO 13584-24:2003 provides for the description of classes, properties, information sources, datatypes, tables, documents and program libraries.

3.16
family of products
set of products represented by the same characterization class

Note 1 to entry: This definition supersedes the definition given in ISO 13584-24:2003, that was the following: “a simple or generic family of parts”.

3.17
feature
aspect of a product that can be described by a characterization class and a set of property-value pairs

Note 1 to entry: In the real world, a feature instance only exists embedded within the product of which it is an aspect.

EXAMPLE 1 The head of a screw is a feature described by a head class and a number of head properties, which depends upon the head class. A screw head only exists when it belongs to a screw.

Note 2 to entry: Features are represented by means of **item_class** whose the **instance_sharable** attribute equals *false*.

Note 3 to entry: The **instance_sharable** attribute allows to specify the conceptual status of an item: either a stand-alone item (**instance_sharable** =*true*), or a feature (**instance_sharable** =*false*). It does not imply any constraint at the data representation level. In the common ISO13584/IEC61360 dictionary model, representing several real world instances that share the same EXPRESS representation by a single EXPRESS entity, or by several EXPRESS entities is considered as implementation dependant. There exist no mechanism for specifying whether data value of a feature instance may or may not be shared.

EXAMPLE 2 The same instance of a screw head class can be referenced by several instances of a screw class. It means that there exists several screw heads, but that all these screw heads have the same characterization class and the same set of property values. The **instance_sharable** attribute allows to specify that changing this instance of the screw head class would change several instances of the screw class.

3.18
imported property
property defined in a class that is selected by another class of the same or of a different reference dictionary, by means of the case-of relationship, to become applicable to the latter class

Note 1 to entry: Only properties that are visible and/or applicable in a class can be imported from this class.

Note 2 to entry: Importation between classes of different reference dictionaries allows reusing properties, defined for example in a standard reference dictionary, without redefining them.

Note 3 to entry: Importation between classes of the same reference dictionary acknowledges the fact that some products can perform several functions, requiring the capability to import property from several higher level classes.

Note 4 to entry: When it is imported in a new class, a property keeps its original identifier, thus all the attributes do not need to be duplicated.

Note 5 to entry: An imported property is applicable for the class where it is imported.

3.19
information
facts, concepts or instructions

[SOURCE: ISO 10303-1:1994, definition 3.2.20]

3.20
information model
formal model of a bounded set of facts, concepts or instructions to meet a specified requirement

[SOURCE: ISO 10303-1:1994, definition 3.2.21]

3.21
information supplier
supplier
organization that delivers an ontology, i.e. a data dictionary, or a supplier library that is responsible for its content

Note 1 to entry: This definition of supplier supersedes the definition of information supplier from ISO 13584-1:2001 that was the following: “organization that delivers a supplier library in the standard format defined in this International Standard and is responsible for its content”.

3.22
international registration data identifier
internationally unique identifier for a certain object of the application domain as defined in ISO/IEC 11179-5

Note 1 to entry: Only international registration data identifiers compliant with ISO/TS 29002-5 are used in the context of the ISO 13584 standard series.

Note 2 to entry: An international registration data identifier may be used for representing the content of a basic semantic unit that identifies a dictionary element as a string.

Note 3 to entry: An international registration data identifier may also be used for identifying the content of an attribute of a dictionary element.

EXAMPLE The unit of measure of a property, a value of a property or a constraint over a property may be identified by an IRDI.

3.23
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 to entry: This mechanism is usually called “inheritance”.

Note 2 to entry: In the common ISO13584/IEC61360 dictionary model, the is-a relationship can only be defined between characterization classes. It is advisable that it defines a single hierarchy and it ensures that both visible and applicable properties are inherited.

3.24
is-case-of relationship
case-of
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 to entry: The goal of the *case-of* relationship is to allow connecting together several class inclusion hierarchies while ensuring that referenced hierarchies can be updated independently.

Note 2 to entry: There is no constraint that the case-of relationship is intended to define single hierarchies.

Note 3 to entry: In the common ISO13584/IEC61360 dictionary model, the case-of relationship can in particular be used 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 e. g., A, and by case-of to B.

Note 4 to entry: The EXPRESS resource constructs for modeling the case-of relationships are defined in 4.5 and after.

3.25
item
thing that can be characterized by means of a characterization class to which it belongs and a set of property value pairs

Note 1 to entry: This definition supersedes the definition given in ISO 13584-24:2003, that was the following: “a thing that can be captured by a class structure and a set of properties”.

Note 2 to entry: In the ISO 13584 standard series, both products and features of products that correspond to composite properties are items.

3.26
leaf characterization class
characterization class that is not further specialized into more precise characterization classes

EXAMPLE Countersunk flat head screw with cross recess (type Y) and hexagon socket head cap screw with metric fine pitch thread are leaf characterization classes defined in ISO 13584-511.

3.27
non-leaf characterization class
characterization class that is further specialized into more precise characterization classes

EXAMPLE *Externally-threaded component* and *metric threaded bolt/screw* are non-leaf characterization classes defined in ISO 13584-511.

3.28
non-quantitative data element type
data element type that identifies or describes an object by means of codes, abbreviations, names, references or descriptions

3.29
part
material or functional element that is intended to constitute a component of different products

[SOURCE: ISO 13584-1:2001, definition 3.1.16]

3.30
parts library
computer-sensible product ontology and computer-sensible description of a set of products by means of references to this ontology

Note 1 to entry: This definition supersedes the definition given in the first edition of this part of IEC 61360, which was the following: “identified set of data and possibly programs which can generate information about a set of parts”.

3.31
product
thing or substance produced by a natural or artificial process

Note 1 to entry: In this part of IEC 61360, the term product is taken in its widest sense to include devices, systems and installations as well as materials, processes, software and services.

3.32
product categorization
part categorization
categorization
recursive partition of a set of products into subsets for a specific purpose

Note 1 to entry: Subsets which appear in a product categorization are called product categorization classes, or product categories.

Note 2 to entry: A product categorization is not a product ontology. It cannot be used for characterizing products.

Note 3 to entry: No property is associated with categorizations.

Note 4 to entry: 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 a product categorization that was developed for spend analysis.

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

3.33
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 to entry: No rule is given in this part of IEC 61360 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 to entry: There is no property associated with a categorization class.

3.34
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 a product characterization.

3.35
product characterization class
part characterization class
characterization class
class of products that fulfil the same function and that share common properties

Note 1 to entry: 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-DB. The second one is included in the first one.

3.36
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, of class relations and of identified properties

Note 1 to entry: 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 as identical. It is advisable that the rules defined in ISO 1087-1 be used for formulating designation 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 to entry: Ontologies are not concerned with words but with concepts, independent of any particular language.

Note 3 to entry: “Consensual” means that the conceptualization is agreed upon in some community.

Note 4 to entry: “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, making inferences.

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

Note 6 to entry: The data model for ontology recommended in this part of IEC 61360 is the common ISO13584/IEC61360 dictionary model, whose simplest version is documented in this part of IEC 61360. More complete versions are documented in ISO 13584-25 and IEC 61360-5 (conformance classes 1, 2, 3 and 4 of both documents).

Note 7 to entry: In this part of IEC 61360, each product ontology addressing a particular product domain compliant with the common ISO13584/IEC61360 dictionary model is called a reference dictionary for that domain.

EXAMPLE The reference dictionary for electric components, which is defined in IEC 61360-DB, is a product ontology for electric components compliant with the common ISO13584/IEC61360 dictionary model. It is agreed upon by all member bodies of IEC SC3D. A corporate reference dictionary is agreed upon by experts designated by management on behalf of the company.

3.37
property
defined parameter suitable for the description and differentiation of products

Note 1 to entry: A property describes one aspect of a given object.

Note 2 to entry: 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 IEC 61360.

Note 3 to entry: This part of IEC 61360 has identified three different kinds of properties: product characteristics, context parameters and context-dependent product characteristics.

Note 4 to entry: This definition of property supersedes the previous definition of the previous edition of this part of IEC 61360 that was the following: “an information that can be represented by a data element type”.

Note 5 to entry: In the ISO 13584 standard series, a property value is represented as a data element type.

3.38
property data type
allowed set of values of a property

3.39
property definition class
product characterization class in the context of which a product property is defined

Note 1 to entry: In the common ISO13584/IEC61360 dictionary model, each product property has one property definition class that defines its domain of application. The property is only meaningful for this class, and all its subclasses, and it is said to be *visible* over this domain.

EXAMPLE In ISO 13584-511, *wrenching height* has *nut* as its property definition class and *major diameter of external thread* has *metric external thread* as its property definition class.

3.40
quantitative data element type
data element type with a numerical value representing a physical quantity, a quantity of information or a count of objects

3.41
reference dictionary
product ontology compliant with the common ISO13584/IEC61360 dictionary model

Note 1 to entry: In the ISO 13584 standard series, a product ontology that addresses a particular product domain, based on the common ISO13584/IEC61360 dictionary model, is called a reference dictionary for that domain.

3.42
resource construct
collection of EXPRESS language entities, types, functions, rules and references that together define a valid description of data

Note 2 to entry: This definition is adapted from the definition of resource construct in ISO 10303-1:1994, i.e., “the collection of EXPRESS language entities, types, functions, rules and references that together define a valid description of product data”.

[SOURCE: ISO 13584-1:2001, definition 3.1.21, modified]

3.43
subclass
class that is one step below another class in a class inclusion hierarchy

Note 1 to entry: In the common ISO13584/IEC61360 dictionary model, class inclusion hierarchies are defined by the *is-a* relationship. They can also be established by the *case-of* relationships.

3.44
superclass
class that is one step above another class in a class inclusion hierarchy

Note 1 to entry: In the common ISO13584/IEC61360 dictionary model, class inclusion hierarchies are defined by the *is-a* relationship. They can also be established by the *case-of* relationships.

Note 2 to entry: In the common ISO13584/IEC61360 dictionary model, a class has at most one superclass specified by means of an *is-a* relationship.

3.45
supplier library
parts library of which the information supplier is different from the library user

Note 1 to entry: This definition supersedes the definition given in ISO 13584-1:2001, that was the following: “set of data, and possibly of programs, for which the supplier is defined and that describes in the standard format defined in this International Standard a set of products and/or a set of representation of products”.

3.46
visible property
property that has a definition meaningful in the scope of a given characterization class, but that does not necessarily apply to the various products belonging to this class

Note 1 to entry: Meaningful in the scope of a given characterization class means that a human observer is able to determine, for any product of the characterization class, whether the property applies, and, if it applies, to which product aspect it corresponds.

Note 2 to entry: The concept of a visible property allows sharing the definition of a property among product characterization classes where this property does not necessarily apply.

EXAMPLE The *non-threaded length* property is meaningful for any class of *screw* but it applies only to those screws that have a non-threaded part. It can be defined as visible at the *screw* level, while becoming applicable only in some subclasses.

Note 3 to entry: All the visible properties of a superclass that is a product characterization class are also visible properties for its subclasses.

Note 4 to entry: To facilitate integration of component libraries and electronic catalogues based on ISO 13584-24:2003 and ISO 13584-25, these parts of ISO 13584 request that only properties that are applicable to a class be used to characterize their instances in component libraries and electronic catalogues.

Note 5 to entry: This definition of a visible property supersedes the previous definition from ISO 13584-24:2003 that was the following: “a property that is defined for some class of products and that does not necessarily apply to the different products of this class of products”.

4 Overview of the common dictionary schema and compatibility with ISO13584_IEC61360_dictionary_schema

4.1 General

In the following subclauses the architecture of the common dictionary schema will be presented and it will be explained how the same information model has to be used in the International Standards to ensure their compatibility.

The common dictionary schema combines the requirements of IEC 61360 series and ISO 13584 series. Therefore, it contains resources to accommodate the specific requirements of both series of International Standards. These resources are provided either as optional capabilities or as subtypes of the types defined to fulfil the common requirements.

4.2 Use of the common dictionary schema to exchange IEC 61360-1 compliant data

The common dictionary schema to exchange IEC 61360-1 compliant data shall be used as follows:

a) the ISO 13584 series specific extensions to support multilingual capability are not required for the exchange of dictionary elements defined according to IEC 61360-1. However these extensions that are present_translations, translated_label and translated_text shall be used in the exchange structure for compatibility reasons;

b) if a component class has a superclass, the coded_name shall be defined as a value_code in the domain of the classifying data element type of the superclass;

c) if a classifying data element type exists within a specific component class, for each value in its domain a subclass and a term shall be defined;

d) a classifying data element type, optional in conformance class 2 in the common dictionary schema, shall always be provided for the component classes defined according to IEC 61360-1;

e) only SI units shall be used although the common dictionary schema enables the use of many kind of system units. When using this schema however for the exchange of IEC 61360 series compliant data, only SI shall be used for quantitative data element types.

4.3 Compatibility with ISO 13584-42

An implementation compliant with this part of IEC 61360 shall support all the entities, types and associated constraints that belong to the conformance class it claims to support. Therefore conformance to conformance class 1 of this part of IEC 61360 requires that all the entities, types and associated constraints defined in the common dictionary schema be supported. ISO 13584 data conforming to the common dictionary schema may thus be processed by an IEC 61360 implementation that conforms to conformance class 1 that includes all the features of conformance class 1. In ISO 13584, a specific conformance class 3 is intended to contain all the entities, types and associated constraints defined in the common dictionary schema. An ISO 13584 compliant implementation conforms to this conformance class shall therefore be able to support IEC data that belongs to conformance class 1 of this part of IEC 61360.

4.4 Naming correspondence between IEC 61360-1 and IEC 61360-2

Due to specific application restrictions – for example the EXPRESS language allows no spaces in entity names –, a number of similar 'EXPRESS names' are created by replacing the blank in a name by an underscore (for example preferred name is presented as preferred_name).

At other places names are used in the EXPRESS model that deviate from those used in IEC 61360-1. This is a consequence of the effort to reach one common EXPRESS information model together with Part Libraries.

The table below presents a help for matching the names used in the two parts of IEC 61360.

Table 1 – Cross refernce table

naming in 61360-2	naming in 61360-1
component_class	Component class
condition_DET	Condition data element type
dependent_P_DET	Data element type
det_classification	Data element type class
(DER)dic_identifier	Identifier
dic_value	Value
material_class	Material class
meaning	Value meaning
non_dependent_P_DET	Data element type
preferred_symbol	Preferred letter symbol

naming in 61360-2	naming in 61360-1
revision	Revision number
source_doc_of_definition	Source document of data element type definition
source_doc_of_definition	Source document of component class definition
synonymous_symbols	Synonymous letter symbol
unit	Unit of measure
value_code	Value code
version	Version number

4.5 Main structure of the common dictionary schema

This subclause explains the main resource constructs provided by the common dictionary schema:

- **dictionary_element** is any element defined in the dictionary;
- **supplier_element** captures the data of suppliers of dictionary elements (classes, properties, data types); class models the dictionary element of classes (families) which are described by properties;
- **property_DET** is the dictionary element of a property;
- **data_type** specifies the type of a property.

These parts of the dictionary schema are presented in more detail in clause 5: **ISO13584_IEC61360_dictionary_schema**.

In the presentation of the common dictionary schema, some overview diagrams are provided as planning models (see Figure 1 to Figure 11). These planning models use the EXPRESS-G graphical notation for the EXPRESS language. For clarification of the diagrams, some of the relationships that are defined in the EXPRESS model are omitted. Figure 1 below outlines as a planning model the main structure of the common dictionary schema. Most of these figures contain overview models (or planning models) but show only that level of detail which is appropriate at a certain place.

For clarification of the diagrams, some of the relationships that are defined in the EXPRESS model are omitted. Figure F.1 below outlines as a planning model the main structure of the common ISO13584/IEC61360 dictionary model.

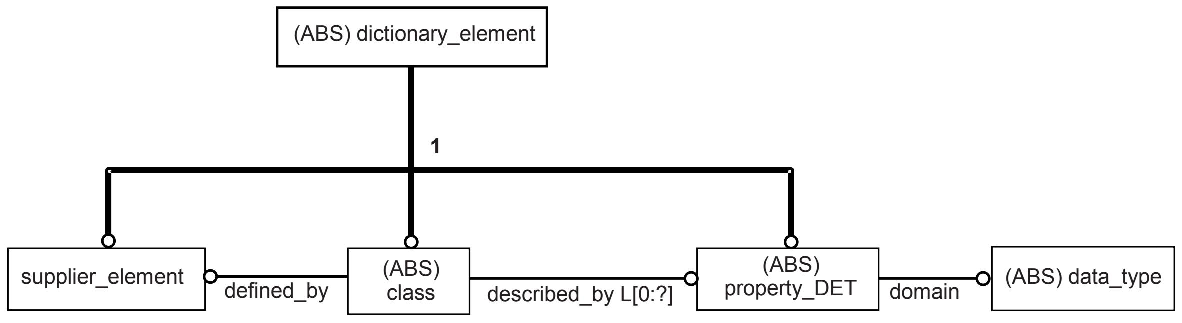

Figure 1 – Overview of the dictionary schema

5 ISO13584_IEC61360_dictionary_schema

5.1 General

This clause, which constitutes the main part of the common information model of ISO 13584-42 and IEC 61360 series, contains the full EXPRESS listing of the dictionary schema, annotated with comments and explanatory text. The order of text in this clause is determined primarily by the order imposed by the EXPRESS language, secondarily by importance.

5.2 Dictionary schema

In the first place, the schema needs to be declared.

EXPRESS specification:

```
*)
SCHEMA ISO13584_IEC61360_dictionary_schema;
(*
```

5.3 References to other schemata

This subclause contains references to other EXPRESS schemata that are used in the dictionary schema. Their source is indicated in the respective comment.

EXPRESS specification:

```
*)
REFERENCE FROM support_resource_schema(identifier, label, text);

REFERENCE FROM person_organization_schema(organization, address);

REFERENCE FROM measure_schema;

REFERENCE FROM ISO13584_IEC61360_language_resource_schema;

REFERENCE FROM ISO13584_IEC61360_class_constraint_schema;

REFERENCE FROM ISO13584_IEC61360_item_class_case_of_schema;

REFERENCE FROM ISO13584_external_file_schema
      (external_item,
      external_file_protocol,
      external_content,
      not_translatable_external_content,
      not_translated_external_content,
      translated_external_content,
      language_specific_content,
      http_file,
      http_class_directory,
      http_protocol);
(*
```

NOTE The schemata referenced above can be found in the following documents:

support_resource_schema	ISO 10303-41
person_organization_schema	ISO 10303-41
measure_schema	ISO 10303-41

ISO13584_IEC61360_language_resource_schema	IEC 61360-2
(which is duplicated for convenience in this document)	
ISO13584_IEC61360_class_constraint_schema	IEC 61360-2
(which is duplicated for convenience in this document)	
ISO13584_IEC61360_item_class_case_of_schema	IEC 61360-2
(which is duplicated for convenience in this document)	
ISO13584_external_file_schema	ISO 13584-24:2003

5.4 Constant definitions

This subclause contains constant definitions used later in type definitions (see 5.11).

EXPRESS specification:

```
*)
CONSTANT
     dictionary_code_len: INTEGER := 131;
     property_code_len: INTEGER := 35;
     class_code_len: INTEGER := 35;
     data_type_code_len:INTEGER := 35;
     supplier_code_len: INTEGER := 149;
     version_len: INTEGER := 10;
     revision_len: INTEGER := 3;
     value_code_len: INTEGER := 35;
     pref_name_len: INTEGER := 255;
     short_name_len: INTEGER := 30;
     syn_name_len: INTEGER := pref_name_len;
     DET_classification_len: INTEGER := 3;
     source_doc_len: INTEGER := 255;
     value_format_len: INTEGER := 80;
     sep_cv: STRING := '#';
     sep_id: STRING := '#';
END_CONSTANT;
(*
```

5.5 Identification of a dictionary

A **dictionary_identification** entity allows to identify unambiguously a particular version of a particular dictionary of a particular information supplier, standard or not. It contains a **code** defined by the dictionary supplier that identifies the dictionary, a **version** number and **revision** number that characterize a particular state of this dictionary.

NOTE 1 The case where dictionary version and revision should be incremented is defined in IEC 61360-1.

EXPRESS specification:

```
*)
ENTITY dictionary_identification;
     code: dictionary_code_type;
     version: version_type;
     revision: revision_type;
     defined_by: supplier_bsu;
DERIVE
     absolute_id: identifier :=
          defined_by.absolute_id + sep_id + code + sep_cv + version;
```

```
UNIQUE
     UR1: absolute_id;
END_ENTITY; -- dictionary_identification
(*
```

Attribute definitions:

code: the code that characterizes the dictionary.

version: the version number that characterizes the version of the dictionary.

revision: the revision number that characterizes the revision of the dictionary.

defined_by: the supplier who defines the dictionary.

absolute_id: the unique identification of the dictionary.

Formal propositions:

UR1: the dictionary identifier defined by the **absolute_id** attribute is unique.

Informal propositions:

IP1: when a dictionary is defined by a standard document that contains only one dictionary, the dictionary **code** shall be the standard number of the document that describes this dictionary if this document only defines one dictionary. It shall be the name defined for the pertinent dictionary in the document that describes it if this document defines several dictionaries. Unless otherwise specified, version shall be set to 1 and revision numbers shall be set to 0 for dictionaries defined by standard documents.

NOTE 2 Representation of the standard numbers of standard documents is specified in 5.1 and 5.2 of ISO 13584-26:2000.

5.6 Basic Semantic Units: defining and using the dictionary

5.6.1 Requirements for exchange

In the exchange of dictionary and part library data, it is customary to partition the data. For example, a dictionary could be updated with some classes that specify their superclass by a reference to a pre-existing class, or when the content of a library is exchanged, dictionary elements are only referenced and not included every time. It shall be possible to refer unambiguously and consistently to the dictionary data.

Thus, it is a clear requirement first, to be able to exchange pieces of data, and second, to have relationships between these pieces. This is depicted in Figure 2.

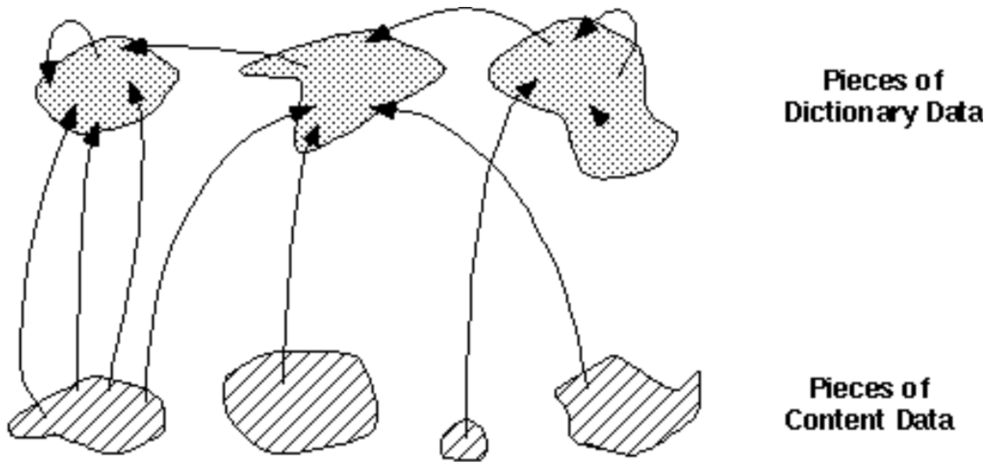

Figure 2 – Pieces of data with relationships

Every one of these pieces corresponds to a physical file (according to ISO 10303-21). EXPRESS (ISO 10303-11:2004) attributes can only contain references to data within the same physical file. Thus it is impossible to use EXPRESS attributes directly to implement inter-piece references.

5.6.2 Three levels architecture of the dictionary data

5.6.2.1 General

In this clause the concept of **basic_semantic_unit** (BSU) is introduced as a means to implement these inter-piece references. A BSU provides a universally unique identification for dictionary descriptions. This is depicted in Figure 3

Assume some piece of content wants to refer a certain dictionary description.

EXAMPLE 1 To convey the value of a property of a component.

It does this by referring to a basic semantic unit through the attribute dictionary_definition.

A dictionary description (dictionary_element) refers to a basic semantic unit through the attribute identified_by. From the correspondence of the absolute identifiers of the basic semantic units this indirect relation is established.

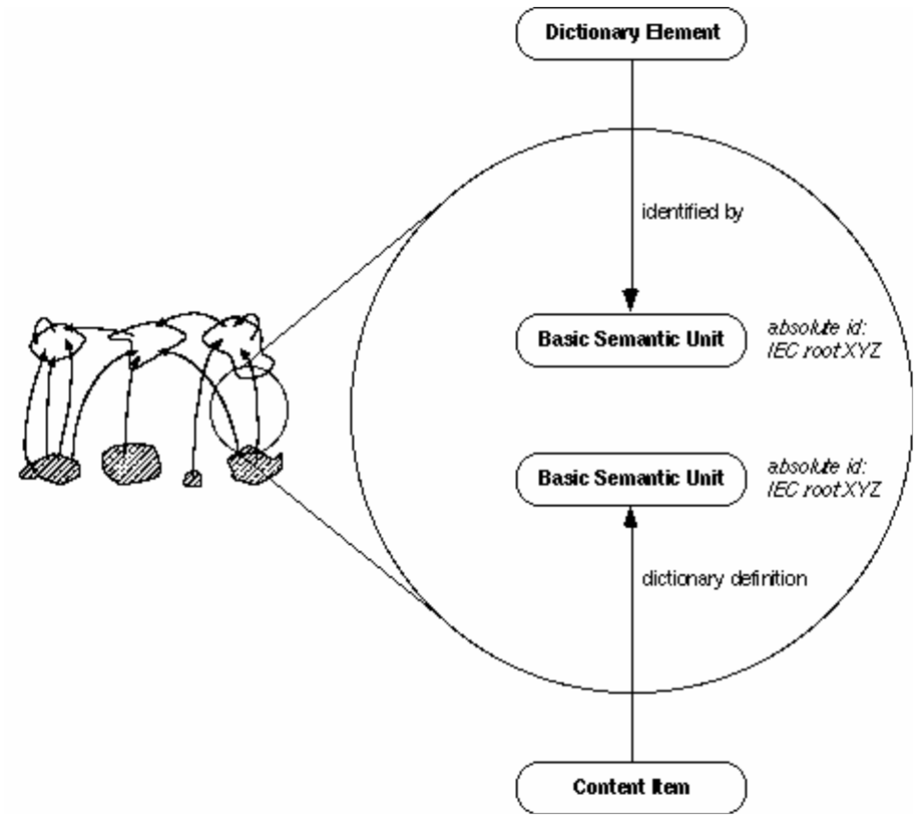

Figure 3 – Implementation of "inter-piece" relationships using basic semantic units

Note that:

- both dictionary element and content item can be present in the same physical file, but need not be;
- the dictionary element does not need to be present for the exchange of some content item referring to it. In this case it is assumed to be present in the dictionary of the target system already. Conversely, dictionary data can be exchanged without any content data;
- the basic semantic unit can be one single instance in the case where both dictionary element and content item instances are in the same physical file;
- the same mechanism applies also to references between various dictionary elements

EXAMPLE 2 Between a class of components and the associated **property_DET**s.

A BSU provides a reference to a dictionary description in any place where this is needed.

EXAMPLE 3 Dictionary delivery, update delivery, library delivery, component data exchange.

The data associated with a property could be exchanged as a pair (**property_BSU**, <value>).

Figure 3 outlines the implementation of this general mechanism.

5.6.2.2 Basic_semantic_unit

A **basic_semantic_unit** is a unique identification of a **dictionary_element**. BSU is the abbreviation of basic semantic unit.

<u>EXPRESS specification:</u>

```
*)
ENTITY basic_semantic_unit
ABSTRACT SUPERTYPE OF(ONEOF(
     supplier_BSU,
     class_BSU,
     property_BSU,
     data_type_BSU,
     supplier_related_BSU,
     class_related_BSU));
     code: code_type;
     version: version_type;
DERIVE
     dic_identifier: identifier := code + sep_cv + version;
INVERSE
     definition: SET [0:1] OF dictionary_element
          FOR identified_by;
     referenced_by: SET [0:1] OF content_item
          FOR dictionary_definition;
END_ENTITY; -- basic_semantic_unit
(*
```

Attribute definitions:

code: the code assigned to identify a certain dictionary element.

version: the version number of a certain dictionary element.

dic_identifier: the full identification, consisting of concatenation of code and version.

definition: a reference to the dictionary element identified by this BSU. If not present in some exchange context, it is assumed to be present in the dictionary of the target system already.

referenced_by: items making use of the dictionary element associated with this BSU.

5.6.2.3 Dictionary_element

A **dictionary_element** is a full definition of the data required to be captured in the semantic dictionary for some concepts. For every concept, a separate subtype is to be used. The **dictionary_element** is associated with a **basic_semantic_unit** (BSU) that serves to uniquely identify this definition in the dictionary.

By including the version attribute in the **basic_semantic_unit** entity, it forms part of the identification of a dictionary element (in contrast to the **revision** and **time_stamps** attributes).

EXPRESS specification:

```
*)
ENTITY dictionary_element
ABSTRACT SUPERTYPE OF(ONEOF(
     supplier_element,
     class_and_property_elements,
     data_type_element));
     identified_by: basic_semantic_unit;
     time_stamps: OPTIONAL dates;
```

```
        revision: revision_type;
        administration: OPTIONAL administrative_data;
        is_deprecated: OPTIONAL BOOLEAN;
        is_deprecated_interpretation: OPTIONAL note_type;
    WHERE
        WR1: NOT EXISTS (SELF.is_deprecated)
                 OR EXISTS (SELF.is_deprecated_interpretation);
    END_ENTITY; -- dictionary_element
    (*
```

Attribute definitions:

identified_by: the BSU identifying this dictionary element.

time_stamps: the optional dates of creation and update of this dictionary element.

revision: the revision number of this dictionary element.

NOTE 1 The type of the **identified_by** attribute will be redefined later to **property_BSU** and **class_BSU** and will then be used to encode together with the code attribute of the BSUs the "Code" attribute for properties and classes respectively. It will also be used to encode the "Version Number" attribute for properties and classes respectively.

NOTE 2 The **time_stamps** attribute will be used as a starting point to encode in the **dates** entity the property and class attributes "Date of Original Definition", "Date of Current Version" and "Date of Current Revision" (see 5.11.3.2).

NOTE 3 The **revision** attribute will be used to encode the property and class attribute "Revision Number".

administration: optional information on the life cycle of the **dictionary_element.**

NOTE 4 The **administration** attribute will be used to represent the information related to the configuration management and translation history.

is_deprecated: an optional Boolean. When true, it specifies that the **dictionary_element** shall no longer be used.

is_deprecated_interpretation: specifies the deprecation rationale and how instance values of the deprecated element, and of its corresponding **BSU**, should be interpreted.

Formal propositions:

WR1: when **is_deprecated** exists, **is_deprecated_interpretation** shall exist.

Informal propositions:

IP1: instance values of i**s_deprecated_interpretation** element shall be defined at the time where deprecation decision was taken.

Figure 4 presents a planning model of the relationship between basic semantic unit and the dictionary element.

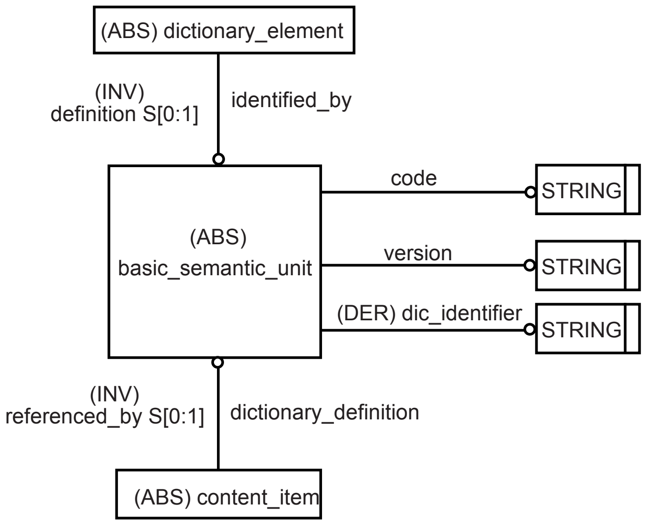

Figure 4 – Relationship between basic semantic unit and dictionary element

5.6.2.4 Content_item

A **content_item** is a piece of data referring to its description in the dictionary. It shall be subtyped.

EXPRESS specification:

```
*)
ENTITY content_item
ABSTRACT SUPERTYPE;
     dictionary_definition: basic_semantic_unit;
END_ENTITY; -- content_item
(*
```

Attribute definitions:

dictionary_definition: the basic semantic unit to be used for referring to the definition in the dictionary.

5.6.3 Overview of basic semantic units and dictionary elements

For every kind of dictionary data, a pair of **basic_semantic_unit** and **dictionary_element** subtypes shall be defined. Figure 5 outlines, as a planning model, the basic semantic units (BSU) and dictionary elements defined later. Note that the relationship between BSU and dictionary elements is redefined for each type of data, so that only corresponding pairs can be related. This is not graphically depicted here, however.

Every kind of dictionary data is treated in one of the following subclauses:

– for suppliers see 5.7;
– for classes see 5.8;
– for properties / data element types see 5.9;
– for data types see 5.10.

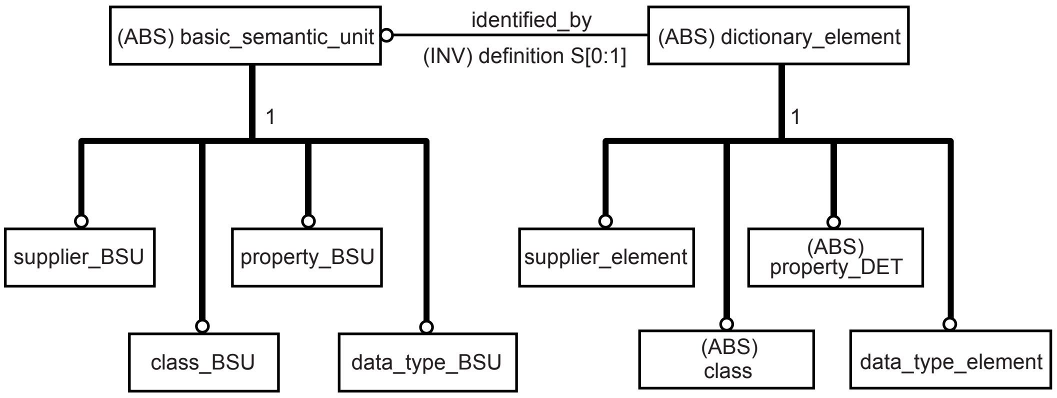

Figure 5 – Current BSUs and dictionary elements

5.6.4 Identification of dictionary elements: three levels structure

The absolute identification of basic semantic units is based on the following three levels structure:

- supplier (of dictionary data);
- supplier-defined dictionary element (any supplier-defined dictionary element defined in the model; in this document supplier-defined dictionary element are **property_DET** and **data_type_element,** but there are provisions to extend this mechanism to other items);
- version of the supplier-defined dictionary element.

An absolute identification can be achieved by concatenation of the applicable code for each level.

NOTE The structure on this absolute identification is different from the structure defined in edition 1 of IEC 61360-2 (duplicated for convenience in ISO 13584-42:1998). In the previous edition, the absolute identification of any **dictionary_element** associated with a **name_scope** (including **property_DET** and **data_type_element**) consisted of: supplier code + class code (corresponding to the **name_scope** class) + dictionary element code + dictionary element version. In this edition, the class code has been removed. Thus, the dictionary element code is unique, for the same type of dictionary element, over all the classes defined by the same supplier. For existing reference dictionaries, registration authorities, maintenance authorities or standardization groups in charge of standard dictionaries should ensure this unicity, possibly by defining new codes prefixed by **name_scope** class codes.

This identification scheme is appropriate within a multi-supplier context. If in a certain application area, only data of one single (data-) supplier are relevant, the corresponding parts of the identification, that are then constant, can be eliminated. For the purpose of exchange, however, all the levels shall be present, to avoid clashes of identifiers.

This identification scheme is described formally in the **absolute_id** attribute of the xxx_BSU entities defined from 5.7 through 5.12.

5.6.5 Extension possibilities for other types of data

5.6.5.1 General

The BSU – dictionary element mechanism is very general and not limited to the four kinds of data used here (see Figure 5). This clause specifies some facilities that allow for extensions for other kinds. Depending on whether the scope of the identifier is given by a class or a supplier, the corresponding **xxx_related_BSU** entity has to be subtyped. It is necessary to redefine the **identified_by** attribute of the entity **dictionary_element** (as is done in 5.7.3 through 5.10 or the current kinds of data).

5.6.5.2 Supplier_related_BSU

The **supplier_related_BSU** provides for the dictionary elements to be associated with suppliers.

EXAMPLE For ISO 13584 series: program libraries.

EXPRESS specification:

```
    *)
    ENTITY supplier_related_BSU
    ABSTRACT SUPERTYPE
    SUBTYPE OF(basic_semantic_unit);
    END_ENTITY; -- supplier_related_BSU
    (*
```

5.6.5.3 Class_related_BSU

The **class_related_BSU** provides for the dictionary elements to be associated with classes.

EXAMPLE For ISO 13584 tables, documents, etc.

EXPRESS specification:

```
    *)
    ENTITY class_related_BSU
    ABSTRACT SUPERTYPE
    SUBTYPE OF(basic_semantic_unit);
    END_ENTITY; -- class_related_BSU
    (*
```

5.6.5.4 Supplier_BSU_relationship

The **supplier_BSU_relationship** is a provision for association of BSUs with suppliers.

EXPRESS specification:

```
    *)
    ENTITY supplier_BSU_relationship
    ABSTRACT SUPERTYPE;
         relating_supplier: supplier_element;
         related_tokens: SET [1:?] OF supplier_related_BSU;
    END_ENTITY; -- supplier_BSU_relationship
    (*
```

Attribute definitions:

relating_supplier: the **supplier_element** that identifies the data supplier.

related_tokens: the set of dictionary elements associated to the supplier identified by the **relating_supplier** attribute.

5.6.5.5 Class_BSU_relationship

The **class_BSU_relationship** entity is a provision for association of BSUs with classes.

EXPRESS specification:

```
*)
ENTITY class_BSU_relationship
ABSTRACT SUPERTYPE;
      relating_class: class;
      related_tokens: SET [1:?] OF class_related_BSU;
END_ENTITY; -- class_BSU_relationship
(*
```

Attribute definitions:

relating_class: the **class** that identifies the dictionary element.

related_tokens: the set of dictionary elements associated to the class identified by the **relating_class** attribute.

5.7 Supplier data

5.7.1 General

This clause contains definitions for the representation of data about a supplier itself. In a multi-supplier environment it is necessary to be able to identify the source of a certain dictionary element. Figure 6 presents a planning model of the data associated with suppliers, followed by the EXPRESS definition.

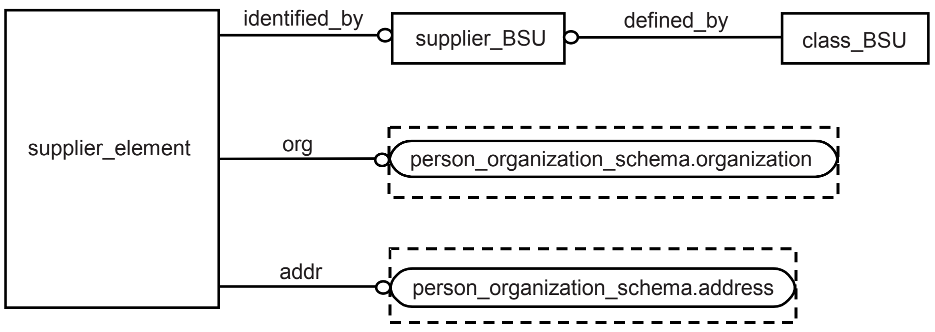
supplier_element
identified_by
supplier_BSU
defined_by
class_BSU
org
person_organization_schema.organization
addr
person_organization_schema.address

Figure 6 – Overview of supplier data and relationships

5.7.2 Supplier_BSU

The **supplier_BSU** entity provides for unique identification of information suppliers.

EXPRESS specification:

```
*)
ENTITY supplier_BSU
SUBTYPE OF(basic_semantic_unit);
      SELF\basic_semantic_unit.code: supplier_code_type;
DERIVE
      SELF\basic_semantic_unit.version: version_type := '1';
      absolute_id: identifier := SELF\basic_semantic_unit.code;
UNIQUE
      UR1: absolute_id;
```

```
END_ENTITY; -- supplier_BSU
(*
```

Attribute definitions:

code: the supplier's code assigned according to ISO 13584-26.

version: the version number of a supplier code shall be equal to 1.

absolute_id: the absolute identification of the supplier.

Formal propositions:

UR1: the supplier identifier defined by the **absolute_id** attribute is unique.

5.7.3 Supplier_element

The **supplier_element** entity gives the dictionary description of suppliers.

EXPRESS specification:

```
*)
ENTITY supplier_element
SUBTYPE OF(dictionary_element);
     SELF\dictionary_element.identified_by: supplier_BSU;
     org: organization;
     addr: address;
INVERSE
     associated_items: SET [0:?] OF supplier_BSU_relationship
          FOR relating_supplier;
END_ENTITY; -- supplier_element
(*
```

Attribute definitions:

identified_by: the **supplier_BSU** used to identify this **supplier_element**.

org: the organizational data of this supplier.

addr: the address of this supplier.

associated_items: allows access to other kinds of data via the BSU mechanism.

EXAMPLE Program library in ISO 13584-24:2003.

5.8 Class data

5.8.1 General

This clause contains definitions for the representation of dictionary data of classes.

Figure 7 outlines, as a planning model, the data associated with classes and their relationship to other dictionary elements.

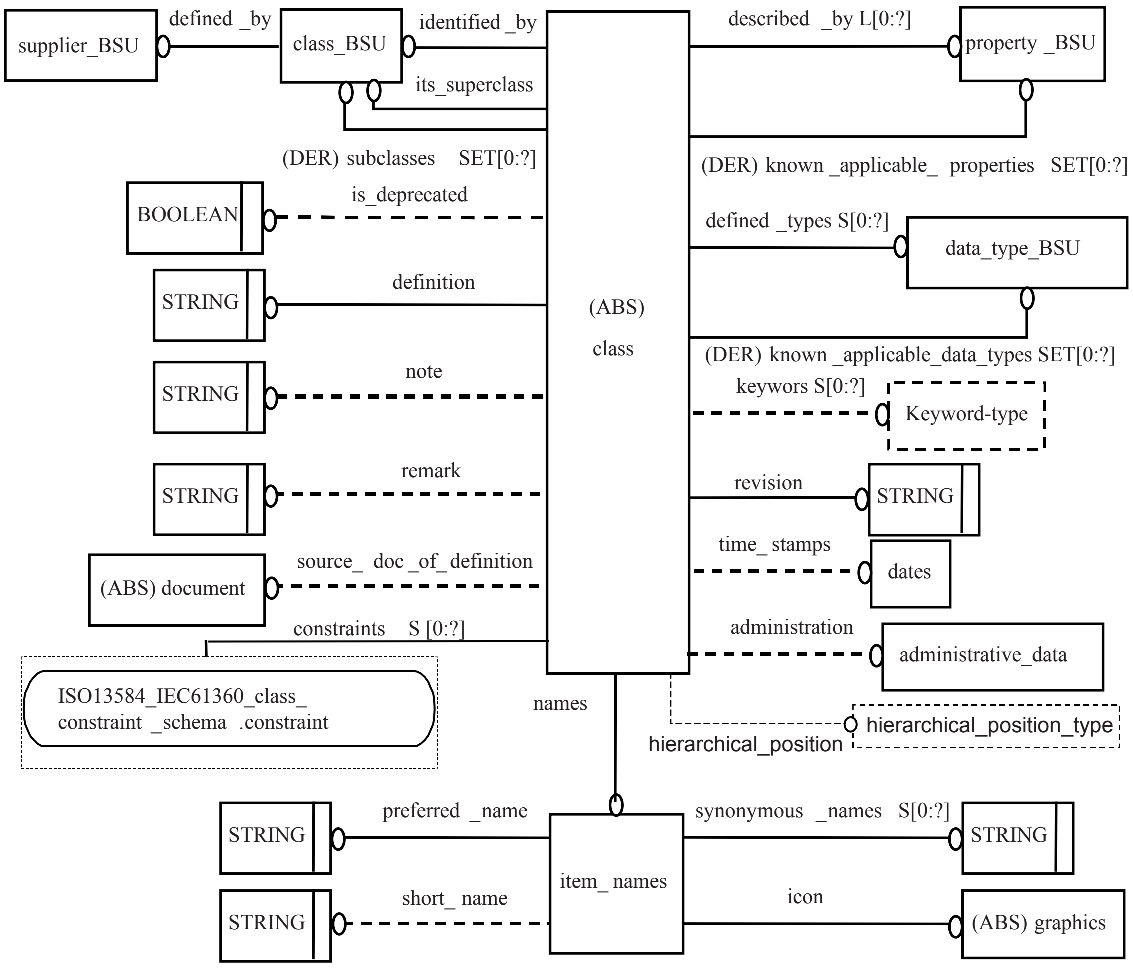

Figure 7 – Overview of class data and relationships

As indicated in the Figure 7 with the **its_superclass** attribute, classes form an inheritance tree. It is important to note that throughout this document, the terms "inheritance" and "to inherit" stand for this relationship between classes (defined in the dictionary), although EXPRESS has an inheritance concept, too. These shall be clearly distinguished to avoid misunderstandings.

The dictionary data for classes (as shown in Figure 7) is spread over three inheritance levels:

– class_and_property_elements defines data common to both classes and property_DETs;

– class allows for other kinds of classes to be specified later;

EXAMPLE Other subtypes of **class**es, in particular **functional_view_class**, **functional_model_class** and **fm_class_view_of** are specified in ISO13584-24:2003. They do not characterize products, but they provide for exchanging particular representations of product, e. g., geometrical representations.

– **item_class** and **categorization_class** are the entities that hold data of different classes of application domain objects.

NOTE 1 Two subtypes of **item_class**, named **component_class** and **material_class**, were defined within the dictionary model of the first edition of this IEC 61360. These subtypes are deprecated, and they are removed from this part of IEC 61360.

NOTE 2 The following changes ensure that the class definitions of a dictionary conforming with the first edition of IEC 61360-2 conforms to this edition: (1) replace **component_class** and **material_class** by **item_class** throughout the reference dictionary; (2) add to each new **item_class** class the **instance_sharable** attribute, the value of which being true; (3) add for each new **item_class** class the optional **hierarchical_position** attribute without setting any value; (4) add for each new **item_class** class the **keywords** attribute, the value of which being an empty collection.

NOTE 3 Another subtype of **item_class**, named **feature_class,** was provided in ISO 13584-24:2003. This subtype is also deprecated and its usage is not allowed in new implementations of this part of IEC 61360.

NOTE 4 The following changes ensure that the class definitions of a dictionary conforming with ISO 13584-25 conforms to this part of IEC 61360: (1) replace **feature_class** by **item_class** throughout the reference dictionary; (2) add to each new **item_class** class the **instance_sharable** attribute, the value of which being false; (3) add for each new **item_class** class the optional **hierarchical_position** attribute without setting any value; (4) add for each new **item_class** class the **keywords** attribute, the value of which being an empty collection.

5.8.2 Structural detail

5.8.2.1 Class_BSU

The **class_BSU** entity provides for the identification of classes.

<u>EXPRESS specification:</u>

```
*)
ENTITY class_BSU
SUBTYPE OF(basic_semantic_unit);
      SELF\basic_semantic_unit.code: class_code_type;
      defined_by: supplier_BSU;
DERIVE
      absolute_id: identifier
           := defined_by.absolute_id + sep_id + dic_identifier;
      known_visible_properties: SET [0:?]OF property_BSU
           := compute_known_visible_properties(SELF);
      known_visible_data_types: SET [0:?]OF data_type_BSU
           := compute_known_visible_data_types(SELF);
INVERSE
      subclasses: SET [0:?] OF class FOR its_superclass;
      added_visible_properties: SET [0:?] OF property_BSU
           FOR name_scope;
      added_visible_data_types: SET [0:?] OF data_type_BSU
           FOR name_scope;
UNIQUE
      UR1: absolute_id;
END_ENTITY; -- class_BSU
(*
```

<u>Attribute definitions</u>:

code: the code assigned to this class by its supplier.

defined_by: the supplier defining this class and its dictionary element.

absolute_id: the unique identification of this class.

known_visible_properties: the set of **property_BSU**s that refer to the class as their **name_scope** attribute or to any known super-class of this class and that are therefore visible for the class (and any of its subclass).

NOTE 1 When some class **dictionary_definition** is not present in the same exchange context (a PLIB exchange context is never assumed to be complete), the super-class of a class may not be known. Therefore the properties defined as visible by this super-class do not belong to the **known_visible_properties** attribute. Only on the receiving system all the **dictionary_definition**s of the BSUs are required to be available. Therefore, on the receiving system, the **known_visible_properties** attribute contains all properties visible for the class.

known_visible_data_types: the set of **data_type_BSU**s that refer to the class as their **name_scope** attribute or to any known super-class of this class and that are therefore visible for the class (and any of its subclass).

NOTE 2 When some class **dictionary_definition** is not present in the same exchange context (a PLIB exchange context is never assumed to be complete), the super-class of a class may not be known. Therefore the **data_type**s defined as visible by this super-class do not belong to the **known_visible_data_types** attribute. Only on the receiving system all the **dictionary_definition**s of the BSUs are required to be available. Therefore, on the receiving system, the **known_visible_data_types** attribute contains all **data_type**s visible for the class.

subclasses: the set of classes specifying this class as their superclass.

added_visible_properties: the set of **property_BSU**s that refer to the class as their **name_scope** and that are therefore visible for the class (and any of its subclass).

NOTE 3 Only the **property_BSU**s that belongs to the same exchange context are referenced by this inverse attribute. On the receiving system they may already exit other **property_BSU**s that refer to this class (a PLIB exchange context is never assumed to be complete).

NOTE 4 The **added_visible_properties** attribute will be used to encode the class attribute "Visible properties".

added_visible_data_types: the set of **data_type_BSU**s that refer to the class as their **name_scope** and that are therefore visible for the class (and any of its subclass).

NOTE 5 Only the **data_type_BSU**s that belongs to the same exchange context are referenced by this inverse attribute. On the receiving system they may already exits other **data_type_BSU**s that refer to this class (a PLIB exchange context is never assumed to be complete).

NOTE 6 The **added_visible_data_types** attribute will be used to encode the class attribute "Visible types".

Formal propositions:

UR1: the concatenation of supplier code and class code is unique.

5.8.2.2 Class_and_property_elements

The **class_and_property_elements** entity captures the attributes that are common to both **class**es and **property_DET**s.

EXPRESS specification:

```
*)
ENTITY class_and_property_elements
ABSTRACT SUPERTYPE OF(ONEOF(
      property_DET,
      class))
SUBTYPE OF(dictionary_element);
      names: item_names;
      definition: definition_type;
      source_doc_of_definition: OPTIONAL document;
      note: OPTIONAL note_type;
      remark: OPTIONAL remark_type;
END_ENTITY; -- class_and_property_elements
(*
```

Attribute definitions:

names: the names describing this dictionary element.

definition: the text describing this dictionary element.

source_doc_of_definition: the source document of this textual description.

note: further information on any part of the dictionary element, which is essential to the understanding.

remark: explanatory text further clarifying the meaning of this dictionary element.

NOTE 1 The **names** attribute will be used as a starting point to encode in the **item_names** entity the property and class attributes "Preferred Name", "Short Name" and "Synonymous Name".

NOTE 2 The **definition** attribute will be used to encode the property attribute "Definition" and the class attribute "Definition".

NOTE 3 The **source_of_doc_definition** attribute will be used to encode the property attribute "Source document of definition" and the class attribute "Source document of definition".

NOTE 4 The **note** attribute will be used to encode the property and class attribute "Note".

NOTE 5 The **remark** attribute will be used to encode the property and class attribute "Remark".

5.8.2.3 Class

The **class** entity is an abstract resource for all kinds of classes.

EXPRESS specification:

```
*)
ENTITY class
ABSTRACT SUPERTYPE OF( ONEOF (item_class, categorization_class))
SUBTYPE OF(class_and_property_elements);
     SELF\dictionary_element.identified_by: class_BSU;
     its_superclass: OPTIONAL class_BSU;
     described_by: LIST [0:?] OF UNIQUE property_BSU;
     defined_types: SET [0:?] OF data_type_BSU;
     constraints: SET [0:?] OF constraint_or_constraint_id;
     hierarchical_position: OPTIONAL hierarchical_position_type;
     keywords: SET [0:?] OF keyword_type;
     sub_class_properties: SET [0:?] OF property_BSU;
     class_constant_values: SET [0:?] OF class_value_assignment;
DERIVE
     subclasses: SET [0:?] OF class := identified_by.subclasses;
     known_applicable_properties: SET [0:?] OF property_BSU
          := compute_known_applicable_properties(
               SELF\dictionary_element.identified_by);
     known_applicable_data_types: SET [0:?] OF data_type_BSU
          := compute_known_applicable_data_types(
               SELF\dictionary_element.identified_by);
     known_property_constraints: SET [0:?] OF property_constraint
          := compute_known_property_constraints(
               [SELF\dictionary_element.identified_by]);
INVERSE
     associated_items: SET [0:?] of class_BSU_relationship
          FOR relating_class;
WHERE
     WR1: acyclic_superclass_relationship(SELF.identified_by, []);
     WR2: NOT all_class_descriptions_reachable(
          SELF\dictionary_element.identified_by)
          OR (list_to_set(SELF.described_by) <=
          SELF\dictionary_element.identified_by
          \class_BSU.known_visible_properties);
     WR3: NOT all_class_descriptions_reachable(
          SELF\dictionary_element.identified_by)
          OR (SELF.defined_types <=
          SELF\dictionary_element.identified_by
          \class_BSU.known_visible_data_types);
     WR5: NOT all_class_descriptions_reachable(
          SELF\dictionary_element.identified_by)
          OR (QUERY (cdp <* described_by
          | (SIZEOF (cdp\basic_semantic_unit.definition)=1)
          AND (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
          +'.DEPENDENT_P_DET') IN TYPEOF
          (cdp\basic_semantic_unit.definition[1]))
          AND NOT
          (cdp\basic_semantic_unit.definition[1].depends_on
          <= known_applicable_properties))=[]);
     WR6: check_datatypes_applicability(SELF);
     WR7: QUERY (cons <* constraints
          | ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
          +'.INTEGRITY_CONSTRAINT' IN TYPEOF (cons))
          AND (SIZEOF (cons\property_constraint.constrained_property
          .definition) =1)
          AND NOT correct_constraint_type(
          cons\integrity_constraint.redefined_domain,
          cons\property_constraint.constrained_property
          .definition[1].domain)) = [];
     WR8: QUERY (cons <* constraints
          | (('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
          +'.CONFIGURATION_CONTROL_CONSTRAINT') IN TYPEOF (cons))
          AND NOT correct_precondition (cons, SELF)) = [];
     WR9: NOT all_class_descriptions_reachable(
          SELF\dictionary_element.identified_by)
          OR (QUERY (cons <* constraints
          | (('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
          +'.PROPERTY_CONSTRAINT') IN TYPEOF (cons))
          AND NOT
          ((cons\property_constraint.constrained_property
          IN SELF\dictionary_element.identified_by
          \class_BSU.known_visible_properties)
          OR (cons\property_constraint.constrained_property
          IN known_applicable_properties)))=[]);
     WR10: (SIZEOF( QUERY (lab <* keywords
          | ('ISO13584_IEC61360_DICTIONARY_SCHEMA'
          +'.LABEL_WITH_LANGUAGE') IN TYPEOF (lab)))
          = SIZEOF( keywords))
          OR (SIZEOF (QUERY (lab <* keywords
          | ('ISO13584_IEC61360_DICTIONARY_SCHEMA'
          +'.LABEL_WITH_LANGUAGE') IN TYPEOF (lab)))
          = SIZEOF( keywords));
     WR11: (('ISO13584_IEC61360_ITEM_CLASS_CASE_OF_SCHEMA'
          + '.A_PRIORI_SEMANTIC_RELATIONSHIP')
          IN TYPEOF (SELF)) OR
          ( QUERY(p <* sub_class_properties
          | NOT(p IN SELF.described_by)) = []);
    WR12: NOT all_class_descriptions_reachable(SELF.identified_by) OR
          (QUERY(va <* class_constant_values |
          NOT is_class_valued_property(
          va.super_class_defined_property, SELF.identified_by)) = []);
    WR13: QUERY(val <* SELF.class_constant_values
          | QUERY (v <* class_value_assigned (
          val.super_class_defined_property, SELF.identified_by)
          | val.assigned_value <> v) <>[]) = [];
END_ENTITY; -- class
(*
```

Attribute definitions:

identified_by: the **class_BSU** identifying this class.

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

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

NOTE 1 A property may also be applicable to a class when this property is imported from another class through an **a_priori_semantic_relationship** as defined in 8.5 of this part of IEC 61360. Therefore the properties referenced by the **described_by** attribute do not define all the applicable properties for a class.

NOTE 2 The list order is the presentation order of the properties suggested by the supplier.

NOTE 3 A property that is a context dependent property (**context_dependent_P_DET**) may become applicable to a class only if all the context parameters (**condition_DET**) on which its value depends are also applicable to this class. This is stated in where rule 5 (WR5).

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

NOTE 4 A **data_type** may also be applicable to a class when this **data_type** is imported from another class through an **a_priori_semantic_relationship** as defined in 8.5 of this part of IEC 61360. Therefore the data types referenced by the **defined_types** attribute do not define all the applicable data types for a class.

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

NOTE 5 Each constraint in the **constraints** attribute should be fulfilled by class instances. Thus the **constraints** attribute is a conjunction of constraints.

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

NOTE 6 This kind of coded name is used in particular in product categorization hierarchies for representing the class inclusion structure through some coding conventions.

EXAMPLE 1 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*.

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

sub_class_properties: declares properties as class-valued, i.e. in subclasses one single value will be assigned per class. See 5.9.6.

class_constant_values: assignments in the current class for class-valued properties declared in superclasses. See 5.9.6.

subclasses: the set of classes specifying this class as their superclass.

known_applicable_properties: the **property_BSU**s that are referenced by the class or any of its known super-class(es) by their **described_by** attribute and that are therefore applicable to this class (and to any of its subclasses).

NOTE 7 When some class **dictionary_definition** is not present in the same exchange context (a PLIB exchange context is never assumed to be complete), the super-class of a class may not be known. Therefore the properties defined as applicable by this super-class do not belong to the **known_applicable_properties** attribute. Only on the receiving system all the **dictionary_definition**s of the **BSU**s are required to be available. Therefore, on the receiving system, the **known_applicable_properties** attribute contains all the properties that are applicable to a class by virtue of being referenced by a **described_by** attribute.

known_applicable_data_types: the **data_type_BSU**s that are referenced by the class or any of its known super-class(es) by their **defined_types** attribute and that are therefore applicable to this class (and to any of its subclasses).

NOTE 8 When some class **dictionary_definition** is not present in the same exchange context (a PLIB exchange context is never assumed to be complete), the super-class of a class may not be known. Therefore the **data_type**s defined as applicable by this super-class do not belong to the **known_applicable_data_types** attribute. Only on the receiving system all the **dictionary_definition**s of the **BSU**s are required to be available. Therefore, on the receiving system, the **known_applicable_data_types** attribute contains all the **data_type**s that are applicable to a class by virtue of being referenced by a **defined_types** attribute.

known_property_constraints: the **constraint**s over a property that are referenced by the class or any of its known is-a superclass by their **constraint**s attribute, or, in case of a class that is a subtype of **a_priori_semantic_relationship,** by its **referenced_constraints** attribute.

associated_items: allows to access other kinds of data using the BSU mechanism.

<u>Formal propositions:</u>

WR1: the inheritance structure defined by the class hierarchy does not contain cycles.

WR2: only those properties that are visible for a class may become applicable to this class by virtue of being referenced by the **described_by** attribute.

WR3: only those data types that are visible for a class may become applicable to this class by virtue of being referenced by the **defined_types** attribute.

WR4: only those properties that are not applicable for a class by inheritance may become applicable to this class by virtue of being referenced by the **described_by** attribute.

WR5: only context dependent properties (**dependent_P_DET**) whose all context parameters (**condition_DET**) are applicable in to class may become applicable for this class by virtue of being referenced by its **described_by** attribute.

WR6: only those data types that are not applicable for a class by inheritance may become applicable to this class by virtue of being referenced by the **defined_types** attribute.

NOTE 9 The **its_superclass** attribute will be used to encode the class attribute "Superclass".

NOTE 10 The **described_by** attribute provides the encoding for the "Applicable Properties" of a class.

NOTE 11 The **defined_types** attribute is used to encode the "Applicable Types" attribute of a class.

WR7: the set of constraints that are property constraints shall define restrictions that are compatible with the domain of values of the properties to which they apply.

WR8: all the properties referenced in the precondition of a **configuration_control_constraint** shall be applicable to the class.

WR9: all the properties referenced in the **constraint** attribute shall be either visible or applicable to the class.

WR10: either all **keyword**s are represented as **label_with_language**s or all are represented as **label**s.

WR11: if the **class** is not an **a_priori_semantic_relationship**, the **sub_class_properties** shall belong to the **described_by** list.

NOTE 12 Through an a_priori_semantic_relationship, sub_class_properties may also be imported.

WR12: the properties referenced in **class_constant_values** are declared as class-valued in some superclass of the current class, or in the current class itself.

NOTE 13 The **sub_class_properties** attribute of the **class** entity is used to encode the "Class valued properties" attribute for classes.

NOTE 14 The **class_constant_values** attribute of the **class** entity is used to encode the "Class constant values" for classes.

WR13: if a property referenced in **class_constant_values** was already assigned a value in a superclass, the value assigned in the current class should be the same.

Informal propositions:

IP1: if all the is-a superclasses of the class are available, then the **known_property_constraints** are all the constraints that apply to the properties that are connected to the class either as visible or as applicable properties.

5.8.3 Item_class

The entity item_class enables the modeling of any type of entity of the application domain that may be captured by a characterization class defined by a class structure and a set of properties. In particular, both instances of products and instances of particular aspects of products represented as features, are mapped onto item_class.

The item_class entity includes an instance_sharable attribute that specifies the conceptual status of an item. If this attribute is true, then each instance represents an independent item, otherwise it is a feature, i.e., a dependent item that has to be component of another item. This does not prescribe any specific implementation at the data representation level.

EXAMPLE The *head of a screw* is a feature described by a number of properties but that may only exist when referenced by a screw. It is represented as an **item_class** with the **instance_sharable** attribute equal to *false*.

NOTE 1 Two subtypes of **item_class**, named **component_class** and **material_class**, were defined within the dictionary model of the first edition of ISO 13584-42 and in IEC 61360-2. These subtypes are deprecated, and they are removed from this edition of IEC 61360.

NOTE 2 Another subtype of **item_class**, named **feature_class,** was provided in ISO 13584-24:2003. This subtype is also deprecated and its usage is not recommended in new implementations of this part of IEC 61360.

EXPRESS specification:

```
*)
ENTITY item_class
SUBTYPE OF(class);
      simplified_drawing: OPTIONAL graphics;
      coded_name: OPTIONAL value_code_type;
```

```
        instance_sharable: OPTIONAL BOOLEAN;
    END_ENTITY; -- item_class
    (*
```

Attribute definitions:

simplified_drawing: optional drawings (**graphics**) that can be associated to the described class.

NOTE 3 The **simplified_drawing** attribute of the **item_class** entity is used to encode the "Simplified Drawing" attribute for classes.

coded_name: may be used as a class constant value to characterize the class in the value domain of a **sub_class_properies** of its superclass.

NOTE 4 This attribute in not used in ISO 13584 series. It is only used in IEC 61360 series.

instance_sharable: 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 5 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** whose **instance_sharable** equals *false* and that is referenced by several instances of **item_class**es at the data model level is interpreted as several real world instances of the same feature.

5.8.4 Categorization_class

The categorization_class entity enables the modeling 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 datatypes, nor constraints are associated, as visible or applicable, with such a class. Moreover, **categorization_class**es 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 attribute, called **categorization_class_superclasses**, allows to record the **categorization_class**es that are superclasses of a **categorization_class** in a case-of hierarchy.

NOTE Using the case-of resource constructs, **item_class**es may also be connected to **categorization_class**es.

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 ISO/TS 23768-1 (to be published), the bearing supplier specifies that his/her class *my_bearings* is case-of the standard bearing class *ball bearing* defined in ISO/TS 23768-1. 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 8 below:

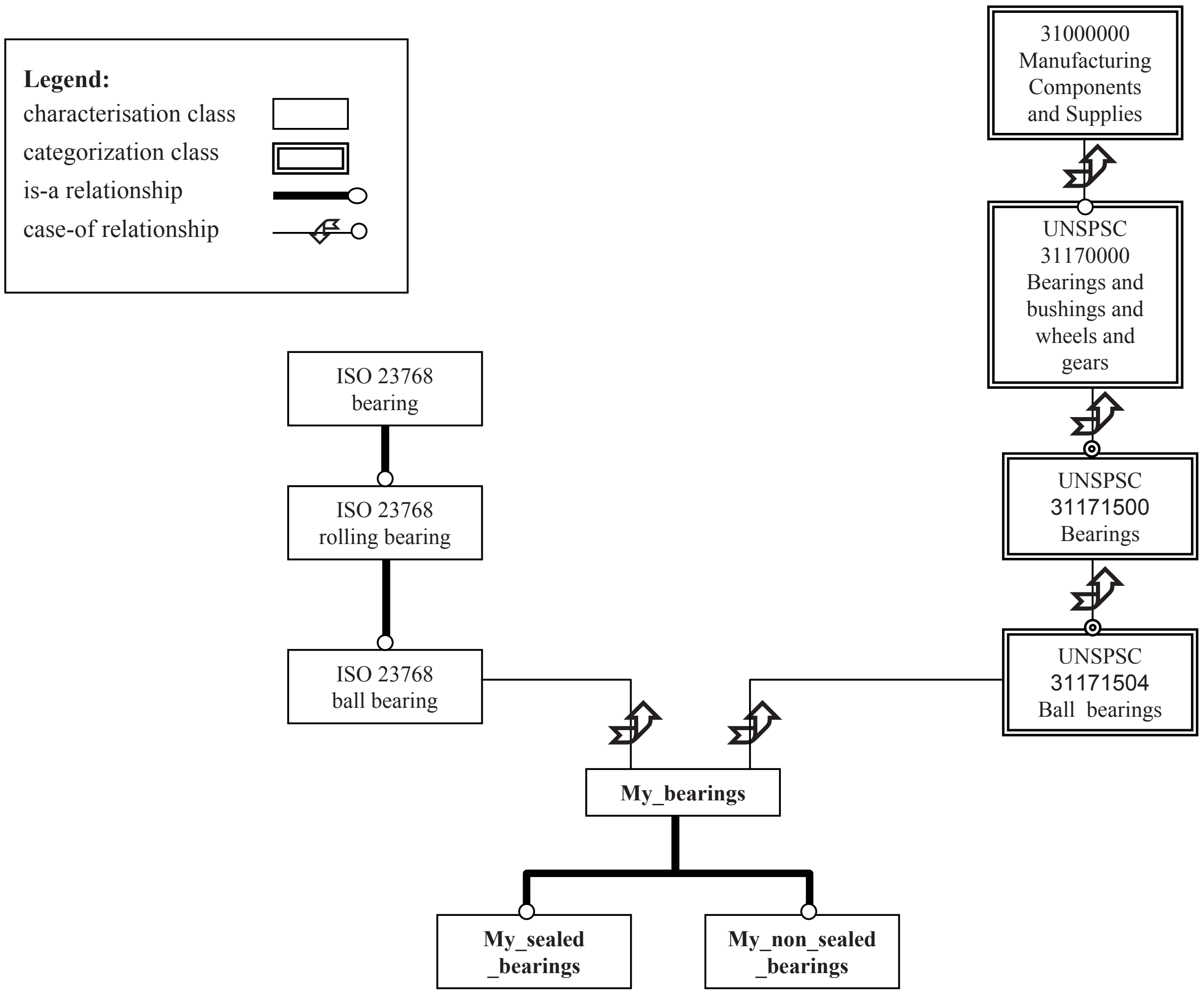

Figure 8 – Example of a supplier ontology

<u>EXPRESS specification:</u>

```
    *)
    ENTITY categorization_class
    SUBTYPE OF(class);
         categorization_class_superclasses: SET [0:?] of class_BSU;
    WHERE
         WR1: QUERY (cl <* SELF. categorization_class_superclasses
              | NOT (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
              +'.CATEGORIZATION_CLASS') IN TYPEOF(cl.definition[1])))
              = [];
         WR2: NOT EXISTS(SELF\class.its_superclass);
         WR3: SIZEOF(SELF\class.described_by) = 0;
         WR4: SIZEOF(SELF\class.defined_types) = 0;
         WR5: SIZEOF(SELF\class.constraints) = 0;
         WR6: SIZEOF(compute_known_visible_properties
              (SELF\dictionary_element.identified_by)) = 0;
         WR7: SIZEOF(SELF\class.sub_class_properties) = 0;
         WR8: SIZEOF(SELF\class.class_constant_values) = 0;
         WR9: SIZEOF(SELF\class.identified_by.known_visible_properties)
              = 0;
         WR10: SIZEOF(SELF\class.identified_by.known_visible_data_types)
              = 0;
    END_ENTITY; -- categorization_class
    (*
```

Attribute definitions:

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

Formal propositions:

WR1: only **categorization_class**es may appear as superclasses of a **categorization_class**.

WR2: a **categorization_class** shall not have is-a superclass.

WR3: no property shall be associated with a **categorization_class**.

WR4: no datatype shall be associated with a **categorization_class**.

WR5: no constraint shall be associated with a **categorization_class**.

WR6: a **categorization_class** shall not be the property definition class of any property.

WR7: no subclass property shall be associated with a **categorization_class**.

WR8: no class constant value shall be associated with a **categorization_class**.

WR9: no visible property shall be associated with a **categorization_class**.

WR10: no visible datatype shall be associated with a **categorization_class**.

5.9 Data element type / properties data

5.9.1 General

This clause contains definitions for the dictionary data for properties.

5.9.2 Property_BSU

The entity **property_BSU** provides for identification of a property.

EXPRESS specification:

```
    *)
    ENTITY property_BSU
    SUBTYPE OF(basic_semantic_unit);
          SELF\basic_semantic_unit.code: property_code_type;
          name_scope: class_BSU;
    DERIVE
          absolute_id: identifier :=
               name_scope.defined_by.absolute_id
               + sep_id + dic_identifier;
    INVERSE
          describes_classes: SET OF class FOR described_by;
    UNIQUE
          UR1: absolute_id;
    WHERE
         WR1: QUERY(c <* describes_classes |
              NOT(is_subclass(c, name_scope.definition[1])))= [];
    END_ENTITY; -- property_BSU
    (*
```

Attribute definitions:

code: to allow for unique identification of the property over all ontologies defined by the same **name_scope.defined_by** supplier.

name_scope: the reference to the class at which or below which the property element is available for reference by the **described_by** attribute.

absolute_id: the unique identification of this property.

describes_classes: the classes declaring this property as available for use in the description of a product.

Formal propositions:

WR1: any class referenced by the **describes_classes** attribute of a **property_BSU** either is the class referenced by its **name_scope** attribute, or is a subclass of this class.

UR1: the property identifier **absolute_id** is unique.

NOTE The **name_scope** attribute of the **property_BSU** entity will be used to encode the "Definition class" attribute for properties.

5.9.3 Property_DET

The **property_DET** entity captures the dictionary description of properties.

EXPRESS specification:

```
    *)
    ENTITY property_DET
    ABSTRACT SUPERTYPE OF(ONEOF(
         condition_DET, dependent_P_DET, non_dependent_P_DET))
    SUBTYPE OF(class_and_property_elements);
         SELF\dictionary_element.identified_by: property_BSU;
         preferred_symbol: OPTIONAL mathematical_string;
         synonymous_symbols: SET [0:?] OF mathematical_string;
         figure: OPTIONAL graphics;
         det_classification: OPTIONAL DET_classification_type;
         domain: data_type;
         formula: OPTIONAL mathematical_string;
    DERIVE
         describes_classes: SET [0:?] OF class
              := identified_by.describes_classes;
    END_ENTITY; -- property_DET
    (*
```

Attribute definitions:

identified_by: the **property_BSU** identifying this property.

preferred_symbol: a shorter description of this property.

synonymous_symbols: synonymous for the shorter description of the property.

figure: an optional **graphics** that describes the property.

det_classification: the ISO 80000/IEC 80000 (formerly ISO 31) class for this property.

domain: the reference to the **data_type** associated to the property.

formula: a mathematical expression for explaining the property.

describes_classes: the classes declaring this property as available for use in the description of a product.

NOTE 1 The **preferred_symbol** attribute is used to encode the "Preferred Letter Symbol" attribute for properties.

NOTE 2 The **synonymous_symbols** attribute is used to encode the "Synonymous Letter Symbol" attribute for properties.

NOTE 3 The **det_classification** attribute is used to encode the "Property Type Classification" attribute of a property.

NOTE 4 The **domain** attribute is used as a starting point for the encoding of the property attribute "Data Type". The entity **data_type** will be subtyped for various possible data types.

NOTE 5 The **formula** attribute is used to encode the "Formula" attribute for properties.

Figure 9 presents a planning model of the data associated with **property_DET**s.

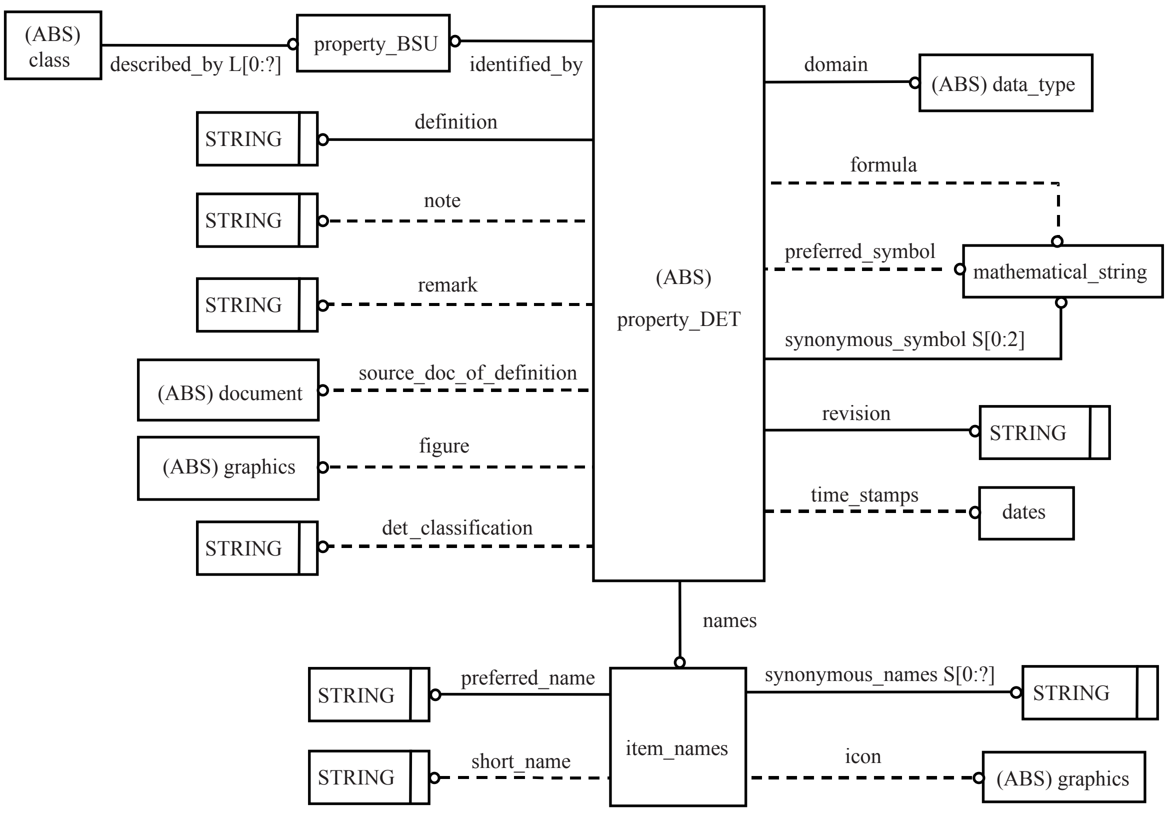

Figure 9 – Overview of property data element type data and relationships

5.9.4 Condition, dependent and non-dependent Data Element Types

Figure 10 depicts the various kinds of Data Element Types in the format of a planning model.

Note that Figure 10 is simplified: the "**depends_on**" relation essentially is implemented with a BSU reference, but a constraint is specified that the referred-to **property_DET** shall be a **condition_DET** (see entity **dependent_P_DET**).

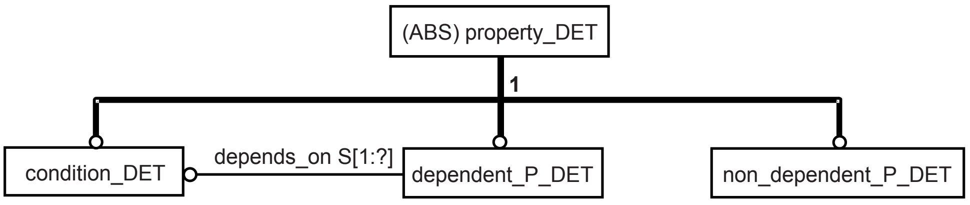

Figure 10 – Kinds of data element types

5.9.5 Structural detail

5.9.5.1 Condition_DET

A **condition_DET** is a property on which other properties may depend upon.

EXPRESS specification:

```
*)
ENTITY condition_DET
SUBTYPE OF(property_DET);
END_ENTITY; -- condition_DET
(*
```

5.9.5.2 Dependent_P_DET

A **dependent_P_DET** is a property whose value depends explicitly on the value(s) of some condition(s).

EXAMPLE The resistance of a thermistor depends upon the ambiant temperature. Thermistor resistance should be presented as a **dependent_P_DET** and thermistor ambiant temperature as a **condition_DET**.

EXPRESS specification:

```
*)
ENTITY dependent_P_DET
SUBTYPE OF(property_DET);
     depends_on: SET [1:?] OF property_BSU;
WHERE
     WR1: QUERY(p <* depends_on | NOT(definition_available_implies(
          p, ('ISO13584_IEC61360_DICTIONARY_SCHEMA.CONDITION_DET'
          IN TYPEOF(p.definition[1]))))) = [];
END_ENTITY; -- dependent_P_DET
(*
```

Attribute definitions:

depends_on: the set of basic semantic units identifying the properties on which this property depends on.

Formal propositions:

WR1: only **condition_DET**s shall be used in the **depends_on** set.

5.9.5.3 Non_dependent_P_DET

A **non_dependent_P_DET** is a property that does not depend explicitly on certain conditions.

EXPRESS specification:

```
*)
ENTITY non_dependent_P_DET
SUBTYPE OF(property_DET);
END_ENTITY; -- non_dependent_P_DET
(*
```

NOTE 1 The three subtypes **condition_DET**, **dependent_P_DET** and **non_dependent_P_DET** of the entity **property_DET** are used to encode the different kinds of properties (see Clause 7). **Condition_DET** is used for context parameters, **dependent_P_DET** is used for context dependent characteristics and the **non_dependent_P_DET** entity is used for product characteristics.

NOTE 2 The **depends_on** attribute of the **dependent_P_DET** entity is used to encode the "Condition" attribute for properties.

5.9.6 Class_value_assignment

Class-valued properties are those properties whose value cannot be assigned individually for an instance of a class but can only be assigned for all instances belonging to a class. Such properties are declared by being included in the **sub_class_properties** list of an **item_class** entity. Then, such a property may be assigned a value for all instances of any **item_class** that is a subclass of the class where the class-valued property is declared, or in this class itself. A value of a class-valued property is assigned to an **item_class** by a **class_value_assignment** referenced by the **class_constant_values** attribute of this class.

NOTE Class-valued properties may be of any data type.

EXPRESS specification:

```
*)
ENTITY class_value_assignment;
      super_class_defined_property: property_BSU;
      assigned_value: primitive_value;
WHERE
      WR1: definition_available_implies(super_class_defined_property,
           compatible_data_type_and_value(super_class_defined_property.
           definition[1]\property_DET.domain, assigned_value));
END_ENTITY; -- class_value_assignment
(*
```

Attribute definitions:

super_class_defined_property: the reference to the property (defined in the class or in any of its superclasses as belonging to the **sub_class_properties** set) to which the **assigned_value** value is assigned.

assigned_value: the value assigned to the property, valid for the whole class referring this **class_value_assignment** instance in its **class_constant_values** set, and all its subclasses.

Formal proposition:

WR1: the value assigned to the **super_class_defined_property** shall be type compatible with the value domain of the **super_class_defined_property**.

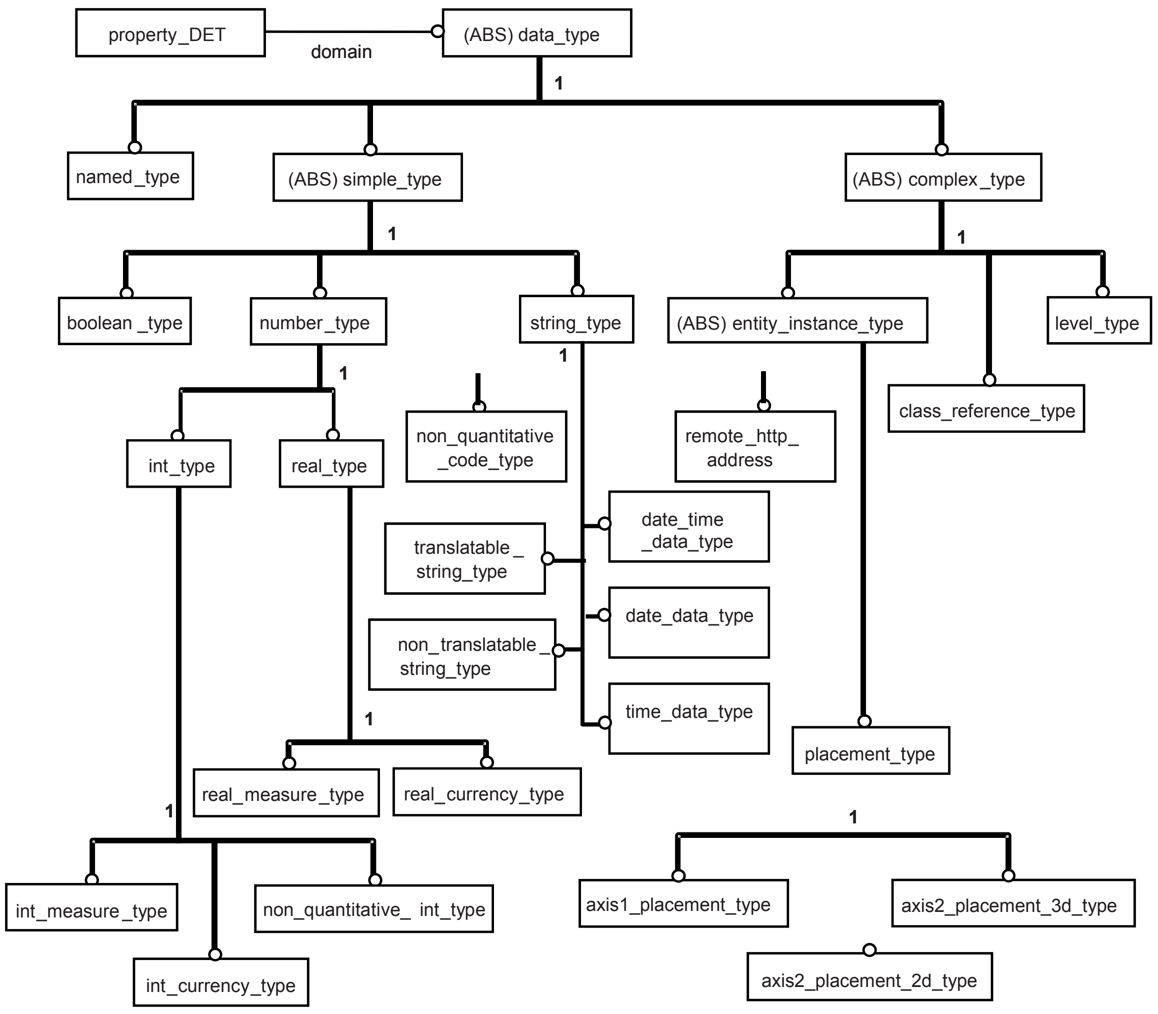

Figure 11 – Entity hierarchy for the type system

5.10 Domain data: the type system

5.10.1 General

This clause contains definitions for the representation of the data types of a **property_DET**. Figure 11 outlines, as a planning model, the entity hierarchy for data types.

In contrast to the other dictionary elements (Suppliers, Classes, Properties), an identification with the basic semantic unit concept is not mandatory for **data_type**, since it will be attached directly to the **property_DET** in many cases, and thus does not need an identification. However, the entities **data_type_BSU** and **data_type_element** allow for a unique identification where this is suitable. It provides for re-using the same type definition in another **property_DET** definition, even outside the current physical file.

5.10.2 Structural detail

5.10.2.1 Data_type_BSU

The data_type_BSU entity provides for identification of data_type_elements.

EXPRESS specification:

```
*)
ENTITY data_type_BSU
SUBTYPE OF(basic_semantic_unit);
     SELF\basic_semantic_unit.code: data_type_code_type;
     name_scope: class_BSU;
DERIVE
     absolute_id: identifier :=
          name_scope.defined_by.absolute_id     (* Supplier*)
          + sep_id + dic_identifier;       (* Data_type *)
INVERSE
     defining_class: SET OF class FOR defined_types;
UNIQUE
     absolute_id;
WHERE
     WR1: is_subclass(defining_class[1], name_scope.definition[1]);
END_ENTITY; -- data_type_BSU
(*
```

Attribute definitions:

code: to allow for unique identification of the data type over all ontologies defined by the same **name_scope.defined_by** supplier.

name_scope: the reference to the class at which or below which the data type element is available for reference by the **defined_types** attribute.

absolute_id: the unique identification of this property.

defining_class: the classes declaring this **data_type** as available for use in the description of a product.

Formal propositions:

WR1: the class used in the **name_scope** attribute is a superclass of the one where this **data_type** is defined.

NOTE The **name_scope** attribute is used to encode the reference to a class the related data type belongs to. This itself, beside the **data_type_element** entity (see below), is part of the encoding of the class attribute "Visible types".

5.10.2.2 Data_type_element

The **data_type_element** entity describes the dictionary element for types. Note that it is not necessary in every case to have BSU and **dictionary_element** for a certain **data_type**, because a **property_DET** can refer to the **data_type** directly. Usage of the BSU relation is only necessary when a supplier wants to refer to the same type in a different physical file.

EXPRESS specification:

```
*)
ENTITY data_type_element
SUBTYPE OF(dictionary_element);
     SELF\dictionary_element.identified_by: data_type_BSU;
     names: item_names;
     type_definition: data_type;
END_ENTITY; -- data_type_element
(*
```

Attribute definitions:

identified_by: the BSU that identifies the described **data_type_element**.

names: the names that allow the description of the defined **data_type_element**.

type_definition: the description of the type carried by the **data_type_element**.

NOTE The re-declared attribute **identified_by** is used to encode the reference to the BSU, this **data_type_element** is related to. This itself, beside the **data_type_BSU** entity (see above), is used to encode the class attribute "Visible types".

5.10.3 The type system

5.10.3.1 Data_type

The **data_type** entity serves as a common supertype for the entities used to indicate the type of the associated DET.

EXPRESS specification:

```
*)
ENTITY data_type
ABSTRACT SUPERTYPE OF(ONEOF(
     simple_type,
     complex_type,
     named_type));
     constraints: SET [0:?] OF domain_constraint;
WHERE
     WR1: QUERY (cons <* constraints
          |NOT correct_constraint_type(cons, SELF)) = [];
END_ENTITY; -- data_type
(*
```

Attribute definitions:

constraints: the set of domain constraints that restrict the domain of values of the data type.

NOTE Each domain constraint in the **constraints** attribute should be fulfilled. Thus the **constraints** attribute is a conjunction of constraints.

Formal proposition:

WR1: the set of domain constraints shall define restrictions that are compatible with the domain of values of the data type.

5.10.3.2 Simple_type

The **simple_type** entity serves as a common supertype for the entities used to indicate a simple type of the associated DET.

EXPRESS specification:

```
*)
ENTITY simple_type
ABSTRACT SUPERTYPE OF(ONEOF(
      number_type,
      boolean_type,
      string_type))
SUBTYPE OF(data_type);
      value_format: OPTIONAL value_format_type;
END_ENTITY; -- simple_type
(*
```

Attribute definitions:

value_format: the optional encoding of the format of values for properties.

NOTE 1 The **value_format** attribute of the **simple_type** entity is used to encode the "Value Format" attribute for properties.

NOTE 2 If any **string_pattern_constraint** applies to the value of a simple type, then it takes precedence on the **value_format**.

5.10.3.3 Number_type

The **number_type** entity provides for values of DETs that are of type NUMBER.

EXPRESS specification:

```
*)
ENTITY number_type
ABSTRACT SUPERTYPE OF(ONEOF(
      int_type,
      real_type,
      rational_type))
SUBTYPE OF(simple_type);
END_ENTITY; -- number_type
(*
```

5.10.3.4 Int_type

The **int_type** entity provides for values of DETs that are of type INTEGER.

EXPRESS specification:

```
*)
ENTITY int_type
SUPERTYPE OF(ONEOF(
      int_measure_type,
      int_currency_type,
      non_quantitative_int_type))
SUBTYPE OF(number_type);
```

```
    END_ENTITY; -- int_type
    (*
```

5.10.3.5 Int_measure_type

The **int_measure_type** entity provides for values of DETs that are measures of type INTEGER. It specifies a **unit** or a unit identifier (**unit_id**), in which values exchanged as single integer are expressed. It may also specify alternative units, or alternative unit identifiers, that are also allowed for use when each value is explicitly associated with its unit.

NOTE 1 Either a **unit** or a **unit_id** is mandatory. In case where both are provided, the **unit** takes precedence.

NOTE 2 When both **alternative_units** and **alternative_unit_ids** are provided, both have the same size and the **alternative_units** attribute takes precedence.

NOTE 3 The **dic_unit_identifier** used in **unit_id** and in **alternative_unit_ids** attributes are unit identifiers that may resolved to a **dic_unit** from an ISO/TS 29002-20 server.

NOTE 4 Each **dic_unit** defined in the **alternative_units** attribute, and each **dic_unit** identified in the **alternative_unit_ids** attribute are required to be associated with a **string_representation**, whose **text_representation** may be used for characterizing the alternative unit used at the instance level.

EXPRESS specification:

```
    *)
    ENTITY int_measure_type
    SUBTYPE OF(int_type);
          unit: OPTIONAL dic_unit;
          alternative_units: OPTIONAL LIST [1:?] OF dic_unit;
          unit_id: OPTIONAL dic_unit_identifier;
          alternative_unit_ids: OPTIONAL LIST [1:?] OF dic_unit_identifier;
    WHERE
          WR1: EXISTS(unit) OR EXISTS(unit_id);
          WR2: NOT EXISTS(alternative_units) OR
               NOT EXISTS(alternative_unit_ids) OR
               (SIZEOF(alternative_units) = SIZEOF(alternative_unit_ids));
          WR3: NOT EXISTS(alternative_units)
               OR (QUERY (un <* SELF.alternative_units
               |NOT EXISTS (un.string_representation))
               = []);
    END_ENTITY; -- int_measure_type
    (*
```

Attribute definitions:

unit: the default unit of reference associated with the value of the **int_measure_type**.

alternative_units: the list of other units that may be used to express the value of the **int_measure_type**.

NOTE 5 The list order is used to ensure that **alternative_units** and **alternative_unit_ids**, if both exist defines the same unit in the same order.

unit_id: the identifier of the default unit of reference associated to the described measure.

NOTE 6 The attribute **unit** and the attribute **unit_id** are both used to encode the "Unit" attribute for properties. When both are provided, **unit** takes precedence

NOTE 7 If the value of a property whose domain is an **int_measure_type** is exchanged as a single integer number, this means that this value is expressed in the **unit** or **unit_id** unit of measure.

alternative_unit_ids: the list of identifiers of other units that may be used to express the value of the **int_measure_type**.

NOTE 8 When the value of a property whose domain is an **int_measure_type** is evaluated in a unit either defined by means of the **alternative_units** attribute or identified by means of the **alternative_unit_ids** attribute, its value cannot be represented as a single integer. It needs to be represented as a pair (value, unit).

Formal propositions:

WR1: one of the two attributes **unit** and **unit_id** shall exist.

WR2: if both attributes **alternative_units** and **alternative_unit_ids** exist, they shall have the same length.

WR3: each **dic_unit** in the **alternative_units** shall have a **string_representation**.

Informal propositions:

IP1: the **dic_unit_identifier**s used in **unit_id** and in **alternative_unit_ids** attributes shall be resolved to a **dic_unit** from an existing ISO/TS 29002-20 server.

IP2: when both **unit** and **unit_id** attributes are provided, they shall define the same unit.

IP3: when both **alternative_units** and **alternative_unit_ids** attributes are provided, they shall define the same list of units in the same order.

IP4: when the **alternative_unit_ids** attribute is provided, all the units the attribute identifies shall resolve to a **dic_unit** that has a **string_representation**.

5.10.3.6 Int_currency_type

The **int_currency_type** entity provides for values of DETs that are integer currencies.

EXPRESS specification:

```
*)
ENTITY int_currency_type
SUBTYPE OF(int_type);
     currency: OPTIONAL currency_code;
END_ENTITY; -- int_currency_type
(*
```

Attribute definitions:

currency: the associated code of the described currency according to ISO 4217. If not present, the currency code has to be exchanged together with the data (values).

5.10.3.7 Non_quantitative_int_type

The **non_quantitative_int_type** entity is an enumeration type where elements of the enumeration are represented with an INTEGER value (see also entity **non_quantitative_code_type** and Figure 12).

EXPRESS specification:

```
*)
ENTITY non_quantitative_int_type
SUBTYPE OF(int_type);
      domain: value_domain;
WHERE
      WR1: QUERY(v <* domain.its_values |
           'ISO13584_IEC61360_DICTIONARY_SCHEMA.VALUE_CODE_TYPE' IN
           TYPEOF(v.value_code)) = [];
END_ENTITY; -- non_quantitative_int_type
(*
```

Attribute definitions:

domain: the set of enumerated values described in the **value_domain** entity.

Formal propositions:

WR1: the values associated with the **domain.its_values** list shall not contain a **value_code_type**.

5.10.3.8 Real_type

The **real_type** entity provides for values of DETs that are of type REAL.

EXPRESS specification:

```
*)
ENTITY real_type
SUPERTYPE OF(ONEOF(
      real_measure_type,
      real_currency_type))
SUBTYPE OF(number_type);
END_ENTITY; -- real_type
(*
```

5.10.3.9 Real_measure_type

The **real_measure_type** entity provides for values of DETs that are measures of type REAL. It specifies a **unit** or a unit identifier, in which values exchanged as single real are expressed. It may also specify alternative units, or alternative unit identifiers, that are also allowed for use when each value is explicitly associated with its unit.

NOTE 1 Either a **unit** or a **unit_id** is mandatory. In case where both are provided, the **unit** takes precedence.

NOTE 2 When both **alternative_units** and **alternative_unit_ids** are provided, both have the same size and the **alternative_units** attribute takes precedence.

NOTE 3 The **dic_unit_identifier** used in **unit_id** and in **alternative_unit_ids** attributes are unit identifiers that may resolve to a **dic_unit** from an ISO/TS 29002-20 server.

NOTE 4 Each **dic_unit** defined in the **alternative_units** attribute, and each **dic_unit** identified in the **alternative_unit_ids** attribute are required to be associated with a **string_representation**, whose **text_representation** may be used for characterizing the alternative unit used at the instance level.

EXPRESS specification:

```
*)
ENTITY real_measure_type
SUBTYPE OF(real_type);
      unit: OPTIONAL dic_unit;
      alternative_units: OPTIONAL LIST [1:?] OF dic_unit;
      unit_id : OPTIONAL dic_unit_identifier;
      alternative_unit_ids: OPTIONAL LIST [1:?] OF dic_unit_identifier;
WHERE
      WR1: EXISTS(unit) OR EXISTS(unit_id);
      WR2: NOT EXISTS(alternative_units)
           OR NOT EXISTS(alternative_unit_ids)
           OR (SIZEOF(alternative_units) =
           SIZEOF(alternative_unit_ids));
      WR3: NOT EXISTS(alternative_units)
           OR (QUERY (un <* SELF.alternative_units
           |NOT EXISTS (un.string_representation))
           = []);
END_ENTITY; -- real_measure_type
(*
```

Attribute definitions:

unit: the default unit of reference associated with the value of the **real_measure_type**.

alternative_units: the list of other units that may be used to express the value of the **real_measure_type**.

NOTE 5 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.

unit_id: the identifier of the default unit of reference associated to the described measure.

NOTE 6 The attribute **unit** and the attribute **unit_id** are both used to encode the "Unit" attribute for properties. When both are provided, **unit** takes precedence.

NOTE 7 If the value of a property whose domain is a **real_measure_type** is exchanged as a single real number, this means that this value is expressed in the **unit** or **unit_id** unit of measure.

alternative_unit_ids: the list of identifiers of other units that may be used to express the value of the **real_measure_type**.

NOTE 8 When the value of a property whose domain is a **real_measure_type** is evaluated in a unit either defined by means of the **alternative_units** attribute or identified by means of the **alternative_unit_ids** attribute, its value cannot be represented as a single real. It will be represented as a pair (value, unit).

Formal propositions:

WR1: one of the two attributes **unit** and **unit_id** shall exist.

WR2: if both attributes **alternative_units** and **alternative_unit_ids** exist, they shall have the same length.

WR3: each **dic_unit** in the **alternative_units** shall have a **string_representation**.

Informal propositions:

IP1: the **dic_unit_identifier**s used in **unit_id** and in **alternative_unit_ids** attributes shall resolve to a **dic_unit** from an existing ISO/TS 29002-20 server.

IP2: when both **unit** and **unit_id** attributes are provided, they shall define the same unit.

IP3: when both **alternative_units** and **alternative_unit_ids** attributes are provided, they shall define the same list of units in the same order.

IP4: when the **alternative_unit_ids** attribute is provided, all the units the attribute identifies shall resolve to a **dic_unit** that has a **string_representation**.

5.10.3.10 Real_currency_type

The **real_currency_type** entity defines real currencies.

EXPRESS specification:

```
*)
ENTITY real_currency_type
SUBTYPE OF(real_type);
     currency: OPTIONAL currency_code;
END_ENTITY; -- real_currency_type
(*
```

Attribute definitions:

currency: the associated code of the described currency according to ISO 4217. If not present, the currency code has to be exchanged together with the data (values).

5.10.3.11 Rational_type

The **rational_type** entity provides for values of DETs that are of type rational.

NOTE In ISO 13584-32, rational values are represented by three integer XML elements: the whole part, the numerator and the denominator. In ISO 13584-24:2003 rational values are represented by an array of three integers: the whole part, the numerator and the denominator.

EXPRESS specification:

```
*)
ENTITY rational_type
SUPERTYPE OF(
     rational_measure_type)
SUBTYPE OF(number_type);
END_ENTITY; -- rational_type
(*
```

5.10.3.12 Rational_measure_type

The **rational_measure_type** entity provides for values of DETs that are measures of type RATIONAL.

EXAMPLE Screw diameter: 4 1/8 inches.

The **rational_measure_type** entity specifies a **unit** or a unit identifier, in which values exchanged as rational are expressed. It may also specify alternative units, or alternative unit identifiers, that are also allowed for use when each value is explicitly associated with its unit.

NOTE 1 Either a **unit** or a **unit_id** is mandatory. In case where both are provided, the **unit** takes precedence.

NOTE 2 When both **alternative_units** and **alternative_unit_ids** are provided, both have the same size and the **alternative_units** attribute takes precedence.

NOTE 3 The **dic_unit_identifier**s used in **unit_id** and in **alternative_unit_ids** attributes are unit identifiers that may resolve to a **dic_unit** from an ISO/TS 29002-20 server.

NOTE 4 Each **dic_unit** defined in the **alternative_units** attribute, and each **dic_unit** identified in the **alternative_unit_ids** attribute are associated with a **string_representation**, whose **text_representation** may be used for characterizing the alternative unit used at the instance level.

EXPRESS specification:

```
*)
ENTITY rational_measure_type
SUBTYPE OF(rational_type);
      unit: OPTIONAL dic_unit;
      alternative_units: OPTIONAL LIST [1:?] OF dic_unit;
      unit_id: OPTIONAL dic_unit_identifier;
      alternative_unit_ids: OPTIONAL LIST [1:?] OF dic_unit_identifier;
WHERE
      WR1: EXISTS(unit) OR EXISTS(unit_id);
      WR2: NOT EXISTS(alternative_units)
           OR NOT EXISTS(alternative_unit_ids)
           OR (SIZEOF(alternative_units) =
           SIZEOF(alternative_unit_ids));
      WR3: NOT EXISTS(alternative_units) OR (QUERY (un <*
           SELF.alternative_units |
           NOT EXISTS (un.string_representation)) = []);
END_ENTITY; -- rational_measure_type
(*
```

Attribute definitions:

unit: the default unit of reference associated with the value of the **rational_measure_type**.

alternative_units: the list of other units that may be used to express the value of the **rational_measure_type**.

NOTE 5 The list order is used to ensure that **alternative_units** and **alternative_unit_ids**, if both exist defines the same unit in the same order.

unit_id: the identifier of the default unit of reference associated to the described measure.

NOTE 6 The attribute **unit** and the attribute **unit_id** are both used to encode the "Unit" attribute for properties. When both are provided, **unit** takes precedence.

NOTE 7 If the value of a property whose domain is a **rational_measure_type** is exchanged as a single rational number, this means that this value is expressed in the **unit** or **unit_id** unit of measure.

alternative_unit_ids: the list of identifiers of other units that may be used to express the value of the **rational_measure_type**.

NOTE 8 When the value of a property whose domain is a **rational_measure_type** is evaluated in a unit either defined by means of the **alternative_units** attribute or identified by means of the **alternative_unit_ids** attribute, its value cannot be represented as a single rational. It needs to be represented as a pair (value, unit).

Formal propositions:

WR1: one of the two attributes **unit** and **unit_**id shall exist.

WR2: if both attributes **alternative_units** and **alternative_unit_ids** exist, they shall have the same length.

WR3: each **dic_unit** in the **alternative_units** shall have a **string_representation**.

Informal propositions:

IP1: the **dic_unit_identifier**s used in **unit_id** and in **alternative_unit_ids** attributes shall resolve to a **dic_unit** from an existing ISO/TS 29002-20 server.

IP2: when both **unit** and **unit_id** attributes are provided, they shall define the same unit.

IP3: when both **alternative_units** and **alternative_unit_ids** attributes are provided, they shall define the same list of units in the same order.

IP4: when the **alternative_unit_ids** attribute is provided, all the units the attribute identifies shall resolve to a **dic_unit** that has a **string_representation**.

5.10.3.13 boolean_type

The **boolean_type** entity provides for values of DETs that are of type BOOLEAN.

EXPRESS specification:

```
*)
ENTITY boolean_type
SUBTYPE OF(simple_type);
END_ENTITY; -- boolean_type
(*
```

5.10.3.14 String_type

The **string_type** provides for values of DETs that are of type STRING.

EXPRESS specification:

```
*)
ENTITY string_type
SUPERTYPE OF ( ONEOF (
      translatable_string_type,
      non_translatable_string_type,
      URI_type,
      non_quantitative_code_type,
      date_time_data_type,
      date_data_type,
      time_data_type))
SUBTYPE OF(simple_type);
END_ENTITY; -- string_type
 (*
```

5.10.3.15 Translatable_string_type

The **translatable_string_type** provides for values of DETs that are of type STRING, but that are supposed to be represented as different strings in different languages.

NOTE 1 Values of such properties cannot be used for product identification.

NOTE 2 Values of such properties may be either a simple **string_value** when a **global_language_assignment** defines a current language, or a **translated_string_value** where each string value is associated with a language.

NOTE 3 Two values of the same property whose **data_type** is **translatable_string_type** may only be compared for equality if the corresponding property as a **source_language** defined as part of its **administrative_data** and if these values are available in this **source_language**. It is not assumed that in languages different from this **source_language** the same meaning is represented by the same string.

EXPRESS specification:

```
*)
ENTITY translatable_string_type
SUBTYPE OF(string_type);
END_ENTITY; -- translatable_string_type
(*
```

5.10.3.16 Non_translatable_string_type

The **non_translatable_string_type** provides for values of DETs that are of type STRING, but that are represented in the same way in any language.

NOTE Values of such properties can be used for product identification.

EXPRESS specification:

```
*)
ENTITY non_translatable_string_type
SUBTYPE OF(string_type);
END_ENTITY; -- non_translatable_string_type
(*
```

5.10.3.17 URI_type

The **URI_type** provides for values of DETs that are of type STRING, but contains a URI.

NOTE A **URI_type** allows in particular to provide URL.

EXPRESS specification:

```
*)
ENTITY URI_type
SUBTYPE OF(string_type);
END_ENTITY; -- URI_type
(*
```

5.10.3.18 Date_time_data_type

The date_time_data_type provides for values of DETs that are of type STRING, but contains a specific instant of time specified according to a particular representation compliant with ISO 8601.

NOTE 1 Only a subset of the lexical representations allowed by ISO 8601 is allowed for values of **date_time_data_type**. This is specified by IP1.

NOTE 2 The above restriction of ISO 8601 representations is the same as the one defined by XML Schema.

EXPRESS specification:

```
*)
ENTITY date_time_data_type
SUBTYPE OF(string_type);
END_ENTITY; -- date_time_data_type
(*
```

Informal propositions:

IP1: the value of a property whose data type is **date_time_data_type** shall comply with the following lexical representation, which is a subset of the lexical representations allowed by ISO 8601. This lexical representation is the ISO 8601 extended format CCYY-MM-DDThh:mm:ss where "CC" represents the century order, the order of the first century being "00", "YY" the year, "MM" the month and "DD" the day, preceded by an optional leading "-" sign to indicate a negative number. If the sign is omitted, "+" is assumed. The letter "T" is the date/time separator and "hh", "mm", "ss" represent hour, minute and second respectively. Additional digits can be used to increase the precision of fractional seconds if desired i.e., the format ss.ss... with any number of digits after the decimal point is supported. The fractional second part is optional; other parts of the lexical form are not optional. To accommodate year values greater than 9999 additional digits can be added to the left of this representation. Leading zeros are required if the year value would otherwise have fewer than four digits; otherwise they are forbidden. The year 0000 is prohibited. The CCYY field shall have at least four digits, the MM, DD, SS, hh, mm and ss fields exactly two digits each (not counting fractional seconds); leading zeroes shall be used if the field would otherwise have too few digits. This representation may be immediately followed by a "Z" to indicate Coordinated Universal Time (UTC) or, to indicate the time zone, i.e. the difference between the local time and Coordinated Universal Time, immediately followed by a sign, + or -, followed by the difference from UTC represented as hh:mm (note: the minutes part is required). See ISO 8601 for details about legal values in the various fields. If the time zone is included, both hours and minutes shall be present.

EXAMPLE To indicate 1:20 p.m. on May the 31st, 1999 for Eastern Standard Time which is 5 hours behind Coordinated Universal Time (UTC), one would write: `1999-05-31T13:20:00-05:00`.

5.10.3.19 Date_data_type

The **date_data_type** provides for values of DETs that are of type STRING, but contains a specific calendar date specified according to a particular representation compliant with ISO 8601.

NOTE 1 Only a subset of the lexical representations allowed by ISO 8601 is allowed for values of **date_data_type**. This is specified by IP1.

NOTE 2 The above restriction of ISO 8601 representations is the same as the one defined by XML Schema.

EXPRESS specification:

```
*)
ENTITY date_data_type
SUBTYPE OF(string_type);
END_ENTITY; -- date_data_type
(*
```

Informal propositions:

IP1: the value of a property whose data type is **date_data_type** shall comply with the following lexical representation, which is a subset of the lexical representations allowed by ISO 8601. The lexical representation for **date_data_type** is the reduced (right truncated) lexical representation for **date_time_data_type**: CCYY-MM-DF. No left truncation is allowed. An optional following time zone qualifier is allowed as for **date_time_data_type**. To accommodate year values outside the range from 0001 to 9999, additional digits can be added to the left of this representation and a preceding "-" sign is allowed.

EXAMPLE 1999-05-31 is the **date_data_type** representation of: May the 31st, 1999.

5.10.3.20 Time_data_type

The **time_data_type** provides for values of DETs that are of type STRING, but contains a specific time specified according to a particular representation compliant with ISO 8601. A value of **time_data_type** represents an instant of time that recurs every day.

NOTE 1 Only a subset of the lexical representations allowed by ISO 8601 is allowed for values of **time_data_type**. This is specified by IP1.

NOTE 2 The above restriction of ISO 8601 representations is the same as the one defined by XML Schema.

NOTE 3 Since the lexical representation allows an optional time zone indicator, **time_data_type** values are partially ordered because it may not be able to determine the order of two values one of which has a time zone and the other does not.

EXPRESS specification:

```
*)
ENTITY time_data_type
SUBTYPE OF(string_type);
END_ENTITY; -- time_data_type
(*
```

Informal propositions:

IP1: the value of a property whose data type is **time_data_type** shall comply with the following lexical representation, which is a subset of the lexical representations allowed by ISO 8601. The lexical representation for **time_data_type** is the left truncated lexical representation for **date_time_data_type** hh:mm:ss.sss with optional following time zone indicator.

EXAMPLE 13:20:00-05:00 is the **time_data_type** representation of: 1.20 p.m. for Eastern Standard Time which is 5 hours behind Coordinated Universal Time (UTC).

5.10.3.21 Non_quantitative_code_type

The **non_quantitative_code_type** entity is an enumeration type where elements of the enumeration are represented with a STRING value (see also ENTITY **non_quantitative_int_type** and Figure 12).

EXPRESS specification:

```
*)
ENTITY non_quantitative_code_type
SUBTYPE OF(string_type);
      domain: value_domain;
WHERE
      WR1: QUERY(v <* domain.its_values |
          NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA.VALUE_CODE_TYPE' IN
          TYPEOF(v.value_code))) = [];
END_ENTITY; -- non_quantitative_code_type
(*
```

Attribute definitions:

domain: the set of enumerated values described in the **value_domain** entity.

Formal propositions:

WR1: the values associated with the **domain.its_values** list shall only contain elements of type **value_code_type**.

5.10.3.22 Complex_type

The **complex_type** entity provides for the definition of types of which the values are represented as EXPRESS instances.

EXPRESS specification:

```
*)
ENTITY complex_type
ABSTRACT SUPERTYPE OF(ONEOF(
      level_type,
      class_reference_type,
      entity_instance_type))
SUBTYPE OF(data_type);
END_ENTITY; -- complex_type
(*
```

5.10.3.23 Level_type

The **level_type** is a complex type indicating that the value of a property consists of one up to four real measure or integer measure values, each one qualified by a particular indicator specifying the meaning of the value.

NOTE 1 Instance values of a **level_type** contain values for and only for the indicators specified in the **levels** attribute. When some of these values are not available, they are represented by **null_values**.

EXAMPLE If the **level_type** specifies that only *minimum* and *typical* values are to be provided as integer, any instance contains integer values (or **null_values**) only for the *minimum* and *typical* levels of the **level_type** instance value.

EXPRESS specification:

```
*)
ENTITY level_type
SUBTYPE OF(complex_type);
      levels: LIST [1:4] OF UNIQUE level;
      value_type: simple_type;
WHERE
      WR1: ('ISO13584_IEC61360_DICTIONARY_SCHEMA.INT_MEASURE_TYPE'
           IN TYPEOF(value_type))
           OR ('ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_MEASURE_TYPE'
           IN TYPEOF(value_type));
      WR2: NOT EXISTS(SELF.levels[2]) OR
```

```
              (SELF.levels[1] < SELF.levels[2]);
        WR3: NOT EXISTS(SELF.levels[2]) OR NOT EXISTS(SELF.levels[3]) OR
             (SELF.levels[2] < SELF.levels[3]);
        WR4: NOT EXISTS(SELF.levels[3]) OR NOT EXISTS(SELF.levels[4]) OR
             (SELF.levels[3] < SELF.levels[4]);
    END_ENTITY; -- level_type
    (*
```

Attribute definitions:

levels: the list of unique indicators that specifies which qualified values shall be associated with the property.

value_type: the data type of the qualified values of the **level_type**.

Formal propositions:

WR1: the **SELF.value_type** shall be of type **int_measure_type** or of type **real_measure_type**.

WR2: the order of the first and second **level**, if both exist, shall follow the enumeration order of the **level** type.

WR3: the order of the second and third **level**, if both exist, shall follow the enumeration order of the **level** type.

WR4: the order of the third and fourth **level**, if both exist, shall follow the enumeration order of the **level** type.

5.10.3.24 Level

The **level** entity specifies the various indicators that may be used to qualify a value in a **level_type**.

These indicators are as follows:

– minimum: 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;
– nominal: value of a quantity used to designate and identify a component, device, equipment, or system;
– typical: 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;
– maximum: 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 The values that are provided for a level type-valued property are specified in the dictionary.

NOTE 3 It is advised that the use of the level type is restricted to those DETs 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.

EXPRESS specification:

```
*)
TYPE level = ENUMERATION OF(
     min,      (* the minimal value of the physical quantity *)
     nom,      (* the nominal value of the physical quantity *)
     typ,      (* the typical value of the physical quantity *)
     max);     (* the maximal value of the physical quantity *)
END_TYPE; -- level
(*
```

5.10.3.25 Class_reference_type

The **class_reference_type** entity provides for values of DETs that are represented as instances of a **class**. It is used, in particular, for the description of assemblies or to describe the material a (part of a) component consists of.

EXPRESS specification:

```
*)
ENTITY class_reference_type
SUBTYPE OF(complex_type);
     domain: class_BSU;
END_ENTITY; -- class_reference_type
(*
```

Attribute definitions:

domain: the **class_BSU** referring to the **class** representing the described type.

5.10.3.26 Entity_instance_type

The **entity_instance_type** entity provides for values of DETs that are represented as instances of some EXPRESS entity data types. A **type_name** attribute enables the specification what are the allowed data types. These data types are strings contained in a set. This attribute, together with the EXPRESS TYPEOF function applied to the value, permits strong type checking and polymorphism. This entity will be subtyped below for some data types that are allowed for use in the dictionary schema.

NOTE When an EXPRESS entity is the value of a given DET the data type of which is an **entity_instance_type**, it is possible to check the correct typing by applying the EXPRESS TYPEOF function on this DET value and compare the results of this TYPEOF application with the strings contained in the **type_name** set attribute.

EXPRESS specification:

```
*)
ENTITY entity_instance_type
SUBTYPE OF(complex_type);
     type_name: SET OF STRING;
END_ENTITY; -- entity_instance_type
(*
```

Attribute definitions:

type_name: the set of strings that describe, in the format of the EXPRESS TYPEOF function, the EXPRESS entity data type names that shall belong to the result of the EXPRESS TYPEOF function when it is applied to a value that references the present entity as its data type.

5.10.3.27 Placement_type

The **placement_type** entity provides for values of DETs that are instances of **placement** entity data type.

NOTE 1 **Placement** entities are imported from ISO 10303-42. According to ISO 10303-42, an instance of **placement** may only exist if it is related to an instance of **geometric_representation_context** in some instance of **representation**. Therefore, if some class properties have instances of **placement** as their values, this class contains a **geometric_representation_context** (that defines the context of these placements) and a **representation** (that gathers these **placement**s with their context). Both **geometric_representation_context** and **representation** being not imported in this part of IEC 61360, the **placement** entities cannot be used when only ISO 13584-42 schemas are used. These entities are introduced as resources for the other parts of ISO 13584.

NOTE 2 **Placement** entities are in particular used in ISO 13584-32 (OntoML) and in ISO 13584-25.

EXPRESS specification:

```
*)
ENTITY placement_type
SUPERTYPE OF(ONEOF(
      axis1_placement_type,
      axis2_placement_2d_type,
      axis2_placement_3d_type))
SUBTYPE OF(entity_instance_type);
WHERE
      WR1: 'GEOMETRY_SCHEMA.PLACEMENT'
           IN SELF\entity_instance_type.type_name;
END_ENTITY; -- placement_type
(*
```

Formal propositions:

WR1: the string 'GEOMETRY_SCHEMA.PLACEMENT' shall be contained in the set defined by the **SELF\entity_instance_type.type_name** attribute.

5.10.3.28 Axis1_placement_type

The **axis1_placement_type** entity provides for values of DETs that are instances of **axis1_placement** entity data type (see ISO 10303-42 for details).

EXPRESS specification:

```
*)
ENTITY axis1_placement_type
SUBTYPE OF(placement_type);
WHERE
      WR1: 'GEOMETRY_SCHEMA.AXIS1_PLACEMENT' IN
           SELF\entity_instance_type.type_name;
END_ENTITY; -- axis1_placement_type
(*
```

Formal propositions:

WR1: the string 'GEOMETRY_SCHEMA.AXIS1_PLACEMENT' shall be contained in the set defined for the **SELF\entity_instance_type.type_name** attribute.

5.10.3.29 Axis2_placement_2d_type

The **axis2_placement_2d_type** entity provides for values of DETs that are instances of **axis2_placement_2d** entity data type (see ISO 10303-42 for details).

EXPRESS specification:

```
*)
ENTITY axis2_placement_2d_type
SUBTYPE OF(placement_type);
WHERE
     WR1: 'GEOMETRY_SCHEMA.AXIS2_PLACEMENT_2D'
          IN SELF\entity_instance_type.type_name;
END_ENTITY; -- axis2_placement_2d_type
(*
```

Formal propositions:

WR1: the string 'GEOMETRY_SCHEMA.AXIS2_PLACEMENT_2D' shall be contained in the set defined for the **SELF\entity_instance_type.type_name** attribute.

5.10.3.30 Axis2_placement_3d_type

The **axis2_placement_3d_type** entity provides for values of DETs that are instances of **axis2_placement_3d** entity data type (see ISO 10303-42 for details).

EXPRESS specification:

```
*)
ENTITY axis2_placement_3d_type
SUBTYPE OF(placement_type);
WHERE
     WR1: 'GEOMETRY_SCHEMA.AXIS2_PLACEMENT_3D'
          IN SELF\entity_instance_type.type_name;
END_ENTITY; -- axis2_placement_3d_type
(*
```

Formal propositions:

WR1: the string 'GEOMETRY_SCHEMA.AXIS2_PLACEMENT_3D' shall be contained in the set defined for the **SELF\entity_instance_type.type_name** attribute.

5.10.3.31 Named_type

The **named_type** entity provides for referring to other types via the BSU mechanism.

EXPRESS specification:

```
*)
ENTITY named_type
```

```
    SUBTYPE OF(data_type );
         referred_type: data_type_BSU;
    END_ENTITY; -- named_type
    (*
```

Attribute definitions:

referred_type: the BSU identifying the **data_type** the present entity refers to.

5.10.4 Values

This clause contains definitions for non-quantitative data element types (see **non_quantitative_int_type** and **non_quantitative_code_type** entities).

Figure 12 outlines, as a planning model, the main data associated with non-quantitative data element types.

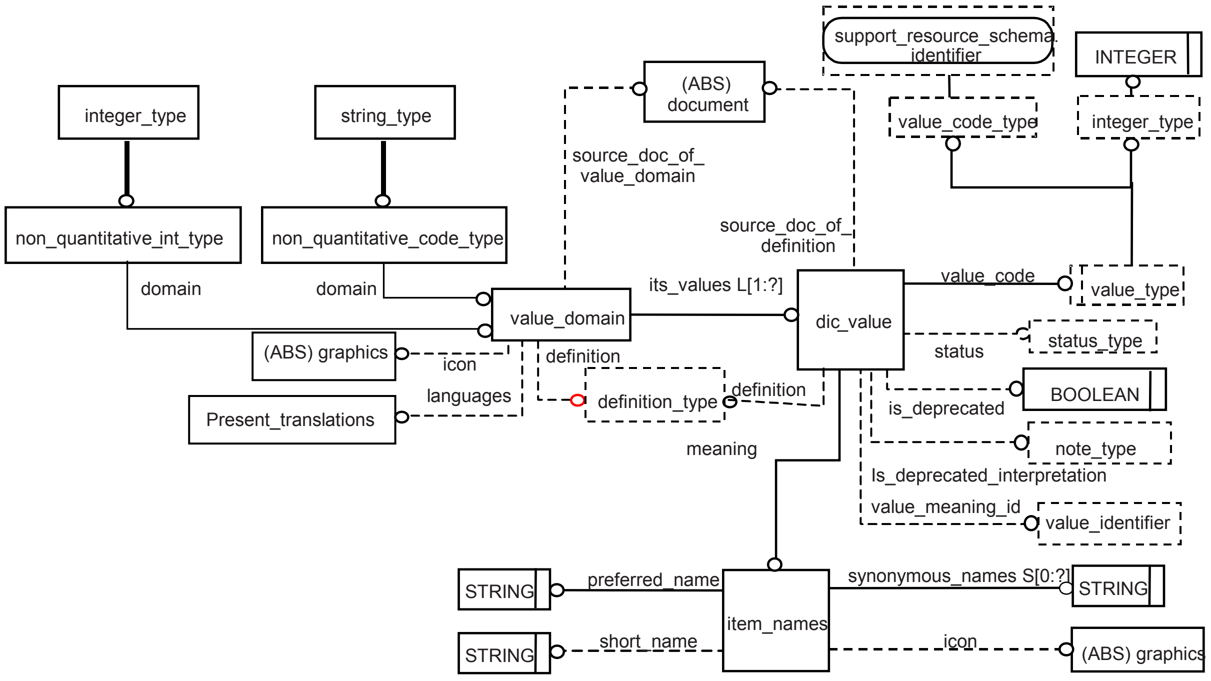

Figure 12 – Overview of non-quantitative data element types

5.10.5 Structural detail

5.10.5.1 Value_domain

The **value_domain** entity describes the set of allowed values for a non-quantitative data element type.

EXPRESS specification:

```
    *)
    ENTITY value_domain;
         its_values: LIST [1:?] OF dic_value;
         source_doc_of_value_domain: OPTIONAL document;
         languages: OPTIONAL present_translations;
         terms: LIST [0:?] OF item_names;
         definition: OPTIONAL definition_type;
```

```
          icon: OPTIONAL graphics;
     WHERE
          WR1: NOT EXISTS(languages) OR (QUERY(v <* its_values |
               languages :<>: v.meaning.languages) = []);
          WR2: codes_are_unique(its_values);
          WR3: EXISTS(languages) OR (QUERY(v <* its_values |
               EXISTS(v.meaning.languages)) = []);
          WR4: EXISTS(languages) OR (QUERY(v <* its_values |
               EXISTS(v.definition.languages)) = []);
     END_ENTITY; -- value_domain
     (*
```

Attribute definitions:

its_values: the enumeration list of values contained in the described domain.

source_doc_of_value_domain: the document describing the domain associated to the described **value_domain** entity.

languages: the optional languages in which the translations are provided.

terms: the list of **item_names** to allow for IEC 61360 the link to the terms dictionary.

definition: the optional text describing the **value_domain**.

icon: an optional **icon** which graphically represents the description associated with the **value_domain**.

Formal propositions:

WR1: if the value meanings are provided in more than one language, then the set of languages used shall be the same for the whole set of values.

WR2: value codes shall be unique within this data type.

WR3: if no **languages** is provided, the value meanings shall not be assigned any language.

WR4: if no **languages** is provided, the value definition shall not be assigned any language.

5.10.5.2 Value_type

Each value of a non-quantitative data element is associated with a code that characterizes the value. A **value_type** may be either an INTEGER or a **value_code_type**.

EXPRESS specification:

```
     *)
     TYPE integer_type = INTEGER;
     END_TYPE; -- integer_type

     TYPE value_type = SELECT(value_code_type, integer_type);
     END_TYPE; -- value_type
     (*
```

5.10.5.3 Dic_value

The **dic_value** entity is one of the values of a **value_domain** entity.

EXPRESS specification:

```
*)
ENTITY dic_value;
      value_code: value_type;
      meaning: item_names;
      source_doc_of_definition: OPTIONAL document;
      definition: OPTIONAL definition_type;
      status: OPTIONAL status_type;
      is_deprecated: OPTIONAL BOOLEAN;
      is_deprecated_interpretation: OPTIONAL note_type;
      value_meaning_id: OPTIONAL dic_value_identifier;
WHERE
      WR1: NOT EXISTS (SELF. is_deprecated)
                OR EXISTS (SELF. is_deprecated_interpretation);
END_ENTITY; -- dic_value
(*
```

Attribute definitions:

value_code: the code associated to the described value. It can be either a **value_code_type** or an **integer_type**.

meaning: the meaning associated to this value. It is provided by names.

source_doc_of_definition: the optional source document in which the value is defined.

definition: the optional text describing the **dic_value**.

status: a **status_type** that defines the life cycle state of the **dic_value**.

NOTE 1 Allowed values of a **status_type** are not standardized. They are defined for each particular dictionary by its information supplier.

EXAMPLE 1 A set of allowed values for the status of items proposed for standardization to an ISO standard maintenance agency are defined in the ISO directives.

EXAMPLE 2 A set of allowed values for the status of items in an IEC database standard is defined in the IEC directives.

NOTE 2 For those **dic_value**s that are not yet released for insertion, representation of draft **dic_value**s might be useful.

EXAMPLE 3 For experimentation purposes before validation.

NOTE 3 If the **status** attribute is not provided for a **dic_value**, and if the use of this **dic_value** is not deprecated as denoted by a possible **is_deprecated** attribute, then the **dic_value** has the same standardization status as the whole dictionary. In particular, if the dictionary is standardized, this **dic_value** is part of the current edition of the standard.

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

NOTE 4 When **is_deprecated** has no value, the **dic_value** is not deprecated.

NOTE 5 Deprecated **dic_value**s are left in the **value_domain**s for upward compatibility reasons.

is_deprecated_interpretation: specify the deprecation rationale and how instance values of the deprecated element should be interpreted.

value_meaning_id: a **dic_value_identifier** that is a global identifier of the **dic_value**, independently of the **value_domain** in which it is included.

NOTE 6 This identifier allows to reuse the same **dic_value** definition in various domains.

Formal proposition:

WR1: when **is_deprecated** exists, **is_deprecated_interpretation** shall exist.

Informal proposition:

IP1: instance values of **is_deprecated_interpretation** element shall be defined at the time where deprecation decision was taken.

5.10.5.4 Administrative_data

An **administrative_data** entity records information about the life cycle of a dictionary element.

EXPRESS specification:

```
*)
ENTITY administrative_data;
      status: OPTIONAL status_type;
      translation: LIST[0:?] of translation_data;
      source_language: language_code;
INVERSE
      administrated_element: dictionary_element FOR administration;
WHERE
      WR1: one_language_per_translation(SELF);
      WR2: SIZEOF(QUERY (trans <* SELF.translation |
           trans.language = source_language)) = 0;
END_ENTITY; -- administrative_data
(*
```

Attribute definitions:

status: a **status_type** that defines the life cycle state of the dictionary element.

NOTE 1 Allowed values of a **status_type** are not standardized. They are defined for each particular dictionary by its information supplier.

EXAMPLE 1 A set of allowed values for the status of items proposed for standardization to an ISO standard maintenance agency are defined in the ISO directives.

EXAMPLE 2 A set of allowed values for the status of items in an IEC database standard is defined in the IEC directives.

NOTE 2 For those **dictionary_element**s that are not yet released for insertion, representation of draft **dictionary_element**s might be useful.

EXAMPLE 3 For experimentation purposes before validation.

NOTE 3 If the **status** attribute is not provided, and if this **dictionary_element** use is not deprecated as denoted by a possible **is_deprecated** attribute, then the **dictionary_element** has the same standardization status as the whole dictionary. In particular, if the dictionary is standardized, this **dictionary_element** is part of the current edition of the standard.

translation: description of responsible translators in the various languages.

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

NOTE 4 A dictionary may contain **dictionary_element**s whose **source_language**s are different, for instance because they where imported from different dictionaries. It is the responsibility of the dictionary data supplier to ensure that the information about these various elements is provided in the same language or languages.

administrated_element: the **dictionary_element** of which life cycle data are recorded in the **administrative_data**.

Formal propositions:

WR1: the languages of translation associated to an **administrative_data** are unique.

WR2: the **source_language** is not present in the languages of translation associated to an **administrative_data**.

5.10.5.5 Translation_data

The **translation_data** entity records information about the possible translations of a dictionary element.

EXPRESS specification:

```
*)
ENTITY translation_data;
      language: language_code;
      responsible_translator: supplier_BSU;
      translation_revision: revision_type;
      date_of_current_translation_revision: OPTIONAL date_type;
INVERSE
      belongs_to: administrative_data FOR translation;
END_ENTITY; -- translation_data
(*
```

Attribute definitions:

language: the language in which the dictionary element is translated.

NOTE 1 In case of discrepancy between the initial definition of a dictionary element and some of its translation, the actual meaning of the dictionary element is the one of the source definition language.

responsible_translator: the organization responsible for the translation in the language element.

translation_revision: the revision number of the corresponding translation.

NOTE 2 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** should not be changed. However, any change of a translation will imply change of the corresponding **translation_revision**.

date_of_current_translation: the date of the last revision of the corresponding translation.

belongs_to: the **administrative_data** that references the **translation_data** record.

5.10.6 Extension to ISO 10303-41 unit definitions

5.10.6.1 General

This clause defines the resources for description of units in a dictionary. It extends the resources defined in ISO 10303-41.

5.10.6.2 Non_si_unit

The **non_si_unit** entity extends the unit model of ISO 10303-41 to allow for the representation of non-SI-units that are neither context dependent, nor conversion-based (see ISO 10303-41 for details).

EXPRESS specification:

```
*)
ENTITY non_si_unit
SUBTYPE OF(named_unit);
     name: label;
END_ENTITY; -- non_si_unit
(*
```

Attribute definitions:

name: the **label** used to name the described unit.

5.10.6.3 Assert_ONEOF rule

The **assert_ONEOF** rule asserts that ONEOF holds between the following subtypes of **named_unit**: **si_unit**, **context_dependent_unit**, **conversion_based_unit**, **non_si_unit**.

EXPRESS specification:

```
*)
RULE assert_ONEOF FOR(named_unit);
WHERE
     QUERY(u <* named_unit |
          ('ISO13584_IEC61360_DICTIONARY_SCHEMA.NON_SI_UNIT'
          IN TYPEOF(u)) AND
          ('MEASURE_SCHEMA.SI_UNIT' IN TYPEOF(u))
          OR ('ISO13584_IEC61360_DICTIONARY_SCHEMA.NON_SI_UNIT'
          IN TYPEOF(u)) AND
          ('MEASURE_SCHEMA.CONTEXT_DEPENDENT_UNIT' IN TYPEOF(u))
          OR ('ISO13584_IEC61360_DICTIONARY_SCHEMA.NON_SI_UNIT'
          IN TYPEOF(u)) AND
          ('MEASURE_SCHEMA.CONVERSION_BASED_UNIT' IN TYPEOF(u))
          ) = [];
END_RULE; -- assert_ONEOF
(*
```

5.10.6.4 Dic_unit

The basic representation of units is in structured form according to ISO 10303-41. But since one of the purposes of storing units in the dictionary is for the presentation to the user, a structured representation alone is not sufficient. It shall be supplemented by a string representation.

The present definitions allow various possibilities:

– the function **string_for_unit** (see 5.12) can be used. For a given structured representation of a unit, it returns a string representation corresponding to the one used in Annex B of IEC 61360-1:2009;
– a string representation can be supplied in plain text form (entity **mathematical_string**, attribute **text_representation**);
– a MathML representation can be supplied to allow for an enhanced presentation of the unit including sub- and superscripts etc. (entity **mathematical_string**, attribute **MathML_representation**).

The **dic_unit** entity describes a unit to be stored in a dictionary.

EXPRESS specification:

```
*)
ENTITY dic_unit;
      structured_representation: unit;
      string_representation: OPTIONAL mathematical_string;
END_ENTITY; -- dic_unit
(*
```

Attribute definitions:

structured_representation: structured representation, from ISO 10303-41, including extension defined in 5.10.6.

string_representation: the function **string_for_unit** can be used to compute a string representation from the **structured_representation**, for the case where no **string_representation** is present.

NOTE The **structured_representation** attribute of the entity **dic_unit** is used to encode the property attribute "Unit".

5.11 Basic type and entity definitions

5.11.1 Basic type definitions

This subclause contains the basic type and entity definitions that were used in the main part of the model. The following section contains basic type and entity definitions, sorted alphabetically.

5.11.2 Structural detail

5.11.2.1 Class_code_type

The **class_code_type** identifies the allowed values for a class code.

EXPRESS specification:

```
*)
TYPE class_code_type = code_type;
WHERE
      WR1: LENGTH(SELF) <= class_code_len;
END_TYPE; -- class_code_type
(*
```

Formal propositions:

WR1: the length of values corresponding to **class_code_type** shall be less or equal to the length of **class_code_len** (i.e., 35).

5.11.2.2 Code_type

The **code_type** identifies the allowed values for a code type used to identify.

NOTE If the code is also intended to be exchanged using ISO/TS 29002-5, it is recommended to fulfil the requirements defined by that standard. For a code, only "safe characters" are allowed. Safe characters include: upper case letters, digits, colons, periods, or underscore. Moreover, the minus character '-' is allowed for particular purposes.

EXPRESS specification:

```
*)
TYPE code_type = identifier;
WHERE
     WR1: NOT(SELF LIKE '*#*');
     WR2: NOT(SELF LIKE '* *');
     WR3: NOT(SELF = '');
END_TYPE; -- code_type
(*
```

Formal propositions:

WR1: the '#' shall not be contained in a **code_type** value. '#' is used to concatenate identifiers, (see: CONSTANT **sep_id**), or code and version (see: CONSTANT **sep_cv**).

WR2: spaces are not allowed, to avoid problems with leading and trailing blanks when concatenating codes.

WR3: a **code_type** shall not be an empty string.

5.11.2.3 Currency_code

5.11.2.3.1 General

The **currency_code** identifies the values allowed for a currency code. These values are defined according to ISO 4217.

EXAMPLE Values are: "CHF" for Swiss Francs, "CNY" for Yuan Renminbi (Chinese), "JPY" for Yen (Japanese), "SUR" for SU Rouble, "USD" for US Dollars, "EUR" for EURO.

EXPRESS specification:

```
*)
TYPE currency_code = identifier;
WHERE
     WR1: LENGTH(SELF) = 3;
END_TYPE; -- currency_code
(*
```

Formal propositions:

WR1: the length of a **currency_code** value shall be equal to 3.

5.11.2.3.2 Data_type_code_type

The **data_type_code_type** identifies the values allowed for a data type code.

<u>EXPRESS specification:</u>

```
*)
TYPE data_type_code_type = code_type;
WHERE
     WR1: LENGTH(SELF) = data_type_code_len;
END_TYPE; -- data_type_code_type
(*
```

<u>Formal propositions:</u>

WR1: the length of a **data_type_code_type** value shall be equal to the value of a **data_type_code_len** (i.e., 35).

5.11.2.3.3 Date_type

The **date_type** identifies the values allowed for a date. These values are defined according to ISO 8601.

EXAMPLE "1994-03-21".

<u>EXPRESS specification:</u>

```
*)
TYPE date_type = STRING(10) FIXED;
WHERE
     WR1: SELF LIKE '####-##-##';
END_TYPE; -- date_type
(*
```

5.11.2.4 Definition_type

The **definition_type** identifies the values allowed for a definition.

<u>EXPRESS specification:</u>

```
*)
TYPE definition_type = translatable_text;
END_TYPE; -- definition_type
(*
```

5.11.2.5 DET_classification_type

The **DET_classification_type** identifies the values allowed for a DET classification. These values are used for DET classification according to ISO 80000/IEC 80000 (formerly ISO 31).

<u>EXPRESS specification:</u>

```
*)
TYPE DET_classification_type = identifier;
WHERE
     WR1: LENGTH(SELF) = DET_classification_len;
END_TYPE; -- DET_classification_type
```

```
(*
```

Formal propositions:

WR1: the length of a **DET_classification_type** value shall be equal to the value of a **DET_classification_len** (i.e., 3).

5.11.2.6 Note_type

The **note_type** identifies the values allowed for a note.

EXPRESS specification:

```
*)
TYPE note_type = translatable_text;
END_TYPE; -- note_type
(*
```

5.11.2.7 Pref_name_type

The **pref_name_type** identifies the values allowed for a preferred name.

EXPRESS specification:

```
*)
TYPE pref_name_type = translatable_label;
WHERE
     WR1: check_label_length(SELF, pref_name_len);
END_TYPE; -- pref_name_type
(*
```

Formal propositions:

WR1: the length of a **pref_name_type** value shall not exceed the length of **pref_name_len** (i.e., 255).

5.11.2.8 Property_code_type

The **property_code_type** identifies the values allowed for a property code.

EXPRESS specification:

```
*)
TYPE property_code_type = code_type;
WHERE
     WR1: LENGTH(SELF) <= property_code_len;
END_TYPE; -- property_code_type
(*
```

Formal propositions:

WR1: the length of a **property_code_type** value shall not exceed the length of **property_code_len** (i.e., 35).

5.11.2.9 Remark_type

The **remark_type** identifies the values allowed for a remark.

EXPRESS specification:

```
*)
TYPE remark_type = translatable_text;
END_TYPE; -- remark_type
(*
```

5.11.2.10 Hierarchical_position_type

The **hierarchical_position_type** identifies the values allowed for a hierarchical position.

EXPRESS specification:

```
*)
TYPE hierarchical_position_type = identifier;
END_TYPE; -- hierarchical_position_type
(*
```

NOTE The representation of a hierarchical position in a **hierarchical_position_type** is based on some coding conventions. This coding convention is not defined by this part of IEC 61360.

5.11.2.11 Revision_type

The **revision_type** identifies the values allowed for a revision.

NOTE When a new version is issued, its revision value is set to '0'.

EXPRESS specification:

```
*)
TYPE revision_type = code_type;
WHERE
     WR1: LENGTH(SELF) <= revision_len;
     WR2: EXISTS(VALUE(SELF)) AND ('INTEGER' IN TYPEOF(VALUE(SELF)))
          AND (VALUE(SELF) >= 0);
END_TYPE; -- revision_type
(*
```

Formal propositions:

WR1: the length of a **revision_type** value shall not exceed the length of **revision_len** (i.e., 3).

WR2: the **revision_type** shall contain digits only and the integer it represents shall be a natural integer.

5.11.2.12 Short_name_type

The **short_name_type** identifies the values allowed for a short name.

EXPRESS specification:

```
*)
TYPE short_name_type = translatable_label;
```

```
WHERE
     WR1: check_label_length(SELF, short_name_len);
END_TYPE; -- short_name_type
(*
```

Formal propositions:

WR1: the length of a **short_name_type** value shall not exceed the length of **short_name_len** (i.e., 30).

5.11.2.13 Supplier_code_type

The **supplier_code_type** identifies the values allowed for a supplier code.

NOTE If the supplier code is also intended to be exchanged using ISO/TS 29002-5, the various parts of the supplier code as defined by ISO 6523 (ICD, OI, OPI, OPIS and AI) are separed by '-'.

EXPRESS specification:

```
*)
TYPE supplier_code_type = code_type;
WHERE
     WR1: LENGTH(SELF) <= supplier_code_len;
END_TYPE; -- supplier_code_type
(*
```

Formal propositions:

WR1: the length of a **supplier_code_type** value shall not exceed the length of **supplier_code_len** (i.e., 149).

5.11.2.14 Syn_name_type

The **syn_name_type** identifies the values allowed for a synonymous name.

EXPRESS specification:

```
*)
TYPE syn_name_type = SELECT(label_with_language, label);
WHERE
     WR1: check_syn_length(SELF, syn_name_len);
END_TYPE; -- syn_name_type
(*
```

Formal propositions:

WR1: the length of a **syn_name_type** value shall not exceed the length of **syn_name_len** (i.e., 255).

5.11.2.15 Keyword_type

The **keyword_type** identifies the values allowed for a keyword.

EXPRESS specification:

```
*)
TYPE keyword_type = SELECT(label_with_language, label);
END_TYPE; -- keyword_type
(*
```

5.11.2.16 ISO_29002_IRDI_type

The **ISO_29002_IRDI_type** is a global identifier that identifies an administrated item in a registry. The structure of this identifier complies with identifier syntax defined in ISO/TS 29002-5.

NOTE 1 An **ISO_29002_IRDI_type** may be used for any kind of information considered in ISO/TS 29002-5 and that may be associated with an IRDI identifier. Three special cases are identified below because they are specifically used in the **ISO13584_IEC61360_dictionary_schema**: **constraint_identifier**, **dic_unit_identifier** and **dic_value_identifier.**

EXPRESS specification:

```
*)
TYPE ISO_29002_IRDI_type = identifier;
WHERE
     WR1: LENGTH (SELF) <= 290;
END_TYPE; -- syn_name_type
(*
```

Formal propositions:

WR1: as specified in ISO/TS 29002-5, the length of the identifier shall not be greater than 290.

Informal propositions:

IP1: the identifier shall fulfil the requirements specified in ISO/TS 29002-5 for an "international registration data identifier" (IRDI).

NOTE 2 According to ISO/TS 29002-5 an IRDI consists either of a string that do not contain the ''#' character, to identify an organization, or of three sub-strings that do not contain the '#' character and that are separated by the '#' character to identify any other administrated item.

NOTE 3 In the case where the IRDI is not used for identifying an organization:

– the first sub-string, called the called the Registration Authority Identifier (RAI), identifies the organization which is responsible for the administration of the administrated item;

– the second sub-string, called the Data Identifier (DI), contains both a categorization of the administrated item, represented by two characters followed by the minus ('-') sign as defined in ISO/TS 29002-5 (for instance: class, property, unit), and the identifier assigned to the administrated item by the RAI;

– the third sub-string corresponds to the Version Identifier (VI) of the IRDI.

5.11.2.17 Constraint_identifier

The **constraint_identifier** is an **ISO_29002_IRDI_type** identifier that provides a global identifier to a constraint represented as a **constraint** entity. The structure of this identifier complies with identifier syntax defined in ISO/TS 29002-5.

NOTE A **constraint_identifier** may be associated with a resolution service compliant with ISO/TS 29002-20. This service would be able to return a formal definition of the constraint identified by the **constraint_identifier** compliant with the **constraint** EXPRESS model in the syntax defined by ISO 13584-32 (OntoML), and possibly in ISO 10303-21 syntax.

EXPRESS specification:

```
*)
TYPE constraint_identifier = ISO_29002_IRDI_type;
END_TYPE; -- constraint_identifier
(*
```

Informal propositions:

IP1: the part of the identifier after the second '#' character, that is the Data Identifier, shall start by '04-' to identify a constraint as specified in ISO/TS 29002-5.

5.11.2.18 Dic_unit_identifier

The **dic_unit_identifier** is an **ISO_29002_IRDI_type** identifier that identifies a unit of which the **dic_unit** representation shall be downloadable from an ISO/TS 29002-20 server. The structure of this identifier complies with identifier syntax defined in ISO/TS 29002-5.

EXPRESS specification:

```
*)
TYPE dic_unit_identifier = ISO_29002_IRDI_type;
END_TYPE; -- dic_unit_identifier
(*
```

Informal propositions:

IP1: a **dic_unit_identifier** shall be associated with a resolution service compliant with ISO/TS 29002, and this service shall be able to return a formal definition of the unit identified by the **dic_unit_identifier** compliant with the **dic_unit** EXPRESS model in the syntax defined by ISO 13584-32 (OntoML), and possibly in the ISO 10303-21 syntax.

IP2: the part of the identifier after the second '#' character, that is the Data Identifier, shall start by '05-' to identify a unit as specified in ISO/TS 29002-5.

NOTE A **dic_unit_identifier** constitutes an International Registration Data Identifier (IRDI) as defined by ISO/IEC 11179-5.

5.11.2.19 Dic_value_identifier

The **dic_value_identifier** is an **ISO_29002_IRDI_type** identifier that provides a global identifier to a property value represented as a **dic_value** entity. The structure of this identifier complies with identifier syntax defined in ISO/TS 29002-5.

NOTE 1 Assigning a **dic_value_identifier** allows to reuse the same **dic_value** definition in several **value_domains**.

NOTE 2 A **dic_value_identifier** may be associated with a resolution service compliant with ISO/TS 29002. This service would be able to return a formal definition of the value identified by the **dic_value_identifier** compliant with the **dic_value** EXPRESS model in the syntax defined by ISO 13584-32 (OntoML), and possibly in ISO 10303-21 syntax.

EXPRESS specification:

```
*)
TYPE dic_value_identifier = ISO_29002_IRDI_type;
END_TYPE; -- dic_value_identifier
(*
```

Informal proposition:

IP1: the part of the identifier after the second '#' character, that is the Data Identifier, shall start by '07-' to identify a property value as specified in ISO/TS 29002-5.

NOTE 3 A **dic_value_identifier** constitutes an International Registration Data Identifier (IRDI) as defined by ISO/IEC 11179-5.

5.11.2.20 Value_code_type

The **value_code_type** identifies the values allowed for a value code.

EXPRESS specification:

```
*)
TYPE value_code_type = identifier;
WHERE
     WR1: LENGTH(SELF) <= value_code_len;
END_TYPE; -- value_code_type
(*
```

Formal propositions:

WR1: the length of a **value_code_type** value shall not exceed the length of **value_code_len** (i.e., 35).

5.11.2.21 Value_format_type

The **value_format_type** identifies the values allowed for a value format. These values are defined according to Annex D.

EXPRESS specification:

```
*)
TYPE value_format_type = identifier;
WHERE
     WR1: LENGTH(SELF) <= value_format_len;
END_TYPE; -- value_format_type
(*
```

Formal propositions:

WR1: the length of a **value_format_type** value shall not exceed the length of **value_format_len** (i.e., 80).

5.11.2.22 Version_type

The version_type identifies the values allowed for a version.

EXPRESS specification:

```
*)
TYPE version_type = code_type;
WHERE
     WR1: LENGTH(SELF) <= version_len;
     WR2: EXISTS(VALUE(SELF)) AND ('INTEGER' IN TYPEOF(VALUE(SELF)))
          AND (VALUE(SELF) >= 0);
END_TYPE; -- version_type
(*
```

Formal propositions:

WR1: the length of a **version_type** value shall not exceed the length of **version_len** (i.e., 10).

WR2: the **version_type** shall contain digits only.

5.11.2.23 Status_type

The **status_type** identifies the values allowed for a status. Allowed values of a **status_type** are not standardized. They shall be defined for each particular dictionary by the supplier of dictionary data.

EXAMPLE 1 A set of allowed values for the status of items proposed for standardization to an ISO standard maintenance agency are defined in the ISO directives.

EXAMPLE 2 A set of allowed values for the status of items in an IEC database standard is defined in the IEC directives.

NOTE A status may be associated both with a **dictionary_element** or with a **dic_value**.

EXPRESS specification:

```
*)
TYPE status_type = identifier;
END_TYPE; -- status_type
(*
```

5.11.2.24 Dictionary_code_type

The **dictionary_code_type** is a **code_type** that identifies a dictionary.

EXPRESS specification:

```
*)
TYPE dictionary_code_type = identifier;
WHERE
     WR1: LENGTH(SELF) <= dictionary_code_len;
END_TYPE; -- dictionary_code_type
(*
```

Formal propositions:

WR1: the length of a **dictionary_code_type** value shall not exceed the length of **dictionary_code_len** (i.e., 131).

5.11.3 Basic entity definitions

5.11.3.1 General

This subclause contains the basic entity definitions, sorted alphabetically.

5.11.3.2 Dates

The **dates** entity describes the three dates associated respectively to the first stable description, to the current version and to the current revision for a given description.

NOTE For each particular dictionary management rules, it is the responsibility of the dictionary information supplier to choose which point in time corresponds to the first stable description of an item.

EXPRESS specification:

```
*)
ENTITY dates;
     date_of_original_definition: date_type;
     date_of_current_version: date_type;
     date_of_current_revision: OPTIONAL date_type;
END_ENTITY; -- dates
(*
```

Attribute definitions:

date_of_original_definition: the date associated to the first stable version of an item.

date_of_current_version: the date associated to the present version.

date_of_current_revision: the date associated to the last revision.

5.11.3.3 Document

The **document** entity is an abstract resource that stands for a document. The dictionary schema only provides for exchanging the identification of documents (see below). The **document** entity may also be subtyped with entities implementing a means for exchanging document data.

EXAMPLE By reference to an external file and exact specification of the format of the file.

EXPRESS specification:

```
*)
ENTITY document
ABSTRACT SUPERTYPE;
END_ENTITY; -- document
(*
```

5.11.3.4 Graphics

The **graphics** entity is an abstract resource to be subtyped with entities implementing a means for exchanging graphical data.

EXAMPLE By reference to an external file and exact specification of the format of the file.

EXPRESS specification:

```
*)
ENTITY graphics
ABSTRACT SUPERTYPE;
END_ENTITY; -- graphics
(*
```

5.11.3.5 External_graphics

The **external_graphics** entity provides for exchanging graphical data by means of external files referenced by a **graphic_files** entity.

EXPRESS specification:

```
*)
ENTITY external_graphics
SUBTYPE OF (graphics);
     representation: graphic_files;
END_ENTITY; -- external_graphics
(*
```

Attribute definitions:

representation: representation of a graphics by means of external files.

5.11.3.6 Graphic_files

A **graphic_files** is an **external_item** whose content is a picture.

NOTE 1 An **external_item** entity, defined in ISO 13584-24:2003, is an item whose content may be provided as library external file(s). It refers to an **external_file_protocol** that specifies how the library external file(s) should be processed, and to an **external_content** that specifies the library external file(s) that represents its content.

NOTE 2 Both **external_file_protocol** and **external_content** entities are defined in ISO 13584-24:2003.

NOTE 3 Only **external_content**s that consist of **http_file**s and only the **http_protocol external_file_protocol**s are referenced by the **ISO13584_IEC61360_dictionary_schema** and may be used in the context of this part of IEC 61360.

NOTE 4 The files of a **graphic_files** may depend upon the language; this is specified by the following subtypes of external_content: **not_translatable_external_content**, **not_translated_external_content** and **translated_external_content**.

NOTE 5 **http_file**, **http_protocol**, **not_translatable_external_content**, **not_translated_external_content** and **translated_external_content**, are defined in ISO 13584-24:2003 and referenced by the **ISO13584_IEC61360_dictionary_schema**.

EXPRESS specification:

```
*)
ENTITY graphic_files
SUBTYPE OF (external_item);
END_ENTITY; -- graphic_files
(*
```

5.11.3.7 Identified_document

The **identified_document** entity describes a document identified by its label.

EXPRESS specification:

```
*)
ENTITY identified_document
SUBTYPE OF(document);
     document_identifier: translatable_label;
WHERE
     WR1: check_label_length(SELF.document_identifier,source_doc_len);
END_ENTITY; -- identified_document
(*
```

Attribute definitions:

document_identifier: the label of the described document.

Formal propositions:

WR1: the length of a **document_identifier** value shall not exceed the length of **source_doc_len** (i.e., 255).

5.11.3.8 Item_names

The **item_names** entity identifies the names that can be associated to a given description. It states the preferred name, the set of synonymous names, the short name and the **languages** in which the different names are provided. It may be associated with an icon.

EXPRESS specification:

```
*)
ENTITY item_names;
     preferred_name: pref_name_type;
     synonymous_names: SET OF syn_name_type;
     short_name: OPTIONAL short_name_type;
     languages: OPTIONAL present_translations;
     icon: OPTIONAL graphics;
WHERE
     WR1: NOT(EXISTS(languages )) OR (
          ('ISO13584_IEC61360_LANGUAGE_RESOURCE_SCHEMA'
          + '.TRANSLATED_LABEL' IN TYPEOF(preferred_name))
          AND (languages :=:
          preferred_name\translated_label.languages)
          AND (NOT(EXISTS(short_name)) OR
          ('ISO13584_IEC61360_LANGUAGE_RESOURCE_SCHEMA'
          + '.TRANSLATED_LABEL' IN TYPEOF(short_name))
          AND (languages :=: short_name\translated_label.languages))
          AND (QUERY(s <* synonymous_names |
          NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
          '.LABEL_WITH_LANGUAGE' IN TYPEOF(s))) = []));
     WR2: NOT EXISTS(languages) OR (QUERY(s <* synonymous_names |
          EXISTS(s.language) AND NOT(s.language IN
          QUERY(l <* languages.language_codes | TRUE
          ))) = []);
     WR3: EXISTS(languages) OR
          (('SUPPORT_RESOURCE_SCHEMA.LABEL' IN
          TYPEOF(preferred_name))
```

```
            AND (NOT(EXISTS(short_name)) OR
            ('SUPPORT_RESOURCE_SCHEMA.LABEL' IN
            TYPEOF(short_name)))
            AND (QUERY(s <* synonymous_names |
            'ISO13584_IEC61360_DICTIONARY_SCHEMA.LABEL_WITH_LANGUAGE' IN
            TYPEOF(s)) = []));
    END_ENTITY; -- item_names
    (*
```

Attribute definitions:

preferred_name: the name that is preferred for use.

synonymous_names: the set of synonymous names.

short_name: the abbreviation of the preferred name.

languages: the optional list of languages in which the different names are provided.

icon: an optional **icon** which graphically represents the description associated with the **item_names**.

Formal propositions:

WR1: if preferred and short names are provided in more than one language, then all the **languages** attributes of the **translated_label**s shall contain the **present_translations** instance as in the languages attribute of this **item_names** instance.

WR2: if synonymous names are provided in more than one language, then only languages indicated in the **present_translations** instance in the **languages** attribute of this **item_names** instance can be used.

WR3: if no **languages** are provided, **preferred_name**, **short_name** and **synonymous_names** shall not be translated.

NOTE 1 The attributes **preferred_name**, **synonymous_names** and **short_name** are used to encode the "Preferred Name", "Synonymous name" and "Short name" attributes respectively for properties and classes.

NOTE 2 The attribute **languages** is used to define the sequence of translations (if requested for attributes).

5.11.3.9 Label_with_language

The **label_with_language** entity provides resources for associating a label to a language.

EXPRESS specification:

```
    *)
    ENTITY label_with_language;
          l: label;
          language: language_code;
    END_ENTITY; -- label_with_language
    (*
```

Attribute definitions:

l: the label associated to a language.

language: the code of the labeled language.

5.11.3.10 Mathematical_string

The **mathematical_string** entity provides resources defining a representation for mathematical strings. It also allows a representation in the MathML format.

EXPRESS specification:

```
*)
ENTITY mathematical_string;
     text_representation: text;
     MathML_representation: OPTIONAL text;
END_ENTITY; -- mathematical_string
(*
```

Attribute definitions:

text_representation: "linear" form of a mathematical string, using ISO 843, if necessary.

MathML_representation: MathML-Text, marked up according to the XML DTD for MathML (document Type Definition). The MathML text shall be processed so that it will be treated as one single string during the exchange (see ISO 10303-21).

5.12 Function definitions

5.12.1 General

This subclause contains functions that are referenced in WHERE clauses to assert data consistency, or that provide resources for application development.

5.12.2 Acyclic_superclass_relationship function

The **acyclic_superclass_relationship** function checks that there is no cycle in the superclass relationship. By the cardinality of the **its_superclass** attribute in ENTITY class, it is ensured that there is an inheritance tree, no acyclic graph. Thus, this function merely has to check that no class instance refers in the **its_superclass** attribute to another one that is essentially a subclass.

EXPRESS specification:

```
*)
FUNCTION acyclic_superclass_relationship(
     current: class_BSU; visited: SET OF class): LOGICAL;

IF SIZEOF(current.definition) = 1 THEN
     IF current.definition[1] IN visited THEN
          RETURN(FALSE);
     (* wrong: current declares a subclass as its superclass *)
     ELSE
          IF EXISTS(current.definition[1]\class.its_superclass)
          THEN
               RETURN(acyclic_superclass_relationship(
                    current.definition[1]\class.its_superclass,
                    visited + current.definition[1]));
          ELSE
```

```
                RETURN(TRUE);
            END_IF;
        END_IF;
ELSE
        RETURN(UNKNOWN);
END_IF;
END_FUNCTION; -- acyclic_superclass_relationship
(*
```

5.12.3 Check_syn_length function

The **check_syn_length** function checks that the length of **s** does not exceed the length indicated by **s_length**.

EXPRESS specification:

```
*)
FUNCTION check_syn_length(s: syn_name_type; s_length: INTEGER):BOOLEAN;

IF 'ISO13584_IEC61360_DICTIONARY_SCHEMA.LABEL_WITH_LANGUAGE'
        IN TYPEOF(s)
THEN
        RETURN(LENGTH(s.l) <= s_length);
ELSE
        RETURN(LENGTH(s) <= s_length);
END_IF;
END_FUNCTION; -- check_syn_length
(*
```

5.12.4 Codes_are_unique function

The **codes_are_unique** function returns TRUE if the **value_code**s are unique within this list of values.

EXPRESS specification:

```
*)
FUNCTION codes_are_unique(values: LIST OF dic_value): BOOLEAN;
LOCAL
        ls: SET OF STRING := [];
        li: SET OF INTEGER := [];
END_LOCAL;

IF('ISO13584_IEC61360_DICTIONARY_SCHEMA.VALUE_CODE_TYPE' IN
        TYPEOF(values[1].value_code))
THEN
        REPEAT i := 1 TO SIZEOF(values);
            ls := ls + values[i].value_code;
        END_REPEAT;

        RETURN(SIZEOF(values) = SIZEOF(ls));
ELSE
        IF('ISO13584_IEC61360_DICTIONARY_SCHEMA.INTEGER_TYPE' IN
            TYPEOF(values[1].value_code))
        THEN
```

```
            REPEAT i := 1 TO SIZEOF(values);
                li := li + values[i].value_code;
            END_REPEAT;

            RETURN(SIZEOF(values) = SIZEOF(li));
        ELSE
            RETURN(?);
        END_IF;
    END_IF;

    END_FUNCTION; -- codes_are_unique
    (*
```

5.12.5 Definition_available_implies function

The **definition_available_implies** function checks whether the definition corresponding to the **BSU** parameter exists or not. Then, if this definition exists, the **expression** parameter is returned.

EXPRESS specification:

```
    *)
    FUNCTION definition_available_implies(
        BSU: basic_semantic_unit;
        expression: LOGICAL): LOGICAL;

    RETURN(NOT(SIZEOF(BSU.definition) = 1) OR expression);

    END_FUNCTION; -- definition_available_implies
    (*
```

5.12.6 Is_subclass function

The function **is_subclass** returns TRUE if **sub** is either **super** or a subclass of **super**. If some class **dictionary_definition** are not available, the function returns UNKNOWN.

EXPRESS specification:

```
    *)
    FUNCTION is_subclass(sub, super: class): LOGICAL;
        IF (NOT EXISTS(sub)) OR (NOT EXISTS(super)) THEN
            RETURN(UNKNOWN);
        END_IF;

        IF sub = super
        THEN
            RETURN(TRUE);
        END_IF;

        IF NOT EXISTS(sub.its_superclass)
        THEN
            (* end of chain reached, didn't meet super so far *)
            RETURN(FALSE);
        END_IF;
        IF SIZEOF(sub.its_superclass.definition) = 1
        THEN
        (* definition available *)
             IF (sub.its_superclass.definition[1] = super)
             THEN
                  RETURN(TRUE);
             ELSE
                  RETURN(is_subclass(sub.its_superclass.definition[1],
                       super));
             END_IF;
        ELSE
             RETURN(UNKNOWN);
        END_IF;

   END_FUNCTION; -- is_subclass
   (*
```

5.12.7 String_for_derived_unit function

The function **string_for_derived_unit** returns a STRING representation of the **derived_unit** (according to ISO 10303-41) passed as parameter. First, the elements of the derived unit are separated according to the sign of the exponent. If there are elements of both kinds, the '/' notation is used to separate those with positive from those with negative sign. If there are only negative exponents, the u-e notation is used. A dot '.' (decimal code 46 according to ISO/IEC 8859-1, see ISO 10303-21) is used to separate individual elements.

<u>EXPRESS specification:</u>

```
   *)
   FUNCTION string_for_derived_unit(u: derived_unit): STRING;

        FUNCTION string_for_derived_unit_element(
             u: derived_unit_element; neg_exp: BOOLEAN
             (* print negative exponents with power -1 *)): STRING;
             (* returns a STRING representation of the
                  derived_unit_element (according to ISO 10303-41)
                  passed as parameter *)

        LOCAL
             result: STRING;
        END_LOCAL;

        result := string_for_named_unit(u.unit);
        IF (u.exponent <> 0)
        THEN
             IF (u.exponent > 0) OR NOT neg_exp
             THEN
                  result := result + '**' + FORMAT(
                       ABS(u.exponent), '2I')[2];
             ELSE
                  result := result + '**' + FORMAT(u.exponent, '2I')[2];
             END_IF;
        END_IF;
             RETURN(result);
        END_FUNCTION; -- string_for_derived_unit_element
```

```
LOCAL
     pos, neg: SET OF derived_unit_element;
     us: STRING;
END_LOCAL;

(* separate unit elements according to the sign of the exponents: *)
pos := QUERY(ue <* u.elements | ue.exponent > 0);
neg := QUERY(ue <* u.elements | ue.exponent < 0);
us := '';
IF SIZEOF(pos) > 0 THEN
     (* there are unit elements with positive sign *)
     REPEAT i := LOINDEX(pos) TO HIINDEX(pos);
          us := us + string_for_derived_unit_element(pos[i], FALSE);
          IF i <> HIINDEX(pos)
          THEN
               us := us + '.';
          END_IF;
     END_REPEAT;

     IF SIZEOF(neg) > 0
     THEN
          (* there are unit elements with negative sign, use '/'
               notation: *)
          us := us + '/';

          IF SIZEOF(neg) > 1
          THEN
               us := us + '(';
          END_IF;

          REPEAT i := LOINDEX(neg) TO HIINDEX(neg);
               us := us + string_for_derived_unit_element(
                    neg[i], FALSE);
               IF i <> HIINDEX(neg)
               THEN
                    us := us + '.';
               END_IF;
          END_REPEAT;

          IF SIZEOF(neg) > 1
          THEN
               us := us + ')';
          END_IF;
     END_IF;
ELSE
     (* only negative signs, use u-e notation *)
     IF SIZEOF(neg) > 0 THEN
          REPEAT i := LOINDEX(neg) TO HIINDEX(neg);
               us := us + string_for_derived_unit_element(
                    neg[i], TRUE);
               IF i <> HIINDEX(neg)
               THEN
                    us := us + '.';
               END_IF;
```

```
           END_REPEAT;
      END_IF;
END_IF;

RETURN(us);

END_FUNCTION; -- string_for_derived_unit
(*
```

5.12.8 String_for_named_unit function

The **string_for_named_unit** function returns a STRING representation of the **named_unit** (according to ISO 10303-41 and the extension in 5.10.6.2) passed as parameter.

EXPRESS specification:

```
*)
FUNCTION string_for_named_unit(u: named_unit): STRING;

IF 'MEASURE_SCHEMA.SI_UNIT' IN TYPEOF(u) THEN
      RETURN(string_for_SI_unit(u));
ELSE
      IF 'MEASURE_SCHEMA.CONTEXT_DEPENDENT_UNIT' IN TYPEOF(u)
      THEN
           RETURN(u\context_dependent_unit.name);
      ELSE
           IF 'MEASURE_SCHEMA.CONVERSION_BASED_UNIT' IN TYPEOF(u)
           THEN
                RETURN(u\conversion_based_unit.name);
           ELSE
                IF 'ISO13584_IEC61360_DICTIONARY_SCHEMA'
                     +'.NON_SI_UNIT' IN TYPEOF(u)
                THEN
                     RETURN(u\non_si_unit.name);
                ELSE
                     RETURN('name_unknown');
                END_IF;
           END_IF;
      END_IF;
END_IF;

END_FUNCTION; -- string_for_named_unit
(*
```

5.12.9 String_for_SI_unit function

The **string_for_SI_unit** function returns a STRING representation of the **si_unit** (according to ISO 10303-41) passed as parameter.

EXPRESS specification:

```
*)
FUNCTION string_for_SI_unit(unit: si_unit): STRING;

LOCAL
```

```
      prefix_string, unit_string: STRING;
END_LOCAL;

IF EXISTS(unit.prefix) THEN
      CASE unit.prefix OF
           exa       : prefix_string := 'E';
           peta      : prefix_string := 'P';
           tera      : prefix_string := 'T';
           giga      : prefix_string := 'G';
           mega      : prefix_string := 'M';
           kilo      : prefix_string := 'k';
           hecto     : prefix_string := 'h';
           deca      : prefix_string := 'da';
           deci      : prefix_string := 'd';
           centi     : prefix_string := 'c';
           milli     : prefix_string := 'm';
           micro     : prefix_string := 'u';
           nano      : prefix_string := 'n';
           pico      : prefix_string := 'p';
           femto     : prefix_string := 'f';
           atto      : prefix_string := 'a';
      END_CASE;
ELSE
      prefix_string := '';
END_IF;

CASE unit.name OF
      metre              : unit_string:= 'm';
      gram               : unit_string := 'g';
      second             : unit_string := 's';
      ampere             : unit_string := 'A';
      kelvin             : unit_string := 'K';
      mole               : unit_string := 'mol';
      candela            : unit_string := 'cd';
      radian             : unit_string := 'rad';
      steradian          : unit_string := 'sr';
      hertz              : unit_string := 'Hz';
      newton             : unit_string := 'N';
      pascal             : unit_string := 'Pa';
      joule              : unit_string := 'J';
      watt               : unit_string := 'W';
      coulomb            : unit_string := 'C';
      volt               : unit_string := 'V';
      farad              : unit_string := 'F';
      ohm                : unit_string := 'Ohm';
      siemens            : unit_string := 'S';
      weber              : unit_string := 'Wb';
      tesla              : unit_string := 'T';
      henry              : unit_string := 'H';
      degree_Celsius     : unit_string := 'Cel';
      lumen              : unit_string := 'lm';
      lux                : unit_string := 'lx';
      becquerel          : unit_string := 'Bq';
      gray               : unit_string := 'Gy';
      sievert            : unit_string := 'Sv';
END_CASE;

RETURN(prefix_string + unit_string);

END_FUNCTION; -- string_for_SI_unit
(*
```

5.12.10 String_for_unit function

The **string_for_unit** function returns a STRING representation of the **unit** (according to ISO 10303-41) passed as parameter.

NOTE The **string_for_unit** function is not called from the EXPRESS code. It is a utility function allowing to compute a string representation from the **structured_representation** of a **dic_unit**, for the case where no **string_representation** is present. This string representation corresponds to the one used in Annex B of IEC 61360-1:2009.

EXPRESS specification:

```
*)
FUNCTION string_for_unit(u: unit): STRING;
      IF 'MEASURE_SCHEMA.DERIVED_UNIT' IN TYPEOF(u)
      THEN
           RETURN(string_for_derived_unit(u));
      ELSE (* 'MEASURE_SCHEMA.NAMED_UNIT' IN TYPEOF(u) holds true *)
           RETURN(string_for_named_unit(u));
      END_IF;
END_FUNCTION; -- string_for_unit
(*
```

5.12.11 All_class_descriptions_reachable function

The **all_class_descriptions_reachable** function checks if the **dictionary_element**s that describe a class, referred by a **class_BSU**, and all its super-classes, can be computed in the inheritance tree defined by the class hierarchy.

EXPRESS specification:

```
*)
FUNCTION all_class_descriptions_reachable(cl: class_BSU): BOOLEAN;

IF NOT EXISTS(cl)
THEN
      RETURN(?);
END_IF;

IF SIZEOF(cl.definition) = 0
THEN
      RETURN(FALSE);
END_IF;

IF NOT(EXISTS(cl.definition[1]\class.its_superclass))
THEN
      RETURN(TRUE);
ELSE
```

```
          RETURN(all_class_descriptions_reachable(
               cl.definition[1]\class.its_superclass));
     END_IF;

     END_FUNCTION; -- all_class_descriptions_reachable
     (*
```

5.12.12 Compute_known_visible_properties function

The **compute_known_visible_properties** function computes the set of properties that are visible for a given class. When a definition is not available, it returns only the visible properties that may be computed.

NOTE When some class **dictionary_definition** is not present in the same exchange context (a PLIB exchange context is never assumed to be complete), the super-class of a class may not be known. Therefore the properties defined as visible by this super-class cannot be computed by the **compute_known_visible_properties** function. Only on the receiving system all the **dictionary_definition**s of the **BSU**s are available. Therefore, on the receiving system, the **compute_known_visible_properties** function computes all the properties that are visible to a class by virtue of referencing it (or any of its superclass) by their **name_scope** attribute.

EXPRESS specification:

```
     *)
     FUNCTION compute_known_visible_properties(cl: class_BSU):
          SET OF property_BSU;
     LOCAL
          s: SET OF property_BSU := [];
     END_LOCAL;

     s := s + USEDIN(cl, 'ISO13584_IEC61360_DICTIONARY_SCHEMA' +
          '.PROPERTY_BSU.NAME_SCOPE');
     IF SIZEOF(cl.definition) = 0
     THEN
          RETURN(s);
     ELSE
          IF EXISTS(cl.definition[1]\class.its_superclass) THEN
               s := s + compute_known_visible_properties(
                    cl.definition[1]\class.its_superclass);
          END_IF;

          RETURN(s);
     END_IF;

     END_FUNCTION; -- compute_known_visible_properties
     (*
```

5.12.13 Compute_known_visible_data_types function

The **compute_known_visible_data_types** function computes the set of **data_type**s that are visible for a given class. When a definition is not available, it returns only the visible **data_type**s that may be computed.

NOTE When some class **dictionary_definition** is not present in the same exchange context (a PLIB exchange context is never assumed to be complete), the super-class of a class may not be known. Therefore the **data_type**s defined as visible by this super-class cannot be computed by the **compute_known_visible_data_types** function. Only on the receiving system all the **dictionary_definition**s of the **BSU**s are available. Therefore, on the receiving system, the **compute_known_visible_data_types** function computes all the **data_type**s that are visible to a class by virtue of referencing it (or any of its superclass) by their **name_scope** attribute.

EXPRESS specification:

```
*)
FUNCTION compute_known_visible_data_types(cl: class_BSU):
      SET OF data_type_BSU;
LOCAL
      s: SET OF data_type_BSU :=[ ];
END_LOCAL;

s := s + USEDIN(cl, 'ISO13584_IEC61360_DICTIONARY_SCHEMA' +
      '.DATA_TYPE_BSU.NAME_SCOPE');

IF SIZEOF(cl.definition) = 0
THEN
      RETURN(s);
ELSE
      IF EXISTS(cl.definition[1]\class.its_superclass)
      THEN
           s := s + compute_known_visible_data_types(
                cl.definition[1]\class.its_superclass);
      END_IF;

      RETURN(s);
END_IF;

END_FUNCTION; -- compute_known_visible_data_types
(*
```

5.12.14 Compute_known_applicable_properties function

The **compute_known_applicable_properties** function computes the set of properties that are applicable for a given class. When a definition is not available, it returns only the applicable properties that may be computed.

NOTE When some class **dictionary_definition** is not present in the same exchange context (a PLIB exchange context is never assumed to be complete), the super-class of a class may not be known. Therefore the properties defined as applicable by this super-class cannot be computed by the **compute_known_applicable_properties** function. Only on the receiving system all the **dictionary_definition**s of the **BSU**s are available. Therefore, on the receiving system, the **compute_known_applicable_properties** function computes all the properties that are applicable to a class by virtue of being referenced by a **described_by** attribute or of being imported through an **a_priori_semantic_relationship**.

EXPRESS specification:

```
*)
FUNCTION compute_known_applicable_properties(cl: class_BSU):
      SET OF property_BSU;

LOCAL
      s: SET OF property_BSU := [];
END_LOCAL;

IF SIZEOF(cl.definition)=0
THEN
      RETURN(s);
ELSE
      REPEAT i := 1 TO SIZEOF(cl.definition[1]\class.described_by);
```

```
          s := s + cl.definition[1]\class.described_by[i];
     END_REPEAT;

     IF (('ISO13584_IEC61360_ITEM_CLASS_CASE_OF_SCHEMA.'
          + 'A_PRIORI_SEMANTIC_RELATIONSHIP')
          IN TYPEOF (cl.definition[1]))
     THEN
          s                    :=                    s                    +
               cl.definition[1]\a_priori_semantic_relationship.referenc
           ed_properties;
     END_IF;

     IF EXISTS(cl.definition[1]\class.its_superclass)
     THEN
          s := s + compute_known_applicable_properties(
               cl.definition[1]\class.its_superclass);
     END_IF;

     RETURN(s);
END_IF;
END_FUNCTION; -- compute_known_applicable_properties
(*
```

5.12.15 Compute_known_applicable_data_types function

The **compute_known_applicable_data_types** function computes the set of **data_type**s that are applicable for a given class. When a definition is not available, it returns only the applicable **data_type**s that may be computed.

NOTE When some class **dictionary_definition** is not present in the same exchange context (a PLIB exchange context is never assumed to be complete), the super-class of a class may not be known. Therefore the **data_type**s defined as applicable by this super-class cannot be computed by the **compute_known_applicable_data_types** function. Only on the receiving system all the **dictionary_definition**s of the **BSU**s are available. Therefore, on the receiving system, the **compute_known_applicable_data_types** function computes all the **data_type**s that are applicable to a class by virtue of being referenced by a **defined_types** attribute or of being imported through an **a_priori_semantic_relationship**.

EXPRESS specification:

```
*)
FUNCTION compute_known_applicable_data_types(cl: class_BSU):
     SET OF data_type_BSU;
LOCAL
     s: SET OF data_type_BSU := [];
END_LOCAL;

IF SIZEOF(cl.definition) = 0
THEN
     RETURN(s);
ELSE
     REPEAT i := 1 TO SIZEOF(cl.definition[1]\class.defined_types);
          s := s + cl.definition[1]\class.defined_types[i];
     END_REPEAT;

     IF (('ISO13584_IEC61360_ITEM_CLASS_CASE_OF_SCHEMA.'
          + 'A_PRIORI_SEMANTIC_RELATIONSHIP')
```

```
            IN TYPEOF (cl.definition[1]))
        THEN
            s                    :=                    s                    +
                cl.definition[1]\a_priori_semantic_relationship.referenc
             ed_data_types;
        END_IF;

        IF EXISTS(cl.definition[1]\class.its_superclass)
        THEN
            s := s + compute_known_applicable_data_types(
                cl.definition[1]\class.its_superclass);
        END_IF;

        RETURN(s);
    END_IF;

    END_FUNCTION; -- compute_known_applicable_data_types
    (*
```

5.12.16 List_to_set function

The **list_to_set** function creates a SET from a LIST named **l**, the type of element for the SET will be the same as that in the original LIST.

EXPRESS specification:

```
    *)
    FUNCTION list_to_set(l: LIST [0:?] OF GENERIC:type_elem):
        SET OF GENERIC: type_elem;

    LOCAL
        s: SET OF GENERIC: type_elem := [];
    END_LOCAL;

    REPEAT i := 1 TO SIZEOF(l);
        s := s + l[i];
    END_REPEAT;

    RETURN(s);
    END_FUNCTION; -- list_to_set
    (*
```

5.12.17 Check_properties_applicability function

The check_properties_applicability function checks that only those properties that are not applicable for a class by inheritance may become applicable to this class by virtue of being referenced by the **described_by** attribute.

EXPRESS specification:

```
    *)
    FUNCTION check_properties_applicability(cl: class): LOGICAL;
    LOCAL
        inter: SET OF property_bsu := [];
    END_LOCAL;
```

```
IF EXISTS(cl.its_superclass)
THEN
      IF (SIZEOF(cl.its_superclass.definition)=1)
      THEN
           inter := (list_to_set(cl.described_by) *
                cl.its_superclass.definition[1]\class.
                known_applicable_properties);
           RETURN(inter = []);
      ELSE
           RETURN(UNKNOWN);
      END_IF;
ELSE
      RETURN(TRUE);
END_IF;

END_FUNCTION; -- check_properties_applicability
(*
```

5.12.18 Check_datatypes_applicability function

The **check_datatypes_applicability** function checks that only those datatypes that are not applicable for a class by inheritance may become applicable to this class by virtue of being referenced by the **defined_types** attribute.

<u>EXPRESS specification:</u>

```
*)
FUNCTION check_datatypes_applicability(cl: class): LOGICAL;
LOCAL
      inter: SET OF data_type_bsu := [];
END_LOCAL;

IF EXISTS(cl.its_superclass)
THEN
      IF (SIZEOF(cl.its_superclass.definition) = 1)
      THEN
           inter := cl.defined_types *
                cl.its_superclass.definition[1]\class.
                known_applicable_data_types;
           RETURN(inter = []);
      ELSE
           RETURN(UNKNOWN);
      END_IF;
ELSE
      RETURN(TRUE);
END_IF;

END_FUNCTION; -- check_datatypes_applicability

(*
```

5.12.19 One_language_per_translation function

The **one_language_per_translation** function checks that the languages of translation in **administrative_data** are unique.

EXPRESS specification:

```
*)
FUNCTION one_language_per_translation (adm: administrative_data)
                         : LOGICAL;
     LOCAL
          count: INTEGER;
          lang: language_code;
     END_LOCAL;

     REPEAT i := 1 TO SIZEOF (adm.translation);
          lang := adm.translation[i].language;
          count := 0;
          REPEAT j :=1 TO SIZEOF (adm.translation);
               IF lang = adm.translation[j].language
               THEN
                    count := count+1;
               END_IF;
          END_REPEAT;
          IF count >1
          THEN RETURN (FALSE);
          END_IF;
     END_REPEAT;
     RETURN(TRUE);

END_FUNCTION; -- one_language_per_translation
(*
```

5.12.20 Allowed_values_integer_types function

The **allowed_values_integer_types** function computes the set of **integer_type** values allowed by a **non_quantitative_int_type.** If the parameter is indeterminate, it returns the indeterminate value.

EXPRESS specification:

```
*)
FUNCTION allowed_values_integer_types (nqit: non_quantitative_int_type)
                         : SET OF integer_type;
LOCAL
     s : SET OF integer_type :=[];
END_LOCAL;

REPEAT i:=1 TO SIZEOF (nqit.domain.its_values);
     s := s + nqit.domain.its_values[i].value_code;
END_REPEAT;
RETURN(s);

END_FUNCTION; -- allowed_values_integer_types
(*
```

5.12.21 Is_class_valued_property function

The **is_class_valued_property** function returns TRUE if the property **prop** is defined as a class valued property for class **cl** by means of a **sub_class_properties** attribute in class **cl** or in any of its superclasses. If some class **dictionary_definition**s are not available to compute all the superclasses of **cl**, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION is_class_valued_property(
      prop: property_BSU; cl: class_BSU): LOGICAL;
      IF (SIZEOF(cl.definition) = 0)
      THEN
           RETURN (UNKNOWN);
      ELSE
           IF NOT (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
                +'.ITEM_CLASS') IN TYPEOF(cl.definition[1]))
           THEN
                RETURN (FALSE);
           END_IF;
           IF prop IN cl.definition[1].sub_class_properties
           THEN RETURN (TRUE);
           END_IF;
           IF NOT EXISTS(cl.definition[1].its_superclass)
           THEN
           (* end of chain reached, didn't meet super so far *)
                RETURN(FALSE);
           END_IF;
           RETURN(is_class_valued_property(prop,
                     cl.definition[1].its_superclass));
      END_IF;

END_FUNCTION; -- is_class_valued_property
(*
```

5.12.22 Class_value_assigned function

The **class_value_assigned** function returns the set of values of the property **prop** that have been assigned to a class **cl** by means of a **class_constant_values** attribute in class **cl** or in any superclass of class **cl**. If several values are assigned in several superclasses the function returns the set of all assigned values. If some class **dictionary_definition**s are not available to compute all the superclasses of **cl**, only the values computed are returned.

EXPRESS specification:

```
*)
FUNCTION class_value_assigned(prop: property_BSU;
      cl: class_BSU) : SET OF primitive_value;
      LOCAL
           val:SET OF primitive_value :=[];
           cva : SET OF class_value_assignment :=[];
      END_LOCAL;
      IF (SIZEOF(cl.definition) = 0)
      THEN
           RETURN (val);
```

```
        END_IF;
        IF NOT (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
             +'.ITEM_CLASS') IN TYPEOF(cl.definition[1]))
        THEN
             RETURN (val);
        END_IF;
        IF EXISTS(cl.definition[1])
        THEN
             cva:= QUERY
                  (a <* cl.definition[1].class_constant_values
                  | a.super_class_defined_property = prop);
             REPEAT i :=1 TO SIZEOF (cva);
                  val := val + cva[i].assigned_value;
             END_REPEAT;
             IF NOT EXISTS(cl.definition[1].its_superclass)
             THEN
                  RETURN (val);
             ELSE RETURN (val + class_value_assigned
                  (prop,cl.definition[1].its_superclass));
             END_IF;
        END_IF;
   END_FUNCTION; -- class_value_assigned


   END_SCHEMA; -- ISO13584_IEC61360_dictionary_schema
   (*
```

6 ISO13584_IEC61360_language_resource_schema

6.1 Overview

The following schema provides resources for permitting strings in various languages. It has been extracted from the dictionary schema, since it could be used in other schemata. It is largely based on the **support_resource_schema** from ISO 10303-41, and can be seen as an extension to that. It allows for the usage of one specific language throughout an exchange context (physical file) without the overhead introduced when multiple languages are used. See Figure 13 for a graphical depiction.

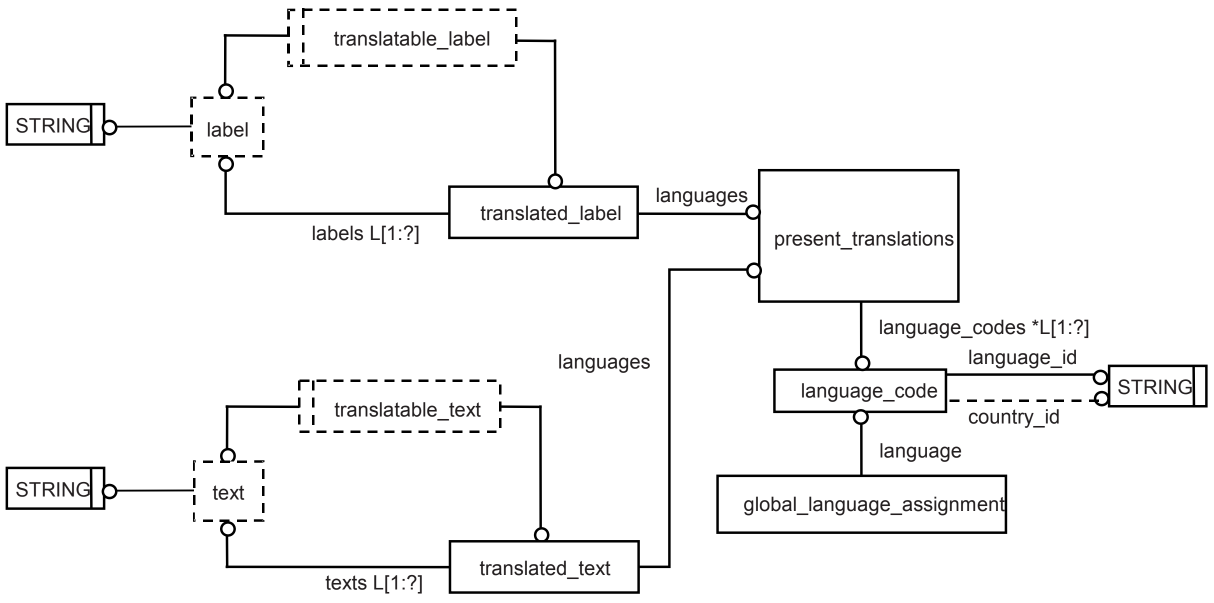

Figure 13 – ISO13584_IEC61360_language_resource_schema and support_resource_schema

EXPRESS specification:

```
*)
SCHEMA ISO13584_IEC61360_language_resource_schema;

REFERENCE FROM support_resource_schema(identifier, label, text);

(*
```

NOTE The **support_resource_schema** schema referenced above can be found in ISO 10303-41.

6.2 ISO13584_IEC61360_language_resource_schema type and entity definitions

6.2.1 general

This subclause contains the EXPRESS type and entity definitions in the **ISO13584_IEC61360_language_resource_schema**.

6.2.2 Language_code

The **language_code** entity enables to identify a language according to ISO 639-1. It contains two codes:

– the language as defined in ISO 639-1 or ISO 639-2, and, optionally
– the country code, as defined in ISO 3166-1, specifying in which country the language is spoken.

EXPRESS specification:

```
*)
ENTITY language_code;
      language_id: identifier;
      country_id: OPTIONAL identifier;
```

```
WHERE
     WR1: (LENGTH (language_id) = 2) OR (LENGTH (language_id) = 3);
     WR2: LENGTH (country_id) = 2;
END_ENTITY; -- language_code
(*
```

Attribute definitions:

language_id: the code that specifies the language as defined by ISO 639-1 or ISO 639-2.

country_id: the code that specifies the country where the language is spoken as defined by ISO 3166-1.

Formal propositions:

WR1: the length of **language_id** value shall be equal to 2 or 3.

WR2: the length of a **country_id** value shall be equal to 2.

6.2.3 Global_language_assignment

The **global_language_assignment** entity specifies the language for **translatable_label** and **translatable_text**, if **label** and **text** are selected, respectively (i.e., without explicit language indication as is done in **translated_label** and **translated_text)**.

EXPRESS specification:

```
*)
ENTITY global_language_assignment;
     language: language_code;
END_ENTITY; -- global_language_assignment
(*
```

Attribute definitions:

language: the code of the assigned language.

6.2.4 Present_translations

The **present_translations** entity serves to indicate the languages used for **translated_label** and **translated_text**.

EXPRESS specification:

```
*)
ENTITY present_translations;
     language_codes: LIST [1:?] OF UNIQUE language_code;
UNIQUE
     UR1: language_codes;
END_ENTITY; -- present_translations
(*
```

Attribute definitions:

language_codes: the list of unique language codes corresponding to the language in which a translation is made.

Formal proposition:

UR1: for each list of **language_code** there shall be a unique instance of **present_translations**.

6.2.5 Translatable_label

A translatable_label defines a type of values that can be labels or translated_labels.

EXPRESS specification:

```
*)
TYPE translatable_label = SELECT(label, translated_label);
END_TYPE; -- translatable_label
(*
```

6.2.6 Translated_label

The **translated_label** entity defines the labels that are translated and the corresponding languages of translation.

EXPRESS specification:

```
*)
ENTITY translated_label;
      labels: LIST [1:?] OF label;
      languages: present_translations;
WHERE
      WR1: SIZEOF(labels) = SIZEOF(languages.language_codes);
END_ENTITY; -- translated_label
(*
```

Attribute definitions:

labels: the list of **label**s that are translated.

languages: the list of **languages** in which each label is translated.

Formal propositions:

WR1: the number of **label**s contained in the **labels** list shall be equal to the number of languages provided in the **languages.language_codes** attribute.

Informal propositions:

IP1: the content of **labels[i]** is in the language identified by **languages.language_codes[i]**.

6.2.7 Translatable_text

A **translatable_text** defines a type of values that can be **text**s or **translated_text**s.

EXPRESS specification:

```
*)
TYPE translatable_text = SELECT(text, translated_text);
END_TYPE; -- translatable_text
(*
```

6.2.8 Translated_text

The **translated_text** entity defines the **text**s that are translated and the corresponding **language**s of translation.

EXPRESS specification:

```
*)
ENTITY translated_text;
      texts: LIST [1:?] OF text;
      languages: present_translations;
WHERE
      WR1: SIZEOF(texts) = SIZEOF(languages.language_codes);
END_ENTITY; -- translated_text
(*
```

Attribute definitions:

texts: the list of **text**s that are translated.

languages: the list of languages in which each text is translated.

Formal propositions:

WR1: the number of **text**s contained in the **texts** list shall be equal to the number of languages provided in the **languages.language_codes** attribute.

Informal propositions:

IP1: the content of **texts[i]** is in the language identified by **languages.language_codes[i]**.

6.3 ISO13584_IEC61360_language_resource_schema function definitions

6.3.1 General

This subclause contains a function that is referenced in WHERE clauses to assert data consistency.

6.3.2 Check_label_length function

The **check_label_length** function checks that no label in **l** exceeds the length indicated by **l_length**.

EXPRESS specification:

```
*)
FUNCTION check_label_length(l: translatable_label;
      l_length: INTEGER): BOOLEAN;

IF 'ISO13584_IEC61360_LANGUAGE_RESOURCE_SCHEMA.TRANSLATED_LABEL'
```

```
          IN TYPEOF(l)
     THEN
          REPEAT i :=1 TO SIZEOF(l.labels);
               IF LENGTH(l.labels[i]) > l_length
               THEN
                    RETURN(FALSE);
               END_IF;
          END_REPEAT;

          RETURN(TRUE);

     ELSE (* the argument l is a single string *)
          RETURN(LENGTH(l) <= l_length);
     END_IF;
     END_FUNCTION; -- check_label_length
     (*
```

6.4 ISO13584_IEC61360_language_resource_schema rule definition

The rule **single_language_assignment** asserts that only one language may be assigned to be used in **translatable_label** and **translatable_text**.

EXPRESS specification:

```
     *)
     RULE single_language_assignment FOR(global_language_assignment);
     WHERE
          SIZEOF(global_language_assignment) <= 1;
     END_RULE; -- single_language_assignment


     END_SCHEMA; -- ISO13584_IEC61360_language_resource_schema
     (*
```

7 ISO13584_IEC61360_class_constraint_schema

7.1 General

This clause defines the requirements for **class_constraint_schema**. The following EXPRESS declaration introduces the **ISO13584_IEC61360_class_constraint_schema** block and identifies the necessary external references.

EXPRESS specification:

```
     *)
     SCHEMA ISO13584_IEC61360_class_constraint_schema;

     REFERENCE FROM ISO13584_IEC61360_dictionary_schema (
          class_BSU,
          property_BSU,
          definition_available_implies,
          is_subclass,
          data_type,
          simple_type,
          complex_type,
          named_type,
          allowed_values_integer_types);

     REFERENCE FROM ISO13584_extended_dictionary_schema
          (data_type_typeof,
          data_type_class_of,
          data_type_type_name);

     REFERENCE FROM ISO13584_instance_resource_schema
          (Boolean_value,
          compatible_class_and_class,
          complex_value,
          dic_class_instance,
          entity_instance_value,
          int_level_spec_value,
          integer_value,
          level_spec_value,
          number_value,
          primitive_value,
          rational_value,
          real_level_spec_value,
          real_value,
          right_values_for_level_spec,
          same_translations,
          simple_value,
          string_value,
          translatable_string_value,
          translated_string_value,
          property_or_data_type_BSU);

     REFERENCE FROM ISO13584_aggregate_value_schema
          (aggregate_entity_instance_value,
          list_value,
          set_value,
          bag_value,
          array_value,
          set_with_subset_constraint_value,
          compatible_complete_types_and_value);
     (*
```

NOTE The schemata referenced above can be found in the following documents:

ISO13584_IEC61360_dictionary_schema (which is duplicated for convenience in 4.5 and after).	IEC 61360-2
ISO13584_extended_dictionary_schema	ISO 13584-24:2003
ISO13584_instance_resource_schema	ISO 13584-24:2003
ISO13584_aggregate_value_schema	ISO 13584-25

7.2 Introduction to the ISO13584_IEC61360_class_constraint_schema

The **ISO13584_IEC61360_class_constraint_schema** provides EXPRESS constructs allowing to redefine, by restriction, the domain of values of a given property when it is applied to a subclass of the characterization class where the property was defined as visible. This constraint shall only make explicit a restriction of the domain of values that already results from the class structure.

EXAMPLE In ISO 13584-511, the class *metric threaded bolt/screw* is a class defined as follows: "headed externally threaded fastener with a cylindrical shank, which may be partly or fully threaded and the head may be furnished with a driving feature". This class has, among other, two properties called *type of head,* and *head properties*. The domain of values of the *type of head* property is a non quantitative data type that includes in particular the following values: *hexagon_head*, *octagonal_head* and *round_head*. The *head properties* property is a feature. It means that it has an *item_class* data type, whose domain is a class *head* that defines any kind of head. The *head* class has several subclasses including: *hexagon head*, associated with all the properties allowing to describe a hexagon head (e. g., *width across flats)*, and *round head*, associated with all the properties allowing to describe a round head (e. g., *head diameter*).

The class *metric threaded bolt/screw* has a subclass called *hexagon head screw* defined as follows: "metric externally threaded fastener with a hexagon head threaded up to the head". This class inherits the properties *type of head* and *head*. From the definition of the *hexagon head screw* subclass, it is clear that the *type of head* property could only take the *hexagon_head* value, and that the *head properties* could only be an instance of *hexagon head* feature class. But these constraints are implicit: they are just stated informally in the definition.

Thus these constraints are not computer sensible. The constraints defined in the **ISO13584_IEC61360_class_constraint_schema** would allow to make these two constraints explicit by associating with the *hexagon head screw* class: (1) an **enumeration_constraint** for the *type of head* property (allowing only the *hexagon_head* code) and (2) a **subclass_constraint** *for the head properties* property (allowing only the *hexagon head* feature class).

Constraints are inherited. When a property whose domain of values has already been restricted in a class C through a constraint needs to be further restricted in a subclass of C through another constraint; both constraints apply together. Thus the real domain of values in the C subclass is the intersection of the two domains defined by the two constraints. The proposed mechanism is similar to the type mechanism redefinition operation available in the EXPRESS language.

This schema allows to express constraints that may apply to data types from the type system of the **ISO13584_IEC61360_dictionary_schema**. Rules are used in those entities that reference a constraint to ensure that each constraint may apply to the data type to which it is related.

7.3 ISO13584_IEC61360_class_constraint_schema entity definitions

7.3.1 General

This clause defines the entities in the ISO13584_IEC61360_class_constraint_schema.

7.3.2 Constraint

The **constraint** entity allows to define a constraint.

EXPRESS specification:

```
*)
ENTITY constraint
ABSTRACT SUPERTYPE OF ( ONEOF (
                    property_constraint,
                    class_constraint));
      constraint_id: OPTIONAL constraint_identifier;
END_ENTITY; -- constraint
(*
```

Attribute definitions:

constraint_id: the **constraint_identifier** that identifies the constraint.

7.3.3 Property_constraint

The **property_constraint** entity 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.

EXPRESS specification:

```
*)
ENTITY property_constraint
ABSTRACT SUPERTYPE OF ( ONEOF (
                   integrity_constraint,
                   context_restriction_constraint))
SUBTYPE OF (constraint);
     constrained_property: property_BSU;
END_ENTITY; -- property_constraint
(*
```

Attribute definitions:

constrained_property: the **property_BSU** for which the constraint applies.

7.3.4 Class_constraint

The **class_constraint** entity is a constraint that restricts the allowed set of instances of a class by constraining several properties or global constraints.

EXPRESS specification:

```
*)
ENTITY class_constraint
ABSTRACT SUPERTYPE OF (configuration_control_constraint)
SUBTYPE OF (constraint);
END_ENTITY; -- class_constraint
(*
```

7.3.5 Configuration_control_constraint

The **configuration_control_constraint** entity 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 its **constraints** attribute or inherits of a class that does so. The **configuration_control_constraint** entity 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.

EXAMPLE A *bolted assembly* consists of the following set of fasteners: one e*xternally threaded fastener*, any number of *washers* and one or more *nuts*. There exist various kinds of threads, including *tapping screw thread, wood screw thread, metric external thread, metric internal thread, imperial internal thread* and *imperial external thread*. Let us assume that one wants to describe a *metric bolted assembly.* One needs to ensure that whatever be the precise structure of the assembly, both the e*xternally threaded fastener* and all the *nut*s involved in the assembly have metric thread. This can be done by specifying in the *metric bolted assembly* class the **configuration_control_constraint** that ensures that any ISO 13584-511 -described fastener referenced by any instance of this class, or any of its subclass, should belong to classes that either do not have value for the *type of thread* property (e. g., *washer*), or whose values should belong to the set: {*metric external thread, metric internal thread*}.

NOTE 1 Both **precondition** and **postcondition** may only restrict properties whose data type is **non_quantitative_code_type**. 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., **sub_class_properties** in a **class**.

NOTE 2 Properties referenced in the **precondition** should be applicable to the class that references the **configuration_control_constraint**.

NOTE 3 In a **configuration_control_constraint**, **filter**s are used both to represent precondition on the referencing instance and to express constraints on the referenced instances.

EXPRESS specification:

```
*)
ENTITY configuration_control_constraint
SUBTYPE OF (class_constraint);
     precondition: SET [0:?] OF filter;
     postcondition: SET [1:?] OF filter;
END_ENTITY; -- configuration_control_constraint
(*
```

Attribute definitions:

precondition: the **filter**s that shall hold on the referencing instance for the restriction to apply.

NOTE 4 If the set of filters is empty, the restriction applies on any referencing instance.

postcondition: the **filter**s that shall hold on a referenced instance for being allowed for reference.

7.3.6 Filter

The **filter** entity is an **enumeration_constraint** that restricts the allowed domain of a property whose data type is either **non_quantitative_code_type** or **non_quantitative_int_type**.

EXPRESS specification:

```
*)
ENTITY filter;
     referenced_property: property_BSU;
     domain: enumeration_constraint;
WHERE
     WR1: definition_available_implies (
          referenced_property,
          (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
          +'.NON_QUANTITATIVE_CODE_TYPE') IN TYPEOF(
          referenced_property.
          definition[1]\property_DET.domain))
          OR
          (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
          +'.NON_QUANTITATIVE_INT_TYPE') IN TYPEOF(
          referenced_property.
          definition[1]\property_DET.domain)));
     WR2: definition_available_implies (
          referenced_property,
          correct_constraint_type(domain,
          referenced_property.definition[1].domain));
END_ENTITY; -- filter
(*
```

Attribute definitions:

referenced_property: the property whose domain of values is restricted by the **filter**.

domain: the **enumeration_constraint** that restricts the domain of values of the referenced property.

Formal propositions:

WR1: the data type of the **referenced_property** shall be either **non_quantitative_code_type** or **non_quantitative_int_type**.

WR2: the **domain** shall define a domain of values that may restrict the initial domain of values of the property.

7.3.7 Integrity_constraint

The **integrity_constraint** entity is a particular property constraint that 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.

EXAMPLE In the reference dictionary defined for fasteners in ISO 13584-511, a *metric threaded bolt/screw* has a *head properties* property that may take, 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.

NOTE In the example above, the integrity constraint does not change at all the meaning of the *head properties* property inherited from *metric threaded bolt/screw* into *hexagon head screw*. It just makes explicit the fact that in the context of the *hexagon head screw* subclass, only a subset of the values allowed for this property in the context of the *metric threaded bolt/screw* class remains allowed.

EXPRESS specification:

```
*)
ENTITY integrity_constraint
SUBTYPE OF (property_constraint);
     redefined_domain: domain_constraint;
WHERE
     WR1: definition_available_implies (constrained_property,
          correct_constraint_type(redefined_domain,
          constrained_property.definition[1].domain));
END_ENTITY; -- integrity_constraint
(*
```

Attribute definitions:

redefined_domain: the constraint that applies on the domain of values of the constrained property.

Formal propositions:

WR1: the **redefined_domain** shall define a domain of values that restricts the initial domain of values of the property.

7.3.8 Context_restriction_constraint

The **context_restriction_constraint** entity is a **property_constraint** that restricts the allowed domain of values for the context parameters on which a context dependent property depends.

EXPRESS specification:

```
*)
ENTITY context_restriction_constraint
SUBTYPE OF (property_constraint);
     context_parameter_constraints: SET [1:?] OF property_constraint;
WHERE
     WR1: definition_available_implies(constrained_property,
          QUERY (cp <*SELF.context_parameter_constraints
          | NOT (cp.constrained_property IN
          constrained_property.definition[1].depends_on))=[]);
     WR2: QUERY (cp <*SELF.context_parameter_constraints
          | NOT (('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
          +'.INTEGRITY_CONSTRAINT') IN TYPEOF (cp))) =[];
     WR3:definition_available_implies(constrained_property,
          'ISO13584_IEC61360_DICTIONARY_SCHEMA.DEPENDENT_P_DET'
          IN TYPEOF(constrained_property.definition[1]));
END_ENTITY; -- context_restriction_constraint
(*
```

Attribute definitions:

context_parameter_constraints: the constraint that applies on the domain of values of the context parameters.

Formal propositions:

WR1: the set of properties whose domain is constrained by the **context_parameter_constraints** property shall be context parameters on which the constrained property depends.

WR2: all the c**ontext_parameter_constraints** shall be **integrity_constraint**s.

WR3: the **constrained_property** shall be a context dependent **property dependent_P_DET**.

7.3.9 Domain_constraint

A **domain_constraint** defines a constraint that restricts the domain of values of a data type.

EXPRESS specification:

```
*)
ENTITY domain_constraint
ABSTRACT SUPERTYPE OF(ONEOF(
     subclass_constraint,
     entity_subtype_constraint,
     enumeration_constraint,
     range_constraint,
     string_size_constraint,
     string_pattern_constraint,
     cardinality_constraint
     ));
END_ENTITY; -- domain_constraint
(*
```

7.3.10 Subclass_constraint

A **subclass_constraint** restricts the domain of values of a **class_reference_type** to one or several subclasses of the class that defines its initial domain.

EXPRESS specification:

```
    *)
    ENTITY subclass_constraint
    SUBTYPE OF(domain_constraint);
        subclasses: SET [1:?] OF class_BSU;
    END_ENTITY; -- subclass_constraint
    (*
```

Attribute definitions:

subclasses: the **class_BSUs** which redefine the class to which the value of the **constrained_property** shall belong.

7.3.11 Entity_subtype_constraint

An **entity_subtype_constraint** restricts the domain of values of an **entity_instance_type** to a subtype of the ENTITY that defines its initial domain.

EXPRESS specification:

```
    *)
    ENTITY entity_subtype_constraint
    SUBTYPE OF(domain_constraint);
        subtype_names: SET[1:?] OF STRING;
    END_ENTITY; -- entity_subtype_constraint
    (*
```

Attribute definitions:

subtype_names: the set of strings that describe, in the format of the EXPRESS TYPEOF function, the EXPRESS entity data type names that shall belong to the result of the EXPRESS TYPEOF function when it is applied to a value that references the **constrained_property** redefined property.

7.3.12 Enumeration_constraint

An **enumeration_constraint** restricts the domain of values of a data type to a list of values 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 subset by means of a **non_quantitative_int_type**, of which the i-the value describes the meaning of the i-the value of the subset.

For those subtypes of **number_type** that are associated with a **dic_unit** and alternative units, and possibly with a **dic_unit_identifier** and alternative unit identifiers, the constraint applies to the value corresponding to the **dic_unit**, or to the single **dic_unit_identifier**. If both exist, they correspond to the same unit.

For those subtypes of **number type** that are associated with a currency, the constraint applies to the currency specified in their data type definition. If no currency is specified in the data type definition, the constraint shall not be used.

For those values that belong to **translatable_string_type**s, 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 **source_language** attribute of the **administrative_data** of the property. If this attribute does not exist, this source language is supposed to be known by the dictionary user.

If another **enumeration_constraint** is applied on 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.

EXAMPLE 1 If, in class C1, this property is associated with an **enumeration_constraint** whose **subset** attribute equals {1, 3, 5, 7}, then in class C1, and any of its subclasses, property P1 may only takes one of the four following values: 1 or 3 or 5 or 7.

EXAMPLE 2 If the data type of property P1 is LIST [1:4] OF INTEGER, and if in class C1 this property is associated with an **enumeration_constraint** whose **subset** attribute equals { {1}, {3, 5}, {7}, {1, 3, 7} } then, in class C1 and any of its subclasses, property P1 may only takes one of the four following values: {1} or {3, 5} or {7} or {1, 3, 7}.

<u>EXPRESS specification:</u>

```
*)
ENTITY enumeration_constraint
SUBTYPE OF (domain_constraint);
      subset: LIST [1:?] OF UNIQUE primitive_value;
      value_meaning: OPTIONAL non_quantitative_int_type;
WHERE
      WR1: (NOT(EXISTS(SELF.value_meaning)))
           OR
           (integer_values_in_range(1, SIZEOF(SELF.subset))
                = allowed_values_integer_types(SELF.value_meaning));
END_ENTITY; -- enumeration_constraint
(*
```

<u>Attribute definitions</u>:

subset: the list describing the subset of values that are allowed as possible values for the property identified by **constrained_property**.

value_meaning: the set of **dic_value**s that define the meaning of each value belonging to the **subset**.

<u>Formal propositions</u>:

WR1: if the **value_meaning non_quantitative_int_type** exists, then the set of **value_code**s of its **dic_value**s shall be in 1.. SIZE_OF(subset).

7.3.13 Range_constraint

A **range_constraint** entity restricts the domain of values of an ordered type to a subset of values defined by a range.

NOTE 1 Strings are not considered as ordered types and cannot be constrained by a **range_constraint**.

For those subtypes of **number_type** that are associated with a **dic_unit** and alternative units, and possibly with a **dic_unit_identifier** and alternative unit identifiers, the constraint applies to the value corresponding to the **dic_unit**, or to the single **dic_unit_identifier**. If both exists, they correspond to the same unit.

For those subtypes of **number_type** that are associated with a currency, the constraint applies to the currency specified in their data type definition. If no currency is specified in the data type definition, the constraint shall not be used.

NOTE 2 For **non_quantitative_int_type** the constraint applies to the **value_code**.

EXPRESS specification:

```
*)
ENTITY range_constraint
SUBTYPE OF (domain_constraint);
      min_value, max_value: OPTIONAL NUMBER;
      min_inclusive, max_inclusive: OPTIONAL BOOLEAN;
WHERE
      WR1: min_value <= max_value;
      WR2: TYPEOF(min_value) = TYPEOF(max_value);
      WR3: NOT EXISTS (min_value) OR EXISTS (min_inclusive);
      WR4: NOT EXISTS (max_value) OR EXISTS (max_inclusive);
END_ENTITY; -- range_constraint
(*
```

Attribute definitions:

min_value: the number defining the low bound of the range of values; not existing value means no lower bound.

max_value: the number defining the high bound of the range of values; not existing value means no upper bound.

min_inclusive: specifies whether **min_value** belongs to the range; not existing value means that there is no low bound.

max_inclusive: specifies whether **max_value** belongs to the range; not existing value means hat there is no high bound.

Formal propositions:

WR1: **min_value** shall be less than or equal to **max_value**.

WR2: **min_value** and **max_value** shall have the same data types.

WR3: if **min_value** has a value, then **min_inclusive** shall also have a value.

WR4: if **max_value** has a value, then **max_inclusive** shall also have a value.

7.3.14 String_size_constraint

A **string_size_constraint** restricts the length of the STRING values allowed for a STRING type, or any of its subtypes.

NOTE 1 A string_type property value domain is either a string_type, a non_translatable_string_type, a translatable_string_type, a URI_type, a non_quantitative_code_type, a date_data_type, a time_data_type or a date_time_data_type.

NOTE 2 For **non_quantitative_code_type** the constraint applies to the code.

For those values that belong to **translatable_string_type**s, 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 **source_language** attribute of the **administrative_data** of the property. If this attribute does not exist, this source language is supposed to be known by the dictionary user.

EXPRESS specification:

```
*)
ENTITY string_size_constraint
SUBTYPE OF (domain_constraint);
     min_length: OPTIONAL INTEGER;
     max_length: OPTIONAL INTEGER;
WHERE
     WR1: (min_length >= 0) AND (max_length >= min_length);
END_ENTITY; -- string_size_constraint
(*
```

Attribute definitions:

min_length: the minimal length for the strings that are allowed as values for the property identified by the **constrained_property** property.

max_length: the maximal length for the strings that are allowed as values for the property identified by the **constrained_property** property.

NOTE 3 If the **min_length** value does not exist, 0 is understood. If the **max_length** value does not exist, unbounded is understood.

Formal propositions:

WR1: **min_length** and **max_length** define correct bounds.

7.3.15 String_pattern_constraint

A **string_pattern_constraint** restricts the domain of values of a **string_type**, or of any of its subtypes, to string values that match a particular pattern. The **pattern** syntax is defined by an XML regular expression and the associated matching algorithms that are defined by the XML Schema Part 2: Datatypes recommendation.

NOTE 1 A **string_type** property value domain is either a **string_type**, a **non_translatable_string_type**, a **translatable_string_type**, an **URI_type**, a **non_quantitative_code_type**, **a date_data_type**, a **time_data_type** or a **date_time_data_type**.

For string_type, non_translatable_string_type, URI_type, non_quantitative_code_type, date_data_type, time_data_type or date_time_data_type the constraint applies to the (unique) string that is the value of the data type. For non_quantitative_code_type the constraint applies to the code.

For those values that belong to translatable_string_types, 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 source_language attribute of the administrative_data of the property, if this attribute does not exist, this source language is supposed to be known by the dictionary user.

NOTE 2 For **non_quantitative_code_type**, **date_data_type**, **time_data_type** or **date_time_data_type**, the **pattern** should comply with the informal propositions defined in the corresponding data types.

EXPRESS specification:

```
*)
ENTITY string_pattern_constraint
SUBTYPE OF (domain_constraint);
     pattern: STRING;
END_ENTITY; -- string_pattern_constraint
(*
```

Attribute definition:

pattern: the pattern of string values that are allowed as values for the property identified by the constrained property.

Informal proposition:

IP1: the **pattern** syntax shall complies with the XML regular expression syntax 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 “2009-05-31”.

7.3.16 Cardinality_constraint

A **cardinality_constraint** restricts the cardinality of an aggregate 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_constraint**s are not allowed on **array_type**.

EXPRESS specification:

```
*)
ENTITY cardinality_constraint
SUBTYPE OF (domain_constraint);
     bound_1: OPTIONAL INTEGER;
     bound_2: OPTIONAL INTEGER;
WHERE
     WR1: (bound_1 >= 0) AND (bound_2 >= bound_1);
END_ENTITY; -- cardinality_constraint
(*
```

Attribute definitions:

bound_1: the lower bound of the cardinality.

bound_2: the upper bound of the cardinality.

NOTE 3 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.

Formal propositions:

WR1: **bound_1** and **bound_2** define correct bounds.

7.4 ISO13584_IEC61360_class_constraint_schema type definitions

7.4.1 General

This subclause defines the type in the ISO13584_IEC61360_class_constraint_schema.

7.4.2 Constraint_or_constraint_id

The constraint_or_constraint_id is either a constraint or a constraint_identifier.

EXPRESS specification:

```
*)
TYPE constraint_or_constraint_id =
      SELECT (constraint, constraint_identifier);
END_TYPE; -- constraint_or_constraint_id
(*
```

7.5 ISO13584_IEC61360_class_constraint_schema function definition

7.5.1 General

This subclause defines the functions in the **ISO13584_IEC61360_class_constraint_schema**.

7.5.2 Integer_values_in_range function

The **integer_values_in_range** function computes the integer values that belong to an integer range defined by its low bound and its high bound. It returns indeterminate (?) when either bounds are indeterminate.

EXPRESS specification:

```
*)
FUNCTION integer_values_in_range(
      low_bound, high_bound: INTEGER): SET OF INTEGER;
LOCAL
      i: INTEGER;
      result: SET OF INTEGER:= [];
END_LOCAL;
      IF EXISTS (low_bound) AND EXISTS (high_bound)
      THEN
           REPEAT i := low_bound TO high_bound;
                result := result + [i];
           END_REPEAT;
           RETURN(result);
      ELSE
           RETURN(?);
      END_IF;
END_FUNCTION; -- integer_values_in_range
(*
```

7.5.3 Correct_precondition function

The **correct_precondition** function checks that the precondition of the **configuration_control_constraint** defined by **cons** uses only properties that are applicable to the **cl** class. It returns a logical that is UNKNOWN when the complete set of applicable properties of the class cannot be computed.

<u>EXPRESS specification:</u>

```
*)
FUNCTION correct_precondition(
     cons: configuration_control_constraint; cl:class): LOGICAL;
LOCAL
     prop: SET OF property_BSU:= [];
END_LOCAL;
     REPEAT i := 1 to SIZEOF (cons.precondition);
          prop := prop + cons.precondition[i].referenced_property;
     END_REPEAT;

     IF prop <= cl.known_applicable_properties
     THEN RETURN (TRUE);
     ELSE
          IF all_class_descriptions_reachable(cl.identified_by)
          THEN RETURN (FALSE);
          ELSE RETURN (UNKNOWN);
          END_IF;
     END_IF;
END_FUNCTION; -- correct_precondition
(*
```

7.5.4 Correct_constraint_type function

The **correct_constraint_type** function checks that the **domain_constraint** defined by **cons** is compatible with the **data_type** defined by **typ**. It returns a logical that is UNKNOWN when the **domain_constraint** defined by **cons** is not one of the subtypes defined in the **ISO13584_IEC61360_class_constraint_schema**.

<u>EXPRESS specification:</u>

```
*)
FUNCTION correct_constraint_type(
     cons: domain_constraint; typ:data_type): LOGICAL;

(*case subclass constraint*)

IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
     +'SUBCLASS_CONSTRAINT') IN TYPEOF(cons)
THEN
     (*the data type shall be class_reference_type*)
     IF NOT
          ('ISO13584_IEC61360_DICTIONARY_SCHEMA.CLASS_REFERENCE_TYPE'
          IN TYPEOF (typ))
     THEN RETURN(FALSE);
     END_IF;

     (*the cons.subclasses shall consist of subclasses for the class
      that defined the initial domain of typ.*)
      IF NOT (QUERY (sc <* cons.subclasses |
               definition_available_implies
           (sc,definition_available_implies
           (typ\class_reference_type.domain,is_subclass(sc.definition[1]
           , typ\class_reference_type.domain.definition[1])))= false)
           = [])
      THEN RETURN(FALSE);
      END_IF;

      RETURN (TRUE);
END_IF;
```

(*case entity subtype constraint*)

```
IF (('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
      +'ENTITY_SUBTYPE_CONSTRAINT') IN TYPEOF (CONS))
THEN

(* the data type is a class_reference_type*)
      IF NOT (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
      +'.ENTITY_INSTANCE_TYPE') IN TYPEOF (typ))
      THEN RETURN(FALSE);
      END_IF;
(* the subtype_name shall define a subtype for the entity_instance_type
of the constrained *)
      IF NOT (cons\entity_subtype_constraint.subtype_names
           >= typ\entity_instance_type.type_name)
      THEN RETURN(FALSE);
      END_IF;

RETURN (TRUE);
END_IF;
```

(*case enumeration_constraint *)

```
IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
      +'.ENUMERATION_CONSTRAINT') IN TYPEOF (CONS)
THEN

(* all the values belonging to the subset of values shall be compatible
with the typ data type *)
      IF (QUERY (val<*cons.subset |
           NOT compatible_data_type_and_value ( typ, val))<> [])
      THEN RETURN(FALSE);
      END_IF;

RETURN (TRUE);
END_IF;
```

(*case range_constraint *)

```
IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA.RANGE_CONSTRAINT'
       IN TYPEOF (CONS))
```

```
THEN

(*if the data type is an integer_type then min_value and max_value
shall be INTEGERs.*)
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.INTEGER_TYPE'
          IN TYPEOF (typ)) AND
          NOT ('INTEGER' IN TYPEOF (cons.min_value))
     THEN RETURN(FALSE);
     END_IF;

(*if the data type is a rational_type then min_value and max_value
shall be rational.*)
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.RATIONAL_TYPE'
          IN TYPEOF (typ)) AND
          NOT   ('ISO13584_INSTANCE_RESOURCE_SCHEMA.RATIONAL_VALUE'   IN
TYPEOF (cons.min_value))
     THEN RETURN(FALSE);
     END_IF;

(*if the data type is a real_type then min_value and max_value shall be
REALs.*)
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_TYPE'
          IN TYPEOF (typ)) AND NOT ('REAL' IN TYPEOF (cons.min_value))
     THEN RETURN(FALSE);
     END_IF;

(*all values of the range shall belong to the allowed values defined by
the type.*)
     IF (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
          + '.NON_QUANTITATIVE_INT_TYPE') IN TYPEOF (typ))
     AND NOT
          (integer_values_in_range(cons.min_value, cons.max_value)
          <= allowed_values_integer_types (typ))
     THEN RETURN(FALSE);
     END_IF;

RETURN (TRUE);
END_IF;
```

(*case entity string_size_constraint*)

```
IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
     +'.STRING_SIZE_CONSTRAINT') IN TYPEOF (CONS)
THEN

(* the data type shall be a string_type or any of its subtypes *)
IF NOT    ('ISO13584_IEC61360_DICTIONARY_SCHEMA.STRING_TYPE'
          IN TYPEOF (typ))
THEN RETURN(FALSE);
END_IF;

RETURN (TRUE);
END_IF;
```

(*case entity string_pattern_constraint *)

```
IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
      +'.STRING_PATTERN_CONSTRAINT') IN TYPEOF (CONS)
THEN

(* the data type shall be a string_type or any of its subtypes *)
      IF NOT ('ISO13584_IEC61360_DICTIONARY_SCHEMA.STRING_TYPE'
           IN TYPEOF (typ))
      THEN RETURN(FALSE);
      END_IF;
RETURN (TRUE);
END_IF;

(*case entity cardinality_constraint *)

IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
      +'.CARDINALITY_CONSTRAINT') IN TYPEOF (CONS)
THEN

(* the data type shall be an aggregate type but not an array*)
      IF (NOT(
      ('ISO13584_IEC61360_DICTIONARY_AGGREGATE_EXTENSION_SCHEMA'
      + '.ENTITY_INSTANCE_TYPE_FOR_AGGREGATE')
       IN TYPEOF(typ)))
      THEN
           RETURN(FALSE);
      END_IF;

      IF ('ISO13584_IEC61360_DICTIONARY_AGGREGATE_EXTENSION_SCHEMA'
           + '.ARRAY_TYPE' IN TYPEOF(typ.type_structure))
      THEN
           RETURN(FALSE);
      END_IF;
      RETURN (TRUE);
END_IF;

RETURN (UNKNOWN);

END_FUNCTION; -- correct_constraint_type
(*
```

7.5.5 Compatible_data_type_and_value function

The function **compatible_data_type_and_value** checks if a value **val** of a **primitive_value** is type compatible with the type defined by a type **dom**. It returns a LOGICAL that is TRUE when they are compatible and FALSE when they are not. This function returns UNKNOWN if the **val** data type is an **uncontrolled_instance_value** (see ISO 13584-24:2003) or when its type is not one of the types defined in the **ISO13584_instance_resource_schema**.

NOTE The value **val** may or may not exist.

EXPRESS specification:

```
*)
FUNCTION compatible_data_type_and_value(dom: data_type;
      val: primitive_value): LOGICAL;
```

```
LOCAL
     temp: class_BSU;
     set_string: SET OF STRING := [];
     set_integer: SET OF INTEGER := [];
     code_type: non_quantitative_code_type;
     int_type: non_quantitative_int_type;
END_LOCAL;

(* The following express statements deal with simple types *)

IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA.INTEGER_VALUE' IN TYPEOF(val))
THEN
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.' +
          'NON_QUANTITATIVE_INT_TYPE' IN TYPEOF (dom))
     THEN
          set_integer := [];
          int_type := dom;
          REPEAT j := 1 TO SIZEOF(int_type.domain.its_values);
               set_integer := set_integer +
                    int_type.domain.its_values[j].value_code;
          END_REPEAT;

          RETURN(val IN set_integer);

     ELSE
          RETURN(('ISO13584_IEC61360_DICTIONARY_SCHEMA.INT_TYPE'
               IN TYPEOF (dom)) OR
               (('ISO13584_IEC61360_DICTIONARY_SCHEMA.NUMBER_TYPE'
               IN TYPEOF (dom))
               AND NOT(('ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_TYPE'
               IN TYPEOF (dom))
               OR ('ISO13584_IEC61360_DICTIONARY_SCHEMA.RATIONAL_TYPE'
               IN TYPEOF (dom)))));


     END_IF;
END_IF;

IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA.REAL_VALUE' IN TYPEOF(val))
THEN
     RETURN(('ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_TYPE'
          IN TYPEOF (dom)) OR
          (('ISO13584_IEC61360_DICTIONARY_SCHEMA.NUMBER_TYPE'
          IN TYPEOF (dom))
          AND NOT(('ISO13584_IEC61360_DICTIONARY_SCHEMA.INT_TYPE'
          IN TYPEOF (dom))
          OR ('ISO13584_IEC61360_DICTIONARY_SCHEMA.RATIONAL_TYPE'
               IN TYPEOF (dom)))));
END_IF;

IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA.RATIONAL_VALUE' IN TYPEOF(val))
THEN
     RETURN(('ISO13584_IEC61360_DICTIONARY_SCHEMA.RATIONAL _TYPE'
          IN TYPEOF (dom)) OR
          (('ISO13584_IEC61360_DICTIONARY_SCHEMA.NUMBER_TYPE'
          IN TYPEOF (dom))
          AND NOT(('ISO13584_IEC61360_DICTIONARY_SCHEMA.INT_TYPE'
          IN TYPEOF (dom))
          OR ('ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_TYPE'
               IN TYPEOF (dom)))));
END_IF;

IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA.STRING_VALUE'
      IN TYPEOF(val))
THEN
      IF (('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
           '.NON_QUANTITATIVE_CODE_TYPE') IN TYPEOF (dom))
      THEN
           set_string := [];
           code_type := dom;
           REPEAT j := 1 TO SIZEOF(code_type.domain.its_values);
                set_string := set_string +
                     code_type.domain.its_values[j].value_code;
           END_REPEAT;

                RETURN(val IN set_string);

      ELSE
           RETURN('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
                '.STRING_TYPE' IN TYPEOF (dom));
      END_IF;
END_IF;

IF ('ISO13584_INSTANCE_RESOURCE_SCHEMA.TRANSLATED_STRING_VALUE'
      IN TYPEOF(val))
THEN
           RETURN('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
                '.TRANSLATABLE_STRING_TYPE' IN TYPEOF (dom));
END_IF;

(* The following express statements deal with complex types *)

IF 'ISO13584_INSTANCE_RESOURCE_SCHEMA.DIC_CLASS_INSTANCE'
      IN TYPEOF(val)
THEN
      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.CLASS_REFERENCE_TYPE'
           IN TYPEOF (dom))
      THEN
           temp := dom.domain;
           RETURN(compatible_class_and_class(temp,
                val\dic_class_instance.class_def));
      ELSE
           RETURN(FALSE);
      END_IF;
END_IF;

IF 'ISO13584_INSTANCE_RESOURCE_SCHEMA.LEVEL_SPEC_VALUE' IN TYPEOF(val)
THEN
```

```
      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.LEVEL_TYPE'
           IN TYPEOF (dom))
      THEN
           RETURN(compatible_level_type_and_instance(
                dom.levels,
                TYPEOF(dom.value_type),
                val));
      ELSE
           RETURN(FALSE);
      END_IF;
END_IF;


(* The following express statements deal with aggregate types *)

IF 'ISO13584_AGGREGATE_VALUE_SCHEMA.AGGREGATE_ENTITY_INSTANCE_VALUE' IN
TYPEOF(val) THEN

      IF (NOT(
      'ISO13584_IEC61360_DICTIONARY_SCHEMA.ENTITY_INSTANCE_TYPE'
       IN TYPEOF(dom)))
      THEN
           RETURN(FALSE);
      END_IF;

      IF (NOT(
           'ISO13584_IEC61360_DICTIONARY_AGGREGATE_EXTENSION_SCHEMA'
       + '.AGGREGATE_TYPE' IN dom.type_name))
      THEN
           RETURN(FALSE);
      END_IF;

      RETURN(compatible_aggregate_type_and_value(dom, val));

END_IF;

IF 'ISO13584_INSTANCE_RESOURCE_SCHEMA.ENTITY_INSTANCE_VALUE'
      IN TYPEOF(val)
THEN
      IF 'ISO13584_INSTANCE_RESOURCE_SCHEMA' +
           '.UNCONTROLLED_ENTITY_INSTANCE_VALUE'
           IN TYPEOF(val)
      THEN
           RETURN(UNKNOWN);
      END_IF;
      IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.ENTITY_INSTANCE_TYPE'
           IN TYPEOF (dom))
           AND (dom.type_name <= TYPEOF(val))
      THEN
           RETURN(TRUE);
      ELSE
           RETURN(FALSE);
      END_IF;
END_IF;
```

```
RETURN(UNKNOWN);

END_FUNCTION; -- compatible_data_type_and_value
(*
```

7.6 ISO13584_IEC61360_class_constraint_schema rule definition

7.6.1 General

This subclause defines the rule in the ISO13584_IEC61360_class_constraint_schema.

7.6.2 Unique_constraint_id

The **unique_constraint_id** rule asserts that two **constraint_identifier**s associated with two different **constraint**s have different values.

EXPRESS specification:

```
*)
RULE unique_constraint_id FOR(constraint);
WHERE
      QUERY(c1 <* constraint |
           SIZEOF(QUERY(c2 <* constraint |
           c1.constraint_id = c2.constraint_id))>1) = [];
END_RULE; -- unique_constraint_id
(*

*)
END_SCHEMA; -- ISO13584_IEC61360_class_constraint_schema

(*
```

8 ISO13584_IEC61360_item_class_case_of_schema

8.1 Overview

This clause defines the requirement for the **ISO13584_IEC61360_item_class_case_of_schema**. The following EXPRESS declaration introduces the **ISO13584_IEC61360_item_class_case_of_schema** block and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA ISO13584_IEC61360_item_class_case_of_schema;

REFERENCE FROM ISO13584_IEC61360_dictionary_schema
      (all_class_descriptions_reachable,
      class,
      class_BSU,
      data_type_BSU,
      item_class,
      property_BSU);

REFERENCE FROM ISO13584_IEC61360_class_constraint_schema
```

```
        (constraint,
        integrity_constraint,
        context_restriction_constraint,
        property_constraint,
        domain_constraint);

    REFERENCE FROM ISO13584_extended_dictionary_schema
        (document_BSU,
        table_BSU,
        visible_properties,
        applicable_properties,
        visible_types,
        applicable_types,
        data_type_named_type);

    (*
```

NOTE The schemata referenced above can be found in the following documents:

ISO13584_IEC61360_dictionary_schema	IEC 61360-2
(which is duplicated for convenience in this document).	
ISO13584_IEC61360_class_constraint_schema	IEC 61360-2
(which is duplicated for convenience in this document).	
ISO13584_extended_dictionary_schema	ISO 13584-24:2003

8.2 Introduction to the ISO13584_IEC61360_item_class_case_of_schema

For modularity reasons, the complete common ISO/IEC dictionary model is split among several documents. The kernel model for product ontologies is defined in IEC 61360-2 and duplicated in this part of IEC 61360. Resources for extending this model, including instance representation, document representation, functional model, functional views and table representation are defined in ISO 13584-24:2003. The various standard levels of implementation of the complete ISO/IEC model, called conformance classes, are defined in ISO 13584-25, and duplicated for informative purpose in IEC 61360-5. The first level corresponds precisely to the content of this part of IEC 61360 more the resource for aggregate-structured values, defined in ISO 13584-25. Other conformance classes include more and more resources from ISO 13584-24:2003.

To define an item class as case-of another item class is more and more used by applications based on the common ISO13584/IEC61360 dictionary model. Moreover, this edition of this part of IEC 61360 has required a change of the information model of this concept. Therefore, it has been decided to move the corresponding EXPRESS entity, called **item_class_case_of**, and its superclass, called **a_priori_semantic_relationship**, from ISO 13584-24:2003 to this part of IEC 61360. These entities are included in a new schema, called **ISO13584_IEC61360_item_class_case_of_schema**. ISO 13584-24:2003 and ISO 13584-25 will be updated accordingly by means of a technical corrigendum.

8.3 ISO13584_IEC61360_item_class_case_of_schema entity definitions

8.3.1 A priori semantic relationship

An **a_priori_semantic_relationship** is an abstract **class** that is defined on the basis of other classes, and that can import properties, data types, tables and documents contained in these classes. It also imports all the **constraint**s that restricted the domain of the imported properties in the classes from which they are imported. This abstract resource is intended to be subtyped by classes. When a class specializes an **a_priori_semantic_relationship,** the properties, data types, tables or documents whose definitions are imported by inheritance of **a_priori_semantic_relationship** become applicable to the class that imports them. In particular, properties and data types that are so imported are allowed for use for describing class instances, and the fact, for a product, to have an aspect that corresponds to each imported property is a necessary criteria for being member of the class.

NOTE 1 All imported properties and data types become directly applicable without being visible. Thus, they are not returned by the **compute_known_visible_properties** or **compute_known_visible_data_types** function.

NOTE 2 The inheritance relationship is a well known example of semantic relationship between classes modelled according to the object oriented paradigm. All the properties and other resources defined in a class usually apply implicitly to all its subclasses. This relationship is used in ISO 13584 series where all the properties, data types, tables or documents visible (respectively applicable) to some classes are implicitly visible (respectively applicable) to all its subclasses. As usual, in ISO 13584 this inheritance is implicit (i.e., not declared by means of an **a_priori_semantic_relationship**) and global (i.e., all the properties and data types are inherited by all its subclasses). An **a_priori_semantic_relationship** enables to define other semantic relationships that are useful in the ISO 13584 application domain, and in particular the case-of relationship that allows an explicit and partial importation of properties, and of other resources defined in a class.

<u>EXPRESS specification:</u>

```
*)
ENTITY a_priori_semantic_relationship
ABSTRACT SUPERTYPE
SUBTYPE OF(class);
     referenced_classes: SET [1:?] OF class_BSU;
     referenced_properties: LIST [0:?] OF property_BSU;
     referenced_data_types: SET [0:?] OF data_type_BSU;
     referenced_tables: SET [0:?] OF table_BSU;
     referenced_documents: SET [0:?] OF document_BSU;
     referenced_constraints: SET [0:?] OF constraint_or_constraint_id;
WHERE
     WR1: QUERY (cons <* SELF.referenced_constraints
          | NOT(('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
          '.ISO_29002_IRDI_type') IN TYPEOF(cons))
          AND NOT (('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
          +'.PROPERTY_CONSTRAINT') IN TYPEOF (cons)))
          = [];
     WR2: QUERY (cons <* SELF.referenced_constraints
          | (('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
          +'.PROPERTY_CONSTRAINT') IN TYPEOF (cons))
          AND NOT (cons\property_constraint.constrained_property
          IN SELF.referenced_properties))
          = [];
     WR3: compute_known_referenced_property_constraints(SELF)
          <= SELF.referenced_constraints;
     WR4: QUERY(prop <* SELF.referenced_properties
          | QUERY(cl <* SELF.referenced_classes
          | visible_properties(cl, [prop])
          OR applicable_properties(cl, [prop]))
          = []) = [];
     WR5: QUERY(typ <* SELF.referenced_data_types
          | QUERY(cl <* SELF.referenced_classes
          | visible_types(cl, [typ])
          OR applicable_types(cl, [typ]))
          = []) = [];
END_ENTITY; -- a_priori_semantic_relationship
(*
```

Attribute definitions:

referenced_classes: the class(es) from where the properties, data types, tables or documents are imported.

NOTE 3 The class from which properties, data types, tables or documents are imported cannot be deduced from the identification of the imported properties, data types, tables or documents because they may be imported from a class where they are inherited or imported. For instance, in IEC 61360-DB, "input-voltage" is a property visible at the root level of the IEC classification. If a supplier class imports the "input-voltage" property from the IEC "transistor" class, this means that (1) the supplier class defines a transistor, and (2) these transistors are described by means of an "input voltage" property.

referenced_properties: the properties whose definitions are imported through the **a_priori_semantic_relationship** entity.

NOTE 4 The list order defines the default order for displaying imported properties during user access to the various subtypes of **a_priori_semantic_relationship**.

referenced_data_types: the data types whose definitions are imported through the **a_priori_semantic_relationship** entity.

referenced_tables: the tables whose definitions are imported through the **a_priori_semantic_relationship** entity.

NOTE 5 Detailed resources and rules on the use of tables are defined in ISO 13584-24:2003. They are not used in this part of IEC 61360, nor in the integrated models documented in ISO 13584-32 (OntoML) and ISO 13584-25.

referenced_documents: the documents whose definitions are imported through the **a_priori_semantic_relationship** entity.

NOTE 6 Detailed resources and rules on the use of documents are defined in ISO 13584-24:2003. They are used in the integrated models documented in ISO 13584-32 (OntoML) or ISO 13584-25.

referenced_constraints: the **property_constraints** that apply to the various imported properties.

NOTE 7 Unlike other referenced entities, the referenced_constraints constraints cannot be selected when the **a_priori_semantic_relationship** is designed. These constraints are all the constraints that affect any of the properties defined in the **referenced_properties** attribute in any class of the **referenced_classes** attribute.

Formal propositions:

WR1: all the **referenced_constraints** that are not IRDI shall be **property_constraint**s.

WR2: all the **referenced_constraints** shall constrain properties that are imported through the **referenced_properties** attribute.

WR3: all the **property_constraint**s that constrain one of the **referenced_properties** property in any of the **referenced_classes** class shall be imported through the **referenced_constraint**s attribute.

WR4: the imported properties defined by the **referenced_properties** attribute shall be visible or applicable for one of the classes belonging to the **referenced_classes** attribute.

WR5: the imported types defined by the **referenced_data_types** attribute shall be visible or applicable for one of the classes belonging to the **referenced_classes** attribute.

Informal propositions:

IP1: all the **constraint**s that are represented by **constraint_identifier**s in the **referenced_constraints** set shall correspond to **property_constraint**s that constrain one of the **referenced_properties** properties in one of the **referenced_classes** class. Such constraint shall not be represented, in the same **referenced_constraints** set, both as a **property_constraint** and as a **constraint_identifier**.

NOTE 8 A constraint represented as a **property_constraint** in one of the **referenced_classes** class may be represented in the **referenced_constraints** set either as a **property_constraint** or as a **constraint_identifier**

IP2: all the constraints that are represented by a **constraint_identifier** in one of the **referenced_classes** classes but whose corresponding **constraint** is a **property_constraint** that constrains one of the properties imported through the **referenced_properties** attribute shall be represented by their **constraint_identifier** in the **referenced_constraints** set.

NOTE 9 These two informal rules ensure that the **referenced_constraints** set of constraints is the union of the sets of **property_constraint**s defined in the various **referenced_classes** classes whose **constrained_property** belongs to the **referenced_properties** set, even when the exchange context does not contain the definitions of all the classes involved in the **a_priori_semantic_relationship** and when some **constraint**s are only represented by their **constraint_identifier**s.

8.3.2 Item_class_case_of

An **item_class_case_of** is the description of an item class that is defined as a is-case-of of some other item class(es).

NOTE 1 An **item_class_case_of** defines an a priori semantic relationship.

EXPRESS specification:

```
*)
ENTITY item_class_case_of
SUBTYPE OF(item_class, a_priori_semantic_relationship);
     is_case_of: SET [1:?] OF class_BSU;
     imported_properties: LIST [0:?] OF property_BSU;
     imported_types: SET [0:?]OF data_type_BSU;
     imported_tables: SET [0:?] OF table_BSU;
     imported_documents: SET [0:?] OF document_BSU;
     imported_constraints: SET [0:?] OF constraint_or_constraint_id;
DERIVE
     SELF\a_priori_semantic_relationship.referenced_classes:
          SET [1:?] OF class_BSU := SELF.is_case_of;
     SELF\a_priori_semantic_relationship.referenced_properties:
          LIST [0:?] OF property_BSU := SELF.imported_properties;
     SELF\a_priori_semantic_relationship.referenced_data_types:
          SET [0:?] OF data_type_BSU := SELF.imported_types;
     SELF\a_priori_semantic_relationship.referenced_tables:
          SET [0:?] OF table_BSU := SELF.imported_tables;
     SELF\a_priori_semantic_relationship.referenced_documents:
          SET [0:?] OF document_BSU := SELF.imported_documents;
     SELF\a_priori_semantic_relationship.referenced_constraints:
          SET [0:?] OF property_constraint
          := SELF.imported_constraints;
WHERE
     WR1: superclass_of_item_is_item(SELF);
     WR2: check_is_case_of_referenced_classes_definition(SELF);
     WR3: QUERY(p <* SELF\class.sub_class_properties
          | NOT((p IN SELF.described_by)
          OR (p IN SELF.imported_properties))) = [];
     WR4: QUERY(p <* SELF\class.sub_class_properties
          | (p IN SELF.imported_properties)
          AND (QUERY(cl<*SELF.is_case_of
          | all_class_descriptions_reachable(cl) AND
          (p IN compute_known_applicable_properties(cl)) AND
          (NOT is_class_valued_property(p, cl)))<>[]))
          =[];
     WR5: QUERY(ccv <* SELF\class.class_constant_values
          | (ccv.super_class_defined_property
          IN SELF.imported_properties)
          AND (QUERY(cl<*SELF.is_case_of
          | all_class_descriptions_reachable(cl) AND
          (ccv.super_class_defined_property
           IN compute_known_applicable_properties(cl)) AND
          (QUERY (v<*class_value_assigned(
          ccv.super_class_defined_property, cl)
          |v<> ccv.assigned_value) <> []))<>[]))
          =[];
     WR6: QUERY(prop <* imported_properties
          | (QUERY(cl<*SELF.is_case_of
          | is_class_valued_property(prop, cl)) <>[])
          AND NOT is_class_valued_property(prop, SELF.identified_by))
          =[];
     WR7: QUERY(ccv <* SELF\class.class_constant_values
          | QUERY(cl<*SELF.is_case_of
          | (class_value_assigned
          (ccv.super_class_defined_property, cl) <> [])
          AND (QUERY(v <* class_value_assigned
          (ccv.super_class_defined_property, cl)
          | v <> ccv. assigned_value)<>[])) <> [])
          =[];
END_ENTITY; -- item_class_case_of
(*
```

<u>Attribute definitions</u>:

is_case_of: the **item_class**(es) of which the present **item_class** is-case-of.

imported_properties: the list of properties that are imported from the **item_class**(es) the defined **item_class** is-case-of.

imported_types: the set of data types that are imported from the **item_class**(es) the defined **item_class** is-case-of.

imported_tables: the set of **table_BSU**s that are imported from the **item_class**(es) the defined **item_class** is-case-of.

imported_documents: the set of **document_BSU**s that are imported from the **item_class**(es) the defined **item_class** is-case-of.

imported_constraints: the set of **property_constraint**s or **constraint_id** 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** in the classes of the **is_case_of** attribute from which they are imported. This is specified in a where rule of **a_priori_semantic_relationship**.

<u>Formal propositions</u>:

WR1: the superclass of an **item_class_case_of** shall be an **item_class**.

WR2: an **item_class_case_of** shall be case-of **item_class**(es).

WR3: the **sub_class_properties** shall belong either to the **described_by** list, or to the **imported_properties** list.

WR4: all the class valued properties declared by means of the **sub_class_properties** that are **imported_properties** shall be class valued properties in all the **is-case-of** classes where they are applicable.

WR5: the values assigned to an imported property by means of the **class_constant_value** assignment shall not be different than the possible value assigned to the same property in the referenced classes.

WR6: all the **imported_properties** that are class valued properties in a class from the **is_case_of** set of classes, shall be class valued properties in the current class.

WR7: all the **imported_properties** that are assigned a **class_constant_value** in a class from the **is_case_of** set of classes, shall be assigned the same **class_constant_value** class value in the current class.

8.4 ISO13584_IEC61360_item_class_case_of_schema function definitions

8.4.1 General

This subclause contains functions that are referenced in WHERE clauses to assert data consistency or that provide resources for application development.

8.4.2 Compute_known_property_constraints function

The **compute_known_property_constraints** function computes the set of **property_constraint**s that applies for the properties of a set of classes. Constraints represented by their identifiers are not returned. When the definition of some classes is not available, the function returns only the **property_constraint**s that may be computed.

NOTE When the **dictionary_definition** of a class is not available in the same exchange context (a PLIB exchange context is never assumed to be complete), its own superclass may not be known. Therefore the constraints defined by this superclass cannot be computed by the **compute_known_property_constraints** function. On the contrary, when all the is-a superclasses of a class are available in the same exchange context, all the constraints that apply to this class may be computed by a single traversal of its is-a inheritance tree, even when some of these superclasses import properties by means of **a_priori_semantic_relationship**s such as **item_class_case_of**.

<u>EXPRESS specification:</u>

```
*)
FUNCTION compute_known_property_constraints(classes: SET OF class_BSU):
      SET OF property_constraint;

LOCAL
      s: SET OF property_constraint := [];
END_LOCAL;

REPEAT nb := 1 TO SIZEOF (classes);
      IF SIZEOF(classes[nb].definition)=1
      THEN
           REPEAT i := 1 TO
                SIZEOF(classes[nb].definition[1]\class.constraints);
```

```
                    IF (('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
                    +'.PROPERTY_CONSTRAINT')
                    IN TYPEOF
                         (classes[nb].definition[1]\class.constraints[i]))
                    THEN
                    s := s + classes[nb].definition[1]\class.constraints[i];
                    END_IF;
               END_REPEAT;

               IF (('ISO13584_IEC61360_ITEM_CLASS_CASE_OF_SCHEMA.'
                    + 'A_PRIORI_SEMANTIC_RELATIONSHIP')
                    IN TYPEOF (classes[nb].definition[1]))
               THEN
                    REPEAT i := 1 TO
                     SIZEOF(classes[nb].definition[1]
                    \a_priori_semantic_relationship
                    .referenced_constraints);

                         IF (('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
                              +'.PROPERTY_CONSTRAINT') IN TYPEOF
                              (classes[nb].definition[1]
                              \a_priori_semantic_relationship
                              .referenced_constraints[i]))
                         THEN
                              s := s + classes[nb].definition[1]
                              \a_priori_semantic_relationship
                              .referenced_constraints [i];
                         END_IF;
                    END_REPEAT;
               END_IF;

               IF EXISTS(classes[nb].definition[1]\class.its_superclass)
               THEN
                    s := s + compute_known_property_constraints(
                         [classes[nb].definition[1]\class.its_superclass]);
               END_IF;

          END_IF;
     END_REPEAT;
     RETURN(s);

     END_FUNCTION; -- compute_known_property_constraints
     (*
```

8.4.3 Compute_known_referenced_property_constraints function

The **compute_known_referenced_property_constraints** function computes all the **property_constraints** that should be imported by an **ap a_priori_semantic_relationship** by computing all the constraints that apply to a property that is imported through the **referenced_properties** attribute of **ap**, and that are defined or inherited in any **ap referenced_classes** class whose **class dictionary_definition** is available in the same exchange context.

NOTE 1 In an **a_priori_semantic_relationship** all the **property_constraint**s defined in or inherited by the classes referenced by the **referenced_classes** attribute that apply to a property that is imported through the

referenced_properties attribute of the **a_priori_semantic_relationship** should be imported through its **referenced_constraints** attribute.

NOTE 2 When the **dictionary_definition** of a class belonging to the **referenced_classes** attribute of **ap** is not available in the same exchange context as **ap** (a PLIB exchange context is never assumed to be complete), the constraints belonging to this class cannot be computed. Thus the result of the function **compute_known_referenced_property_constraints** may be only a subset of the constraints that should be imported by **ap**.

NOTE 3 When the **dictionary_definition**s of all the **referenced_classes** classes of **ap** are available in the same exchange context, and when no constraint is represented by a single **constraint_identifier**, the function **compute_known_referenced_property_constraints** returns exactly all the constraints that should be imported by **ap**.

EXPRESS specification:

```
*)
FUNCTION compute_known_referenced_property_constraints(
          ap: a_priori_semantic_relationship):
          SET OF property_constraint;
LOCAL
     s: SET OF property_constraint := []; -- result
     prop: SET OF property_BSU :=
          list_to_set(ap.referenced_properties); --imported properties
     cl: SET OF class_BSU :=
          ap.referenced_classes; --source of importation
     cons: SET OF property_constraint
          := compute_known_property_constraints(cl);
          -- all those property_constraints existing in the various
          -- classes from cl that may be computed in the current
          -- exchange context.
END_LOCAL;

     REPEAT n_cons := 1 TO SIZEOF(cons);
          IF cons[n_cons].constrained_property IN prop
          THEN
               s := s + cons[n_cons];
          END_IF;
     END_REPEAT;
RETURN(s);

END_FUNCTION; -- compute_known_referenced_property_constraints
(*
```

8.4.4 Superclass_of_item_is_item function

The **superclass_of_item_is_item** function checks that the superclass of an **item_class cl**, if it exists, is an **item_class.**

If the **class** associated with a **class_BSU** cannot be computed, the function returns UNKNOWN.

EXPRESS specification:

```
*)
FUNCTION superclass_of_item_is_item(cl: item_class): LOGICAL;

IF NOT EXISTS(cl\class.its_superclass)
THEN
     RETURN(TRUE);
END_IF;
```

```
IF SIZEOF(cl\class.its_superclass.definition) = 0
THEN
     RETURN(UNKNOWN);
END_IF;

RETURN(('ISO13584_IEC61360_DICTIONARY_SCHEMA.ITEM_CLASS')
     IN TYPEOF(cl\class.its_superclass.definition[1]));

END_FUNCTION; -- superclass_of_item_is_item
(*
```

8.4.5 Check_is_case_of_referenced_classes_definition function

The **check_is_case_of_referenced_classes_definition** returns TRUE if the **item_class_case_of is_case_of** set of referenced class dictionary definition(s) is type compatible with the given **cl item_class_case_of** instance. Otherwise, it returns FALSE.

EXPRESS specification:

```
*)
FUNCTION check_is_case_of_referenced_classes_definition(
     cl: item_class_case_of): BOOLEAN;
LOCAL
     class_def_ok: BOOLEAN := TRUE;
END_LOCAL;

REPEAT i := 1 TO SIZEOF(cl.is_case_of);
     IF (SIZEOF(cl.is_case_of[i].definition) = 1)
     THEN
          IF (NOT('ISO13584_IEC61360_DICTIONARY_SCHEMA' +
               '.ITEM_CLASS'
               IN TYPEOF(cl.is_case_of[i].definition[1])))
          THEN
               class_def_ok := FALSE;
          END_IF;

     END_IF;
END_REPEAT;

RETURN(class_def_ok);

END_FUNCTION; -- check_is_case_of_referenced_classes_definition
(*
```

8.5 ISO13584_IEC61360_item_class_case_of_schema rule definitions

8.5.1 General

This subclause defines the rule in the **ISO13584_IEC61360_item_class_case_of_schema**.

8.5.2 Imported_properties_are_visible_or_applicable_rule rule

The **imported_properties_are_visible_or_applicable_rule** rule checks that when a property is imported by a class by means of an **a_priori_semantic_relationship**, this property is visible or applicable for the class it is imported from.

NOTE Applicable properties include the properties imported through a semantic relationship. This rule enables to import properties from a class where they were already imported.

EXPRESS specification:

```
*)
RULE imported_properties_are_visible_or_applicable_rule FOR(
     a_priori_semantic_relationship, property_DET);
WHERE
     WR1: QUERY(rel <* a_priori_semantic_relationship
           | QUERY(prop <* rel.referenced_properties
           | QUERY(cl <* rel.referenced_classes
           | NOT visible_properties(cl, [prop])
          AND NOT applicable_properties(cl, [prop]))
          = rel.referenced_classes) = [])
          = a_priori_semantic_relationship;
END_RULE; -- imported_properties_are_visible_or_applicable_rule
(*
```

8.5.3 Imported_data_types_are_visible_or_applicable_rule rule

The **imported_data_types_are_visible_or_applicable_rule** rule checks that when a data type is imported by a class by means of an **a_priori_semantic_relationship**, this data type is visible or applicable for the class it is imported from.

NOTE Applicable data types include the data types imported through a semantic relationship. This rule enables to import data types from a class where they were already imported.

EXPRESS specification:

```
*)
RULE imported_data_types_are_visible_or_applicable_rule FOR(
     a_priori_semantic_relationship, data_type_element);
WHERE
     WR1: QUERY(rel <* a_priori_semantic_relationship
           | QUERY(typ <* rel.referenced_data_types
           | QUERY(cl <* rel.referenced_classes
           | NOT visible_types(cl, [typ])
          AND NOT applicable_types(cl, [typ]))
          = rel.referenced_classes) = [])
          = a_priori_semantic_relationship;
END_RULE; -- imported_data_types_are_visible_or_applicable_rule
(*
```

8.5.4 Allowed_named_type_usage_rule rule

The **allowed_named_type_usage_rule** rule is related to the usage of a named type. It states that only types that are applicable to a class may be used to specify the domain of the properties declared by a class, through its **described_by** attribute.

EXPRESS specification:

```
*)
RULE allowed_named_type_usage_rule FOR(class);
LOCAL
     named_type_usage_allowed: LOGICAL := TRUE;
     is_app: LOGICAL;
```

```
      prop: property_bsu;
      cl: class;
      dtnt: SET[0:1] OF data_type_bsu := [];
END_LOCAL;

REPEAT i := 1 TO SIZEOF(class);
      cl := class[i];
      REPEAT j := 1 TO SIZEOF(class[i].described_by);
           prop := cl.described_by[j];
           dtnt := data_type_named_type(prop);

           IF (SIZEOF(dtnt) = 1) THEN
                is_app := applicable_types(cl.identified_by, dtnt);
                IF (NOT is_app) THEN
                     named_type_usage_allowed := FALSE;
                END_IF;
           END_IF;
      END_REPEAT;
END_REPEAT;

WHERE
      WR1: named_type_usage_allowed;
END_RULE; -- allowed_named_type_usage_rule
(*


*)
END_SCHEMA; -- ISO13584_IEC61360_item_class_case_of_schema
(*
```

Annex A
(informative)

Example physical file

A.1 Objective

This annex gives some fragments of a physical file for exchanging the data of IEC 61360-DB. It is intended to show the use of the EXPRESS model in Clause 5 "ISO13584_IEC61360_dictionary_schema" together with ISO 10303-21 to exchange corresponding data.

A.2 File Header

```
*/
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('Example physical file'), '2;1');
FILE_NAME('example.spf', '2007-07-18', ('IEC SC3D WG2'), (),
'Version 1', '', '');
FILE_SCHEMA(('example_schema'));
ENDSEC;
DATA;
/*
```

A.3 Supplier data

```
*/
#1=SUPPLIER_BSU('112/2///61360_4_1', *); /*according to ISO 13584-26*/
#2=SUPPLIER_ELEMENT(#1, #3, '01', $, $, $, #4, #5);
#3=DATES('1994-09-16', '1994-09-16', $);
#4=ORGANIZATION('IEC', 'IEC Maintenance Agency', 'The IEC Maintenance
Agency');
#5=ADDRESS('to be determined', $, $, $, $, $, $, $, $, $, $, $);
#10=SUPPLIER_BSU('112/3///_00', *);   /* ISO/IEC ICS */
/*
```

A.4 Root class data

The AAA000 IEC root class provides a name scope corresponding IEC 61360-DB. It covers two trees, one for materials, one for components. It is defined as an **item_class**.

```
*/
#90=CLASS_BSU('OO', '001', #10);
#100=CLASS_BSU('AAA000', '001', #1);
#101=ITEM_CLASS(#100, #3, '01', $, $, $, #102, TEXT('IEC root class that
provides a name scope corresponding to IEC 61360-DB. It covers  two trees,
one for materials, one for components'), $, $, $, #90, (#110), (), (), $,
(), (#110), (), $, $, $);
#102=ITEM_NAMES(LABEL('IEC root class'), (), LABEL('IEC root'), $, $);
#110=PROPERTY_BSU('AAE000', '001', #100);
```

```
#111=NON_DEPENDENT_P_DET(#110, #3, '01', $, $,$, #112, TEXT('the type of
tree: material or component'), $, $, $, $, (), $, $, #113, $);
#112=ITEM_NAMES(LABEL('type of tree'), (), LABEL('tree type'), $, $);
#113=NON_QUANTITATIVE_CODE_TYPE((), 'A..8', #114);
#114=VALUE_DOMAIN((#120,#122), $, $, (), $, $);
#120=DIC_VALUE(VALUE_CODE_TYPE('MATERIAL'), #121, $, $, $, $, $, $);
#121=ITEM_NAMES(LABEL('material tree'), (), LABEL('mat tree'), $, $);
#122=DIC_VALUE(VALUE_CODE_TYPE('COMPONS'), #123, $, $, $, $, $, $);
#123=ITEM_NAMES(LABEL('component tree'), (), LABEL('comp tree'), $, $);
/*
```

A.5 Material data

```
*/
#200=CLASS_BSU('AAA218', '001', #1);
#201=ITEM_CLASS(#200, #3, '01', $, $, $, #202, TEXT('root class of the
materials tree'), $, $, $, #100, (#210,#230), (), (), $, (), (#210),
(#205), $, 'MATERIAL', $);
#202=ITEM_NAMES(LABEL('materials root class'), (), LABEL('materials root'),
$, $);
#205=CLASS_VALUE_ASSIGNMENT(#110, STRING_VALUE('MATERIAL'));
#210=PROPERTY_BSU('AAF311', '005', #100);
#211=NON_DEPENDENT_P_DET(#210, #3, '01', $, $, $, #212, TEXT('code of the
type of material'), $, $, $, $, (), $, 'A57', #213, $);
#212=ITEM_NAMES(LABEL('material type'), (), LABEL('material type'), $, $);
#213=NON_QUANTITATIVE_CODE_TYPE((), 'M..3', #214);
#214=VALUE_DOMAIN((#220,#222,#224,#226), $, $, (), $, $);
#220=DIC_VALUE(VALUE_CODE_TYPE('ACO'), #221, $, $, $, $, $, $);
#221=ITEM_NAMES(LABEL('acoustical'), (), LABEL('acoustical'), $, $);
#222=DIC_VALUE(VALUE_CODE_TYPE('MG'), #223, $, $, $, $, $, $);
#223=ITEM_NAMES(LABEL('magnetic'), (), LABEL('magnetical'), $, $);
#224=DIC_VALUE(VALUE_CODE_TYPE('OP'), #225, $, $, $, $, $, $);
#225=ITEM_NAMES(LABEL('optical'), (), LABEL('optical'), $, $);
#226=DIC_VALUE(VALUE_CODE_TYPE('TH'), #227, $, $, $, $, $, $);
#227=ITEM_NAMES(LABEL('thermal-electric'), (), LABEL('th-electric'), $, $);
#230=PROPERTY_BSU('AAF286', '005', #100);
#231=NON_DEPENDENT_P_DET(#230, #3, '01', $, $,$, #232, TEXT('The nominal
density (in kg/m**3) of a material.'), $, $, $, #233, (), $, 'K02', #234,
$);
#232=ITEM_NAMES(LABEL('density'), (), LABEL('density'), $, $);
#233=MATHEMATICAL_STRING('$r_d', '&rho;<sub>d</sub>');
#234=REAL_MEASURE_TYPE((), 'NR3..3.3ES2', #235, $, $, $);
#235=DIC_UNIT(#236, $);
#236=DERIVED_UNIT((#239,#237));
#237=DERIVED_UNIT_ELEMENT(#238, 1.);
#238=SI_UNIT(*,.KILO.,.GRAM.);
#239=DERIVED_UNIT_ELEMENT(#240, -3.);
#240=SI_UNIT(*, $,.METRE.);
/*
```

A.6 Component data

```
*/
#300=CLASS_BSU('EEE000', '001', #1);
#301=ITEM_CLASS(#300, #3, '01', $, $, $, #302, TEXT('root class of the
components tree'), $, $, $, #100, (#310,#330,#350), (), (), $, (), (#310),
(#305), $, 'COMPONS', $);
#302=ITEM_NAMES(LABEL('components root class'), (), LABEL('components
root'), $, $);
#305=CLASS_VALUE_ASSIGNMENT(#110, STRING_VALUE('COMPONS'));
#310=PROPERTY_BSU('AAE001', '005', #100);
#311=NON_DEPENDENT_P_DET(#310, #3, '01', $, $, $, #312, TEXT('Code of the
main functional class to which a component belongs'), $, $, $, $, (), $,
'A52', #313, $);
#312=ITEM_NAMES(LABEL('main class of component'), (), LABEL('main class'),
$, $);
#313=NON_QUANTITATIVE_CODE_TYPE((), 'M..3', #314);
#314=VALUE_DOMAIN((#320,#322,#324,#326), $, $, (), $, $);
#320=DIC_VALUE(VALUE_CODE_TYPE('EE'), #321, $, $, $, $, $, $);
#321=ITEM_NAMES(LABEL('EE (electric / electronic)'), (), LABEL('EE'), $,
$);
#322=DIC_VALUE(VALUE_CODE_TYPE('EM'), #323, $, $, $, $, $, $);
#323=ITEM_NAMES(LABEL('electromechanical'), (), LABEL('electromech'), $,
$);
#324=DIC_VALUE(VALUE_CODE_TYPE('ME'), #325, $, $, $, $, $, $);
#325=ITEM_NAMES(LABEL('mechanical'), (), LABEL('mechanical'), $, $);
#326=DIC_VALUE(VALUE_CODE_TYPE('MP'), #327, $, $, $, $, $, $);
#327=ITEM_NAMES(LABEL('magnetic part'), (), LABEL('magnetic'), $, $);
#330=PROPERTY_BSU('AAF267', '005', #100);
#331=NON_DEPENDENT_P_DET(#330, #3, '01', $, $,$, #332, TEXT('The nominal
distance (in m) between the inside of the two tapes used for taped products
with axial leads'), $, $, $, #333, (), $, 'T03', #334, $);
#332=ITEM_NAMES(LABEL('inner tape spacing'), (), LABEL('inner tape spac'),
$, $);
#333=MATHEMATICAL_STRING('b_tape', 'b<sub>tape</sub>');
#334=LEVEL_TYPE((), (.NOM.), #335);
#335=REAL_MEASURE_TYPE((), 'NR3..3.3ES2', #336, $, $, $);
#336=DIC_UNIT(#337, $);
#337=SI_UNIT(*, $,.METRE.);
#350=PROPERTY_BSU('AAE022', '005', #100);
#351=NON_DEPENDENT_P_DET(#350, #3, '01', $, $, $,#352, TEXT('The value as
specified by level (miNoMax) of the outside diameter (in m) of a component
with a body of circular cross-section'), $, $, $, #353, (), $, 'T03', #354,
$);
#352=ITEM_NAMES(LABEL('outside diameter'), (), LABEL('outside diam'), $,
$);
#353=MATHEMATICAL_STRING('d_out', 'd<sub>out</sub>');
#354=LEVEL_TYPE((), (.MIN.,.NOM.,.MAX.), #355);
#355=REAL_MEASURE_TYPE((), 'NR3..3.3ES2', #356, $, $, $);
#356=DIC_UNIT(#357, #358);
```

```
#357=SI_UNIT(*, $,.METRE.);
#358=MATHEMATICAL_STRING('m', 'm');
/*
```

A.7 Electric / electronic component data

```
*/
#400=CLASS_BSU('EEE001', '001', #1);
#401=ITEM_CLASS(#400, #3, '01', $, $,$, #402, TEXT('electric / electronic
components'), $, $, $, #300, (#410,#470), (), (), $, (), (#410), (#405), $,
'EE', $);
#402=ITEM_NAMES(LABEL('EE components'), (), LABEL('EE components'), $, $);
#405=CLASS_VALUE_ASSIGNMENT(#310, STRING_VALUE('EE'));
#410=PROPERTY_BSU('AAE002', '005', #100);
#411=NON_DEPENDENT_P_DET(#410, #3, '01', $, $,$, #412, TEXT('Code of the
category to which an electric/electronic component belongs.'), $, $, $, $,
(), $, 'A52', #413, $);
#412=ITEM_NAMES(LABEL('category EE component'), (), LABEL('categ EE comp'),
$, $);
#413=NON_QUANTITATIVE_CODE_TYPE((), 'M..3', #414);
#414=VALUE_DOMAIN((#420,#422,#424,#426,#428
,#430,#432,#434,#436,#438
,#440), $, $, (), $, $);
#420=DIC_VALUE(VALUE_CODE_TYPE('AMP'), #421, $, $, $, $, $, $);
#421=ITEM_NAMES(LABEL('amplifier'), (), LABEL('amplifier'), $, $);
#422=DIC_VALUE(VALUE_CODE_TYPE('ANT'), #423, $, $, $, $, $, $);
#423=ITEM_NAMES(LABEL('antenna (aerial)'), (), LABEL('antenna (aer)'), $,
$);
#424=DIC_VALUE(VALUE_CODE_TYPE('BAT'), #425, $, $, $, $, $, $);
#425=ITEM_NAMES(LABEL('battery'), (), LABEL('battery'), $, $);
#426=DIC_VALUE(VALUE_CODE_TYPE('CAP'), #427, $, $, $, $, $, $);
#427=ITEM_NAMES(LABEL('capacitor'), (), LABEL('capacitor'), $, $);
#428=DIC_VALUE(VALUE_CODE_TYPE('CND'), #429, $, $, $, $, $, $);
#429=ITEM_NAMES(LABEL('conductor'), (), LABEL('conductor'), $, $);
#430=DIC_VALUE(VALUE_CODE_TYPE('DEL'), #431, $, $, $, $, $, $);
#431=ITEM_NAMES(LABEL('delay line'), (), LABEL('delay line'), $, $);
#432=DIC_VALUE(VALUE_CODE_TYPE('DID'), #433, $, $, $, $, $, $);
#433=ITEM_NAMES(LABEL('diode device'), (), LABEL('diode device'), $, $);
#434=DIC_VALUE(VALUE_CODE_TYPE('FIL'), #435, $, $, $, $, $, $);
#435=ITEM_NAMES(LABEL('filter'), (), LABEL('filter'), $, $);
#436=DIC_VALUE(VALUE_CODE_TYPE('IC'), #437, $, $, $, $, $, $);
#437=ITEM_NAMES(LABEL('integrated circuit'), (), LABEL('IC'), $, $);
#438=DIC_VALUE(VALUE_CODE_TYPE('IND'), #439, $, $, $, $, $, $);
#439=ITEM_NAMES(LABEL('inductor'), (), LABEL('inductor'), $, $);
#440=DIC_VALUE(VALUE_CODE_TYPE('LAM'), #441, $, $, $, $, $, $);
#441=ITEM_NAMES(LABEL('lamp'), (), LABEL('lamp'), $, $);
#470=PROPERTY_BSU('AAE754', '005', #100);
```

```
#471=NON_DEPENDENT_P_DET(#470, #3, '01', $, $, $,#472, TEXT('The number of
electrical terminals of an electric/electronic or electromechanical
component'), $, $, $, #473, (), $, 'Q56', #474, $);
#472=ITEM_NAMES(LABEL('number of terminals'), (LABEL('number of pins')),
LABEL('nr of terminals'), $, $);
#473=MATHEMATICAL_STRING('N_term', 'N<sub>term</sub>');
#474=INT_TYPE((), 'NR1..4');

ENDSEC;
END-ISO-10303-21;
```

Annex B
(informative)

EXPRESS-G Diagram

This annex contains the EXPRESS-G diagrams for the Clauses 6 through 8. EXPRESS-G is defined in Annex A of ISO 10303-11:2004.

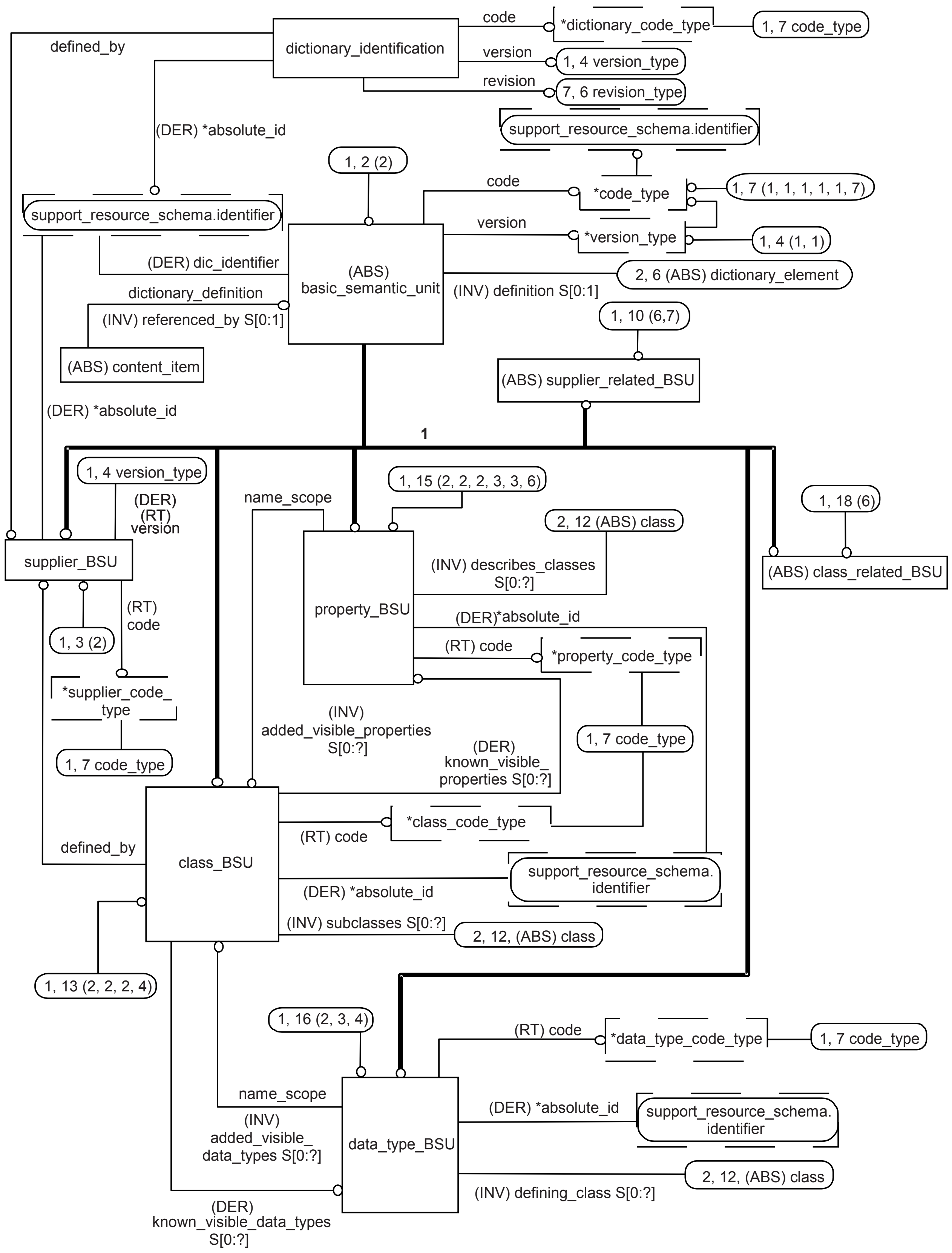

Figure B.1 – ISO13584_IEC61360_dictionary_schema – EXPRESS-G diagram 1 of 7

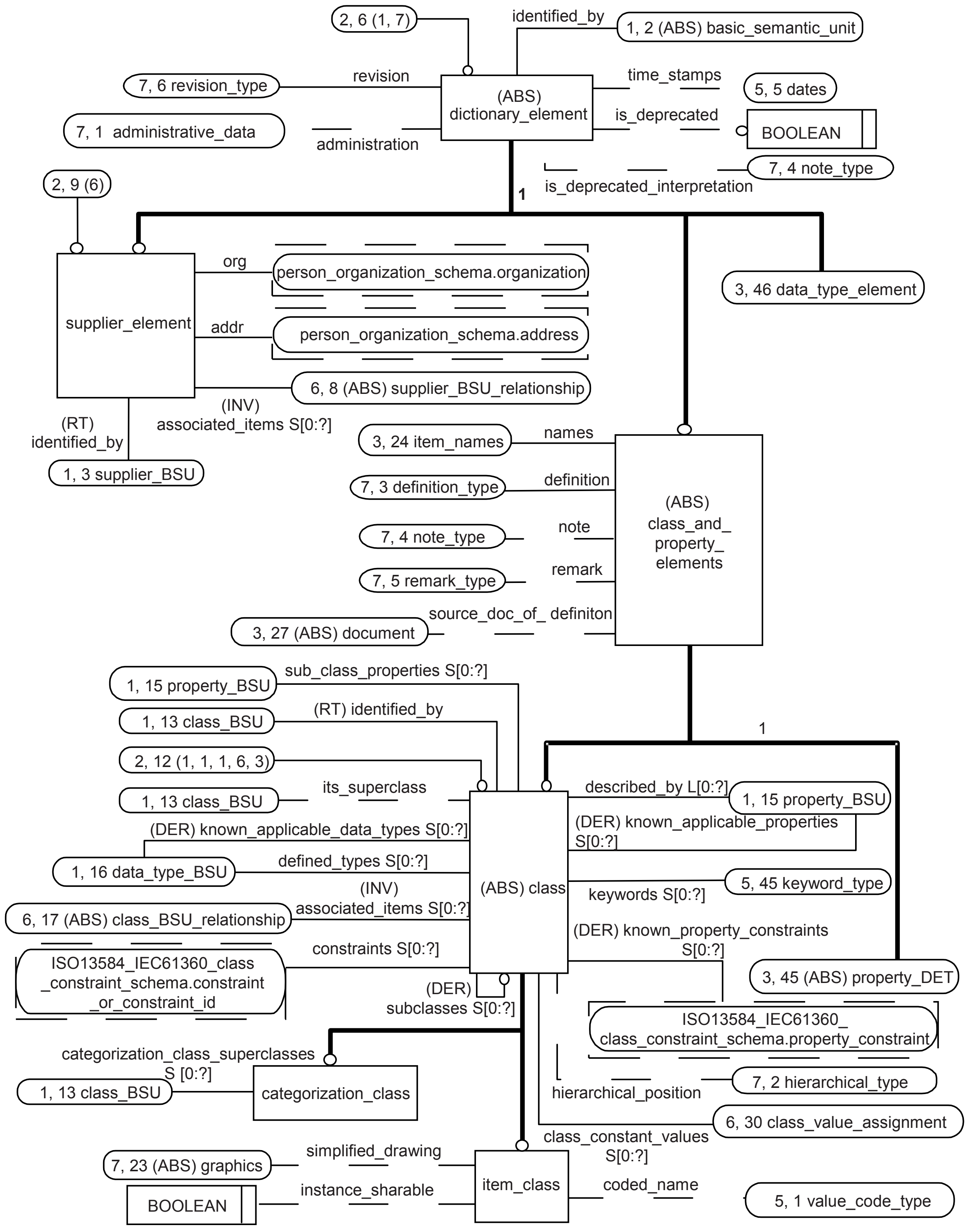

Figure B.2 – ISO13584_IEC61360_dictionary_schema – EXPRESS-G diagram 2 of 7

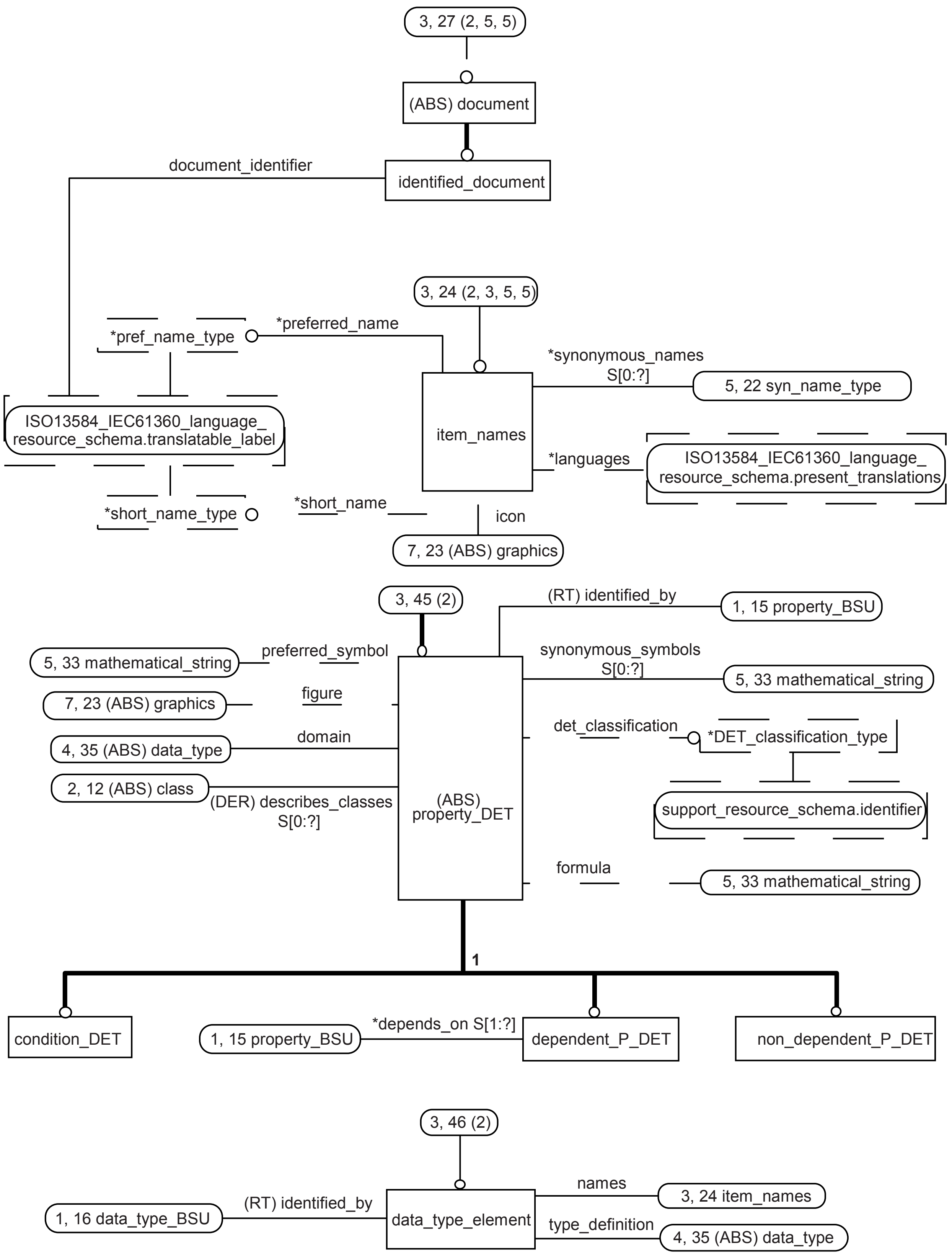

Figure B.3 – ISO13584_IEC61360_dictionary_schema – EXPRESS-G diagram 3 of 7

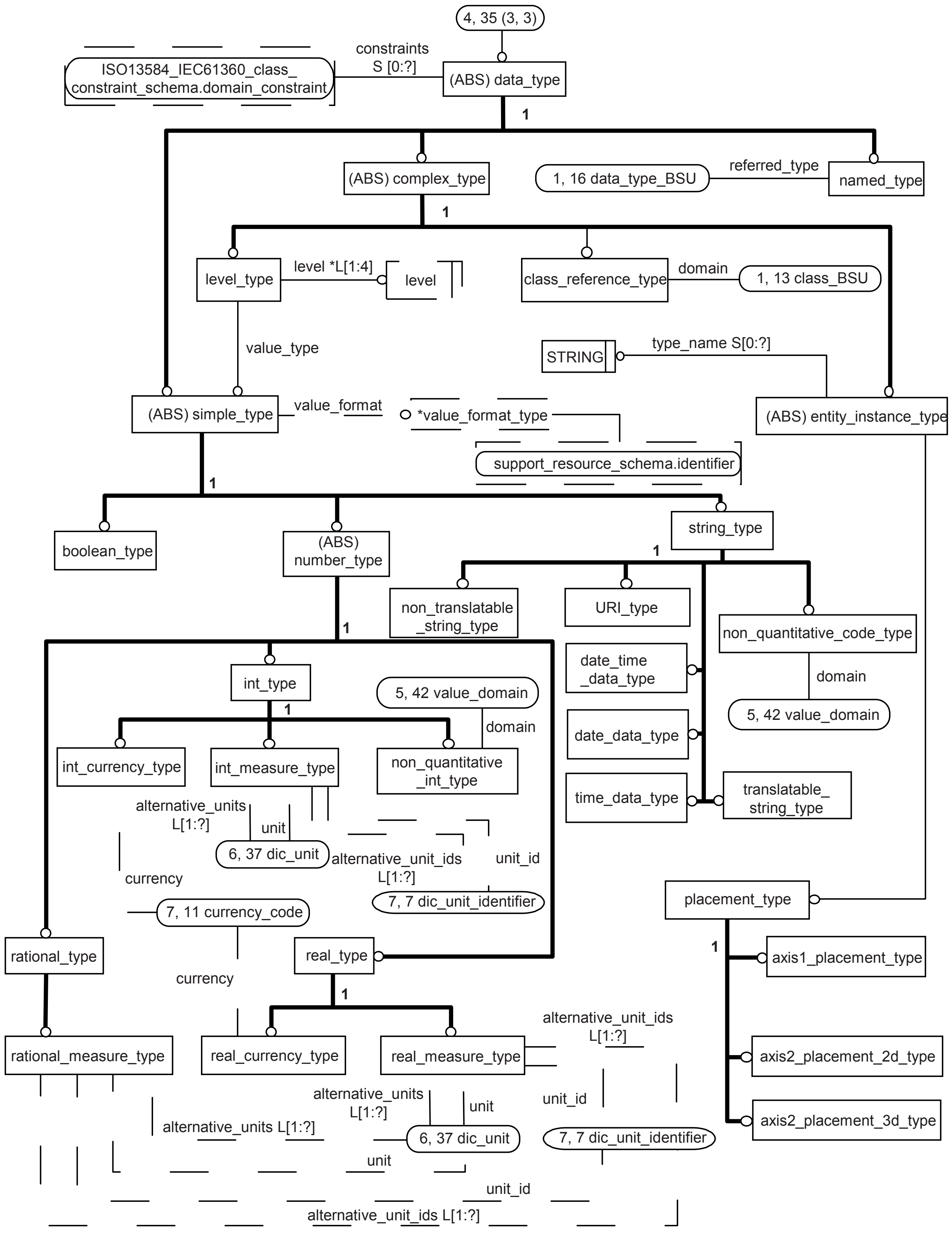

Figure B.4 – ISO13584_IEC61360_dictionary_schema EXPRESS-G diagram 4 of 7

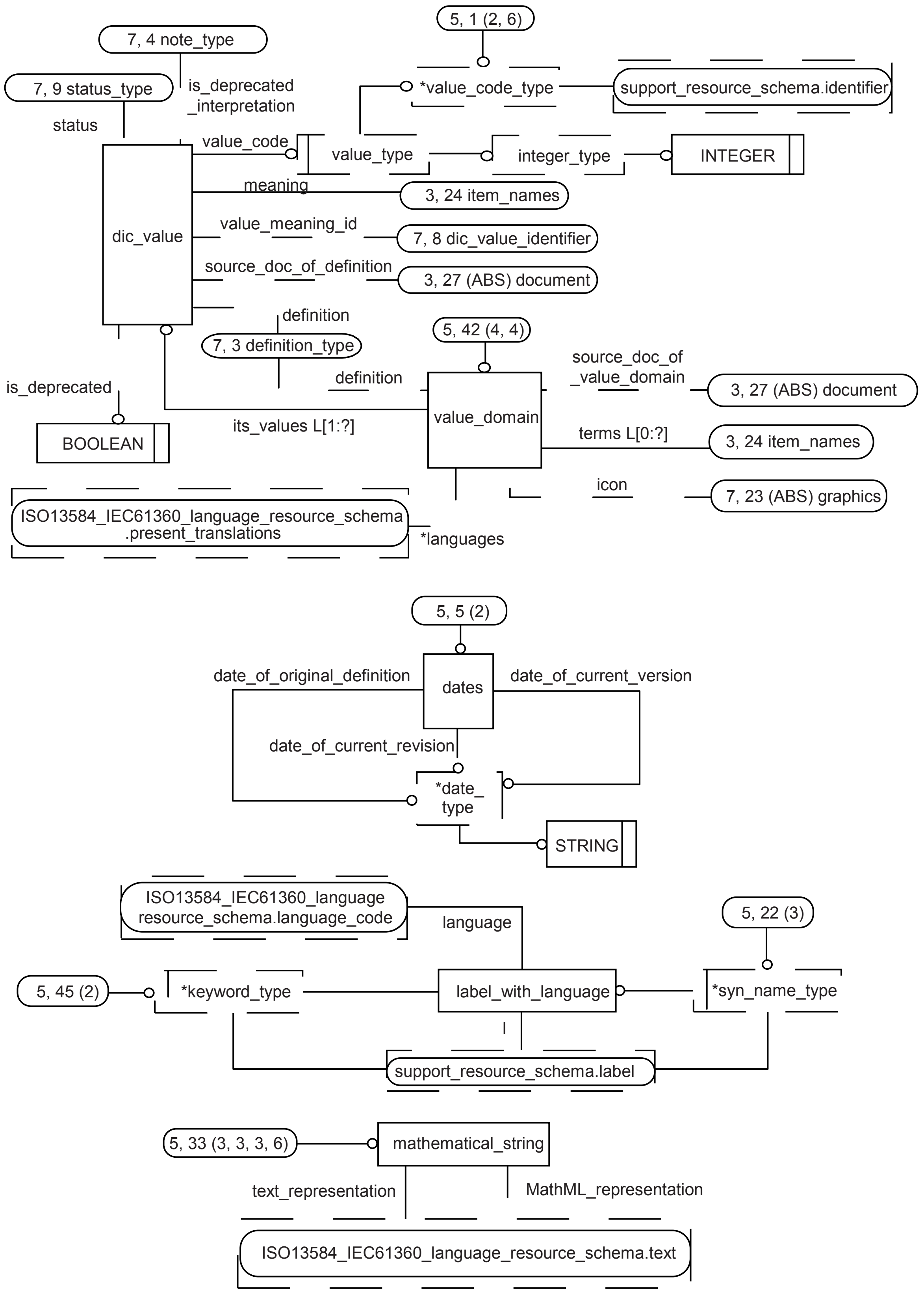

Figure B.5 – ISO13584_IEC61360_dictionary_schema – EXPRESS-G diagram 5 of 7

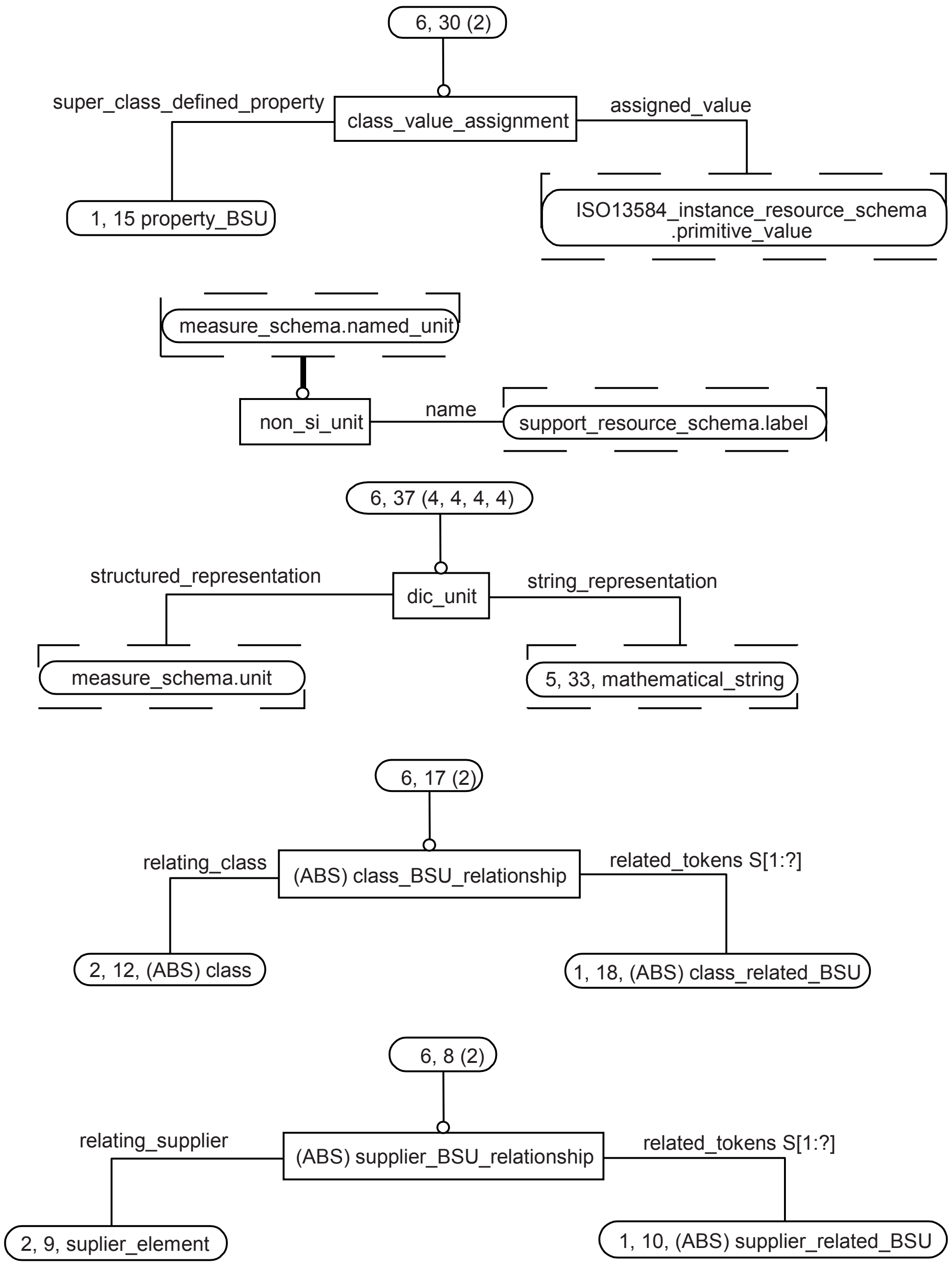
6, 30 (2)
super_class_defined_property
class_value_assignment
assigned_value
1, 15 property_BSU
ISO13584_instance_resource_schema .primitive_value
measure_schema.named_unit
non_si_unit
name
support_resource_schema.label
6, 37 (4, 4, 4, 4)
structured_representation
dic_unit
string_representation
measure_schema.unit
5, 33, mathematical_string
6, 17 (2)
relating_class
(ABS) class_BSU_relationship
related_tokens S[1:?]
2, 12, (ABS) class
1, 18, (ABS) class_related_BSU
6, 8 (2)
relating_supplier
(ABS) supplier_BSU_relationship
related_tokens S[1:?]
2, 9, suplier_element
1, 10, (ABS) supplier_related_BSU

Figure B.6 – ISO13584_IEC61360_dictionary_schema – EXPRESS-G diagram 6 of 7

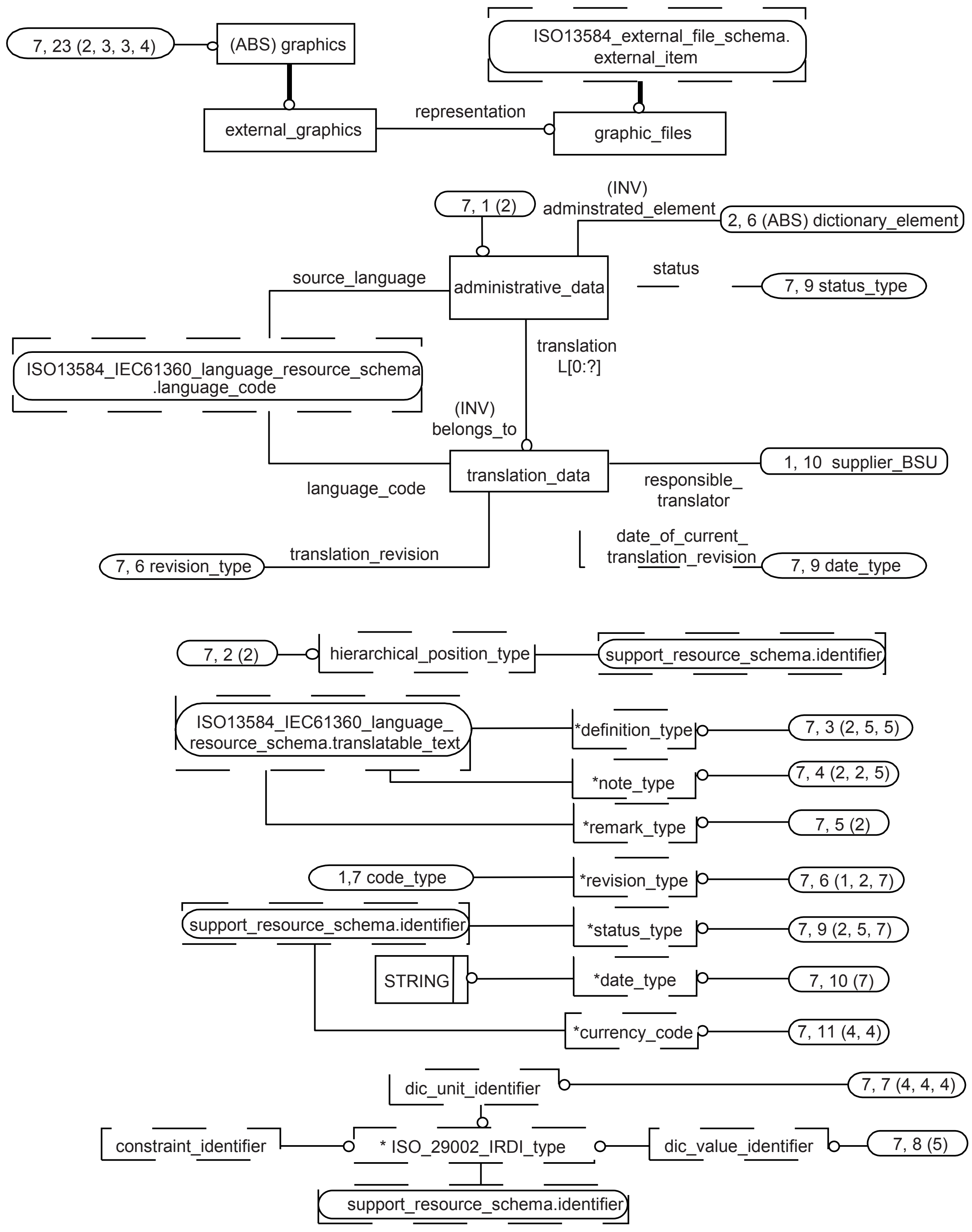

Figure B.7 – ISO13584_IEC61360_dictionary_schema – EXPRESS-G diagram 7 of 7

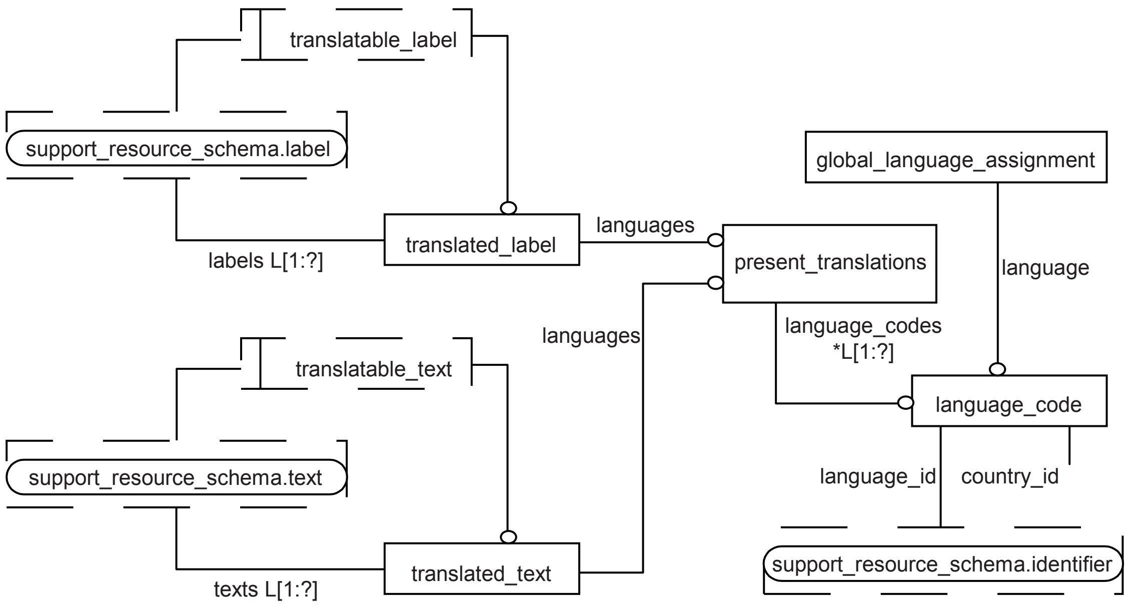

Figure B.8 – ISO13584_IEC61360_language_resource_schema – EXPRESS-G diagram 1 of 1

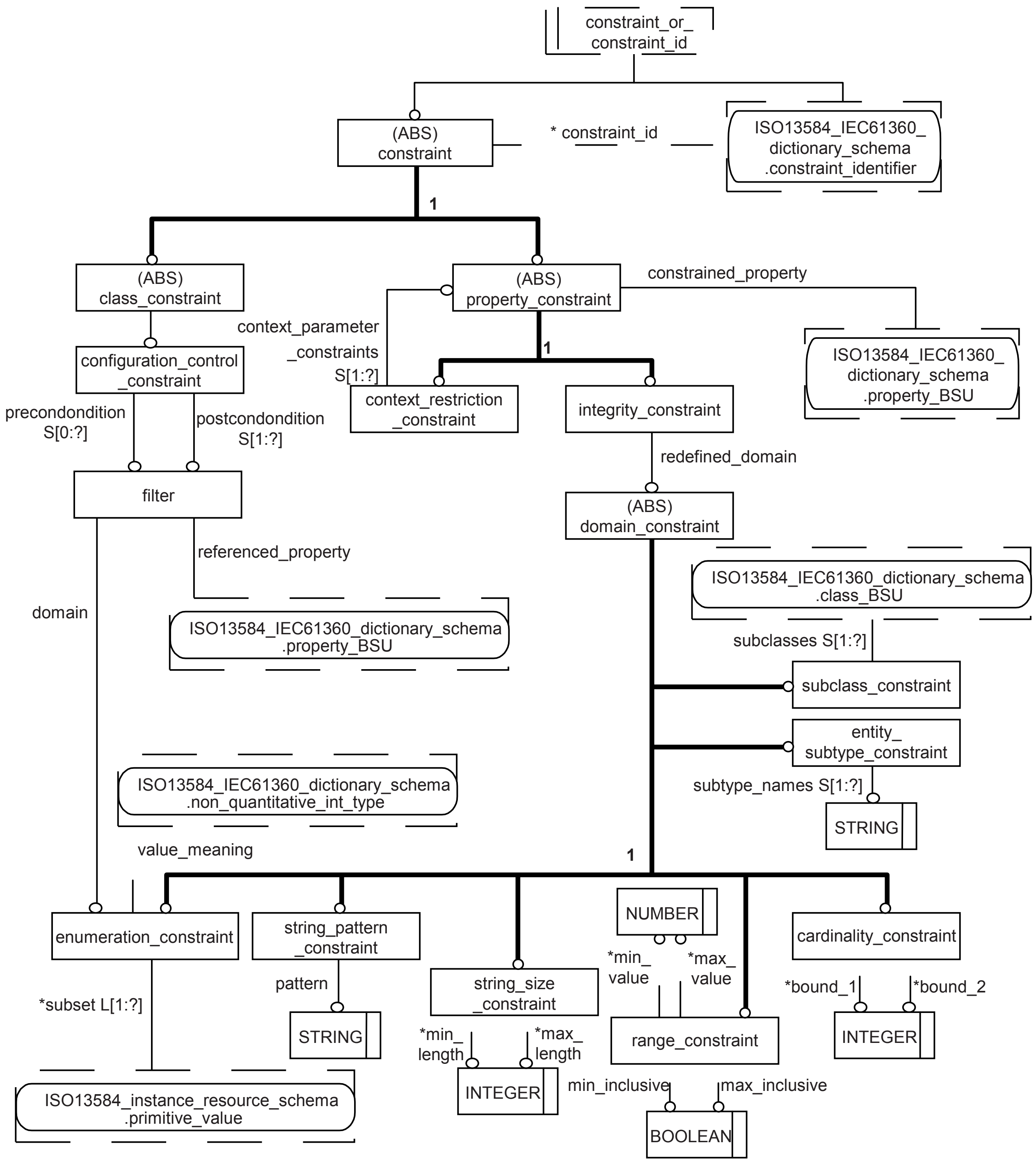
constraint_or_ constraint_id
(ABS) constraint
* constraint_id
ISO13584_IEC61360_ dictionary_schema .constraint_identifier
1
(ABS) class_constraint
(ABS) property_constraint
constrained_property
ISO13584_IEC61360_ dictionary_schema .property_BSU
context_parameter _constraints S[1:?]
configuration_control _constraint
context_restriction _constraint
integrity_constraint
precondondition S[0:?]
postcondondition S[1:?]
redefined_domain
filter
(ABS) domain_constraint
referenced_property
ISO13584_IEC61360_dictionary_schema .class_BSU
domain
ISO13584_IEC61360_dictionary_schema .property_BSU
subclasses S[1:?]
subclass_constraint
entity_ subtype_constraint
subtype_names S[1:?]
ISO13584_IEC61360_dictionary_schema .non_quantitative_int_type
STRING
value_meaning
enumeration_constraint
string_pattern _constraint
string_size _constraint
NUMBER
*min_ value
*max_ value
range_constraint
cardinality_constraint
*bound_1
*bound_2
INTEGER
*subset L[1:?]
pattern
STRING
*min_ length
*max_ length
INTEGER
min_inclusive
max_inclusive
BOOLEAN
ISO13584_instance_resource_schema .primitive_value

Figure B.9 – ISO13584_IEC61360_constraint_schema – EXPRESS-G diagram 1 of 1

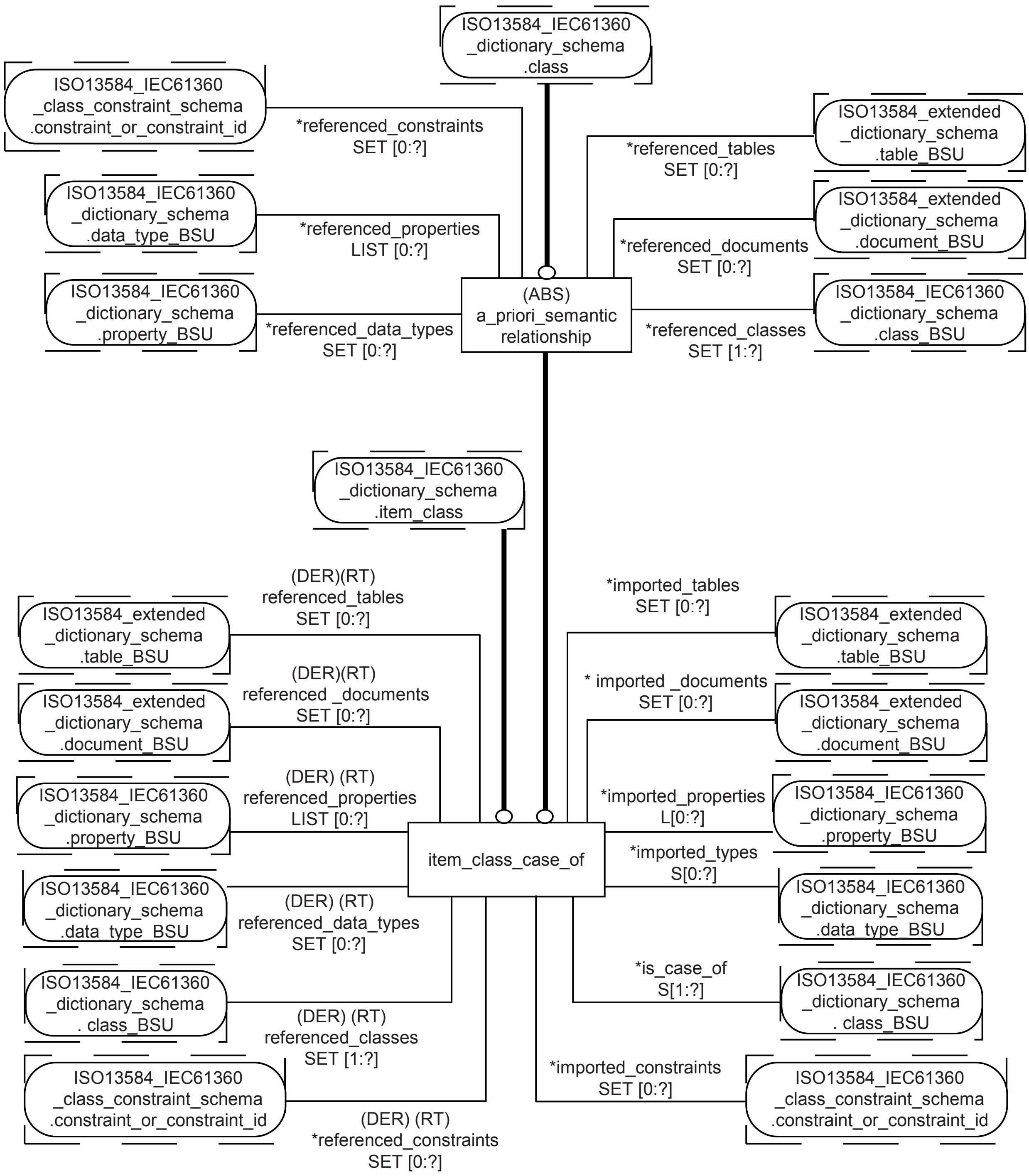
ISO13584_IEC61360 _dictionary_schema .class
ISO13584_IEC61360 _class_constraint_schema .constraint_or_constraint_id
*referenced_constraints SET [0:?]
ISO13584_IEC61360 _dictionary_schema .data_type_BSU
*referenced_properties LIST [0:?]
ISO13584_IEC61360 _dictionary_schema .property_BSU
*referenced_data_types SET [0:?]
(ABS) a_priori_semantic relationship
*referenced_tables SET [0:?]
ISO13584_extended _dictionary_schema .table_BSU
*referenced_documents SET [0:?]
ISO13584_extended _dictionary_schema .document_BSU
*referenced_classes SET [1:?]
ISO13584_IEC61360 _dictionary_schema .class_BSU
ISO13584_IEC61360 _dictionary_schema .item_class
(DER)(RT) referenced_tables SET [0:?]
ISO13584_extended _dictionary_schema .table_BSU
(DER)(RT) referenced _documents SET [0:?]
ISO13584_extended _dictionary_schema .document_BSU
(DER) (RT) referenced_properties LIST [0:?]
ISO13584_IEC61360 _dictionary_schema .property_BSU
ISO13584_IEC61360 _dictionary_schema .data_type_BSU
(DER) (RT) referenced_data_types SET [0:?]
ISO13584_IEC61360 _dictionary_schema . class_BSU
(DER) (RT) referenced_classes SET [1:?]
ISO13584_IEC61360 _class_constraint_schema .constraint_or_constraint_id
(DER) (RT) *referenced_constraints SET [0:?]
item_class_case_of
*imported_tables SET [0:?]
ISO13584_extended _dictionary_schema .table_BSU
* imported _documents SET [0:?]
ISO13584_extended _dictionary_schema .document_BSU
*imported_properties L[0:?]
ISO13584_IEC61360 _dictionary_schema .property_BSU
*imported_types S[0:?]
ISO13584_IEC61360 _dictionary_schema .data_type_BSU
*is_case_of S[1:?]
ISO13584_IEC61360 _dictionary_schema . class_BSU
*imported_constraints SET [0:?]
ISO13584_IEC61360 _class_constraint_schema .constraint_or_constraint_id

Figure B.10 – ISO13584_IEC61360_item_class_case_of_schema – EXPRESS-G diagram 1 of 1

Annex C
(informative)

Partial dictionaries

The EXPRESS data model published in this part of IEC 61360 and duplicated for convenience in ISO 13584-42 allows to describe dictionaries composed of classes, properties and data types and it provides for their unique identification through the Basic Semantic Unit (BSU) mechanism. This data model allows to describe hierarchies of classes structured according to a tree structure using a simple inheritance mechanism. With this data model, only single and self-contained dictionaries were addressed.

The use of this data model leads to several different dictionaries. During a dictionary building process it may happen that designers require to reference a particular class, property or data type already defined in another dictionary. The possibility to import properties and data types in the dictionary under design is offered by the case-of relationship that may be used with the complete common ISO13584/IEC61360 dictionary model, documented both in ISO 13584-25 and in IEC 61360-5. Indeed, this relationship allows to import externally defined properties and data types providing the possibility to support partial dictionaries and to avoid duplication between dictionaries, each dictionary defining its own class structure.

In the complete common ISO13584/IEC61360 dictionary model, two mechanisms are offered:

– the **a_priori_case_of_semantic_relationship** EXPRESS entity allows to directly use properties or data types defined in external dictionary or dictionaries without describing them again;
– the **a_posteriori_case_of_relationship** EXPRESS entity allows, once properties or data types have been already defined in a dictionary under design, to map them onto corresponding properties or data types defined in an external dictionary.

These mechanisms allow to design dictionaries that refers to data elements that are defined in other dictionaries without affecting their semantic meaning. Moreover, the case-of relationships are recorded in the dictionaries that use this capability, providing for automatic integration of dictionaries based on the same standard dictionaries.

This mechanism is also recommended when designing an end-user dictionary. As a rule, an end-user dictionary does not need the whole class structure defined in standard dictionaries, while wishing to be able to exchange information with other users whose dictionaries are based on the same standard dictionary or dictionaries. If the end-user defines its own hierarchy but maps each of his/her classes through case-of onto the corresponding standard class, and if he/she imports all the existing standard properties that prove useful in his/her context, while adding user-specific properties, the user may customise its own dictionary while being able to exchange standard information with other users. This approach for designing end-user dictionaries is the approach recommended in this standard.

Annex D
(normative)

Value format specification

D.1 General

This part of IEC 61360 and ISO 13584-42 provide 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 **data_type**, including **boolean_type**.

NOTE 2 In this part of IEC 61360, 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.

D.2 Notation

Table D.1 summarizes the subset of the ISO/IEC 14977 EBNF syntactic metalanguage used by this part of IEC 61360 to specify value format of properties.

Using these notations, the syntax of the subset of the EBNF metalanguage used by this part of IEC 61360 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 D.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.

D.3 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 D.5 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 D.6 we define the meta-identifiers for the five non-quantitative value formats, together with their meaning at the value level.

D.4 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 combines to form numbers, except the first one.

NOTE If a number comprises only one digit, no `trailingDigit` is present.

D.5 Quantitative value formats

D.5.1 General

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** or any of its subtype;
– **level_type** whose **value_type** are either **real_measure_type** or **int_measure_type**;
– **list_type**, **set_type**, **bag_type**, **array_type** or **set_with_subset_constraint_type** whose **value_type** are **number_type** or any of its subtype.

NOTE 1 **list_type**, **set_type**, **bag_type**, **array_type** or **set_with_subset_constraint_type** are defined in ISO 13584-25.

NOTE 2 For **non_quantitative_int_type** the value format applies to the code.

NOTE 3 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**, since integer values should not have a fractional part.

D.5.2 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 should 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`.

D.5.3 NR2-value format

The NR2-value syntax specifies the format of a 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 should 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.

D.5.4 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 should 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.

D.5.5 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.

D.6 Non-quantitative value formats

D.6.1 General

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** or any of its subtype;
- **list_type**, **set_type**, **bag_type**, **array_type** or **set_with_subset_constraint_type** whose **value_type** are **string_type** or any of its subtype.

NOTE 1 **list_type**, **set_type**, **bag_type**, **array_type** or **set_with_subset_constraint_type** are defined in ISO 13584-25.

NOTE 2 For **translatable_string_type** the value format applies to any language-specific representation of the string.

NOTE 3 For **non_quantitative_code_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.

D.6.2 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 4 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 5 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.

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.

D.6.3 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 D.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.

D.6.4 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 D.2 shows the transposition of the European digits “0” to “9” into Arabic digits.

Table D.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.

D.6.5 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.

D.6.6 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.

D.7 Value examples

Table D.3 below contains some examples for values that may be contained in numbers and strings characterized by the above value format scheme.

Table D.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;

Characters from ISO/IEC 10646-1

The following characters shall be used for the purpose of Mixed value format (M) (see D.5.3):

– 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 D.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 D.4 – Characters from other rows of the Basic Multilingual Plane of ISO/IEC 10646-1

Character	Name	Row	Cell
	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
∫	INTEGRAL	22	2B
∮	CONTOUR INTEGRAL	22	2E
∞	INFINITY	22	1E

Table D.4 *(continued)*

Character	Name	Row	Cell
∇	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

Table D.4 *(continued)*

Character	Name	Row	Cell
⊾	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
′	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

Table D.4 *(continued)*

Character	Name	Row	Cell
⇀	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
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

Table D.4 *(continued)*

Character	Name	Row	Cell
‰	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

Bibliography

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

[2] IEC 60748 (all parts), *Semiconductor devices – Integrated circuits*

[3] IEC 61360-5, *Standard data element types with associated classification scheme for electric components – Part 5: Extensions to the EXPRESS dictionary schema*

[4] IEC 80000 (all parts)[3], *Quantities and units*

[5] ISO/IEC 8824-1, *Information technology – Abstract Syntax Notation One (ASN.1) – Part 1: Specification of basic notation*

[6] ISO/IEC 11179-5, *Information technology – Metadata registries (MDR) – Part 5: Naming and identification principles*

[7] ISO/IEC, *International Classification of Standard (ICS)*

[8] ISO 31 (all parts), *Quantities and units*

[9] ISO 639-1, *Codes for the representation of names of languages – Part 1: Alpha-2 code*

[10] ISO 639-2, *Codes for the representation of names of languages – Part 2: Alpha-3 code*

[11] ISO 704, *Terminology work – Principles and methods*

[12] ISO 843, *Information and documentation – Conversion of Greek characters into Latin characters*

[13] ISO 1087-1, *Terminology work – Vocabulary – Part 1: Theory and application*

[14] ISO 6523, *Data interchange – Structure for the identification of organizations*

[15] ISO 10303-1:1994, *Industrial automation systems and integration – Product data representation and exchange – Part 1: Overview and fundamental principles*

[16] ISO 10303-21, *Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure*

[17] ISO 10303-42:2000, *Industrial automation systems and integration – Product data representation and exchange – Part 42: Integrated generic resource: Geometric and topological representation*[2]

[18] ISO 13584-1:2001, *Industrial automation systems and integration – Parts library – Part 1: Overview and fundamental principles*

[19] ISO 13584-24:2003, *Industrial automation systems and integration – Parts library – Part 24: Logical resource: Logical model of supplier library*

2 A new edition of ISO 10303-41 was published in 2005.

[20] ISO 13584-25, *Industrial automation systems and integration – Parts library – Part 25: Logical resource: Logical model of supplier library with aggregate values and explicit content*

[21] ISO 13584-32, *Industrial automation systems and integration – Parts library – Part 32: Implementation resources: OntoML: Product ontology markup language*

[22] ISO 13584-511, *Industrial automation systems and integration – Parts library – Part 511: Mechanical systems and components for general use – Reference dictionary for fasteners*

[23] ISO/TS 23768-1[3], *Rolling bearings – Parts library – Part 1: Reference dictionary for rolling bearings*

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

[25] ISO/TS 29002-20, *Industrial automation systems and integration – Exchange of characteristic data – Part 20: Concept dictionary resolution services*

[26] ISO 80000 (all parts)[4], *Quantities and units*

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

[28] Mathematical Markup Language (MathML). Second Edition. World Wide Web Consortium Recommendation 21-October-2003
Available from <http://www.w3.org/TR/MathML2/>

3 To be published.

4 The ISO 31 series of parts has been cancelled and replaced by the ISO 80000/IEC 80000 series of parts.

This page deliberately left blank

This page deliberately left blank