IEC 61360-2

Edition 3.0 2012-10

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Standard data element types with associated classification scheme for electric components –
Part 2: EXPRESS dictionary schema

Types normalisés d'éléments de données avec plan de classification pour composants électriques –
Partie 2: Schéma d'un dictionnaire EXPRESS

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

Useful links:

IEC publications search - www.iec.ch/searchpub

The advanced search enables you to find IEC publications by a variety of criteria (reference number, text, technical committee,…).
It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available on-line and also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary (IEV) on-line.

Customer Service Centre - webstore.iec.ch/csc

If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

A propos de la CEI

La Commission Electrotechnique Internationale (CEI) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications CEI

Le contenu technique des publications de la CEI est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Liens utiles:

Recherche de publications CEI - www.iec.ch/searchpub

La recherche avancée vous permet de trouver des publications CEI en utilisant différents critères (numéro de référence, texte, comité d'études,…).
Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

Just Published CEI - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications de la CEI. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org

Le premier dictionnaire en ligne au monde de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans les langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (VEI) en ligne.

Service Clients - webstore.iec.ch/csc

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

IEC 61360-2

Edition 3.0 2012-10

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Standard data element types with associated classification scheme for electric components – Part 2: EXPRESS dictionary schema

Types normalisés d'éléments de données avec plan de classification pour composants électriques – Partie 2: Schéma d'un dictionnaire EXPRESS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE
CODE PRIX XH

ICS 31.020

ISBN 978-2-83220-321-7

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

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

Part 2: EXPRESS dictionary schema

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 61360-2 has been prepared by subcommittee 3D: Product properties and classes and their identification, of IEC technical committee 3: Information structures, documentation and graphical symbols.

This third edition cancels and replaces the second edition published in 2002, and its Amendment 1 (2003). It is a technical revision.

This edition includes the following significant technical changes with respect to the previous edition:

– 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.

The text of this standard is based on the following documents:

FDIS	Report on voting
3D/196/FDIS	3D/204/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

A list of all parts of the IEC 61360 series can be found, under the general title *Standard data elements types with associated classification scheme for electric components*, on the IEC website.

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

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-time* 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.

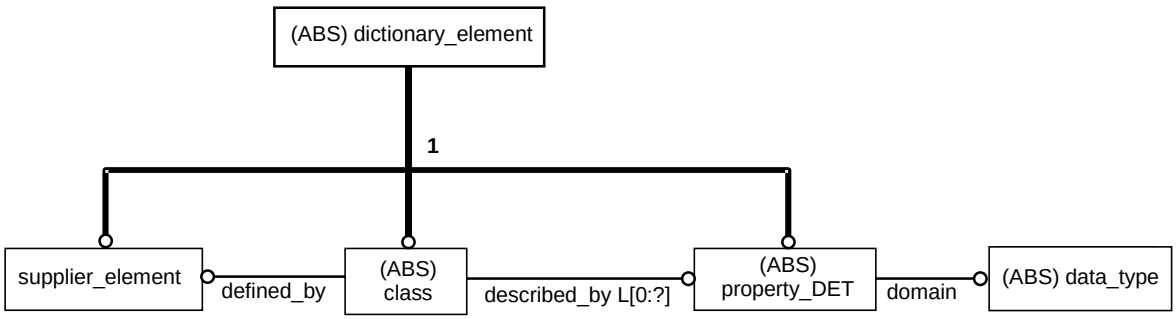


Figure 1 – Overview of the dictionary schema

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.

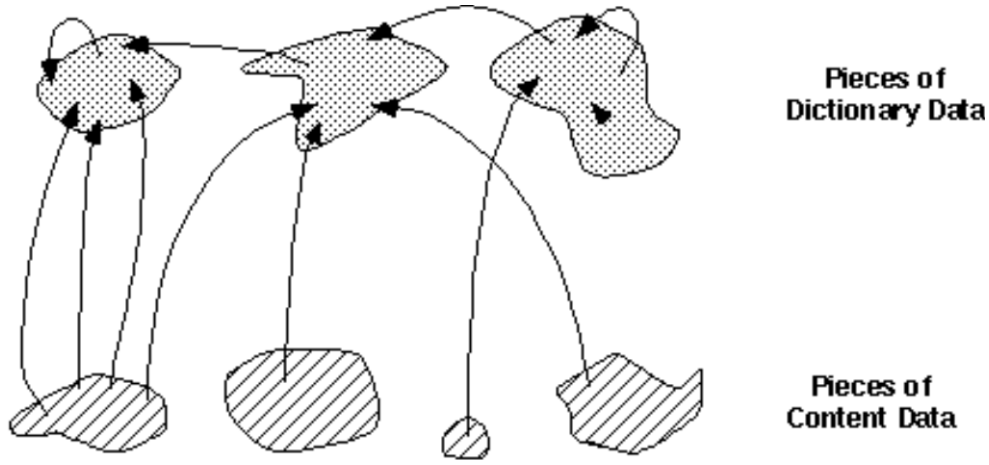


Figure 2 – Pieces of data with relationships

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.

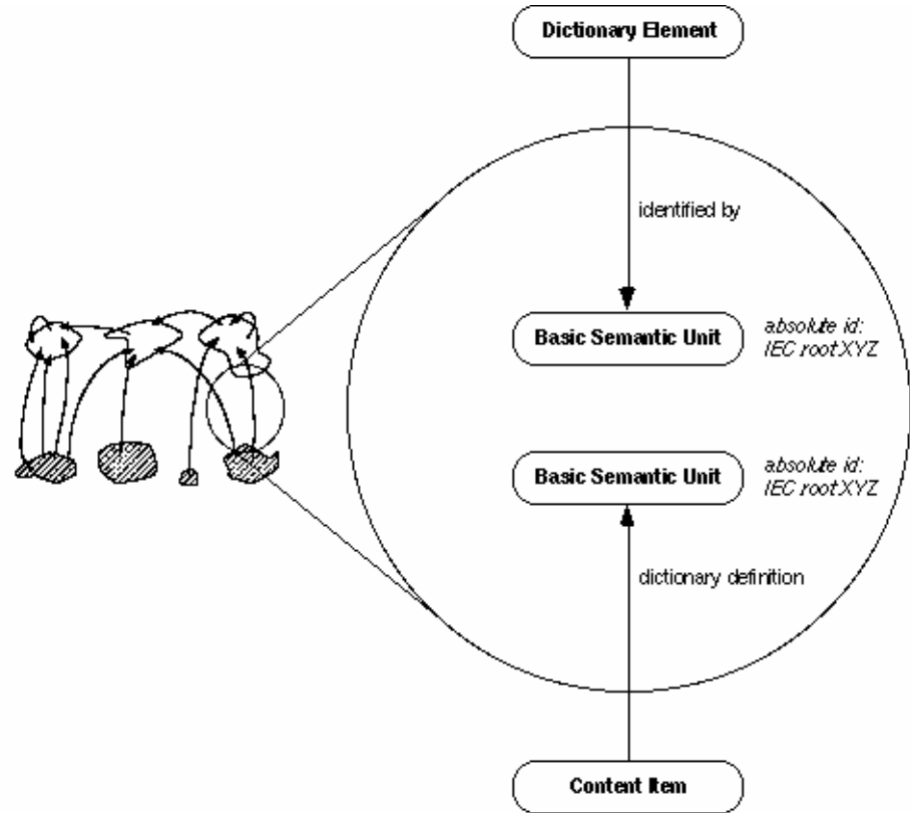


Figure 3 – Implementation of "inter-piece" relationships using basic semantic units

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.

EXPRESS specification:

```
*)
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.

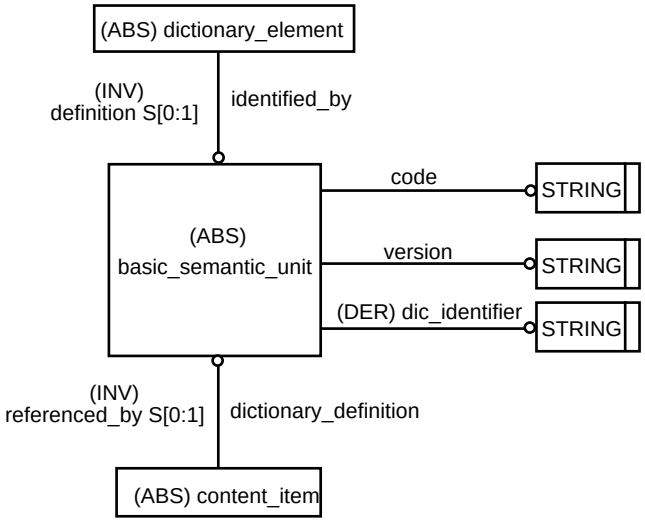


Figure 4 – Relationship between basic semantic unit and dictionary element

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.

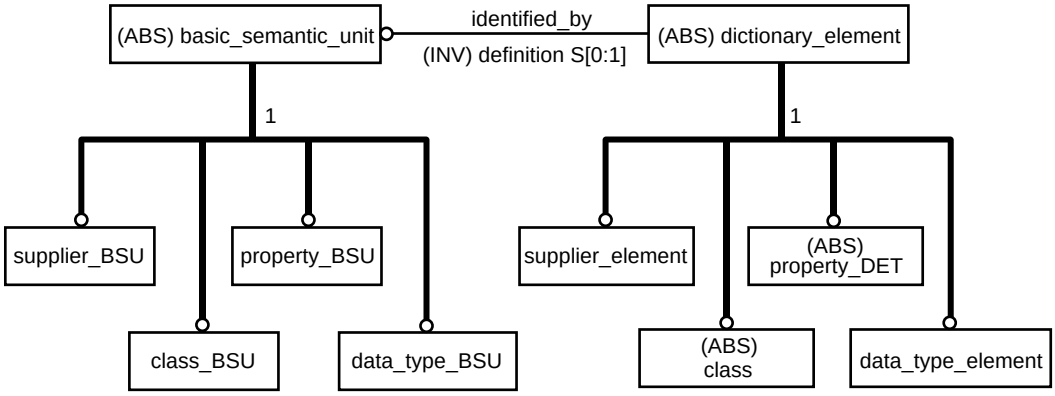


Figure 5 – Current BSUs and dictionary elements

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.

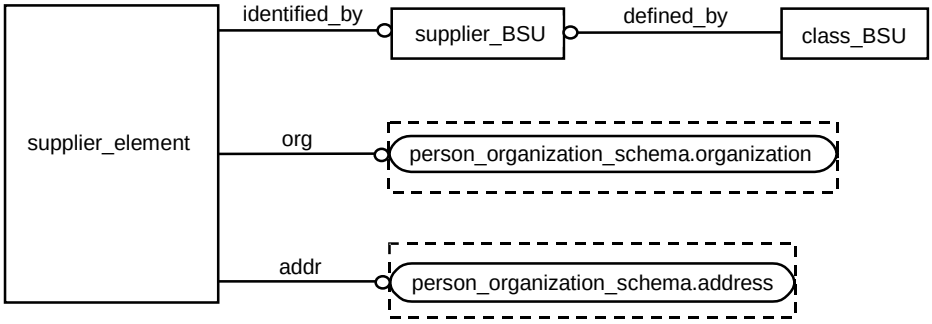


Figure 6 – Overview of supplier data and relationships

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.

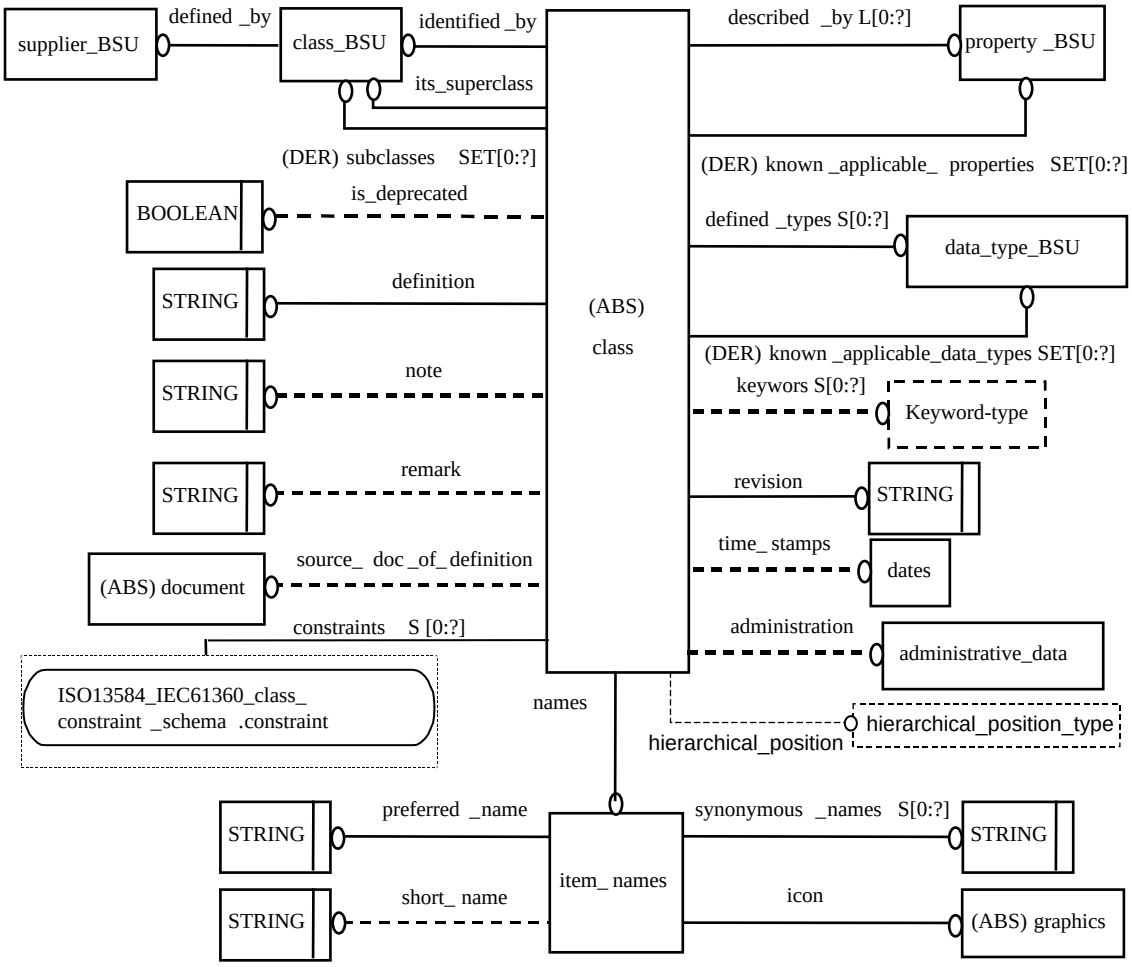


Figure 7 – Overview of class data and relationships

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.

EXPRESS specification:

```
*)
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
(*
```

Attribute definitions:

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.

Formal propositions:

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:

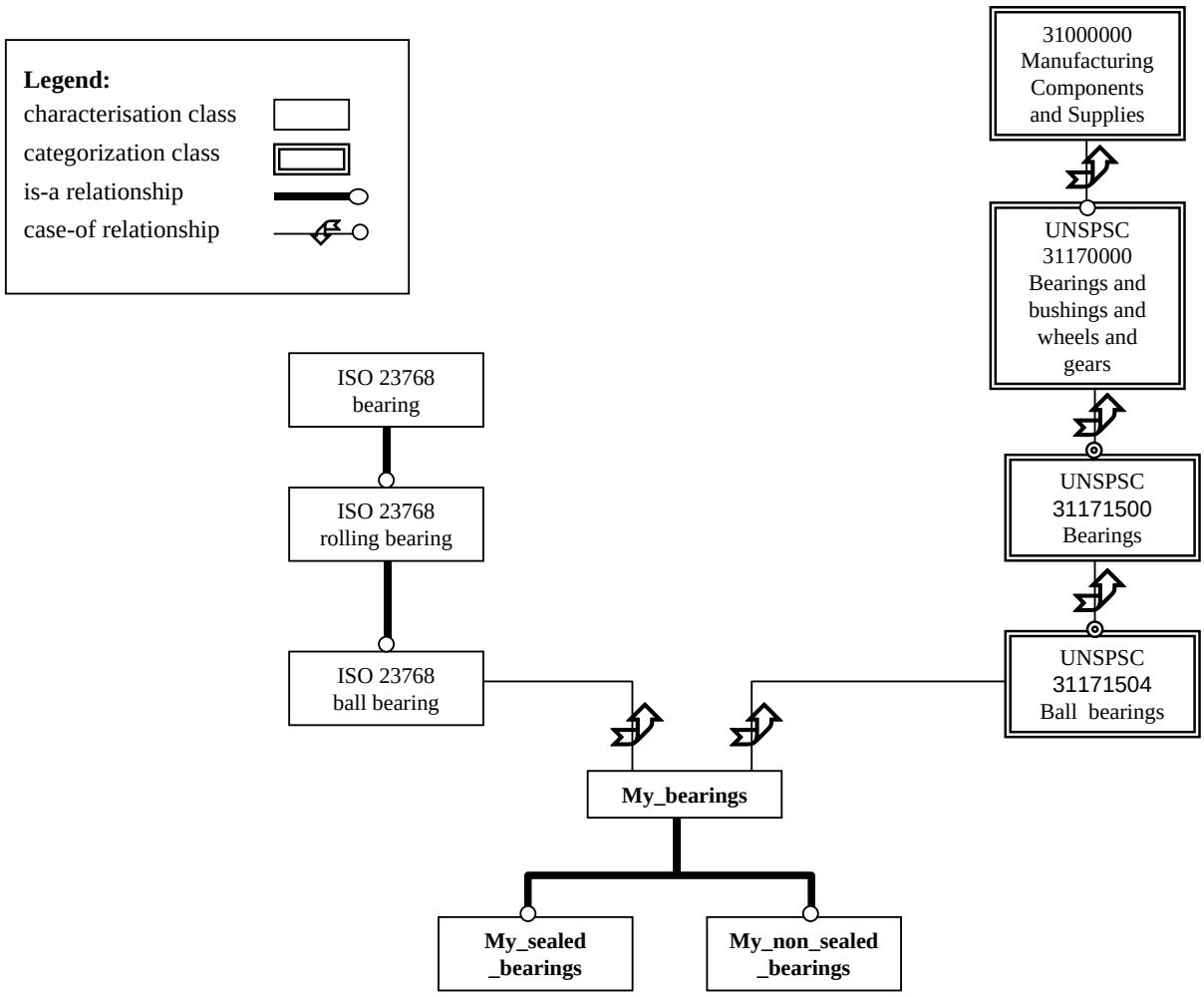


Figure 8 – Example of a supplier ontology

EXPRESS specification:

```
*)
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.

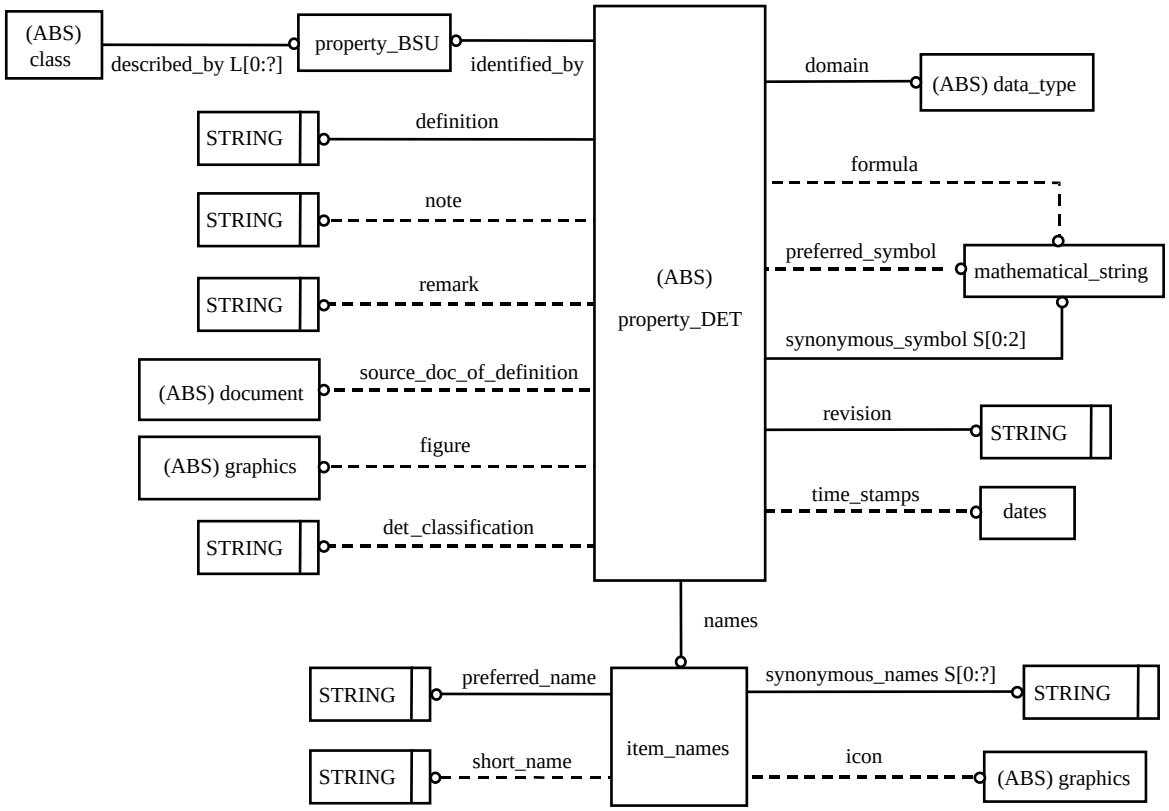


Figure 9 – Overview of property data element type data and relationships

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**).

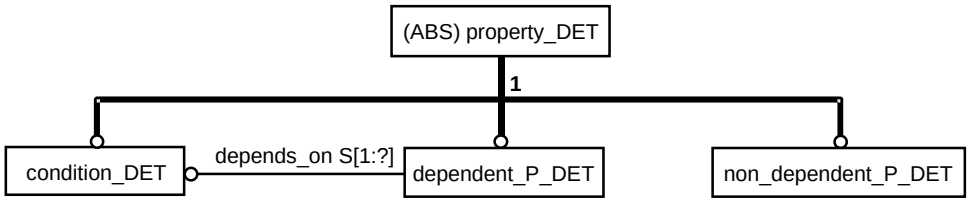


Figure 10 – Kinds of data element types

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**.

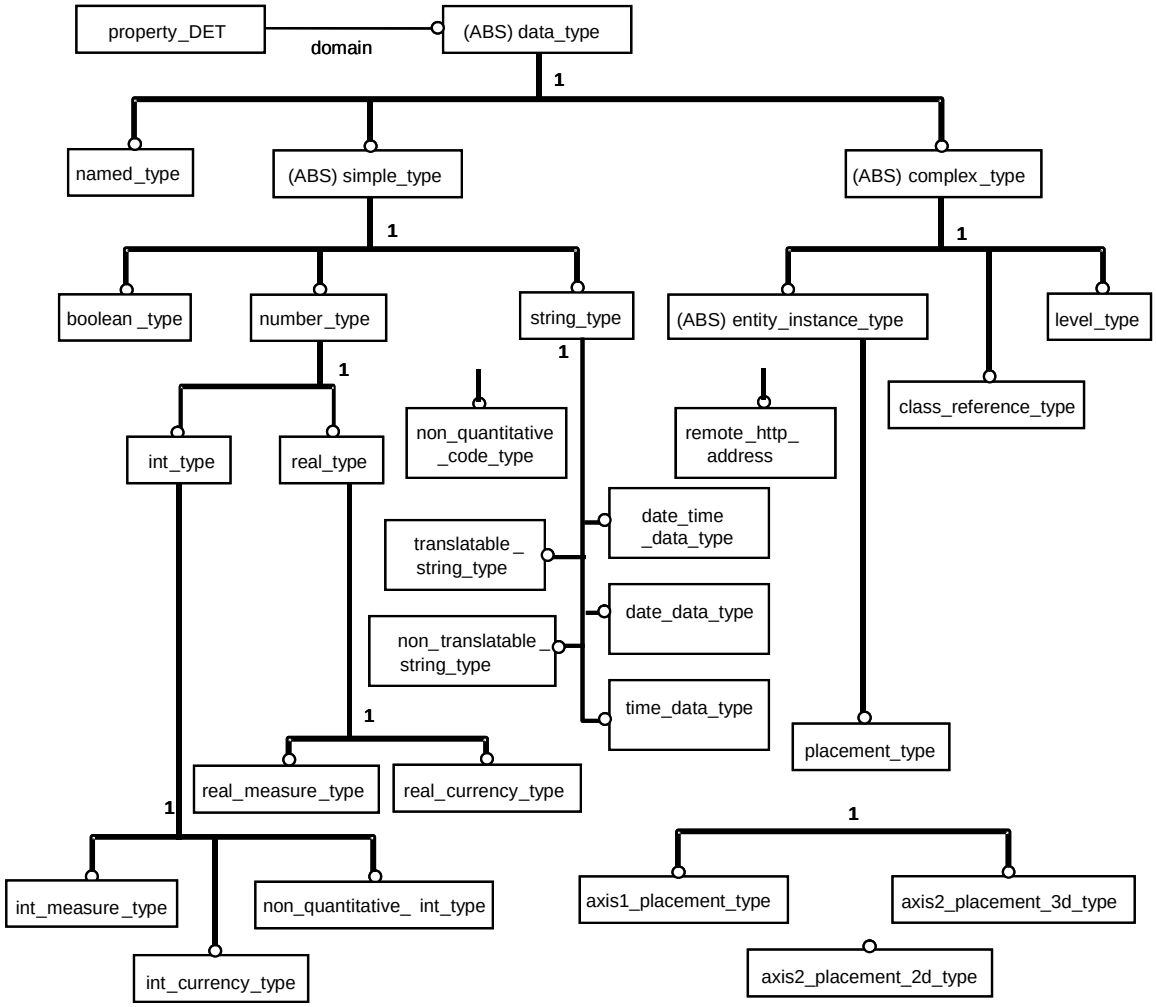


Figure 11 – Entity hierarchy for the type system

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_value**s.

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_value**s) 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.

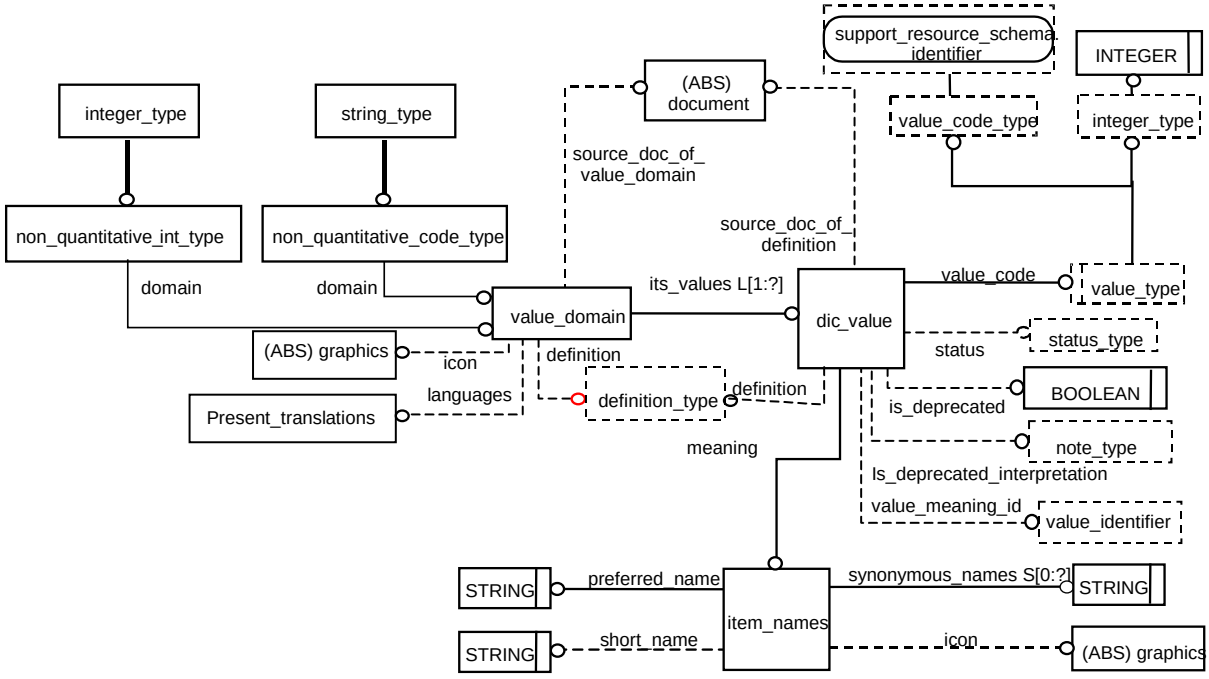


Figure 12 – Overview of non-quantitative data element types

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.

EXPRESS specification:

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

Formal propositions:

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".

EXPRESS specification:

```
*)
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.

EXPRESS specification:

```
*)
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).

EXPRESS specification:

```
*)
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 separated 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.

EXPRESS specification:

```
    *)
    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.

EXPRESS specification:

```
*)
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.

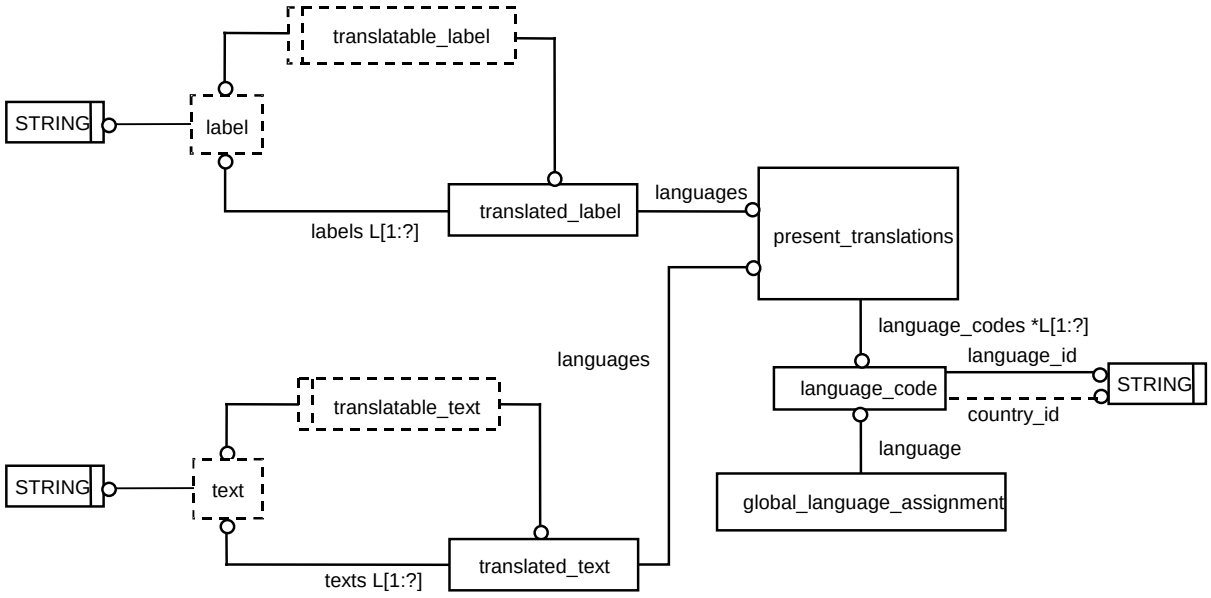


Figure 13 – ISO13584_IEC61360_language_resource_schema and support_resource_schema

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_BSU**s 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}.

EXPRESS specification:

```
*)
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
(*
```

Attribute definitions:

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**.

Formal propositions:

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.

EXPRESS specification:

```
*)
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**.

EXPRESS specification:

```
*)
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.

EXPRESS specification:

```
*)
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
    (*
```

Attribute definitions:

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**.

Formal propositions:

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**.

EXPRESS specification:

```
*)
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.

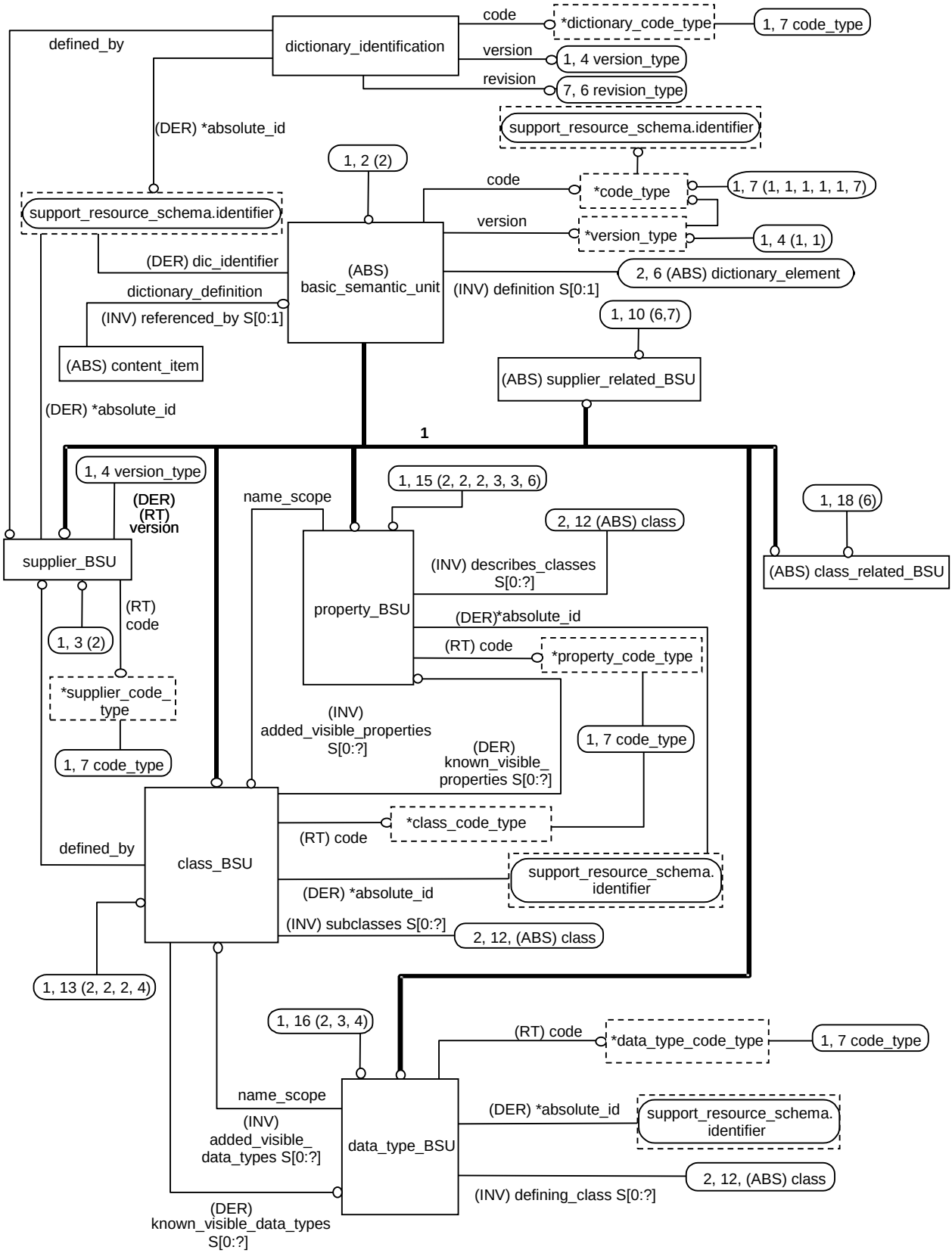


Figure B.1 – ISO13584_IEC61360_dictionary_schema – EXPRESS-G diagram 1 of 7

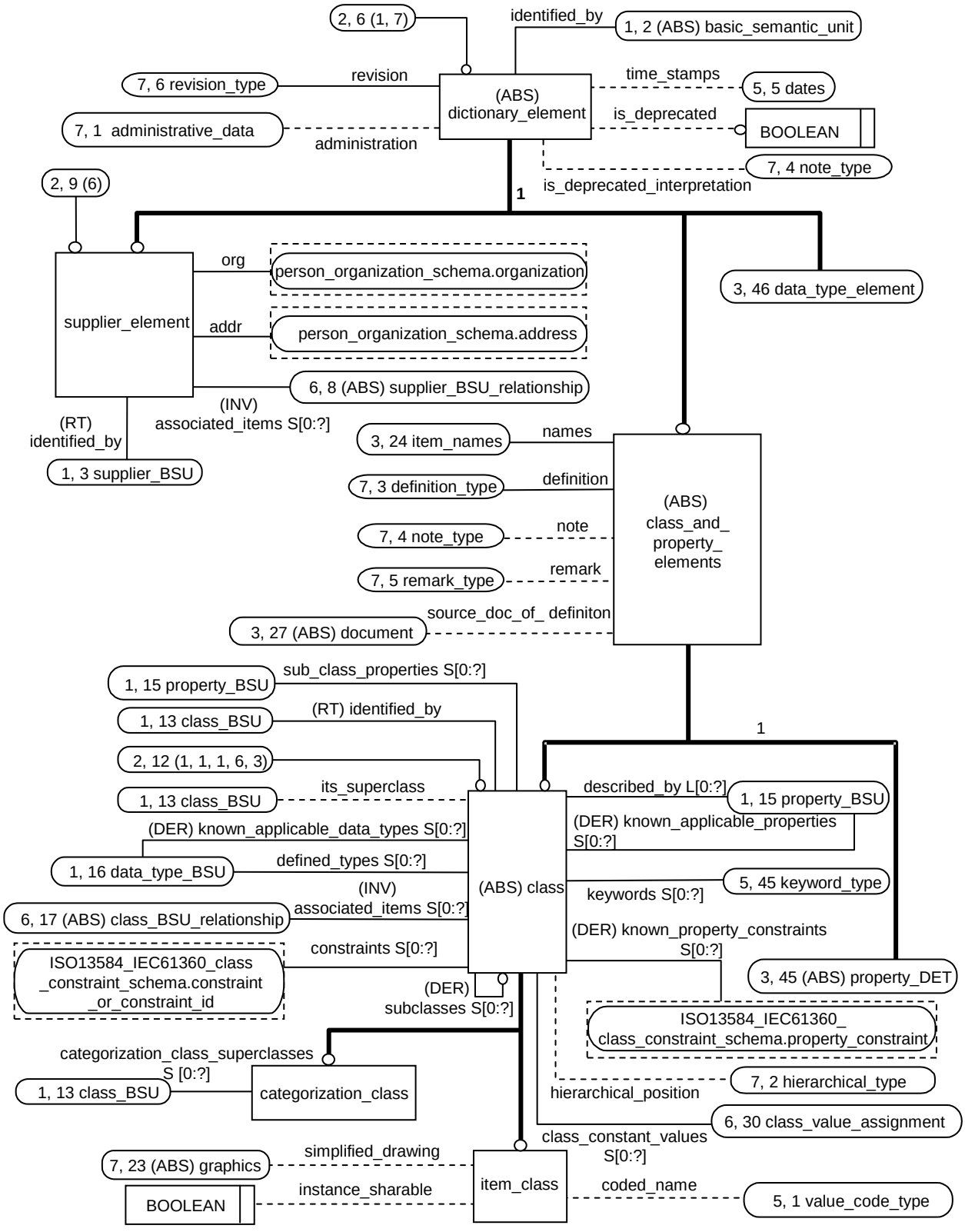


Figure B.2 – ISO13584_IEC61360_dictionary_schema – EXPRESS-G diagram 2 of 7

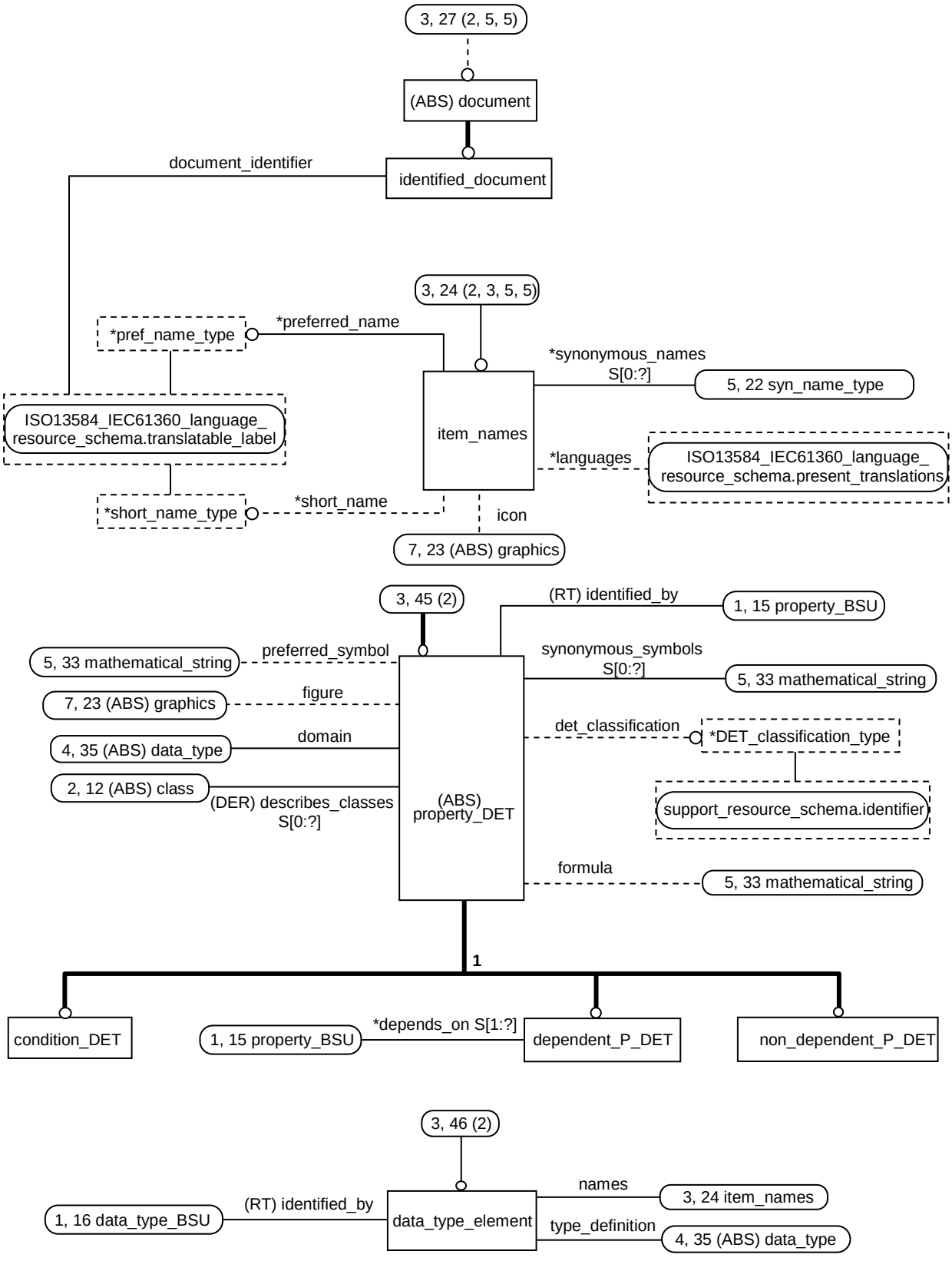


Figure B.3 – ISO13584_IEC61360_dictionary_schema – EXPRESS-G diagram 3 of 7

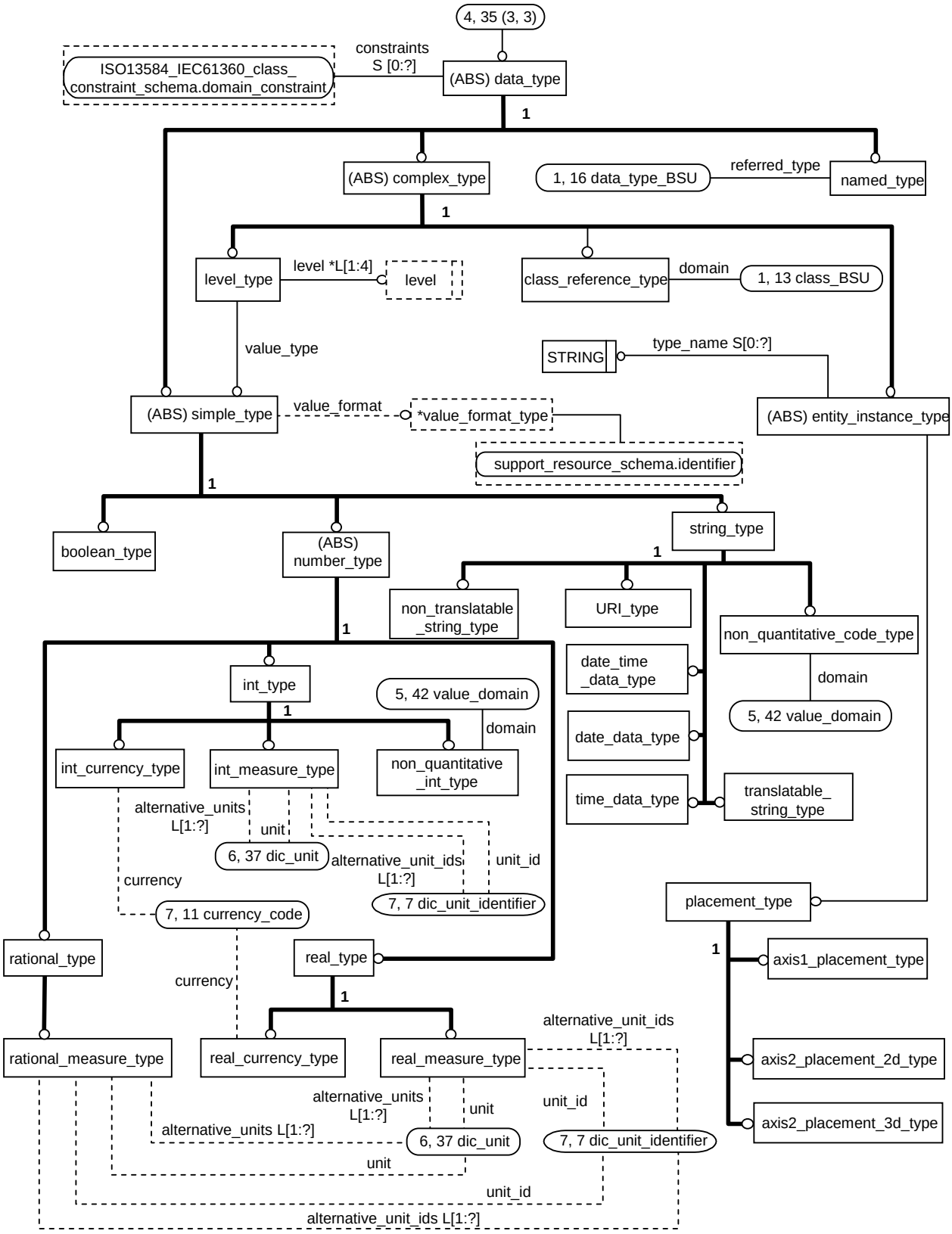


Figure B.4 – ISO13584_IEC61360_dictionary_schema EXPRESS-G diagram 4 of 7

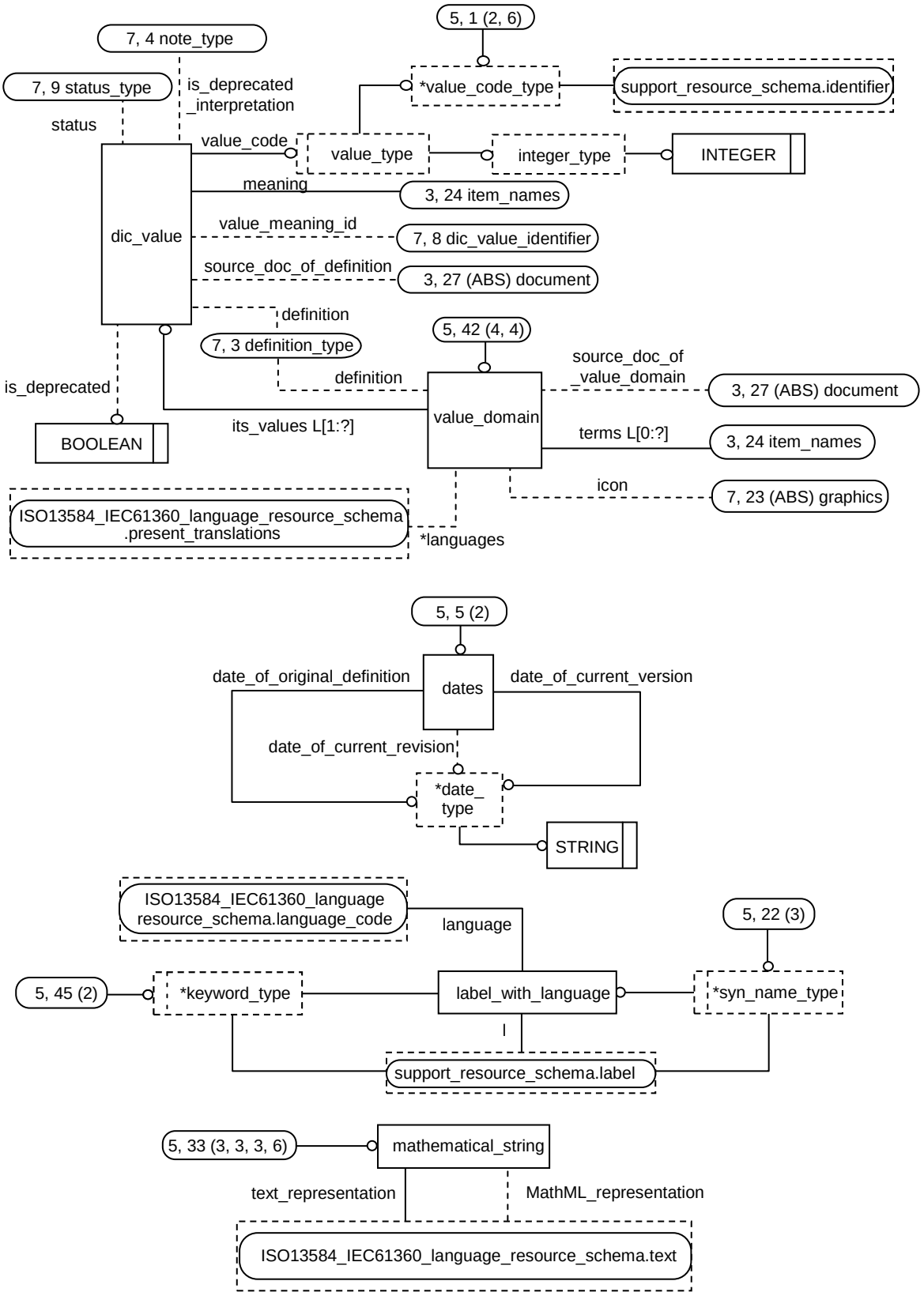


Figure B.5 – ISO13584_IEC61360_dictionary_schema – EXPRESS-G diagram 5 of 7

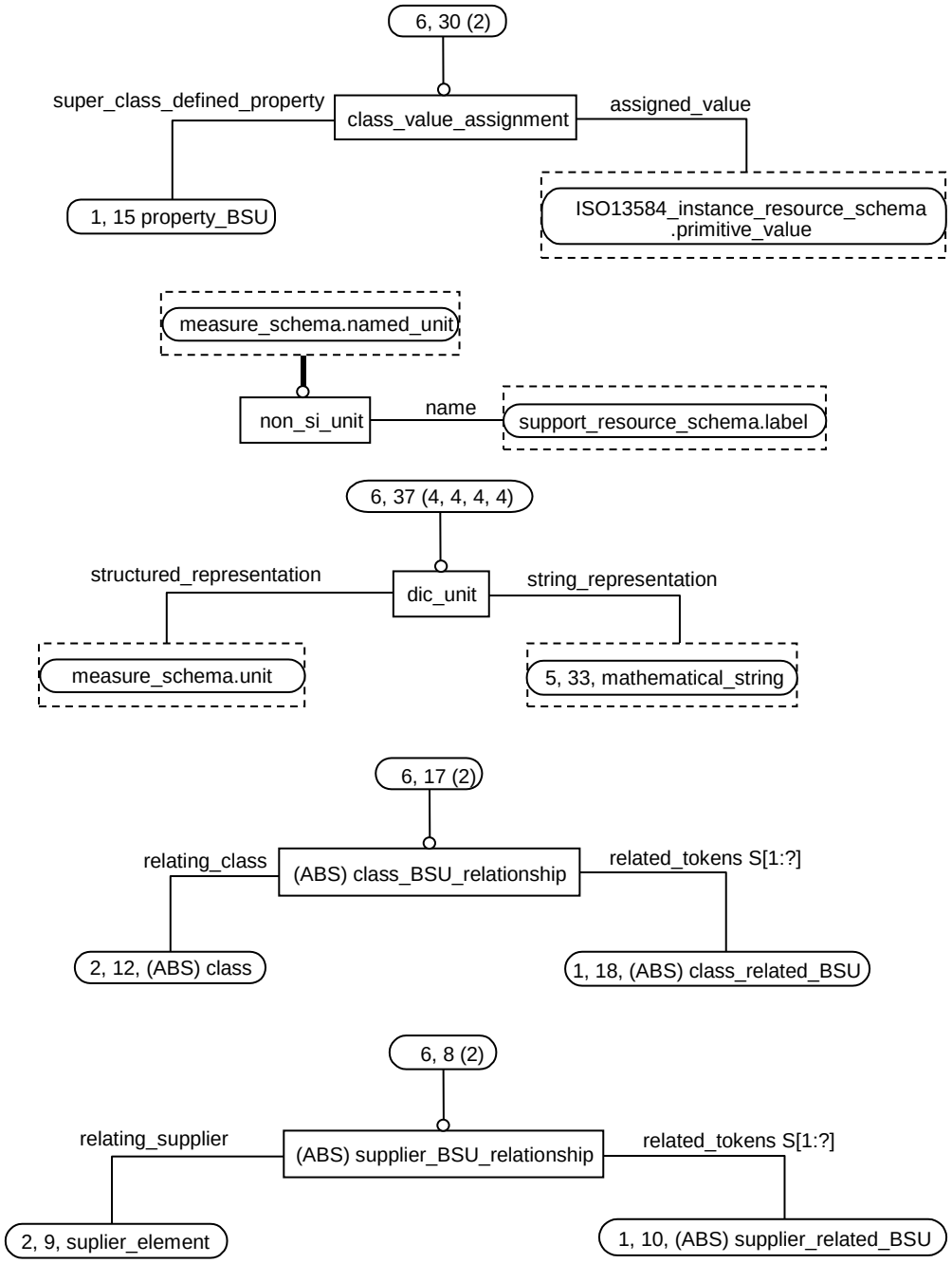


Figure B.6 – ISO13584_IEC61360_dictionary_schema – EXPRESS-G diagram 6 of 7

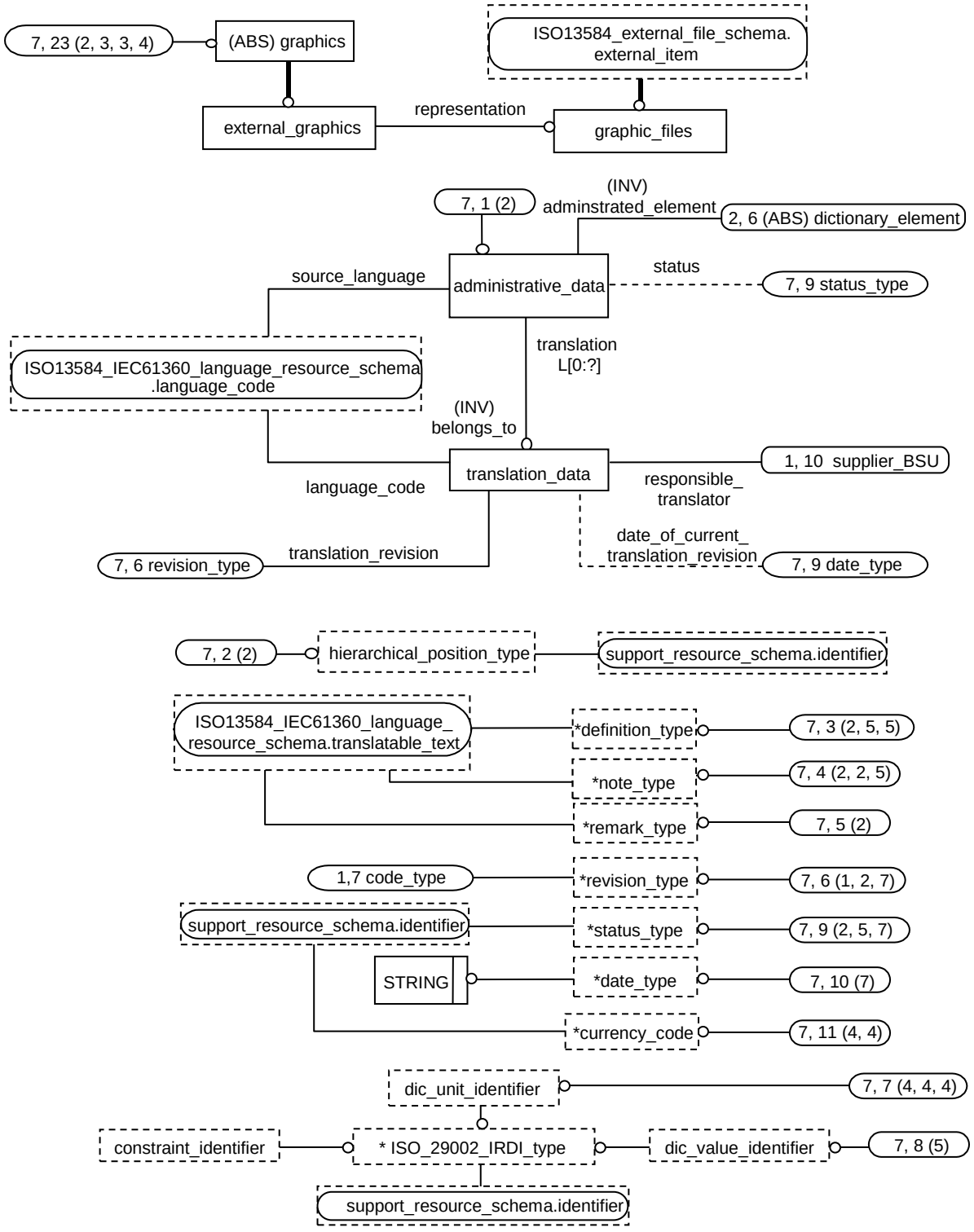


Figure B.7 – ISO13584_IEC61360_dictionary_schema – EXPRESS-G diagram 7 of 7

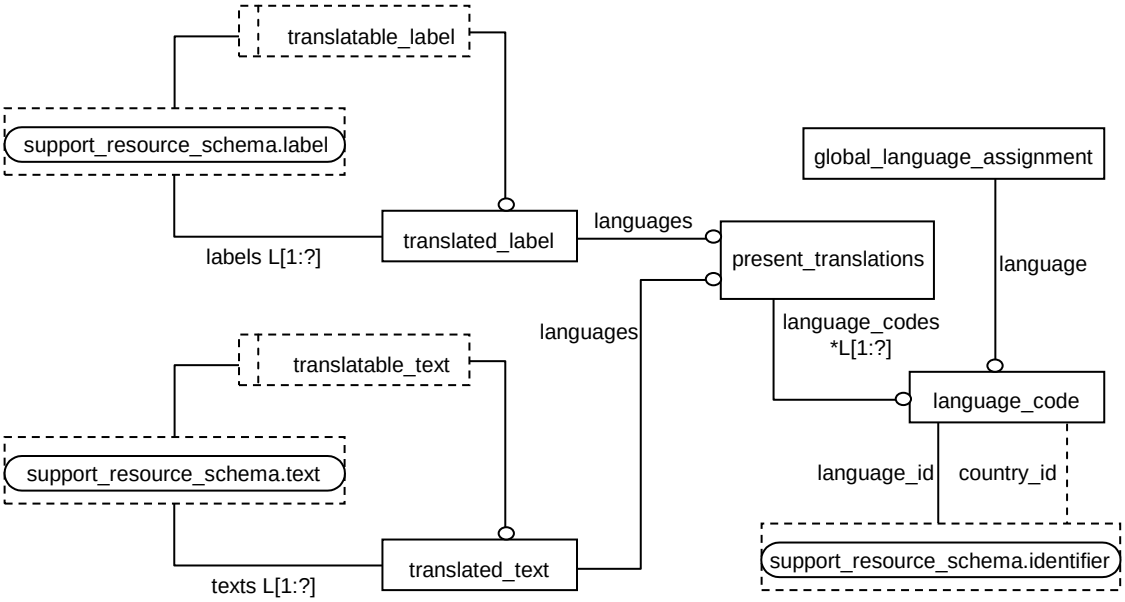


Figure B.8 – ISO13584_IEC61360_language_resource_schema – EXPRESS-G diagram 1 of 1

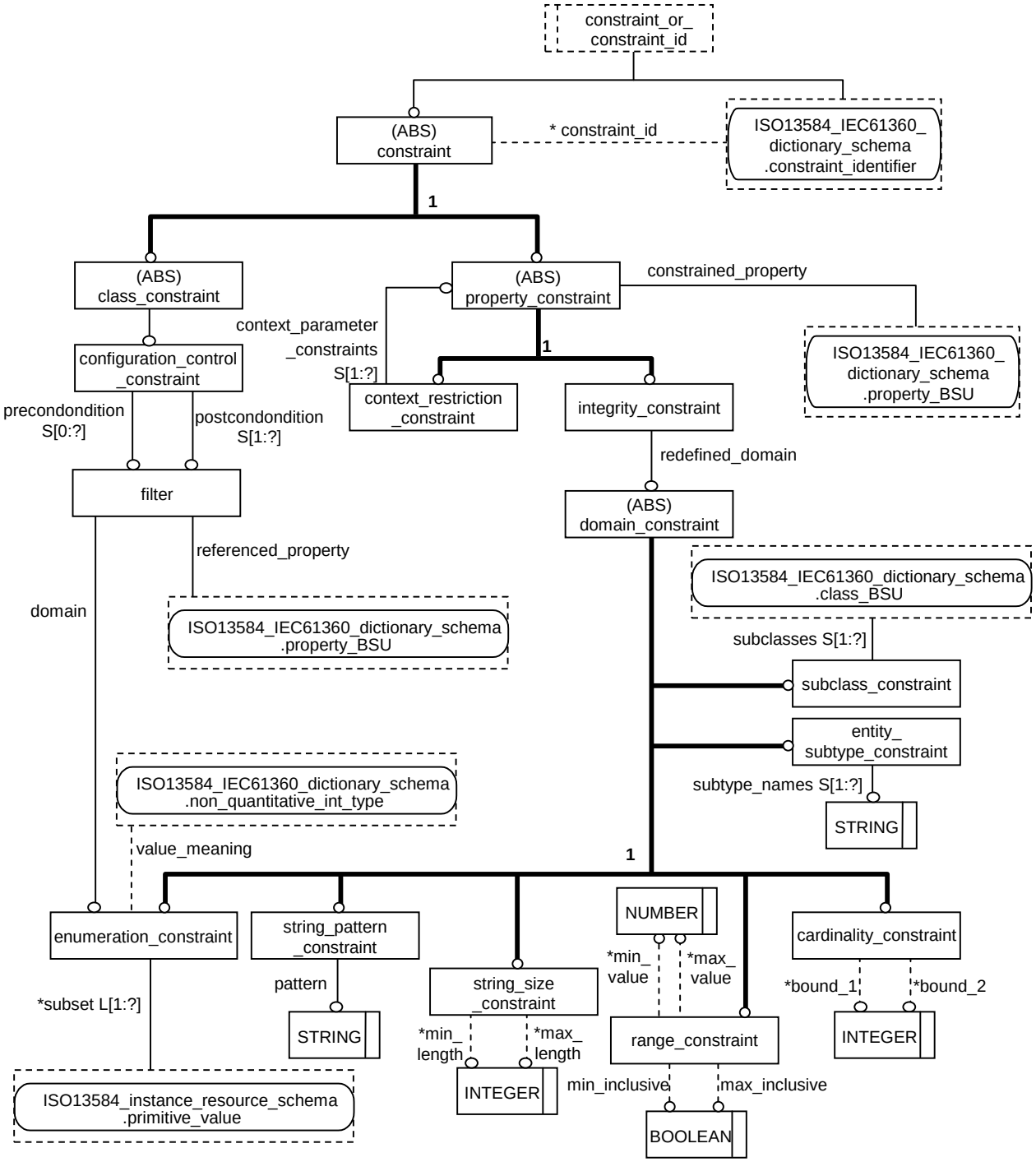


Figure B.9 – ISO13584_IEC61360_constraint_schema – EXPRESS-G diagram 1 of 1

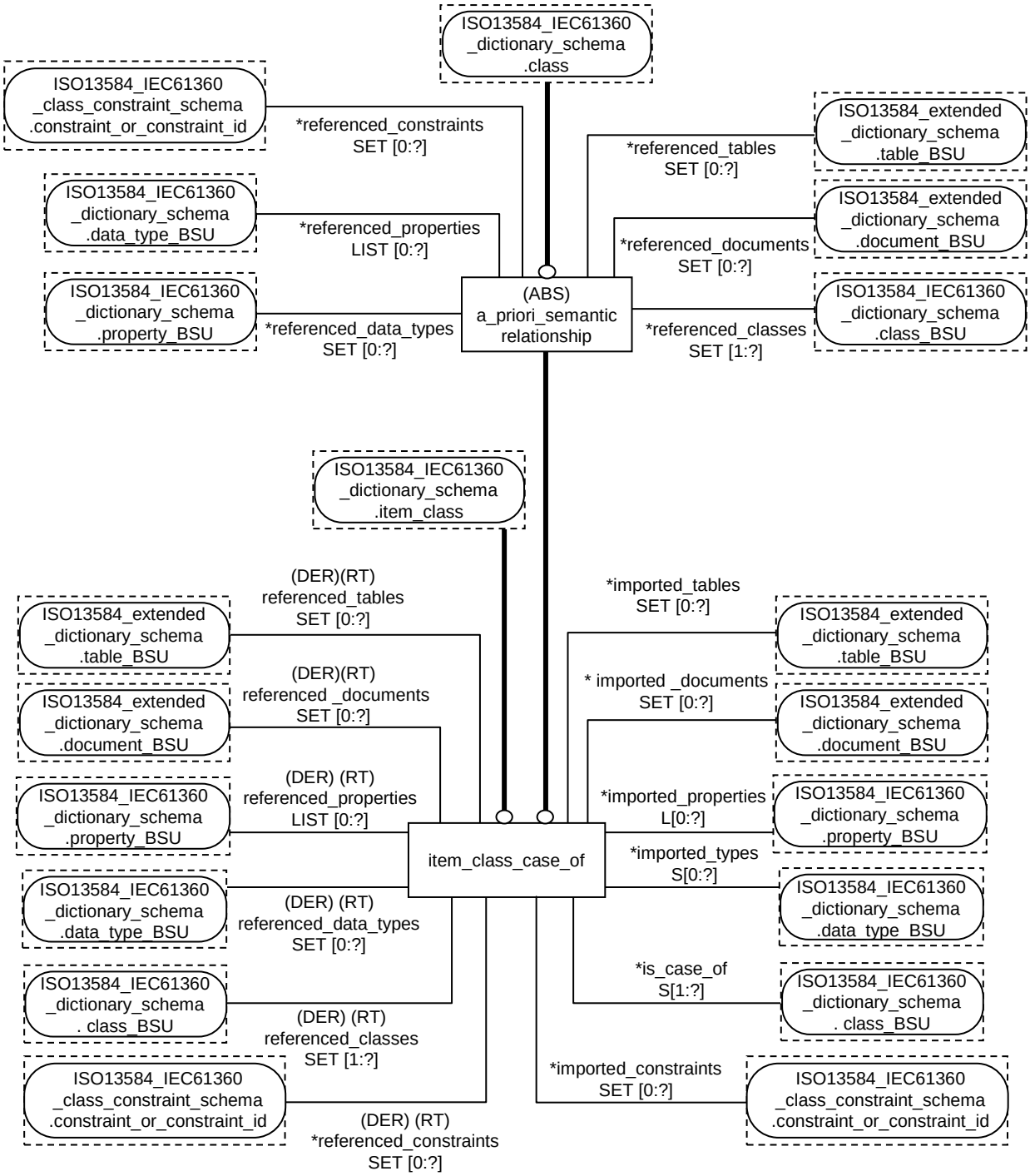


Figure B.10 – ISO13584_IEC61360_item_class_case_of_schema – EXPRESS-G diagram 1 of 1

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.

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

TYPES NORMALISÉS D'ÉLÉMENTS DE DONNÉES AVEC PLAN DE CLASSIFICATION POUR COMPOSANTS ÉLECTRIQUES –

Partie 2: Schéma d'un dictionnaire EXPRESS

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) La CEI elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de la CEI. La CEI n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à la CEI, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de la CEI, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de la CEI ou de toute autre Publication de la CEI, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de la CEI peuvent faire l'objet de droits de brevet. La CEI ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale CEI 61360-2 a été établie par le sous-comité 3D: Propriétés et classes des produits et leur identification, du comité d'études 3 de la CEI: Structures d'informations, documentation et symboles graphiques.

Cette troisième édition annule et remplace la deuxième édition parue en 2002 et son Amendement 1 (2003). Elle constitue une révision technique.

Cette édition inclut les modifications techniques majeures suivantes par rapport à l'édition précédente :

– séparation des concepts entre la classe de categorisation et la classe de caractérisation;
– introduction de contraintes de valeurs pour les classes et les propriétés;

– ajout de nouveaux sous-types différents pour les types de données, y compris l'entité rational_type ;

– amelioration de la représentation de l'unité de mesure.

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
3D/196/FDIS	3D/204/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Une liste de toutes les parties de la série CEI 61360, présentées sous le titre général *Types normalisés d'éléments de données avec plan de classification pour composants électriques,* peut être consultée sur le site web de la CEI.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de la CEI sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

INTRODUCTION

Le schéma commun de dictionnaire ISO/CEI présenté ici est basé sur l'intersection des domaines d'application des normes suivantes:

– CEI 61360-1;
– ISO 13584-42.

Les parties pertinentes des articles du domaine d'application de ces normes comprennent ce qui suit:

CEI 61360-1:2009

«La présente partie de la CEI 61360 donne une base solide pour la définition claire et non ambiguë des propriétés caractéristiques (types d'éléments de données) de tous les éléments des systèmes électrotechniques depuis les composants de base jusqu'aux sous-ensembles et aux systèmes complets. Bien qu'ils aient été conçus à l'origine dans l'optique de fournir une base pour l'échange d'information sur les composants électriques/électroniques, il est admis d'utiliser les principes et les méthodes contenus dans la présente norme dans des domaines autres que ceux de la conception d'origine comme les ensembles de composants et les systèmes et les sous-systèmes électrotechniques.»

ISO 13584-42:2010 (disponible en anglais seulement)

"Cette partie de l'ISO 13584 spécifie les principes à utiliser pour définir les classes de caractérisation des parties et les propriétés des parties qui caractérisent une partie indépendamment de toute identification particulière définie par un fournisseur.

Les règles et lignes directrices données dans cette partie de l'ISO 13584 sont obligatoires pour les comités de normalisation responsables de la création de caractérisation hiérarchisée normalisée.

L'utilisation de ces règles par les fournisseurs et utilisateurs est une méthodologie recommandée pour construire leur propre hiérarchie. "

Le SC 3D de la CEI et le TC184/SC4 de l'ISO ont convenu de NE PAS changer/et ou modifier le modèle EXPRESS proposé de manière indépendante l'un de l'autre afin de garantir l'harmonisation et la réutilisabilité du travail fourni par les deux comités. Par conséquent, il convient d'envoyer les demandes d'amendements aux deux comités. Il convient que ces demandes soient adoptées par les deux comités avant de modifier le modèle d'informations EXPRESS.

TYPES NORMALISÉS D'ÉLÉMENTS DE DONNÉES AVEC PLAN DE CLASSIFICATION POUR COMPOSANTS ÉLECTRIQUES –

Partie 2: Schéma d'un dictionnaire EXPRESS

1 Domaine d'application

La présente partie de la série CEI 61360 fournit un modèle formel pour les données conformément au domaine d'application donné dans la CEI 61360-1 et l'ISO 13584-42 et, donc, fournit un moyen pour la représentation et l'échange des données interprétables par un ordinateur.

L'intention est de fournir un modèle commun d'information pour le travail du SC 3D de la CEI et du TC184/SC4 de l'ISO, permettant ainsi la mise en œuvre de dictionnaires de systèmes traitant des données livrées conformément à l'une ou l'autre des normes établies par les deux comités.

Le domaine d'application de la présente partie de la CEI 61360 est le schéma de dictionnaire commun ISO/CEI basé sur l'intersection des deux domaines d'application des deux normes de base CEI 61360-1 et ISO 13584-42.

Le modèle EXPRESS proposé représente un modèle formel commun pour les deux normes et facilite leur harmonisation.

La CEI 61360-2 constitue le document maître. L'ISO 13584-42 contient une copie du modèle EXPRESS de la CEI 61360-2 dans une annexe informative

Dans un certain d'articles, où le modèle EXPRESS commun donne plus de liberté, la CI a défini un plus grand nombre de restrictions qui peuvent être consultées dans la partie méthodologie de la CEI 61360-1.

Deux schémas sont fournis dans la présente partie de la CEI 61360 définissant deux options qui peuvent être sélectionnées pour une mise en œuvre. Chacune de ces options est appelée "classe de conformité".

– L'ISO13584_IEC61360_dictionary_schema2 permet de prendre en charge la modélisation et l'échange de types d'éléments de données techniques avec le schéma de classification associé utilisé dans les définitions des types d'élément de données. Elle constitue la classe de conformité 1 de la présente partie de la CEI 61360.
– L'ISO13584_IEC61360_language_resource_schema fournit des ressources si les chaînes le permettent dans des langues diverses. Elle a été extraite du schéma de dictionnaire, car elle pourrait être utilisée dans d'autres schémas conceptuels. Elle repose largement sur le support_resource_schema issu de l'ISO 10303-41:2000 et peut être vue comme une extension à cela. Elle permet l'utilisation d'une langue spécifique dans tout un contexte d'échange (Fichier physique) sans la surcharge introduite lorsque plusieurs langues sont utilisées.

Lorsqu'il est utilisé avec l'ISO 10303-21, chaque schéma définit un seul format d'échange. Le format d'échange défini par la classe de conformité 1 est totalement compatible avec la série ISO 13584.

2 Références normatives

Les documents de référence suivants sont indispensables pour l'application du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

CEI 61360-1:2009, *Types normalisés d'éléments de données avec plan de classification pour composants électriques – Partie 1: Définitions – Principes et méthodes*

CEI 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* (disponible en anglais seulement)

ISO/CEI 8859-1:1998, *Techonologies de l'information – Jeux de caractères graphiques codés sur un seul octet – Partie 1 : Alphabet latin n°1*

ISO/CEI 10646-1, *Technologies de l'information – Jeu universel de caractères codés sur plusieurs octets (JUC) – Partie 1:Architecture et plan multilingue de base*

ISO/CEI 14977, *Technologies de l'information – Métalangage syntaxique – BNF étendu*

ISO 639 (toutes les parties), *Codes pour la représentation des noms de langue*

ISO 843:1997, *Information et documentation – Conversion des caractères grecs en caractères latins*

ISO 3166-1, *Codes pour la représentation des noms de pays et de leurs subdivisions – Partie 1: Codes de pays*

ISO 4217:2008, *Codes pour la représentation des monnaies et types de fonds*

ISO 8601:2004, *Éléments de données et formats d'échange – Échange d'information – Représentation de la date et de l'heure*

ISO 10303-11:2004, *Systèmes d'automatisation industrielle et intégration – Représentation et échange de données de produits – Partie 11: Méthodes de description: Manuel de référence du langage EXPRESS*

ISO 10303-21:2002, *Systèmes d'automatisation industrielle et intégration – Représentation et échange de données de produits – Partie 21: Méthodes de mise en application: Encodage en texte clair des fichiers d'échange*

ISO 10303-41:2000, *Systèmes d'automatisation industrielle et intégration – Représentation et échange de données de produits – Partie 41: Ressources génériques intégrées: Principes de description et de support de produits*[1]

ISO 13584-26, *Systèmes d'automatisation industrielle et intégration – Bibliothèque de composants – Partie 26: Ressource logique: Identification des fournisseurs d'information*

ISO 13584-42:2010, *Systèmes d'automatisation industrielle et intégration – Bibliothèque de composants – Partie 42: Méthodologie descriptive: Méthodologie appliquée à la structuration des familles de pièces*

---

1 Une nouvelle edition de l'ISO 10303-41 a été publiée en 2005.

3 Termes et définitions

Pour les besoins du présent document, les termes et définitions suivants s'appliquent.

3.1
classe abstraite
classe dont les membres sont également des membres d'une de ses sous-classes

Note 1 à l'article: Les classes abstraites sont utilisées lorsqu'il est nécessaire de regrouper différentes sortes d'objets en une classe d'une hiérarchie d'inclusion de classes.

Note 2 à l'article: Dans le modèle de dictionnaire commun ISO13584/CEI61360, il est possible de définir aussi bien des classes de catégorisation abstraites que des classes de caractérisation abstraites. Le fait qu'elle soit abstraite est seulement une caractéristique conceptuelle d'une classe. Cette caractéristique n'est pas représentée de manière explicite dans le modèle.

Note 3 à l'article: Par la relation d'héritage, une classe de caractérisation abstraite permet de partager, par exemple, certaines propriétés visibles entre différentes sous-classes qui correspondent à différentes sortes d'éléments.

3.2
propriété applicable d'une classe
propriété applicable possédée nécessairement par chaque partie qui est membre d'une classe de caractérisation

Note 1 à l'article: Chaque partie qui est membre d'une classe de caractérisation possède un aspect correspondant à chaque propriété applicable de cette classe de caractérisation.

Note 2 à l'article: La définition ci-dessus est conceptuelle, il n'y a pas d'exigence demandant que toutes les propriétés applicables d'une classe soient utilisées pour décrire chaque partie de cette classe au niveau du modèle de données.

Note 3 à l'article: Toutes les propriétés applicables d'une superclasse sont aussi des propriétés applicables pour les sous-classes de cette superclasse.

Note 4 à l'article: Seules les propriétés définies ou héritées comme étant des propriétés visibles et importées d'une classe peuvent être des propriétés applicables.

Note 5 à l'article: Pour faciliter l'intégration des bibliothèques de composants et des catalogues électroniques selon l'ISO 13584-24:2003 et l'ISO 13584-25, ces parties de l'ISO 13584 demandent que seules les propriétés qui sont applicables à une classe soient utilisées pour caractériser leurs instances dans les bibliothèques de composants et les catalogues électroniques.

3.3
attribut
élément de données interprétable par un ordinateur pour la description d'une propriété, d'une relation ou d'une classe

Note 1 à l'article: Un attribut décrit un seul détail d'une propriété, d'une classe ou d'une relation.

EXEMPLE Le nom d'une propriété, le code d'une classe, l'unité de mesure dans laquelle sont fournies les valeurs d'une propriété.

3.4
unité sémantique de base
entité qui fournit une identification absolue et universellement unique d'un certain objet du domaine d'application qui est représenté comme un élément du dictionnaire

EXEMPLE 1 Un dictionnaire conforme à la présente partie de la CEI 61360 permet la prise en charge de l'identification de classes, de propriétés, de sources d'informations et de types de données (datatypes).

EXEMPLE 2 Un dictionnaire conforme à l'ISO 13584-24:2003 permet la prise en charge de l'identification de classes, de propriétés, de sources d'informations, de types de données (datatypes), de tableaux, de documents et des bibliothèques de programmes.

EXEMPLE 3 Dans l'ISO 13584-511, la classe des boulons à tête hexagonale est identifiée par une BSU, la propriété "qualité de tolérance de filetage" est également identifiée par une BSU.

Note 1 à l'article: Le contenu d'une unité sémantique de base peut également être représenté par un IRDI.[2]

3.5
caractéristique d'un produit
caractéristique produit
propriété invariable, caractéristique d'un produit, dont la valeur est fixée une fois le produit défini

Note 1 à l'article: Changer la valeur d'une caractéristique d'un produit signifierait changer le produit.

EXEMPLE Pour un roulement à billes, le diamètre intérieur et le diamètre extérieur sont des caractéristiques de produit.

Note 2 à l'article: Adaptée de l'ISO 13584-24:2003, définition 3.12.

3.6
classe
abstraction d'un ensemble de produits semblables

Note 1 à l'article: Un produit qui se conforme à l'abstraction définie par une classe est appelé un "membre de classe".

Note 2 à l'article: Une classe est un concept intentionnel qui peut prendre de nombreuses significations différentes dans des contextes différents.

EXEMPLE L'ensemble des produits utilisés par une entreprise particulière et l'ensemble des produits normalisés par l'ISO sont deux exemples de contextes. Dans ces deux contextes (l'entreprise particulière et l'ISO), l'ensemble des produits qui sont considérés comme étant des membres de la classe *roulements avec une rangée de billes* peut être différent, en particulier parce que les employés de chaque entreprise ignorent l'existence d'un certain nombre de produits roulements avec une rangée de billes.

Note 3 à l'article: Les classes sont structurées par des relations d'inclusion de classes.

Note 4 à l'article: Une classe de produits est un concept général tel que défini dans l'ISO 1087-1. Ainsi, il est conseillé d'utiliser les règles définies dans l'ISO 704 pour définir les attributs de dénomination et de définition des classes de produits.

Note 5 à l'article: Dans le contexte de la série ISO 13584, une classe est soit une classe de caractérisation, associée à des propriétés et utilisable pour caractériser des produits, soit une classe de catégorisation, non associée à des propriétés et non utilisable pour caractériser des produits.

3.7
relation d'inclusion de classes
relation entre classes qui signifie l'inclusion des membres de classes: si A est une superclasse de A1, cela signifie que, dans n'importe quel contexte, tout membre de A1 est également membre de A

EXEMPLE 1 L'ensemble de produits utilisés par une entreprise particulière et l'ensemble de tous les produits normalisés par l'ISO sont deux exemples de contextes.

EXEMPLE 2 Dans n'importe quel contexte, la classe *condensateur* inclut la classe *condensateur électrolytique*.

Note 1 à l'article: L'inclusion de classes définit une structure hiérarchique entre les classes.

Note 2 à l'article: L'inclusion de classes est une relation conceptuelle qui ne prescrit rien au niveau de la représentation des données. En conséquence, elle ne prescrit aucun schéma particulier de base des données ou modèle particulier de données.

Note 3 à l'article: Dans le modèle défini dans la présente partie de la CEI 61360, la relation “is-a” (c'est-à-dire: "est un") assure l'inclusion de classes. La présente partie de la CEI 61360 recommande que la relation “case-of” assure aussi l'inclusion de classe.

Note 4 à l'article: La relation d'inclusion de classe s'appelle également "subsomption" (ou "subsumption").

---

2 IRDI = *International Registration Data Identifier*.

3.8
membre de classe
produit qui se conforme à l'abstraction définie par une classe

3.9
propriété de valeur d'une classe
propriété qui a une seule valeur pour toute une classe de caractérisation de produits

Note 1 à l'article: La valeur d'une propriété de valeur d'une classe n'est pas définie de façon individuelle pour chaque produit d'une caractérisation de classe, mais de façon globale pour la classe elle-même.

Note 2 à l'article: Lorsque tous les produits d'une classe de produits ont la même valeur pour une propriété particulière, le fait de définir la propriété comme étant une propriété de valeur d'une classe permet d'éviter la duplication de la valeur pour chaque instance.

Note 3 à l'article: Les propriétés de valeur d'une classe peuvent également être utilisées pour capturer une certaine identité entre différentes caractérisations de classes lorsque cette identité n'est pas capturée par la structure hiérarchique.

3.10
modèle de dictionnaire commun ISO13584/CEI61360
modèle de données pour ontologie de produits, utilisant le langage de modélisation d'informations EXPRESS, résultant d'un effort conjoint entre le TC 184/SC 4/WG 2 de l'ISO et le SC 3D de la CEI

Note 1 à l'article: Plusieurs niveaux de mises en œuvre autorisées, appelées classes de conformité, sont définis pour le modèle de dictionnaire commun ISO13584/CEI61360. La classe de conformité 1 est constituée des divers schémas documentés dans la présente partie de la CEI 61360 (qui dupliquent des informations contenue dans la présente norme), plus l'ISO13584_IEC61360_dictionary_aggregate_extension_schema documenté dans l'ISO 13584-25 (dupliqué dans la CEI 61360-5). D'autres classes de conformité sont documentées dans l'ISO 13584-25 (classes de conformité 2, 3 et 4).

Note 2 à l'article: Dans la série de normes ISO 13584, chaque ontologie particulière de produits traitant d'un domaine particulier de produits et basée sur le modèle de dictionnaire commun ISO13584/CEI61360 est appelée "dictionnaire de référence" pour le domaine en question.

3.11
caractéristique de produit dépendante du contexte
propriété d'un *produit* dont la valeur dépend de certains *paramètres de contexte*

Note 1 à l'article: Pour un produit donné, une caractéristique dépendant du contexte est définie mathématiquement comme étant une fonction dont le domaine est défini par un certain nombre de paramètres du contexte qui définissent l'environnement du produit.

EXEMPLE Pour un *roulement à billes*, la *durée de vie* est une caractéristique dépendant du contexte qui est fonction de la *charge radiale*, de la *charge axiale* et de la *vitesse de rotation*.

Note 2 à l'article: Adaptée de l'ISO 13584-24:2003, définition 3.22.

3.12
paramètre de contexte
variable dont la valeur caractérise le contexte dans lequel un *produit* est placé

EXEMPLE 1 La *charge dynamique* appliquée à un *palier* est un paramètre de contexte pour ce *palier*.

EXEMPLE 2 La *température ambiante* à laquelle la *valeur de résistance* d'une *résistance* est mesurée est un paramètre de contexte pour cette *résistance*.

Note 1 à l'article: Cette définition annule et remplace la définition donnée dans l'ISO 13584-24:2003, qui était la suivante: "variable dont la valeur caractérise le contexte dans lequel il est prévu de placer un *produit*".

Note 2 à l'article: Dans la série de normes ISO 13584, une valeur de propriété est représentée comme étant un type d'élément de données.

3.13
type d'élément de données
unité de données pour laquelle l'identification, la description et la représentation de la valeur ont été spécifiées

Note 1 à l'article: Dans la série de normes ISO 13584, une valeur de propriété est représentée comme étant un type d'élément de données.

3.14
données de dictionnaire
ensemble des données qui représente des ontologies de produits éventuellement associées à des catégorisations de produits

Note 1 à l'article: Il est conseillée d'échanger les données du dictionnaire en utilisant une certaine classe de conformité du modèle de dictionnaire commun ISO/CEI.

Note 2 à l'article: Cette définition de "données de dictionnaire" annule et remplace la définition antérieure issue de la première édition de la CEI 61360-2 qui était la suivante: “ensemble de données qui décrit des hiérarchies des classes de caractérisation des produits et des propriétés de ces produits”.

3.15
élément de dictionnaire
ensemble d'attributs qui constitue la description du dictionnaire de certains objets du domaine d'application

EXEMPLE 1 Un dictionnaire conforme à la présente partie de la CEI 61360 permet la prise en charge de la description des classes, des propriétés, des sources d'informations et des types de données (datatypes).

EXEMPLE 2 Un dictionnaire conforme à l'ISO 13584-24:2003 permet la prise en charge de la description des classes, des propriétés, des sources d'informations, des types de données (datatypes), des tableaux, des documents et des bibliothèques de programmes.

3.16
famille de produits
ensemble de produits représentés par la même caractérisation de classe

Note 1 à l'article: Cette définition annule et remplace la définition donnée dans l'ISO 13584-24:2003, qui était la suivante: “famille simple ou générique de pièces”.

3.17
caractéristique intrinsèque
aspect d'un produit qui peut être décrit par une classe de caractérisation et un ensemble de paires propriété-valeur

Note 1 à l'article: Dans le monde réel, une instance de caractéristique intrinsèque n'existe qu'intégrée au sein du produit dont elle est un aspect.

EXEMPLE 1 La tête d'une vis est une caractéristique intrinsèque décrite par une classe de têtes et un certain nombre de propriétés de la tête, qui dépend de la classe de têtes. Une tête de vis existe seulement lorsqu'elle appartient à une vis.

Note 2 à l'article: Les caractéristiques intrinsèques sont représentées au moyen d'**item_class** dont l'attribut **instance_sharable** est égal à *false*.

Note 3 à l'article: L'attribut **instance_sharable** permet de spécifier le statut conceptuel d'un article: soit un article autonome (**instance_sharable** =*true*), soit une caractéristique intrinsèque (**instance_sharable** =*false*). Cela n'implique aucune contrainte au niveau de la représentation des données. Dans le modèle de dictionnaire commun ISO13584/CEI61360, le fait de représenter plusieurs instances du monde réel qui partagent la même représentation EXPRESS par une seule entité EXPRESS ou par plusieurs entités EXPRESS est considéré comme étant dépendant de la mise en œuvre. Il n'existe aucun mécanisme pour spécifier si une valeur de données d'une instance de caractéristique intrinsèque peut ou peut ne pas être partagée.

EXEMPLE 2 La même instance d'une classe de têtes de vis peut être référencée par plusieurs instances d'une classe de vis. Cela signifie qu'il existe plusieurs têtes de vis, mais toutes les têtes de vis ont la classe de caractérisation et le même ensemble de valeurs de propriétés. L'attribut **instance_sharable** permet de spécifier que le fait de changer cette instance de la classe têtes de vis changerait plusieurs instances de la classe vis.

3.18
propriété importée
propriété définie dans une classe qui est sélectionnée par une autre classe du même dictionnaire de référence ou d'un dictionnaire de référence différent, au moyen de la relation "case-of", pour devenir applicable à cette dernière classe

Note 1 à l'article: Seules des propriétés qui sont visibles et/ou applicables dans une classe peuvent être importées de cette classe.

Note 2 à l'article: L'importation entre classes de dictionnaires de référence différentes permet de réutiliser des propriétés, définies par exemple dans un dictionnaire de référence normalisé, sans les redéfinir.

Note 3 à l'article: L'importation entre classes du même dictionnaire de référence reconnaît le fait que certains produits puissent accomplir plusieurs fonctions, ce qui exige la capacité d'importer une propriété de plusieurs classes de niveau supérieur.

Note 4 à l'article: Lorsqu'elle est importée dans une nouvelle classe, une propriété conserve son identificateur initial et, de ce fait, tous les attributs peuvent ne pas être dupliqués.

Note 5 à l'article: Une propriété importée est applicable pour la classe dans laquelle elle est importée.

3.19
information
fait, conception ou instruction

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

3.20
modèle d'information
modèle formel d'un ensemble borné de faits, de concepts ou d'instructions pour satisfaire à une exigence spécifiée

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

3.21
fournisseur d'information
fournisseur
organisation qui délivre une ontologie, c'est-à-dire un dictionnaire de données ou une bibliothèque fournisseur qui est responsable de son contenu

Note 1 à l'article: Cette définition de "fournisseur" annule et remplace la définition de "fournisseur d'information" issue de l'ISO 13584-1:2001 qui était la suivante: “organisation qui délivre une bibliothèque fournisseur dans le format normalisé défini la présente Norme Internationale et qui est responsable de son contenu".

3.22
identificateur international de données d'enregistrement
identificateur unique au niveau international pour un certain objet du domaine d'application tel que défini dans l'ISO/CEI 11179-5

Note 1 à l'article: Seuls les identificateurs internationaux de données d'enregistrement conformes à l'ISO/TS 29002-5 sont utilisés dans le contexte de la série de normes de l'ISO 13584.

Note 2 à l'article: Un identificateur international de données d'enregistrement peut être utilisé pour représenter le contexte d'une unité sémantique de base qui identifie un élément de dictionnaire comme étant une chaîne.

Note 3 à l'article: Un identificateur international de données d'enregistrement peut aussi être utilisé pour identifier le contenu d'un attribut d'un élément du dictionnaire.

EXEMPLE L'unité de mesure d'une propriété, d'une valeur d'une propriété ou d'une contrainte sur une propriété peut être identifiée par un identificateur international de données d'enregistrement (IRDI).[3]

3.23
relation "is-a"
relation d'inclusion de classes associée à l'héritage: si A1 *is-a* A, alors chaque produit appartenant à A1 appartient à A et tout ce qui est décrit dans ce contexte de A est automatiquement dupliqué dans le contexte de A1

Note 1 à l'article: Ce mécanisme est habituellement appelé "héritage".

---

3 IRDI = International Registration Data Identifier.

Note 2 à l'article: Dans le modèle de dictionnaire commun ISO13584/CEI61360, la relation is-a peut seulement être définie entre des classes de caractérisation. Il est conseillé qu'elle ne définisse qu'une seule hiérarchie et elle assure que les propriétés tant visibles qu'applicables sont héritées.

3.24
relation "is-case-of"
case-of
mécanisme d'importation de propriétés: si A1 is *case-of* A, alors la définition des produits de A couvre également les produits de A1 et, donc, A1 peut importer n'importe quelle propriété provenant de A

Note 1 à l'article: Le but de la relation *case-of* est de permettre de connecter ensemble plusieurs hiérarchies d'inclusion de classes tout en assurant que les hiérarchies référencées peuvent être mises à jour de façon indépendante.

Note 2 à l'article: Il n'existe pas de contrainte pour que la relation "case-of" soit destinée à définir des hiérarchies simples.

Note 3 à l'article: Dans le modèle de dictionnaire commun ISO13584/CEI61360, la relation "case-of" peut être utilisée en particulier dans quatre cas: (1) pour lier une classe de caractérisation à une classe de catégorisation, (2) pour importer, dans le contexte de certains dictionnaires de référence normalisés, un certain nombre de propriétés définies dans d'autres dictionnaires de référence normalisés, (3) pour relier un dictionnaire utilisateur de référence à un ou plusieurs dictionnaires de référence normalisés, (4) pour décrire un produit en utilisant les propriétés de différentes classes: lorsque des produits de classe A1 remplissent deux fonctions différentes et sont donc logiquement décrits par des propriétés associées à deux classes différentes, A et B, A1 peut être reliée par une relation "is-a", par exemple, à A et par une relation "case-of" à B.

Note 4 à l'article: Les constructions de ressources EXPRESS pour modéliser les relations "case-of" sont définies en 4.5 et après.

3.25
article
chose qui peut être caractérisée au moyen d'une classe de caractérisation à laquelle elle appartient et d'un ensemble de paires propriété-valeur

Note 1 à l'article: Cette définition remplace la définition donnée dans l'ISO 13584-24:2003, qui était la suivante: “un objet dont la description peut être capturée par une structure de classe et un ensemble de propriétés”.

Note 2 à l'article: Dans la série de normes ISO 13584, des produits et des caractéristiques intrinsèques de produits qui correspondent à des propriétés composées sont des articles.

3.26
classe de caractérisation feuille
classe de caractérisation qui n'est pas encore spécialisée en classes de caractérisation plus précises

EXEMPLE Vis à tête fraisée plate à empreinte cruciforme (type Y) et vis à tête cylindrique à six pans creux avec filetage métrique à pas fin sont des classes de caractérisation feuilles définies dans l'ISO 13584-511.

3.27
classe de caractérisation non feuille
classe de caractérisation qui est encore spécialisée en classes de caractérisation plus précises

EXEMPLE *Composant à filetage externe* et *boulon/vis à filetage métrique* sont des classes de caractérisation non feuilles définies dans l'ISO 13584-511.

3.28
type d'élément de données non quantitatif
type d'élément de données qui identifie ou décrit un objet au moyen de codes, d'abréviations, de noms, de références ou de descriptions

3.29
composant
élement matériel ou fonctionnel conçu pour constituer un composant de différents produits

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

3.30
bibliothèque de composants
ontologie de produits interprétable par un ordinateur et description interprétable par un ordinateur d'un ensemble de produits au moyen des références à cette ontologie

Note 1 à l'article: Cette définition remplace la définition donnée dans la première édition de la présente partie de la CEI 61360, qui était la suivante: “ensemble identifié de données et éventuellement de programmes qui peuvent générer des informations relatives à un ensemble de composants”.

3.31
produit
chose ou substance produite par un processus naturel ou artificiel

Note 1 à l'article: Dans la présente partie de la CEI 61360, le terme "produit" est pris dans son sens le plus large afin d'inclure les dispositifs, les systèmes et les installations ainsi que les matériels, les processus, les logiciels et les services.

3.32
catégorisation de produits
catégorisation de composants
catégorisation
partitionnement récursif d'un ensemble de produits en sous-ensembles dans un but spécifique

Note 1 à l'article: Les sous-ensembles qui apparaissent dans une catégorisation de produit sont appelés "classes de catégorisation de produits" ou "catégories de produits".

Note 2 à l'article: Une catégorisation de produits n'est pas une ontologie de produits. Elle ne peut pas être utilisée pour caractériser des produits.

Note 3 à l'article: Aucune propriété n'est associée à des catégorisations.

Note 4 à l'article: Plusieurs catégorisations du même ensemble de produits sont possibles en fonction de leur usage visé.

EXEMPLE La classification UNSPSC (United Nations Standard Products and Service Code, définie par l'Organisation des Nations Unies) est un exemple de catégorisation de produits qui a été développée pour l'analyse des dépenses.

Note 5 à l'article: En utilisant la relation *is-case-of*, plusieurs hiérarchies de classes de caractérisation de produits peuvent être reliées à une hiérarchie de catégorisation pour créer une seule structure.

3.33
classe de catégorisation de produits
classe de catégorisation de composants
classe de catégorisation
classe de produits qui constitue un élément d'une catégorisation

EXEMPLE *Composants de fabrication et Fournitures*, et *Optiques industrielles* sont des exemples d'une classe catégorisation de produits définie dans l'UNSPSC.

Note 1 à l'article: Aucune règle n'est donnée dans la présente partie de la CEI 61360 quant à la manière de sélectionner des classes de catégorisation. Ce concept est introduit (1) pour clarifier sa différence avec "classe de caractérisation", et (2) pour expliquer que la même classe de caractérisation peut être reliée à un nombre quelconque de classes de catégorisation.

Note 2 à l'article: Il n'y a aucune propriété associée à une classe de catégorisation.

3.34
caractérisation de produits
caractérisation de composants
description d'un produit au moyen d'une classe de caractérisation de produits à laquelle il appartient et d'un ensemble de paires propriété-valeur

EXEMPLE Hexagon_head_bolts_ISO_4014 (Qualités de produits = A, thread_type=M, longueur= 50, Diamètre = 8) est un exemple de caractérisation de produits.

3.35
classe de caractérisation de produits
classe de caractérisation de composants
classe de caractérisation
classe de produits qui remplissent la même fonction et qui partagent des propriétés communes

Note 1 à l'article: Des classes de caractérisation de produits peuvent être définies à divers niveaux de détails, définissant ainsi une hiérarchie d'inclusion de classes.

EXEMPLE *Boulon/vis à filetage métrique* et *boulon six pans* sont des exemples de classes de caractérisation de produits définies dans l'ISO 13584-511. La première classe de caractérisation est incluse dans la seconde. *Transistor* et *transistor de puissance bipolaire* sont des exemples de classes de caractérisation de produits définies dans la CEI 61360-DB. La seconde est incluse dans la première.

3.36
ontologie de produits
ontologie de composants
ontologie
modèle de connaissances des produits, réalisé par une représentation formelle et consensuelle des concepts d'un domaine de produits en termes de classes de caractérisation identifiée, de relations de classes et de propriétés identifiées

Note 1 à l'article: Les ontologies de produits reposent sur un modèle classe-instance qui permet de reconnaître et de dénommer les ensembles de produits, appelés "classes de caractérisation", qui ont une fonction semblable (par exemple, *roulement à billes*, *condensateur*), mais aussi de discerner au sein d'une classe les divers sous-ensembles de produits, appelés "instances", qui sont considérés comme étant identiques. Il est conseillé d'utiliser les règles définies dans l'ISO 1087-1 pour formuler la dénomination et les définitions des classes de caractérisation. Les instances n'ont pas de définition. Elles sont désignées par la classe à laquelle elles appartiennent et un ensemble de paires propriété-valeur.

Note 2 à l'article: Les ontologies ne sont pas concernées par des mots, mais par des concepts indépendants de tout langage particulier.

Note 3 à l'article: “Consensuel” signifie que la conceptualisation est obtenue à la suite d’un accord commun dans une certaine communauté.

Note 4 à l'article: “Formel” signifie que l'ontologie est censée être interprétable par une machine. Un certain niveau de raisonnement machine est logiquement possible sur l'ontologie, par exemple vérification de cohérence, en utilisant des déductions.

Note 5 à l'article: “Identifié” signifie que chaque classe de caractérisation d'ontologie et des propriétés sont associées à un identificateur unique au niveau global permettant de référencer ce concept à partir de n'importe quel contexte.

Note 6 à l'article: Le modèle de données pour ontologie recommandé dans la présente partie de la CEI 61360 est le modèle de dictionnaire commun ISO13584/CEI61360, dont la version la plus simple est documentée dans la présente partie de la CEI 61360. Des versions plus complètes sont documentées dans l'ISO 13584-25 et la CEI 61360-5 (classes de conformité 1, 2, 3 et 4 de ces deux documents).

Note 7 à l'article: Dans la présente partie de la CEI 61360, chaque ontologie de produits traitant d'un domaine particulier de produits conforme au modèle de dictionnaire commun ISO13584/CEI61360 est appelée "dictionnaire de référence" pour le domaine en question.

EXEMPLE Le dictionnaire de référence pour composants électriques, qui est défini dans la CEI 61360-DB, est une ontologie de produits pour composants électriques conforme au modèle de dictionnaire commun ISO13584/CEI61360. Il est obtenu à la suite d’un commun accord entre tous les organismes membres du SC 3D de la CEI. Un dictionnaire de référence d'entreprise est obtenu à la suite d'un accord commun entre des experts désignés par la direction, et ce, au nom de la société.

3.37
propriété
paramètre défini adapté pour la description et la différenciation de produits

Note 1 à l'article: Une propriété décrit un seul aspect d'un objet donné.

Note 2 à l'article: Une propriété est définie par la totalité de ses attributs associés. Les types et nombre d'attributs qui décrivent une propriété avec une exactitude élevée sont documentés dans la présente partie de la CEI 61360.

Note 3 à l'article: La présente partie de la CEI 61360 a identifié trois sortes différentes de propriétés: caractéristiques de produits, paramètres de contexte et caractéristiques de produits dépendantes du contexte.

Note 4 à l'article: Cette définition de "propriété" remplace la définition antérieure issue de la première édition de la présente partie de la CEI 61360 qui était la suivante: “information pouvant être représentée par un type d'élément de données”.

Note 5 à l'article: Dans la série de normes ISO 13584, une valeur de propriété est représentée comme étant un type d'élément de données.

3.38
type de données de propriété
ensemble autorisé de valeurs d'une propriété

3.39
classe de définition de propriété
classe de caractérisation de produits dans le contexte duquel une propriété de produit est définie

Note 1 à l'article: Dans le modèle de dictionnaire commun ISO13584/CEI61360, chaque propriété de produit a une seule classe de définitions de propriété qui définit son domaine d'application. La propriété a un sens seulement pour cette classe, et toutes ses sous-classes, et elle est dite *visible* sur ce domaine.

EXEMPLE Dans l'ISO 13584-511, *wrenching height* («hauteur de prise de clé») possède *nut* («écrou») comme sa classe de définition de propriétés et *major diameter of external thread* («grand diamètre du filetage externe») possède *metric external thread* («filetage externe métrique») comme sa classe de définition de produits.

3.40
type d'élément de données quantitatif
type d'élément de données doté d'une valeur numérique représentant une grandeur physique, une somme d'informations ou un total d'objets

3.41
dictionnaire de référence
ontologie de produits conforme au modèle de dictionnaire commun ISO 13584/CEI 61360

Note 1 à l'article: Dans la série de normes ISO 13584, une ontologie de produits qui traite d'un domaine particulier de produits, basée sur le modèle de dictionnaire commun ISO13584/CEI61360, est appelée "dictionnaire de référence" pour le domaine en question.

3.42
construction de ressource
collection d'entités, types, fonctions, règles et références du langage EXPRESS, définissant ensemble une description valide de données

Note 1 à l'article: Cette définition est adaptée de la définition de "construction de ressource" donnée dans l'ISO 10303-1:1994, à savoir “collection d'entités, types, fonctions, règles et références du langage EXPRESS, définissant ensemble une description valide de données de produit”.

[SOURCE: ISO 13584-1:2001, définition 3.1.21, modifiée]

3.43
sous-classe
classe inférieure d’un niveau à une autre classe dans une hiérarchie d'inclusion de classes

Note 1 à l'article: Dans le modèle de dictionnaire commun ISO13584/CEI61360, les hiérarchies d'inclusion de classes sont définies par la relation *is-a*. Elles peuvent également être établies par les relations *case-of*.

3.44
superclasse
classe supérieure d’un niveau à une autre classe dans une hiérarchie d'inclusion de classes

Note 1 à l'article: Dans le modèle de dictionnaire commun ISO13584/CEI61360, les hiérarchies d'inclusion de classes sont définies par la relation *is-a*. Elles peuvent également être établies par les relations *case-of*.

Note 2 à l'article: Dans le modèle de dictionnaire commun ISO13584/CEI61360, une classe à tout au plus une superclasse spécifiée au moyen d'une relation *is-a*.

3.45
bibliothèque fournisseur
bibliothèque de composants dont le fournisseur d'informations est différent de l'utilisateur de la bibliothèque

Note 1 à l'article: Cette définition remplace la définition donnée dans l'ISO 13584-1:2001, qui était la suivante: “ensemble de données et éventuellement de programmes, pour lesquels le fournisseur est défini et décrivant dans le format normalisé défini dans la présente Norme internationale un ensemble de produits et/ou un ensemble de représentations de produits”.

3.46
propriété visible
propriété qui a une définition ayant un sens dans le domaine d'application d'une classe de caractérisation donnée, mais qui ne s'applique pas nécessairement aux divers produits appartenant à cette classe

Note 1 à l'article: "Ayant un sens dans le domaine d'application d'une classe de caractérisation donnée" signifie qu'un observateur humain est à même de déterminer, pour tout produit de la classe de caractérisation, si la propriété s'applique ou pas, et, si elle s'applique, à quel aspect de produit elle correspond.

Note 2 à l'article: Le concept de propriété visible permet de partager la définition d'une propriété entre des classes de caractérisation de produits dans lesquelles cette propriété ne s'applique pas nécessairement.

EXEMPLE La propriété *longueur non filetée* a un sens pour toute classe de *vis*, mais elle ne s'applique qu'aux vis qui ont une partie non filetée. Elle peut être définie comme étant visible au niveau de la *vis*, tout en devenant applicable seulement dans certaines sous-classes.

Note 3 à l'article: Toutes les propriétés visibles d'une superclasse qui est une classe de caractérisation de produits sont également des propriétés visibles pour ses sous-classes.

Note 4 à l'article: Pour faciliter l'intégration des bibliothèques de composants et des catalogues électroniques selon l'ISO 13584-24:2003 et l'ISO 13584-25, ces parties de l'ISO 13584 demandent que seules les propriétés qui sont applicables à une classe soient utilisées pour caractériser leurs instances dans les bibliothèques de composants et les catalogues électroniques.

Note 5 à l'article: Cette définition d'une "propriété visible" annule et remplace la définition antérieure issue de l'ISO 13584-24:2003 qui était la suivante: “propriété définie pour une certaine famille de produits et qui ne s'applique pas nécessairement aux différents produits de cette famille de produits”.

4 Vue d'ensemble du schéma de dictionnaire commun et de la compatibilité avec l'ISO13584_IEC61360_dictionary_schema

4.1 Généralités

Dans les paragraphes suivants, il sera présenté l'architecture du schéma de dictionnaire commun et il sera expliqué comment le même modèle d'informations doit être utilisé dans les Normes internationales pour assurer leur compatibilité.

Le schéma de dictionnaire commun combine les exigences de la série CEI 61360 et de la série ISO 13584. Par conséquent, il contient des ressources pour prendre en charge les exigences spécifiques des deux séries de Normes internationales. Ces ressources sont fournies soit comme capacités facultatives, soit comme sous-types des types définis pour satisfaire aux exigences communes.

4.2 Utilisation du schéma de dictionnaire commun pour échanger des données conformes à la CEI 61360-1

Le dictionnaire commun pour échanger des données conformes à la CEI 61360-1 doit être utilisé comme suit:

a) les extensions spécifiques à la série ISO 13584 pour prendre en charge la capacité multilingue ne sont pas requises pour l'échange d'éléments de dictionnaire définis

conformément à la CEI 61360-1. Cependant, ces extensions qui sont present_translations, translated_label et translated_text doivent être utilisées dans la structure de l'échange, et ce, pour des raisons de compatibilité;

b) si une classe de composants a une superclasse, le **coded_name** doit être défini comme étant un **value_code** dans le **domaine** du type d'élément des données de classification de la superclasse;

c) si un type d'élément de données de classification existe dans une classe spécifique de composants, pour chaque **value** dans son **domain** une sous-classe et un **term** doivent être définis;

d) un type d'élément de données de classification, facultatif dans la classe de conformité 2 dans le schéma de dictionnaire commun, doit toujours être fourni pour les classes des composants définies conformément à la CEI 61360-1;

e) seules les unités SI doivent être utilisées, même si le schéma de dictionnaire commun permet l'utilisation de plusieurs types de systèmes d'unités. Cependant, lors de l'utilisation de ce schéma pour l'échange des données conformes à la série CEI 61360, seules les unités SI doivent être utilisées pour les types d'élément de données quantitatif.

4.3 Compatibilité avec l'ISO 13584-42

Une mise en œuvre conforme à la présente partie de la CEI 61360 doit prendre en charge toutes les entités, tous les types et toutes les contraintes associées qui appartiennent à la classe de conformité qu'elle prétend prendre en charge. Par conséquent, la conformité à la classe de conformité 1 de la présente partie de la CEI 61360 exige que toutes les entités, tous les types et toutes les contraintes associées définis dans le schéma de dictionnaire commun soient pris en charge. Les données ISO 13584 conformes au schéma de dictionnaire commun peuvent donc être traitées par une mise en œuvre conforme à la CEI 61360 qui se conforme à la classe de conformité 1 qui inclut toutes les caractéristiques intrinsèques de la classe de conformité 1. Dans l'ISO 13584, une classe de conformité 3 spécifique est destinée à contenir toutes les entités, tous les types et toutes les contraintes associées définis dans le schéma de dictionnaire commun. Une mise en œuvre conforme à l'ISO 13584 conforme à cette classe de conformité doit donc être capable de prendre en charge des données CEI qui appartiennent à la classe de conformité 1 de la présente partie de la CEI 61360.

4.4 Correspondance de dénomination entre la CEI 61360-1 et la CEI 61360-2

En raison des restrictions d'applications spécifiques (par exemple: le langage EXPRESS interdit les espaces dans les noms d'entités), un certain nombre de noms 'EXPRESS similaires sont créés en remplaçant le blanc dans un nom par un trait de soulignement (par exemple, "preferred name" est présenté comme étant "preferred_name").

En d'autres endroits, il est utilisé dans le modèle EXPRESS des noms qui s'écartent de ceux utilisés dans la CEI 61360-1. C'est la conséquence de l'effort visant à atteindre un seul modèle d'informations EXPRESS commun accompagné de Libraires de composants.

Le tableau ci-dessous présente une aide pour la mise en correspondance des noms utilisés dans les deux parties de la CEI 61360.

Tableau 1 – Table de références croisées

dénomination dans la CEI 61360-2	dénomination dans la CEI 61360-1
component_class	Component class (Classe de composants)
condition_DET	Condition data element type (Type d'élément de données de condition)
dependent_P_DET	Data element type (Type d'élément de données)
det_classification	Data element type class (Classe d'un type d'élément de données)

dénomination dans la CEI 61360-2	dénomination dans la CEI 61360-1
(DER)dic_identifier	Identifier (Identificateur)
dic_value	Value (Valeur)
material_class	Material class (Classe de matériaux)
meaning (signification)	Value meaning (Signification de la valeur)
non_dependent_P_DET	Data element type (Type d'élément de données)
preferred_symbol	Preferred letter symbol (Symbole littéral préférentiel)
revision (révision)	Revision number (Numéro de révision)
source_doc_of_definition	Source document of data element type definition (Document source de définition de type d'élément de données)
source_doc_of_definition	Source document of component class definition (Document source de définition de classe de composants)
synonymous_symbols	Synonymous letter symbol (Symbole littéral synonyme)
unit (unité)	Unit of measure (Unité de mesure)
value_code	Value code (Code de la valeur)
version	Version number (Numéro de version)

4.5 Structure principale du schéma de dictionnaire commun

Le présent paragraphe explique les principales constructions des ressources fournies par le schéma de dictionnaire commun:

– **dictionary_element** est n'importe quel élément défini dans le dictionnaire;
– **supplier_element** capture les données des fournisseurs des éléments du dictionnaire (classes, propriétés, types de données); la classe modélise l'élément du dictionnaire classes (familles) qui sont décrites par des propriétés;
– **property_DET** est l'élément de dictionnaire d'une propriété;
– **data_type** spécifie le type d'une propriété.

Ces composants du schéma de dictionnaire sont présentés plus en détail à l'Article 5: **ISO13584_IEC61360_dictionary_schema**.

Dans la présentation du schéma de dictionnaire commun, un certain nombre de diagrammes de vue d'ensemble sont fournis suivant le modèle de planification (voir Figure 1 à Figure 11). Ces modèles de planification utilisent la notation graphique EXPRESS-G pour le langage EXPRESS. Pour la clarté des diagrammes, certaines des relations qui sont définies dans le modèle EXPRESS sont omises. La Figure 1 ci-dessous présente sous forme de modèle de planification la structure principale du schéma du dictionnaire commun. La plupart de ces figures contiennent des modèles de vue d'ensemble (ou modèles de planification), mais montrent uniquement le niveau de détail qui est approprié à un certain endroit.

Pour la clarté des diagrammes, certaines des relations qui sont définies dans le modèle EXPRESS sont omises. La Figure F.1 ci-dessus présente, sous la forme d'un modèle de planification, les grandes lignes d'une structure principale du modèle de dictionnaire commun ISO13584/CEI61360.

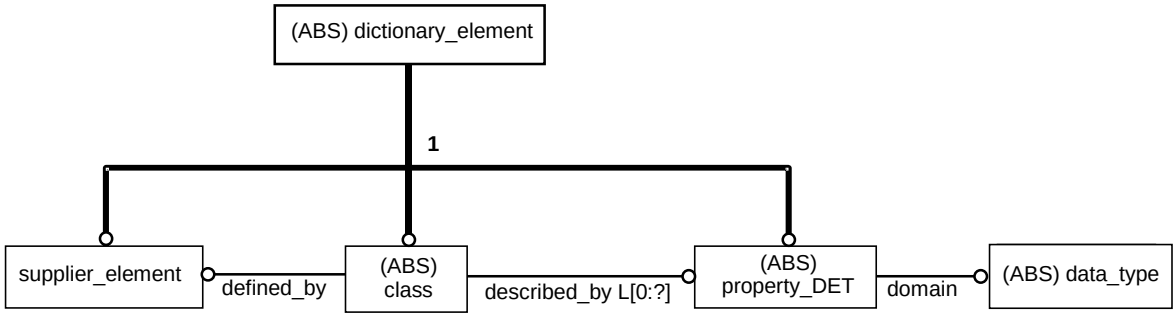


Figure 1 – Vue d'ensemble du schéma de dictionnaire

5 ISO13584_IEC61360_dictionary_schema

5.1 Généralités

Le présent article, qui constitue la partie principale du Modèle d'information commun de l'ISO 13584-42 et de la série CEI 61360, contient l'énumération EXPRESS complète du schéma de dictionnaire, annotée avec des commentaires et du texte explicatif. L'ordre du texte dans le présent article est déterminé en premier lieu par l'ordre imposé par le langage EXPRESS et en second lieu, par l'importance.

5.2 Schéma de dictionnaire

En premier lieu, le schéma a besoin d'être déclaré.

EXPRESS specification:

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

5.3 Références à d'autres schémas conceptuels

Le présent paragraphe contient des références à d'autres schémas conceptuels EXPRESS qui sont utilisés dans le schéma du dictionnaire. Leur source est indiquée dans le commentaire respectif.

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 Les schémas conceptuels ci-dessus référencés peuvent être consultés dans les documents suivants:

support_resource_schema	ISO 10303-41
person_organization_schema	ISO 10303-41
measure_schema	ISO 10303-41
ISO13584_IEC61360_language_resource_schema	CEI 61360-2
(qui est dupliqué pour commodité dans ce document)	
ISO13584_IEC61360_class_constraint_schema	CEI 61360-2
(qui est dupliqué pour commodité dans ce document)	
ISO13584_IEC61360_item_class_case_of_schema	CEI 61360-2
(qui est dupliqué pour commodité dans ce document)	
ISO13584_external_file_schema	ISO 13584-24:2003

5.4 Définitions de constantes

Le présent paragraphe contient des définitions de constantes utilisées dans les définitions de types (voir 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 d'un dictionnaire

Une entité **dictionary_identification** permet d'identifier sans ambiguïté une version particulière d'un dictionnaire particulier d'un fournisseur particulier d'informations, normalisée ou non. Elle contient un **code** défini par le fournisseur du dictionnaire qui identifie le

dictionnaire, un numéro de **version** et un numéro de **revision** qui caractérisent un état particulier de ce dictionnaire.

NOTE 1 La condition pour laquelle il convient d'incrémenter la version et la révision du dictionnaire est définie dans la CEI 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
(*
```

Définitions des attributs:

code: le code qui caractérise le dictionnaire.

version: le numéro de version qui caractérise la version du dictionnaire.

revision: le numéro de révision qui caractérise la révision du dictionnaire.

defined_by: le fournisseur qui définit le dictionnaire.

absolute_id: l'identification unique du dictionnaire.

Propositions formelles:

UR1: l'identificateur de dictionnaire défini par l'attribut **absolute_id** est unique.

Propositions informelles:

IP1: lorsqu'un dictionnaire est défini par un document de norme qui contient un seul dictionnaire, le **code** du dictionnaire doit être le numéro normalisé du document qui décrit ce dictionnaire si ce document définit seulement un seul dictionnaire. Il doit être le nom défini pour le dictionnaire pertinent dans le document qui le décrit si ce document définit plusieurs dictionnaires. Sauf spécification contraire, la version doit être mise à 1 et les numéros de révision doivent être mis à 0 pour les dictionnaires définis par des documents de normes.

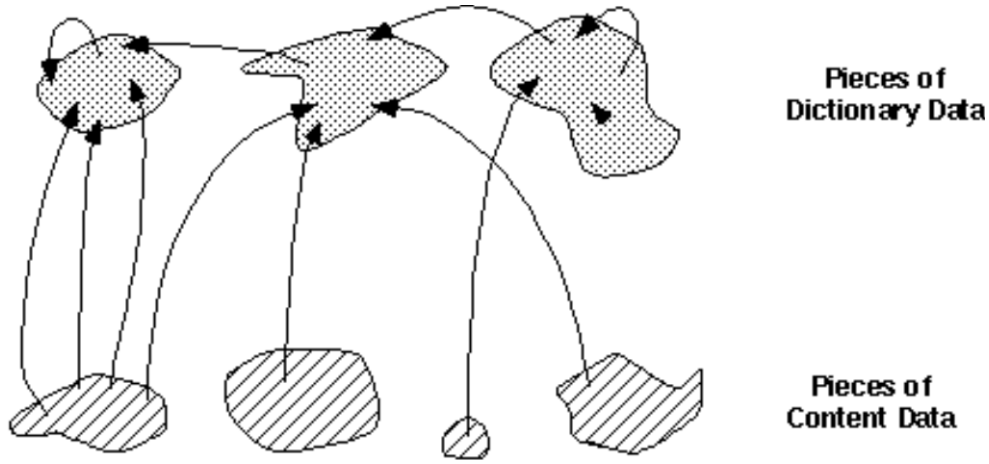
NOTE 2 La représentation des numéros de normes pour les documents de normes est spécifiée en 5.1 et 5.2 de l'ISO 13584-26:2000.

5.6 Basic Semantic Units (Unités sémantiques de base): définition et utilisation du dictionnaire

5.6.1 Exigences pour l'échange

Dans l'échange des données de dictionnaire et de bibliothèque de composants, il est de coutume de diviser les données. Par exemple, un dictionnaire pourrait être mis à jour avec certaines classes qui spécifient leur superclasse par référence à une classe préexistante, ou lorsque le contenu d'une bibliothèque est échangé, des éléments de dictionnaire sont seulement référencés et pas inclus chaque fois. Il doit être possible de se référer sans ambiguïté et de manière cohérente aux données du dictionnaire.

Ainsi, une exigence claire est, en premier lieu, d'être capable d'échanger des éléments de données et, en second lieu, d'avoir des relations entre ces éléments. Ceci est montré à la Figure 2.



Légende

Anglais	Français
Pieces of Dictionary Data	Élément de données du dictionnaire
Pieces of Content Data	Élément de données du contenu

Figure 2 – Éléments de données avec relations

Chacun de ces éléments correspond à un fichier physique (conformément à l'ISO 10303-21). Les attributs EXPRESS (ISO 10303-11:2004) peuvent seulement contenir des références à des données au sein du même fichier physique. Il est donc impossible d'utiliser directement des attributs EXPRESS pour mettre en œuvre des références entre des éléments.

5.6.2 Architecture à trois niveaux des données du dictionnaire

5.6.2.1 Généralités

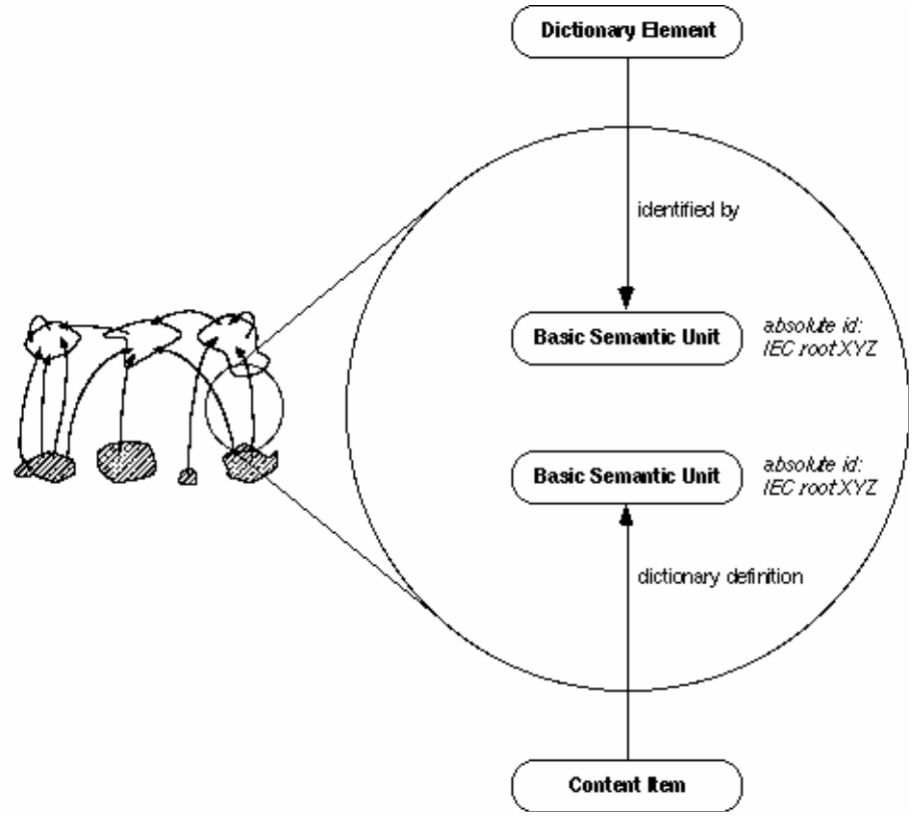
Dans le présent article, le concept de **basic_semantic_unit** (BSU) est introduit comme un moyen de mettre en œuvre ces références entre éléments. Une BSU fournit une identification universellement unique pour les descriptions du dictionnaire. Ceci est montré à la Figure 3.

Supposons qu'un certain élément du contenu souhaite se référer à une certaine description du dictionnaire.

EXEMPLE 1 Pour acheminer la valeur d'une propriété d'un composant.

Il le fait en se référant à une unité sémantique de base par le biais de l'attribut dictionary_definition.

Une description de dictionnaire (dictionary_element) se réfère à une unité sémantique de base par le biais de l'attribut identified_by. Une relation indirecte est établie à partir de la correspondance des identificateurs absolus des unités sémantiques de base.



Légende

Anglais	Français
Dictionary element	Élément du dictionnaire
identified by	identifié par
Basic Semantic Unit	Unité sémantique de base (BSU)
dictionary definition	définition du dictionnaire
Content Item	article du contenu
Absolute id	Identifiant absolu
IEC root XYZ	XYZ IEC racine

Figure 3 – Mise en œuvre de relations "entre éléments" à l'aide d'unités sémantiques de base

Noter que :

– l'élément du dictionnaire et l'article du contenu peuvent être tous les deux présents dans le même fichier physique, sans que cela constitue une exigence;

– l'élément du dictionnaire peut ne pas être présent pour l'échange d'un certain article du contenu le référençant. Dans ce cas, il est supposé être déjà présent dans le dictionnaire du système cible. Réciproquement, des données du dictionnaire peuvent être échangées sans une quelconque donnée de contenu;

– l'unité sémantique de base peut être une seule instance dans le cas où les instances d'élément du dictionnaire et d'article du contenu sont situées dans le même fichier physique;

– le même mécanisme s'applique aussi aux références entre divers éléments du dictionnaire

EXEMPLE 2 Entre une classe de composants et les **property_DET** associés.

Une BSU fournit une référence à une description du dictionnaire partout où elle est nécessaire.

EXEMPLE 3 Livraison de dictionnaire, livraison de mise à jour, livraison de bibliothèque, échange des données composants.

Les données associées à une propriété pourraient être échangées sous la forme d'une paire (**property_BSU**, <value>).

La Figure 3 présente les grandes lignes de la mise en œuvre de ce mécanisme général.

5.6.2.2 Basic_semantic_unit

Une **basic_semantic_unit** est une identification unique d'un **dictionary_element**. BSU est l'abréviation de Basic Semantic Unit (Unité sémantique de base).

EXPRESS specification:

```
*)
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
(*
```

Définitions des attributs:

code: le code assigné pour identifier un certain élément de dictionnaire.

version: le numéro de version d'un certain élément de dictionnaire.

dic_identifier: l'identification complète, consistant en la concaténation de "code" et "version".

definition: une référence à l'élément de dictionnaire identifié par cette BSU. S'il est absent dans un certain contexte d'échange, il est supposé être déjà présent dans le dictionnaire du système-cible.

referenced_by: articles utilisant l'élément de dictionnaire associé à cette BSU.

5.6.2.3 Dictionary_element

Un **dictionary_element** est une définition complète des données qui doit être capturée dans le dictionnaire sémantique pour certains concepts. Pour chaque concept, un sous-type séparé doit être utilisé. Le **dictionary_element** est associé à une **basic_semantic_unit** (BSU) qui sert à identifier de façon univoque cette définition dans le dictionnaire.

En incluant l'attribut version dans l'entité **basic_semantic_unit**, elle est partie intégrante de l'identification d'un élément du dictionnaire (contrairement aux attributs **revision** et **time_stamps**).

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
(*
```

Définitions des attributs:

identified_by: la BSU identifiant cet élément du dictionnaire.

time_stamps: les dates facultatives de création et de mise à jour de cet élément du dictionnaire.

revision: le numéro de révision de cet élément du dictionnaire.

NOTE 1 Le type de l'attribut **identified_by** sera redéfini ultérieurement en **property_BSU** et **class_BSU** et sera ensuite utilisé pour coder, conjointement à l'attribut "code" des BSU, l'attribut "Code" respectif pour les propriétés et les classes. Il sera aussi utilisé pour coder l'attribut "Version Number" (Numéro de version) respectif pour les propriétés et les classes.

NOTE 2 L'attribut **time_stamps** sera utilisé comme point de départ pour coder, dans l'entité **dates**, les attributs de propriétés et de classes "Date of Original Definition" (Date de définition d'origine), "Date of Current Version" (Date de version courante) et "Date of Current Revision" (Date de révision courante) (voir 5.11.3.2).

NOTE 3 L'attribut **revision** sera utilisé pour coder l'attribut de propriété et de classe "Revision Number" (Numéro de révision).

administration: information facultative relative au cycle de vie du **dictionary_element.**

NOTE 4 L'attribut **administration** sera utilisé pour représenter les informations relatives à la gestion de configuration et à l'historique des traductions.

is_deprecated: un booléen (Boolean) facultatif. Lorsqu'il est "true" (vrai), il spécifie que **dictionary_element** ne doit plus être utilisé.

is_deprecated_interpretation: spécifie la justification de la dépréciation et comment il convient d'interpréter les valeurs d'instances de l'élément déconseillé et celles de sa **BSU** correspondante.

Propositions formelles:

WR1: lorsque **is_deprecated** existe, **is_deprecated_interpretation** doit exister.

Propositions informelles:

IP1: les valeurs d'instance de l'élément i**s_deprecated_interpretation** doivent être définies au moment où la décision de dépréciation a été prise.

La Figure 4 présente un modèle de planification de la relation entre l'unité sémantique de base et l'élément du dictionnaire.

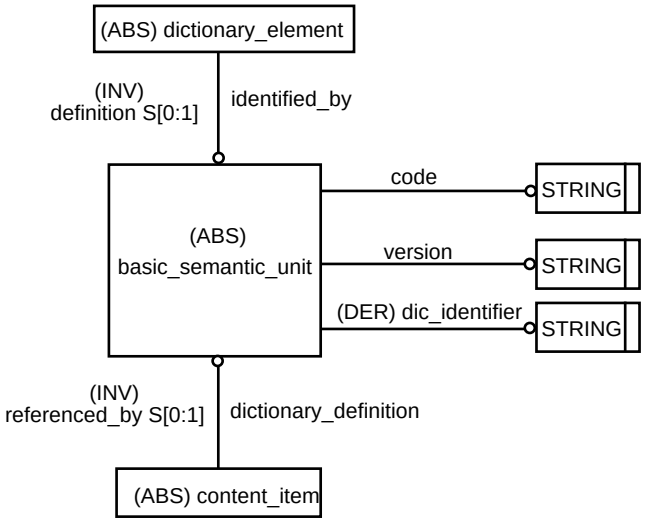


Figure 4 – Relation entre unité sémantique de base et élément de dictionnaire

5.6.2.4 Content_item

Un **content_item** est un élément de données renvoyant à sa description dans le dictionnaire. Il doit être sous-typé.

EXPRESS specification:

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

Définitions des attributs:

dictionary_definition: l'unité sémantique de base devant être utilisée pour se référer à la définition dans le dictionnaire.

5.6.3 Vue d'ensemble d'unités sémantiques de base et d'éléments de dictionnaire

Pour chaque sorte de données du dictionnaire, une paire de sous-types **basic_semantic_unit** et **dictionary_element** doit être définie. La Figure 5 présente, sous la forme d'un modèle de planification, les grandes lignes des unités sémantiques de base (BSU) et des éléments du dictionnaire définis ultérieurement. Noter que la relation entre BSU et éléments du dictionnaire est redéfinie pour chaque type de données et, de ce fait, seules des paires correspondantes peuvent être reliées. Cela n'est toutefois pas montré graphiquement ici.

Chaque catégorie de données du dictionnaire est traitée dans l'un des paragraphes suivants:

– pour les fournisseurs, voir 5.7;
– pour les classes, voir 5.8;
– pour les propriétés / types d'éléments de données, voir 5.9;
– pour les types de données, voir 5.10.

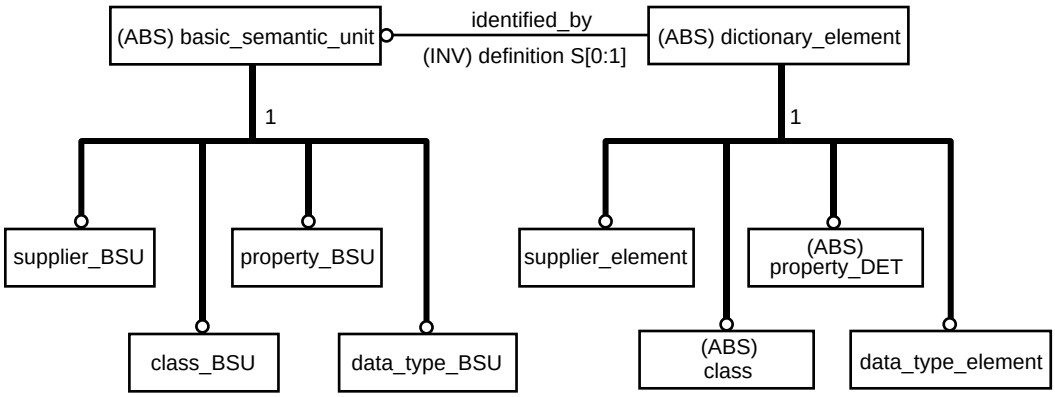


Figure 5 – BSU et éléments de dictionnaire courants

5.6.4 Identification d'éléments de dictionnaire: structure à trois niveaux

L'identification absolue des unités sémantiques de base repose sur la structure à trois niveaux ci-après:

– fournisseur (de données du dictionnaire);
– élément du dictionnaire défini par le fournisseur (tout élément du dictionnaire défini par le fournisseur qui est défini dans le modèle; dans le présent document, les éléments du dictionnaire définis par un fournisseur sont **property_DET** et **data_type_element,** mais il existe des dispositions pour étendre ce mécanisme à d'autres articles);
– version de l'élément du dictionnaire défini par le fournisseur.

Une identification absolue peut être obtenue par la concaténation du code applicable pour chaque niveau.

NOTE La structure sur cette identification absolue diffère de la structure définie dans l'édition 1 de la CEI 61360-2 (dupliquée pour la commodité dans l'ISO 13584-42:1998). Dans l'édition précédente, l'identification absolue de tout **dictionary_element** associé à un **name_scope** (y compris **property_DET** et **data_type_element**) consistait en: code fournisseur + code de classe (correspondant à la classe **name_scope**) + code d'élément du dictionnaire + version d'élément du dictionnaire. Dans la présente édition, le code de classe a été enlevé. Ainsi, le code d'élément du dictionnaire est unique, pour le même type d'élément du dictionnaire, sur toutes les classes définies par le même fournisseur. Pour les dictionnaires de référence existants, il convient que les organismes d'enregistrement, les autorités de maintenance ou les groupes de normalisation chargés des dictionnaires normalisés assurent cette unicité, en définissant éventuellement de nouveaux codes préfixés par les codes de classe **name_scope**.

Ce schéma d'identification est approprié dans le contexte multifournisseur. Si dans un certain domaine d'application, seules les données d'un seul fournisseur (de données) sont pertinentes, les parties correspondantes de cette identification, qui sont alors constantes, peuvent être éliminées. Pour les besoins d'échange, tous les niveaux doivent toutefois être présents, afin d'éviter des conflits d'identificateurs.

Ce schéma d'identification est formellement décrit dans l'attribut **absolute_id** des entités xxx_BSU définies de 5.7 à 5.12.

5.6.5 Possibilités d'extension pour d'autres types de données

5.6.5.1 Généralités

Le mécanisme BSU – élément de dictionnaire est très général et ne se limite pas aux quatre sortes de données utilisées ici (voir Figure 5). Le présent Article spécifie un certain nombre de moyens qui permettent des extensions d'autres sortes. Selon que le domaine d'application de l'identificateur est donné par une classe ou par un fournisseur, l'entité **xxx_related_BSU** correspondante doit être sous-typée. Il est nécessaire de redéfinir l'attribut **identified_by** de l'entité **dictionary_element** (comme cela est réalisé dans les paragraphes 5.7.3 à 5.10 ou dans les catégories de données courantes).

5.6.5.2 Supplier_related_BSU

Le **supplier_related_BSU** offre une prise en charge pour que les éléments de dictionnaire soient associés à des fournisseurs.

EXEMPLE Pour la série ISO 13584: bibliothèques de programmes.

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

Le **class_related_BSU** offre une prise en charge pour que les éléments de dictionnaire soient associés à des classes.

EXEMPLE Pour les tableaux ISO 13584, 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

Le **supplier_BSU_relationship** est une disposition permettant l'association des BSU à des fournisseurs.

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
(*
```

Définitions des attributs:

relating_supplier: le **supplier_element** qui identifie le fournisseur de données.

related_tokens: l'ensemble d'éléments de dictionnaire associés au fournisseur identifié par l'attribut **relating_supplier**.

5.6.5.5 Class_BSU_relationship

L'entité **class_BSU_relationship** est une disposition permettant l'association des BSU à des 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
(*
```

Définitions des attributs:

relating_class: l'attribut **class** qui identifie l'élément de dictionnaire.

related_tokens: l'ensemble des éléments de dictionnaire associés à la classe identifiée par l'attribut **relating_class**.

5.7 Données fournisseur

5.7.1 Généralités

Le présent article contient des définitions pour la représentation des données relatives au fournisseur lui-même. Dans un environnement multifournisseur, il est nécessaire de pouvoir identifier la source d'un certain élément du dictionnaire. La Figure 6 présente un modèle de planification des données associées à des fournisseurs, suivi de la définition EXPRESS.

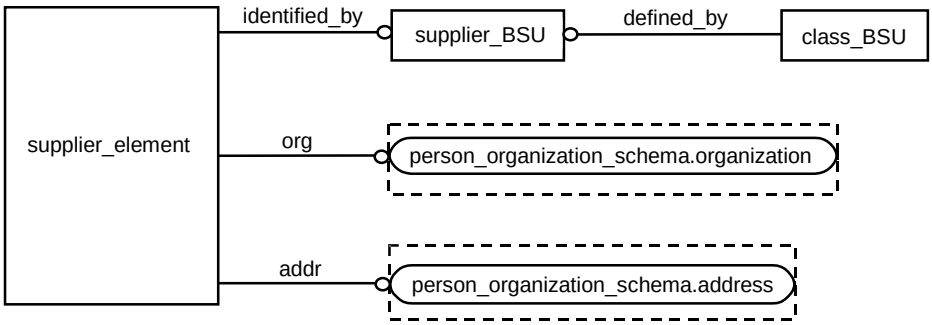


Figure 6 – Vue d'ensemble des données fournisseur et des relations

5.7.2 Supplier_BSU

L'entité **supplier_BSU** permet la prise en charge de l'identification unique des fournisseurs d'informations.

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
(*
```

Définitions des attributs:

code: le code fournisseur attribué conformément à l'ISO 13584-26.

version: le numéro de version d'un code fournisseur doit être égal à 1.

absolute_id: l'identification absolue du fournisseur.

Propositions formelles

UR1: l'identificateur du fournisseur défini par l'attribut **absolute_id** est unique.

5.7.3 Supplier_element

L'entité **supplier_element** donne la description des fournisseurs du dictionnaire.

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
(*
```

Définitions des attributs:

identified_by: l'entité **supplier_BSU** utilisée pour identifier ce **supplier_element**.

org: les données organisationnelles de ce fournisseur.

addr: l'adresse de ce fournisseur.

associated_items: permet l'accès à d'autres sortes de données par le biais du mécanisme de BSU.

EXEMPLE Bibliothèque de programmes dans l'ISO 13584-24:2003.

5.8 Données de classe

5.8.1 Généralités

Le présent article contient des définitions pour la représentation des données de classe du dictionnaire.

Le Figure 7 présente, sous la forme d'un modèle de planification, les grandes lignes des données associées aux classes et leur relation à d'autres éléments du dictionnaire.

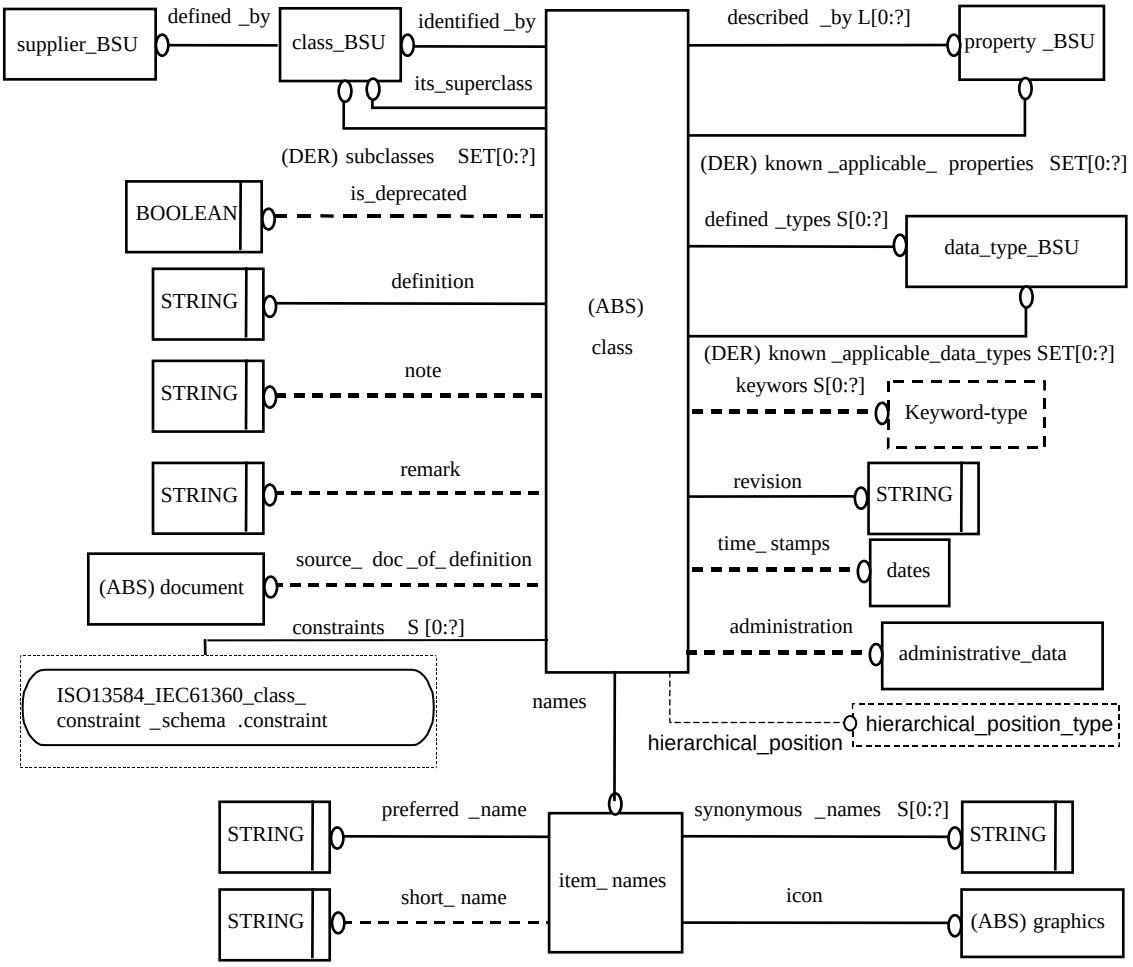


Figure 7 – Vue d'ensemble des données de classe et de leurs relations

Comme indiqué à la Figure 7 avec l'attribut **its_superclass**, les classes forment un arbre d'héritage. Il est important de noter que tout au long du présent document, les termes "héritage" et "hériter" représentent cette relation entre classes (définies dans le dictionnaire), bien que le langage EXPRESS ait aussi un concept d'héritage. Ils doivent être distingués de façon claire et nette pour éviter les incompréhensions.

Les données du dictionnaire pour les classes (telles que montrées à la Figure 7) sont étalées sur trois niveaux d'héritage:

– class_and_property_elements définit les données communes aux classes et aux property_DET;

– class permet que d'autres sortes de classes soient spécifiées ultérieurement;

EXEMPLE D'autres sous-types de **class**, en particulier **functional_view_class**, **functional_model_class** et **fm_class_view_of**, sont spécifiés dans l'ISO 13584-24:2003. Ils ne caractérisent pas les produits, mais ils permettent la prise en charge de l'échange des représentations produit particulières (par exemple: des représentations géométriques).

– **item_class** et **categorization_class** sont les entités qui contiennent les données de différentes classes d'objets du domaine d'application.

NOTE 1 Deux sous-types de **item_class**, appelés **component_class** et **material_class**, avaient été définis dans le modèle de dictionnaire de la première édition de la présente partie de CEI 61360. Ces sous-types sont déconseillés et ils ont été retirés de la présente partie de la CEI 61360.

NOTE 2 Les changements suivants assurent que les définitions de classe d'un dictionnaire conformes à la première édition de la 61360-2 se conforment à la présente édition: (1) remplacer **component_class** et **material_class** par **item_class** dans tout le dictionnaire de référence; (2) à chaque nouvelle class **item_class**, ajouter l'attribut **instance_sharable**, dont la valeur est true; (3) pour chaque nouvelle classe **item_class**, ajouter

l'attribut facultatif **hierarchical_position** sans attribuer de valeur; (4) pour chaque nouvelle classe **item_class**, ajouter l'attribut **keywords**, dont la valeur est une collection vide.

NOTE 3 Un autre sous-type of **item_class**, appelé **feature_class,** avait été fourni par l'ISO 13584-24:2003. Ce sous-type est également déconseillé et son utilisation est interdite dans les nouvelles mises en œuvre de la présente partie de la CEI 61360.

NOTE 4 Les changements suivants assurent que les définitions de classe d'un dictionnaire conformes à l'ISO 13584-25 se conforment à la présente partie de la CEI 61360: (1) remplacer **feature_class** par **item_class** dans tout le dictionnaire de référence; (2) à chaque nouvelle classe **item_class**, ajouter l'attribut **instance_sharable**, dont la valeur est false; (3) pour chaque nouvelle classe **item_class**, ajouter l'attribut facultatif **hierarchical_position** sans attribut de valeur; (4) pour chaque nouvelle classe **item_class**, ajouter l'attribut **keywords**, dont la valeur est une collection vide.

5.8.2 Détail de structure

5.8.2.1 Class_BSU

L'entité **class_BSU** permet la prise en charge de l'identification des classes.

EXPRESS specification:

```
*)
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
(*
```

Définitions des attributs:

code: le code affecté à cette classe par son fournisseur.

defined_by: le fournisseur définissant cette classe et son élément de dictionnaire.

absolute_id: l'identification unique de cette classe.

known_visible_properties: l'ensemble des **property_BSU** qui se réfèrent à la classe comme leur attribut **name_scope** (portée de nom) ou à toute superclasse connue de cette classe et qui sont donc visibles pour la classe (et n'importe laquelle de ses sous-classes)

NOTE 1 Lorsque l'attribut **dictionary_definition** d'une certaine classe est absent dans le même contexte d'échange (un contexte d'échange PLIB n'est jamais supposé être complet), la superclasse d'une classe peut ne

pas être connue. Par conséquent, les propriétés définies comme étant visibles par cette superclasse n'appartiennent pas à l'attribut **known_visible_properties**. C'est seulement pour le système de réception que tous les attributs **dictionary_definition** des BSU doivent être disponibles. Par conséquent, sur le système de réception, l'attribut **known_visible_properties** contient toutes les propriétés visibles pour la classe.

known_visible_data_types: l'ensemble des **data_type_BSU** qui se réfèrent à la classe comme leur attribut **name_scope** ou à toute superclasse connue de cette classe et qui sont donc visibles pour la classe (et n'importe laquelle de ses sous-classes)

NOTE 2 Lorsque l'attribut **dictionary_definition** d'une certaine classe est absent dans le même contexte d'échange (un contexte d'échange PLIB n'est jamais supposé être complet), la superclasse d'une classe peut ne pas être connue. Par conséquent, les entités **data_type** définies comme étant visibles par cette superclasse n'appartiennent pas à l'attribut **known_visible_data_types**. C'est seulement pour le système de réception que tous les attributs **dictionary_definition** des BSU doivent être disponibles. Par conséquent, sur le système de réception, l'attribut **known_visible_data_types** contient toutes les entités **data_type** visibles pour la classe.

subclasses: l'ensemble des classes spécifiant cette classe comme étant leur superclasse.

added_visible_properties: l'ensemble des entités **property_BSU** qui se réfèrent à la classe comme étant leur **name_scope** et qui sont donc visibles pour cette classe (et n'importe laquelle de ses sous-classes).

NOTE 3 Seuls les entités **property_BSU** qui appartiennent au même contexte d'échange sont référencées par cet attribut inverse. Sur le côté de réception, il peut déjà exister d'autres **property_BSU** qui se réfèrent à cette classe (un contexte d'échange PLIB n'est jamais supposé être complet).

NOTE 4 L'attribut **added_visible_properties** sera utilisé pour coder l'attribut de classe "Visible properties" (Propriétés visibles).

added_visible_data_types: l'ensemble des entités **data_type_BSU** qui se réfèrent à la classe comme étant leur **name_scope** et qui sont donc visibles pour cette classe (et n'importe laquelle de ses sous-classes).

NOTE 5 Seuls les entités **data_type_BSU** qui appartiennent au même contexte d'échange sont référencées par cet attribut inverse. Sur le côté de réception, il peut déjà exister d'autres **data_type_BSU** qui se réfèrent à cette classe (un contexte d'échange PLIB n'est jamais supposé être complet).

NOTE 6 L'attribut **added_visible_data_types** sera utilisé pour coder l'attribut de classe "Visible types" (Types visibles).

Propositions formelles

UR1: la concaténation du code fournisseur et du code de classe est unique.

5.8.2.2 Class_and_property_elements

L'entité **class_and_property_elements** capture les attributs qui sont communs aux **class** et aux **property_DET**.

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
(*
```

Définitions des attributs:

names: les noms décrivant cet élément du dictionnaire.

definition: le texte décrivant cet élément du dictionnaire.

source_doc_of_definition: le document source de cette description textuelle.

note: informations supplémentaires sur n'importe quelle partie d'élément du dictionnaire, qui sont essentielles à la compréhension.

remark: texte explicatif clarifiant davantage la signification de cet élément du dictionnaire.

NOTE 1 L'attribut **names** sera utilisé comme un point de départ pour coder dans l'entité **item_names** les attributs de propriété et de classe "Preferred Name" (Nom préférentiel), "Short Name" (Nom court) et "Synonymous Name" (Synonyme).

NOTE 2 L'attribut **definition** sera utilisé pour coder l'attribut propriété "Definition" (Définition) et l'attribut classe "Definition" (Définition).

NOTE 3 L'attribut **source_of_doc_definition** sera utilisé pour coder l'attribut propriété "Source document of definition" (Document de définition source) et l'attribut classe "Source document of definition".

NOTE 4 L'attribut **note** sera utilisé pour coder l'attribut propriété et classe "Note".

NOTE 5 L'attribut **remark** sera utilisé pour coder l'attribut propriété et classe "Remark" (Remarque).

5.8.2.3 Class

L'entité **class** est une ressource abstraite pour toutes les sortes de 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
(*
```

Définitions des attributs:

identified_by: l'entité **class_BSU** identifiant cette classe.

its_superclass: référence à la classe dont la classe courante est une sous-classe.

described_by: la liste des références à des propriétés complémentaires disponibles à l'utilisation dans la description des produits au sein d'une classe, et de n'importe laquelle de ses sous-classes.

NOTE 1 Une propriété peut également être applicable à une classe lorsque cette propriété est importée d'une autre classe par le biais d'une entité **a_priori_semantic_relationship** telle que définie en 8.5 de la présente partie de la CEI 61360. Par conséquent, les propriétés référencées par l'attribut **described_by** ne définissent pas toutes les propriétés applicables pour cette classe.

NOTE 2 L'ordre de la liste est l'ordre de présentation conseillé par le fournisseur.

NOTE 3 Une propriété qui est une propriété dépendante d'un contexte (**context_dependent_P_DET**) peut devenir applicable à une classe seulement si tous les paramètres de contexte (**condition_DET**) dont dépend sa valeur sont également applicables à cette classe. Cela est énoncé dans la règle "WHERE" 5 (WR5).

defined_types: l'ensemble des références aux types qui peuvent être utilisés pour les divers **property_DET** dans tout l'arbre d'héritage descendant de cette classe.

NOTE 4 Une entité **data_type** peut également être applicable à une classe lorsque cette entité **data_type** est importée d'une autre classe par le biais d'une entité **a_priori_semantic_relationship** telle que définie en 8.5 de la présente partie de la CEI 61360. Par conséquent, les types de données référencés par l'attribut **described_types** ne définissent pas tous les types de données applicables pour cette classe.

constraints: l'ensemble des contraintes qui limitent les domaines cibles des valeurs de certaines propriétés de la classe à certains sous-ensembles de leurs domaines hérités de valeurs.

NOTE 5 Il convient que chaque contrainte dans l'attribut **constraints** soit remplie par les instances de classe. Ainsi, l'attribut **constraints** est donc une conjonction de contraintes.

hierarchical_position: la représentation codée de la position d'une classe dans la hiérarchie d'inclusion des classes à laquelle elle appartient; une **hierarchical_position** d'une classe change lorsque la structure des classes d'une ontologie change. Par conséquent, elle ne peut pas servir d'identificateur stable pour des classes.

NOTE 6 Cette sorte de nom codé est utilisée en particulier dans des hiérarchies de catégorisation de produits pour représenter la structure d'inclusion des classes à travers un certain nombre de conventions de codage.

EXEMPLE 1 Dans l'UNSPSC, *Manufacturing Components and Supplies* (Composants et fournitures de fabrication) a la position hiérarchique 31000000, *Hardware* (Équipement matériel) a la position hiérarchique 31160000 et *Bolt* (Boulon) a la position hiérarchique 31161600. Par convention, cette représentation de la position hiérarchique permet de déclarer que *Manufacturing Components and Supplies* est au premier niveau de la hiérarchie, que *Hardware* est au deuxième niveau de la hiérarchie et est inclus dans *Manufacturing Components and Supplies* et que *Bolt* est au troisième niveau de la hiérarchie et est inclus dans *Hardware.*

keywords: un ensemble de mots clés, éventuellement en plusieurs langues, permettant d'effectuer des recherches dans la classe.

sub_class_properties: déclare les propriétés de valeur d'une classe, c'est-à-dire que dans les sous-classes, une seule valeur sera assignée par classe. Voir 5.9.6.

class_constant_values: affectations dans la classe courante pour les propriétés de valeur déclarées d'une classe en des superclasses. Voir 5.9.6.

subclasses: l'ensemble de classes spécifiant cette classe comme étant leur superclasse

known_applicable_properties: Les **property_BSU** qui sont référencées par la classe, ou n'importe laquelle de sa/ses superclasse(s) connue(s) par leur attribut **described_by** et qui sont donc applicables à cette classe (et n'importe laquelle de ses sous-classes).

NOTE 7 Lorsque l'attribut **dictionary_definition** d'une certaine classe est absent dans le même contexte d'échange (un contexte d'échange PLIB n'est jamais supposé être complet), la superclasse d'une classe peut ne pas être connue. Par conséquent, les propriétés définies comme étant applicables par cette superclasse n'appartiennent pas à l'attribut **known_applicable_properties**. C'est seulement pour le système de réception que tous les attributs **dictionary_definition** des **BSU** doivent être disponibles. Par conséquent, sur le système de réception, l'attribut **known_applicable_properties** contient toutes les propriétés qui sont applicables à une classe en vertu du fait d'être référencées par un attribut **described_by**.

known_applicable_data_types: Les **data_type_BSU** qui sont référencées par la classe, ou n'importe laquelle de sa/ses superclasse(s) connue(s) par leur attribut **defined_types** et qui sont donc applicables à cette classe (et n'importe laquelle de ses sous-classes).

NOTE 8 Lorsque l'attribut **dictionary_definition** d'une certaine classe est absent dans le même contexte d'échange (un contexte d'échange PLIB n'est jamais supposé être complet), la superclasse d'une classe peut ne pas être connue. Par conséquent, les entités **data_type** définies comme étant applicables par cette superclasse n'appartiennent pas à l'attribut **known_applicables_data_types**. C'est seulement pour le système de réception que tous les attributs **dictionary_definition** des **BSU** doivent être disponibles. Par conséquent, sur le système de réception, l'attribut **known_applicable_data_types** contient toutes les **data_type** qui sont applicables à une classe en vertu du fait d'être référencées par un attribut **defined_types**.

known_property_constraints: les **constraints** sur une propriété qui sont référencées par la classe, ou par n'importe laquelle de ses superclasses "is-a" connues par leur attribut **constraints** ou, dans le cas d'une classe qui est un sous-type de **a_priori_semantic_relationship,** par son attribut **referenced_constraints**.

associated_items: permet d'accéder à d'autres sortes de données en utilisant le mécanisme de BSU.

Propositions formelles

WR1: la structure d'héritage définie par la hiérarchie des classes ne contient pas de boucles.

WR2: seules les propriétés qui sont visibles pour une classe peuvent devenir applicables à cette classe par le fait d'être référencées par l'attribut **described_by**.

WR3: seules les types de données qui sont visibles pour une classe peuvent devenir applicables à cette classe par le fait d'être référencés par l'attribut **defined_types.**

WR4: seules les propriétés qui ne sont pas applicables pour une classe par héritage peuvent devenir applicables à cette classe par le fait d'être référencées par l'attribut **described_by**.

WR5: seules les propriétés dépendantes d'un contexte (**dependent_P_DET**) dont tous les paramètres de contexte (**condition_DET**) sont applicables dans la classe peuvent devenir applicables pour cette classe par let fait d'être référencées par son attribut **described_by**.

WR6: seuls les types de données qui ne sont pas applicables pour une classe par héritage peuvent devenir applicables à cette classe par le fait d'être référencés par l'attribut **defined_types**.

NOTE 9 L'attribut **its_superclass** sera utilisé pour coder l'attribut "Superclass" (Superclasse) d'une classe.

NOTE 10 L'attribut **described_by** fournit le codage pour l'attribut "Applicable Properties" d'une classe.

NOTE 11 L'attribut **defined_types** est utilisé pour coder l'attribut "Applicable Types" (Types applicables) d'une classe.

WR7: l'ensemble des contraintes qui sont des contraintes de propriété doit définir des restrictions qui soient compatibles avec le domaine des valeurs des propriétés auxquelles elles s'appliquent.

WR8: toutes les propriétés référencées dans la précondition d'une **configuration_control_constraint** doivent être applicables à la classe.

WR9: toutes les propriétés référencées dans l'attribut **constraint** doivent être soit visibles, soit applicables à la classe.

WR10: soit tous les attributs **keywords** sont représentés comme étant des **label_with_languages**, soit ils sont tous représentés comme étant des **labels**.

WR11: Si **class** n'est pas une **a_priori_semantic_relationship**, **sub_class_properties** doit appartenir à la liste d'attributs **described_by**.

NOTE 12 Par le biais d'une a_priori_semantic_relationship, l'attribut sub_class_properties peut également être importé.

WR12: les propriétés référencées dans **class_constant_values** sont déclarées comme étant de valeur pour une classe dans une certaine superclasse de la classe courante ou dans la classe courante elle-même.

NOTE 13 L'attribut **sub_class_properties** de l'entité **class** est utilisé pour coder l'attribut "Class valued properties" (Propriétés valuées d'une classe) des classes.

NOTE 14 L'attribut **class_constant_values** de l'entité **class** est utilisé pour coder l'attribut "Class constant values" (Valeurs de constantes de classe) des classes.

WR13: Si une propriété référencée dans **class_constant_values** a déjà reçu une valeur qui lui est assignée dans une superclasse, il convient que la valeur assignée dans la classe courante soit la même.

Propositions informelles:

IP1: Si toutes les superclasses "is-a" de la classe sont disponibles, les **known_property_constraints** sont toutes les contraintes qui s'appliquent aux propriétés qui sont reliées à cette classe comme propriétés soit visibles, soit applicables.

5.8.3 Item_class

L'entité item_class permet la modélisation de tout type d'entité du domaine d'application qui peut être capturé par une classe de caractérisation définie par une structure de classe et un ensemble de propriétés. En particulier, les instances produits et aussi les instances aspects particuliers des produits représentées comme étant des caractéristiques intrinsèques sont mises en correspondances avec l'entité item_class.

L'entité item_class inclut un attribut instance_sharable qui spécifie le statut conceptuel de l'article. Si cet attribut est "true", chaque instance représente un article indépendant, autrement elle est une caractéristique intrinsèque, c'est-à-dire un article dépendant qui doit être le composant d'un autre article. Ceci ne prescrit aucune mise en œuvre spécifique au niveau de la représentation des données.

EXEMPLE La *tête d'une vis* est une caractéristique intrinsèque décrite par un nombre de propriétés, mais elle peut exister seulement lorsqu'elle est référencée par une vis. Elle est représentée comme étant une **item_class** avec l'attribut **instance_sharable** égal à *false*.

NOTE 1 Deux sous-types de **item_class**, appelés **component_class** et **material_class**, avaient été définis dans le modèle de dictionnaire de la première édition de l'ISO 13584-42 et celle de la CEI 61360-2. Ces sous-types sont déconseillés et ils ont été retirés de la présente édition de la CEI 61360.

NOTE 2 Un autre sous-type of **item_class**, appelé **feature_class,** avait été fourni par l'ISO 13584-24:2003. Ce sous-type est également déconseillé et son utilisation n'est pas recommandée dans les nouvelles mises en œuvre de la présente partie de la CEI 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
(*
```

Définitions des attributs:

simplified_drawing: dessins (**graphics**) facultatifs qui peuvent être associés à la classe décrite.

NOTE 3 L'attribut **simplified_drawing** de l'entité **item_class** est utilisé pour coder l'attribut "Simplified Drawing" (Dessin simplifié) pour des classes.

coded_name: peut être utilisé comme une valeur de constante de classe pour caractériser la classe dans un domaine de valeurs d'un attribut **sub_class_properties** de sa superclasse.

NOTE 4 Cet attribut n'est pas utilisé dans la série ISO 13584. Il est seulement utilisé dans la série CEI 61360.

instance_sharable: lorsqu'il est "false", il spécifie que les instances de **item_class** sont des caractéristiques intrinsèques; lorsqu'il n'est pas donné ou est "true", il spécifie que les instances de **item_class** sont des articles autonomes.

NOTE 5 Dans le modèle de dictionnaire commun ISO13584/CEI61360, c'est la mise en œuvre qui permet de décider si plusieurs instances en monde réel de caractéristiques intrinsèques modélisées par le même ensemble de paires "propriété-valeur" sont représentées par plusieurs éléments de données EXPRESS ou par le même élément de données dans le fichier d'échange de données. Ainsi, une instance de **item_class** dont l'attribut **instance_sharable** est égal à *false* et qui est référencée par plusieurs instances de **item_class** au niveau de modèle de données est interprétée comme plusieurs instances en monde réel de la même caractéristique intrinsèque.

5.8.4 Categorization_class

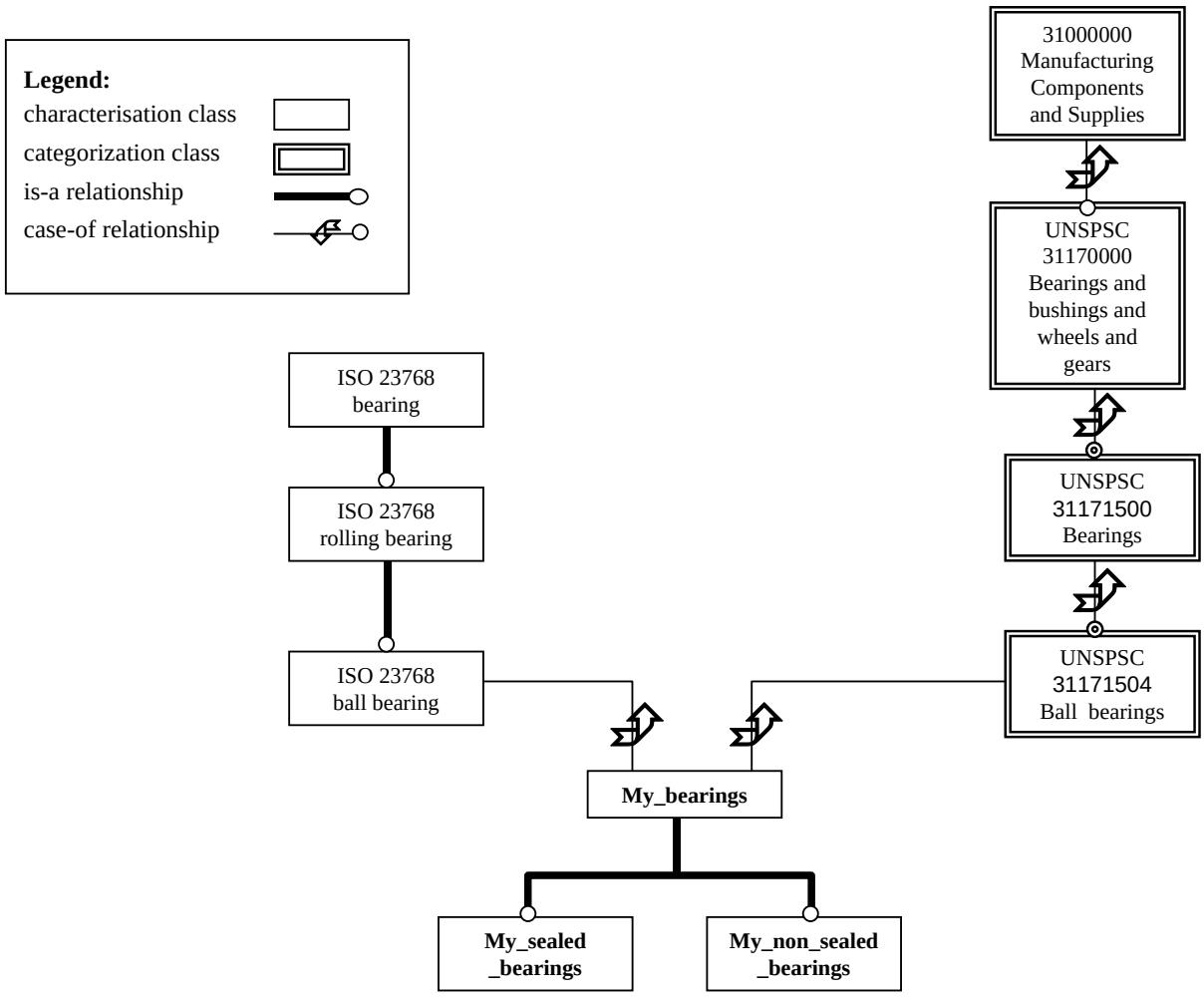
L'entité categorization_class permet la modélisation d'un regroupement d'un ensemble d'objets qui constitue un élément d'une catégorisation.

EXEMPLE 1 "Composants de fabrication et fournitures", et "Optique industrielle" sont des exemples de classes catégorisation de produits définies dans l'UNSPSC.

Ni les propriétés ni les datatypes ni les contraintes ne sont associés, comme étant visibles ou applicables, à une telle classe. En outre, les **categorization_class** ne peuvent pas être reliées les unes aux autres par la relation d'héritage "is-a", mais elles peuvent seulement être reliées les unes aux autres par la relation de classes "is-case-of". Un attribut spécifique, appelé **categorization_class_superclasses**, permet d'enregistrer les **categorization_class** qui sont des superclasses de **categorization_class** dans une hiérarchie "case-of".

NOTE En utilisant les constructions de ressources "case-of", des **item_class** peuvent également être reliées à des **categorization_class**.

EXEMPLE 2 L'exemple ci-après montre comment classes de caractérisation et classes de catégorisation peuvent être reliées pour atteindre un certain nombre d'objectifs particuliers. Un fournisseur de roulements à billes souhaite concevoir sa propre ontologie et la rendre facile à récupérer et à utiliser. Pour atteindre ces objectifs, il souhaite utiliser des propriétés normalisées et être relié à des classifications normalisées. Le fournisseur fournit seulement des roulements à billes, mais certains roulements sont scellés, d'autres non. Des propriétés particulières peuvent être associées aux roulements scellés et aux roulements non scellés, mais ces catégories n'existent pas en tant que classes dans les ontologies des roulements normalisées. Donc, le fournisseur de roulements procède comme suit. (1) Il conçoit une ontologie propriétaire constituée de trois classes de caractérisation: *my_bearings (mes roulements), my_sealed_bearings (mes roulements scellés), my_non_sealed_bearing (mes roulements non scellés)*. Les deux dernières classes sont reliées à la première par la relation d'héritage "is-a" et toutes les propriétés assignées à la première classe sont héritées par la dernière. (2) Pour utiliser certaines des propriétés définies dans l'ISO/TS 23768-1 (à publier), le fournisseur de roulements spécifie que sa classe *my_bearings* est "case-of" la classe normalisée de roulements *ball bearing* définies dans l'ISO/TS 23768-1 (à publier). Par cette relation "case-of", il peut importer dans sa classe *my_bearings* les propriétés définies par la norme: *bore diameter (diamètre d'alésage), outside diameter (diamètre extérieur), ISO tolerance class (classe de tolérance ISO)*. De plus, il créait les propriétés recherchées qui ne sont pas définies dans la norme. (3) Pour faciliter la récupération du serveur qui affiche le catalogue fournisseur, il expose un petit fragment de la classification UNSPSC et une relation "case-of" entre la classe *ball_bearings* de l'UNSPSC et sa propre classe *my_bearing*. Le résultat est présenté à la Figure 8 ci-dessous:



Légende

Anglais	Français
Characterisation class	Classe de caractérisation
categorization class	Classe de catégorisation
is-a relationship	Relation « is-a »
case-of relationship	Relation « case-of »
ISO 23768 bearing	Roulement ISO 23768
ISO 23768 rolling bearing	Roulement à rouleaux ISO 23768
ISO 23768 ball bearing	Roulement à billes ISO 23768
31000000 Manufacturing components and supplies	Composants et fournitures de fabrication 31000000
UNSPSC 31170000 Bearings and bushings and wheels and gears	UNSPSC 31170000 Roulements, douilles, roues et engrenages
UNSPSC 31171500 Bearings	UNSPSC 31171500 Roulements
UNSPSC 31171504 Ball Bearings	UNSPSC 31171504 Roulements à billes

Figure 8 – Exemple d'ontologie fournisseur

EXPRESS specification:

```
*)
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
(*
```

Définitions des attributs:

categorization_class_superclasses: les **categorization_class** qui sont supérieures d'un niveau à la classe de catégorisation dans une hiérarchie de classe "case-of".

Propositions formelles

WR1: seules des **categorization_class** peuvent apparaître comme étant des superclasses d'une entité **categorization_class**.

WR2: une **categorization_class** ne doit pas avoir de superclasse "is-a".

WR3: aucune propriété ne doit être associée à **categorization_class**.

WR4: aucun datatype ne doit être associé à **categorization_class**.

WR5: aucune contrainte ne doit être associée à **categorization_class**.

WR6: une **categorization_class** ne doit pas être la classe définition de propriétés pour une quelconque propriété.

WR7: aucune propriété de sous-classe ne doit être associée à **categorization_class**.

WR8: aucune valeur de constante de classe ne doit être associée à **categorization_class**.

WR9: aucune propriété visible ne doit être associée à **categorization_class**.

WR10: aucun datatype visible ne doit être associé à **categorization_class**.

5.9 Type de données d'un élément / propriétés des données

5.9.1 Généralités

Le présent article contient des définitions des propriétés pour les données du dictionnaire.

5.9.2 Property_BSU

L'entité **property_BSU** permet la prise en charge de l'identification d'une propriété.

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
(*
```

Définitions des attributs:

code: pour permettre l'identification unique de la propriété sur toutes les ontologies définies par le même fournisseur **name_scope.defined_by**.

name_scope: la référence à la classe à laquelle ou en dessous de laquelle l'élément de propriété est disponible comme référence par l'attribut **described_by**.

absolute_id: l'identification unique de cette propriété.

describes_classes: les classes déclarant cette propriété disponible à l'emploi pour la description d'un produit.

Propositions formelles:

WR1: toute classe référencée par l'attribut **describes_classes** de **property_BSU** est soit la classe référencée par son attribut **name_scope**, soit une sous-classe de cette classe.

UR1: l'identificateur de propriété **absolute_id** est unique.

NOTE L'attribut **name_scope** de **property_BSU** sera utilisé pour coder l'attribut "Definition class" (Classe de définition) pour les propriétés.

5.9.3 Property_DET

L'entité **property_DET** capture la description des propriétés d'un dictionnaire.

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
(*
```

Définitions des attributs:

identified_by: l'entité **property_BSU** identifiant cette propriété.

preferred_symbol: une description plus courte de cette propriété.

synonymous_symbols: synonyme de la description plus courte de cette propriété.

figure: une entité **graphics** facultative qui décrit la propriété.

det_classification: la classe ISO 80000/CEI 80000 (anciennement ISO 31) pour cette propriété.

domain: la référence **data_type** associée à la propriété.

formula: expression mathématique pour expliquer la propriété.

describes_classes: les classes déclarant cette propriété comme étant disponible à l'emploi dans la description d'un produit.

NOTE 1 L'attribut **preferred_symbol** est utilisé pour coder l'attribut "Preferred Letter Symbol" (Symbole littéral préférentiel) des propriétés.

NOTE 2 L'attribut **synonymous_symbols** est utilisé pour coder l'attribut "Synonymous Letter Symbol" (Symbole littéral synonyme) des propriétés.

NOTE 3 L'attribut **det_classification** est utilisé pour coder l'attribut "Property Type Classification" (Classification de type de propriété) d'une propriété.

NOTE 4 L'attribut **domain** est utilisé comme point de départ pour le codage de l'attribut de propriété "Data Type". L'entité **data_type** sera sous-typée pour divers types de données possibles.

NOTE 5 L'attribut **formula** est utilisé pour coder l'attribut "Formula" des propriétés.

La Figure 9 présente un modèle de planification des données associées à des **property_DET**.

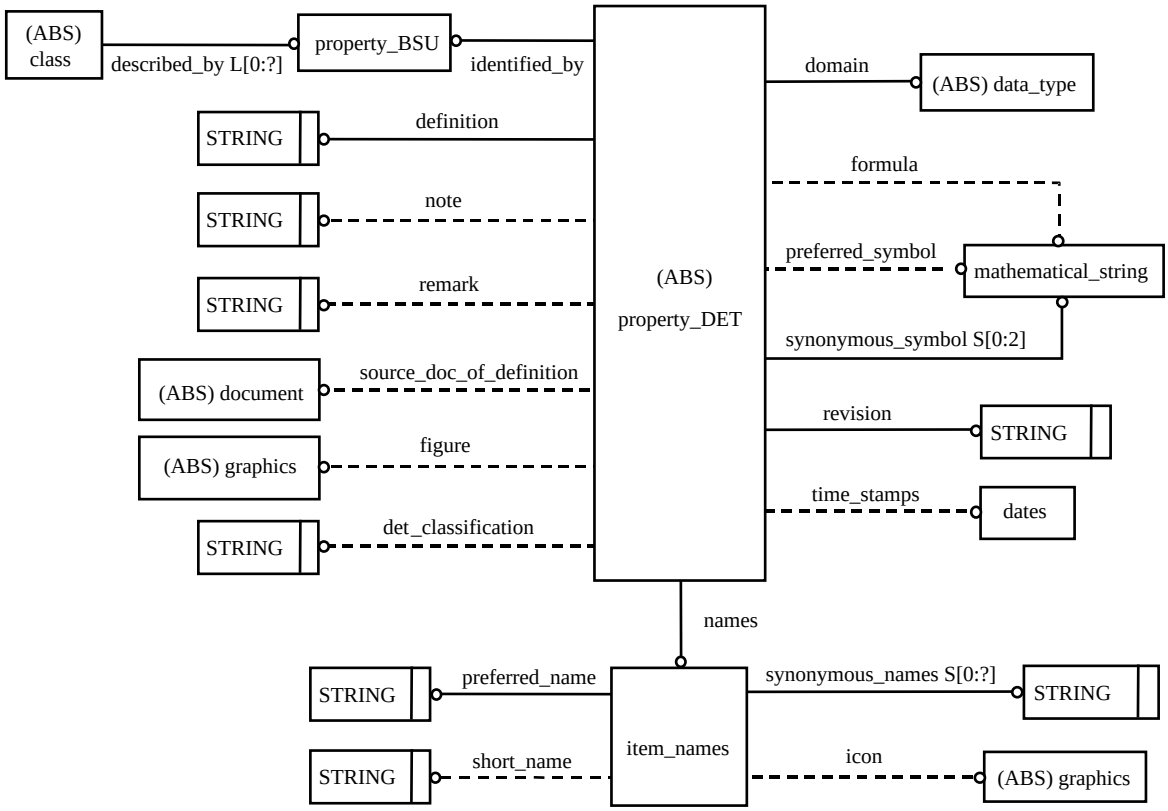


Figure 9 – Vue d'ensemble des attributs des données d'un élément et de leurs relations

5.9.4 Types de données d'un élément, conditionnels, dépendants et indépendants

La Figure 10 décrit les diverses sortes de Data Element Types (types de données d'un élément) dans le format d'un modèle de planification.

Noter que la Figure 10 est simplifiée: la relation "**depends_on**" est essentiellement mise en œuvre avec une référence BSU, mais une contrainte est spécifiée stipulant que **property_DET** référencée doit être une **condition_DET** (voir l'entité **dependent_P_DET**).

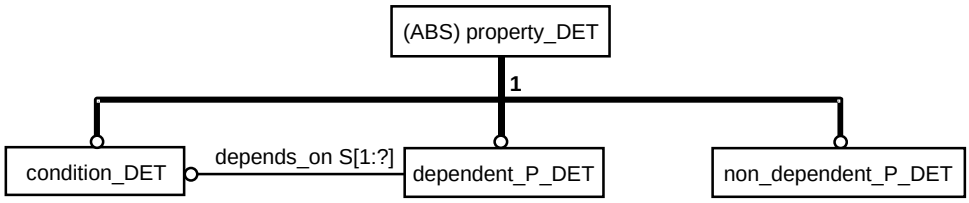


Figure 10 – Genres des types de données d'un élément

5.9.5 Détail de structure

5.9.5.1 Condition_DET

Une **condition_DET** est une propriété dont peuvent dépendre d'autres propriétés.

EXPRESS specification:

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

5.9.5.2 Dependent_P_DET

Une **dependent_P_DET** est une propriété dont la valeur dépend explicitement de la/des valeur(s) d'une/de certaine(s) condition(s).

EXEMPLE La résistance d'une thermistance dépend de la température ambiante. Il convient de présenter la résistance d'une thermistance comme étant une **dependent_P_DET** et la température ambiante d'une thermistance comme étant une **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
(*
```

Définitions des attributs:

depends_on: l'ensemble des unités sémantiques de base identifiant les propriétés dont dépend cette propriété.

Propositions formelles:

WR1: seules des **condition_DET** doivent être utilisées dans l'ensemble **depends_on**.

5.9.5.3 Non_dependent_P_DET

Une **non_dependent_P_DET** est une propriété qui ne dépend pas explicitement de certaines conditions.

EXPRESS specification:

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

NOTE 1 Les trois sous-types **condition_DET**, **dependent_P_DET** et **non_dependent_P_DET** de l'entité **property_DET** sont utilisés pour coder les différentes sortes de propriétés (voir Article 7). L'entité **condition_DET** est utilisée pour les paramètres de contexte, **dependent_P_DET** est utilisée pour les caractéristiques dépendant d'un contexte et **non_dependent_P_DET** est utilisée pour les caractéristiques produit.

NOTE 2 L'attribut **depends_on** de l'entité **dependent_P_DET** est utilisé pour coder l'attribut "Condition" des propriétés.

5.9.6 Class_value_assignment

Les propriétés de valeur d'une classe sont les propriétés qui ne peuvent pas être assignées individuellement à une instance d'une classe, mais peuvent seulement être assignées pour toutes les instances appartenant à une classe. Ces propriétés sont déclarées en étant incluses dans la liste de **sub_class_properties** d'une entité **item_class**. Ensuite, une telle propriété peut avoir une valeur qui lui est assignée pour toutes les instances de toute **item_class** qui est une sous-classe où la propriété de valeur d'une classe est déclarée ou dans cette classe elle-même. Une valeur d'une propriété de valeur d'une classe est assignée à une **item_class** par une **class_value_assignment** référencée par l'attribut **class_constant_values** de cette classe.

NOTE Des propriétés de valeur d'une classe peuvent être de n'importe quel type de données.

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
(*
```

Définitions des attributs:

super_class_defined_property: la référence à la propriété (définie dans la classe ou dans n'importe laquelle de ses superclasses comme appartenant à l'ensemble des **sub_class_properties**) à laquelle la valeur **assigned_value** est assignée.

assigned_value: la valeur assignée à la propriété, valide pour toute la classe référençant cette instance de **class_value_assignment** dans son ensemble de **class_constant_values**, et toutes ses sous-classes.

Proposition formelle:

WR1: la valeur assignée à la **super_class_defined_property** doit avoir un type compatible avec celui du domaine des valeurs de la **super_class_defined_property**.

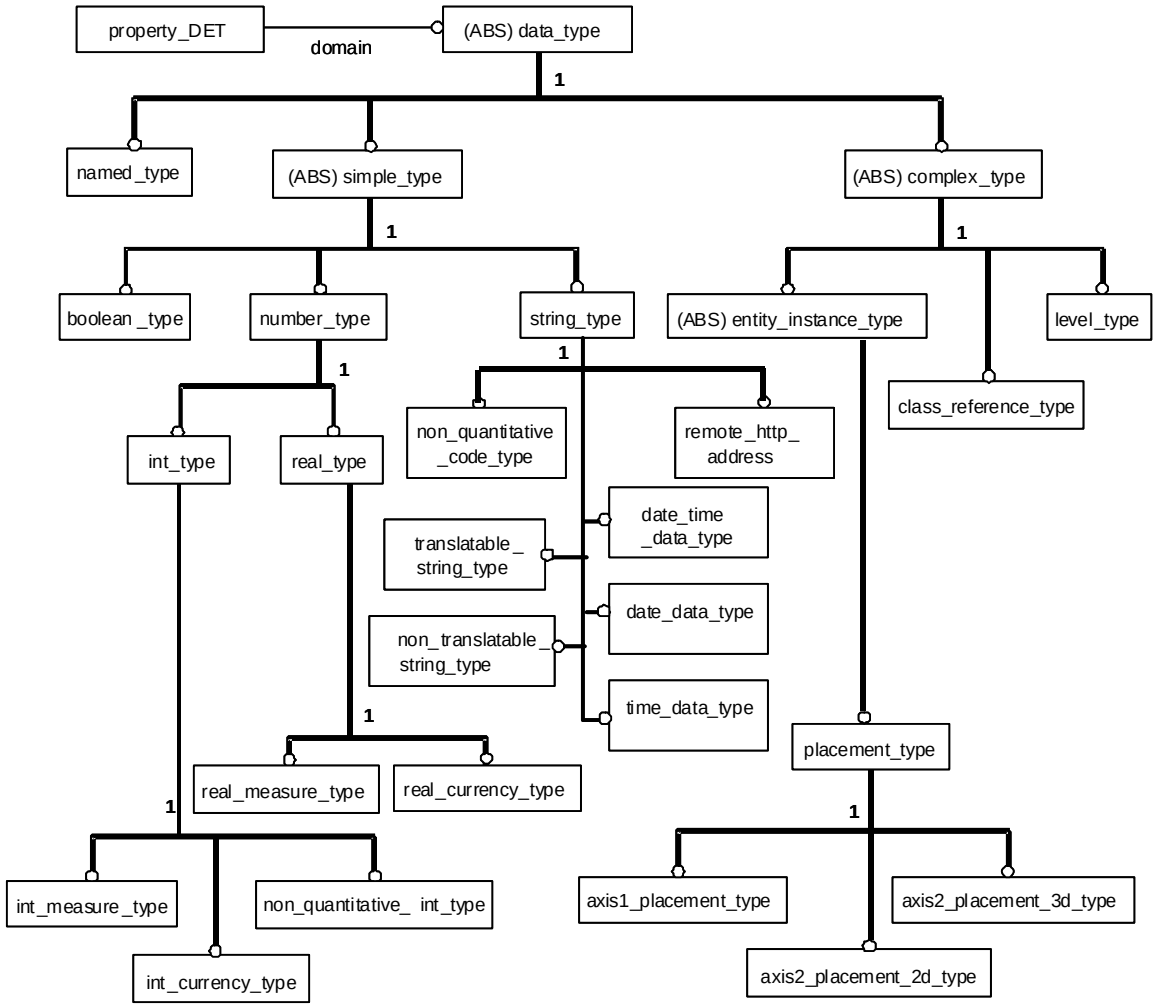


Figure 11 – Hiérarchie d'entités pour le système de types

5.10 Données de domaine: le système de types

5.10.1 Généralités

Le présent article contient des définitions pour la représentation des types de données de **property_DET**. La Figure 11 présente, sous la forme d'un modèle de planification, les grandes lignes d'une hiérarchie d'entités pour les types de données.

Contrairement aux autres éléments de dictionnaire (Fournisseurs, Classes, Propriétés), une identification avec le concept d'unité sémantique de base n'est pas obligatoire pour **data_type**, car elle sera directement attachée à **property_DET** dans un grand nombre de cas et, donc, n'a pas besoin d'une identification. Cependant, les entités **data_type_BSU** et **data_type_element** permettent une identification unique lorsque celle-ci est adaptée. Elle permet la prise en charge de la réutilisation de la même définition type dans une autre définition de **property_DET**, même à l'extérieur du fichier physique courant.

5.10.2 Détail de structure

5.10.2.1 Data_type_BSU

L'entité data_type_BSU permet la prise en charge de l'identification des 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
(*
```

Définitions des attributs:

code: pour permettre l'identification unique du type de données sur toutes les ontologies définies par le même fournisseur **name_scope.defined_by**.

name_scope: la référence à la classe à laquelle ou en dessous de laquelle l'élément type de données est disponible comme référence par l'attribut **defined_types**.

absolute_id: l'identification unique de cette propriété.

defining_class: les classes déclarant **data_type** comme disponible à l'emploi pour la description d'un produit.

Propositions formelles:

WR1: la classe utilisée dans l'attribut **name_scope** est une superclasse de celle où ce **data_type** est définie.

NOTE L'attribut **name_scope** est utilisé pour coder la référence à une classe à laquelle le type de données connexe appartient. Celui-la même, à côte de l'entité **data_type_element** (voir ci-dessous), fait partie du codage de l'attribut de classe "Visible Types".

5.10.2.2 Data_type_element

L'entité **data_type_element** décrit les types d'élément d'un dictionnaire. Noter qu'il n'est pas nécessaire dans chaque cas d'avoir BSU et **dictionary_element** pour un certain **data_type**, car une **property_DET** peut se référer directement à **data_type**. L'utilisation de la relation BSU est seulement nécessaire lorsqu'un fournisseur souhaite se référer au même type dans un fichier physique différent.

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
(*
```

Définitions des attributs:

identified_by: la BSU qui identifie **data_type_element** décrite.

names: les noms qui permettent la description de **data_type_element** définie.

type_definition: la description du type transporté par **data_type_element**.

NOTE L'attribut **identified_by** redéclaré est utilisé pour coder la référence à la BSU, à laquelle ce **data_type_element** est reliée. Celui-la-même qui, à côte de l'entité **data_type_BSU** (voir ci-dessus), est utilisé pour coder l'attribut de classe "Visible types".

5.10.3 Le système de types

5.10.3.1 Data_type

L'entité **data_type** sert de supertype commun pour les entités utilisées pour indiquer le type du DET associé.

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
(*
```

Définitions des attributs:

constraints: l'ensemble des contraintes du domaine qui limitent le domaine des valeurs pour le type de données.

NOTE Il convient que chaque contrainte de domaine dans l'attribut **constraints** soit satisfaite. Ainsi, l'attribut **constraints** est donc une conjonction de contraintes

Proposition formelle:

WR1: l'ensemble des contraintes du domaine doit définir des restrictions qui soient compatibles avec le domaine des valeurs du type de données.

5.10.3.2 Simple_type

L'entité **simple_type** sert de supertype commun pour les entités utilisées pour indiquer un type simple du DET associé.

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
(*
```

Définitions des attributs:

value_format: le codage facultatif du format des valeurs pour des propriétés.

NOTE 1 L'attribut **value_format** de l'entité **simple_type** est utilisé pour coder l'attribut "Value format" des propriétés.

NOTE 2 Si une **string_pattern_constraint** quelconque s'applique à la valeur d'un type simple, elle prévaut sur le **value_format**.

5.10.3.3 Number_type

L'entité **number_type** permet la prise en charge des valeurs des DET qui sont du 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

L'entité **int_type** permet la prise en charge des valeurs des DET qui sont du 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

L'entité **int_measure_type** permet la prise en charge des valeurs des DET qui sont des mesures de type INTEGER. Elle spécifie une **unit**, ou un identificateur d'unité (**unit_id**), dans laquelle sont exprimées les valeurs échangées comme "single integer" (entier simple). Elle peut également spécifier des unités de substitution, ou des identificateurs d'unités de substitution, qu'il est autorisé d'utiliser lorsque chaque valeur est explicitement associée à son unité.

NOTE 1 Soit un attribut **unit**, soit un attribut **unit_id** est obligatoire. S'ils sont fournis tous les deux, l'attribut **unit** prévaut.

NOTE 2 Lorsque les attributs **alternative_units** et **alternative_unit_ids** sont donnés tous les deux, ils ont tous les deux la même taille et l'attribut **alternative_units** prévaut.

NOTE 3 Les **dic_unit_identifier** utilisés dans les attributs **unit_id** et **alternative_unit_ids** sont des identificateurs d'unités qui peuvent être réduits en une **dic_unit** à partir d'un serveur ISO/TS 29002-20.

NOTE 4 Chaque **dic_unit** définie dans l'attribut **alternative_units** et chaque entité **dic_unit** identifiée dans l'attribut **alternative_unit_ids** doivent être associées à un attribut **string_representation**, dont l'attribut **text_representation** peut être utilisé pour caractériser l'unité de remplacement utilisée au niveau d'une instance.

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
(*
```

Définitions des attributs:

unit: l'unité de référence par défaut associée à la valeur de **int_measure_type**.

alternative_units: la liste d'autres unités qui peuvent être utilisées pour exprimer la valeur de **int_measure_type**.

NOTE 5 L'ordre de la liste est utilisé pour assurer que les **alternative_units** et **alternative_unit_ids**, s'ils existent tous les deux, définissent la même unité dans le même ordre.

unit_id: l'identificateur d'unité de référence par défaut associée à la mesure décrite.

NOTE 6 L'attribut **unit** et l'attribut **unit_id** sont utilisés tous les deux pour coder l'attribut "Unit" des propriétés. Lorsqu'ils sont fournis tous les deux, **unit** prévaut.

NOTE 7 Si la valeur d'une propriété dont le domaine est **int_measure_type** est échangée comme étant un nombre entier simple, cela signifie que cette valeur est exprimée dans l'unité de mesure **unit** ou **unit_id**.

alternative_unit_ids: la liste des identificateurs d'autres unités qui peuvent être utilisées pour exprimer la valeur de **int_measure_type**.

NOTE 8 Lorsque la valeur d'une propriété dont le domaine est une **int_measure_type** est évaluée dans une unité soit définie au moyen de l'attribut **alternative_units**, soit identifiée au moyen de l'attribut **alternative_unit_ids**, sa valeur ne peut pas être représentée comme étant un entier simple. Il est nécessaire de la représenter sous forme d'une paire (valeur, unité).

Propositions formelles:

WR1: l'un des deux attributs **unit** et **unit_id** doit exister.

WR2: Si les attributs **alternative_units** et **alternative_unit_ids** existent tous les deux, ils doivent avoir la même longueur.

WR3: chaque **dic_unit** dans **alternative_units** doit avoir un attribut **string_representation**.

Propositions informelles:

IP1: Les **dic_unit_identifier**s utilisés dans les attributs **unit_id** et **alternative_unit_ids** doivent être résolus en une entité **dic_unit** à partir d'un serveur ISO/TS 29002-20 existant.

IP2: lorsque les attributs **unit** et **unit_id** sont fournis tous les deux, ils doivent définir la même unité.

IP3: lorsque les attributs **alternative_units** et **alternative_unit_ids** sont fournis tous les deux, ils doivent définir la même liste d'unités, et ce, dans le même ordre.

IP4: lorsque l'attribut **alternative_unit_ids** est fourni, toutes les unités que l'attribut identifie doivent se résoudre en une entité **dic_unit** qui a un attribut **string_representation**.

5.10.3.6 Int_currency_type

L'entité **int_currency_type** permet la prise en charge des valeurs des DET qui sont des monnaies entières.

EXPRESS specification:

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

Définitions des attributs:

currency: le code associé de la monnaie décrite conforme à l'ISO 4217. S'il est absent, le code de monnaie doit être échangé ensemble avec les données (valeurs).

5.10.3.7 Non_quantitative_int_type

L'entité **non_quantitative_int_type** est un type énumération dans lequel des éléments de l'énumération sont représentés avec une valeur INTEGER (voir aussi l'entité **non_quantitative_code_type** et la 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
(*
```

Définitions des attributs:

domain: l'ensemble des valeurs énumérées décrites dans l'entité **value_domain**.

Propositions formelles:

WR1: les valeurs associées à la liste de **domain.its_values** ne doivent pas contenir un **value_code_type**.

5.10.3.8 Real_type

L'entité **real_type** permet la prise en charge des valeurs des DET qui sont du 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

L'entité **real_measure_type** permet la prise en charge des valeurs des DET qui sont des mesures de type REAL. Elle spécifie une **unit**,ou un identificateur d'unité, dans laquelle sont exprimées les valeurs échangées comme étant "single real" (réelles simples). Elle peut également spécifier des unités de substitution, ou des identificateurs d'unités de substitution, qu'il est autorisé d'utiliser lorsque chaque valeur est explicitement associée à son unité.

NOTE 1 Soit un attribut **unit**, soit un attribut **unit_id** est obligatoire. S'ils sont fournis tous les deux, l'attribut **unit** prévaut.

NOTE 2 Lorsque les attributs **alternative_units** et **alternative_unit_ids** sont donnés tous les deux, ils ont tous les deux la même taille et l'attribut **alternative_units** prévaut.

NOTE 3 Les **dic_unit_identifier** utilisés dans les attributs **unit_id** et **alternative_unit_ids** sont des identificateurs d'unités qui peuvent se résoudre en une entité **dic_unit** à partir d'un serveur ISO/TS 29002-20.

NOTE 4 Chaque **dic_unit** définie dans l'attribut **alternative_units** et chaque **dic_unit** identifiée dans l'attribut **alternative_unit_ids** doivent être associées à un attribut **string_representation**, dont l'attribut **text_representation** peut être utilisé pour caractériser l'unité de remplacement utilisée au niveau d'instance.

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
(*
```

Définitions des attributs:

unit: l'unité de référence par défaut associée à la valeur de **real_measure_type**.

alternative_units: la liste d'autres unités qui peuvent être utilisées pour exprimer la valeur de **real_measure_type**.

NOTE 5 L'ordre de la liste est utilisé pour assurer que les **alternative_units** et **alternative_unit_ids**, s'ils existent tous les deux, définissent la même unité dans le même ordre.

unit_id: l'identificateur d'unité de référence par défaut associée à la mesure décrite.

NOTE 6 L'attribut **unit** et l'attribut **unit_id** sont utilisés tous les deux pour coder l'attribut "Unit" pour des propriétés. Lorsqu'ils sont fournis tous les deux, **unit** prévaut.

NOTE 7 Si la valeur d'une propriété dont le domaine est **real_measure_type** est échangée comme étant un nombre réel simple, cela signifie que cette valeur est exprimée dans l'unité de mesure **unit** ou **unit_id**.

alternative_unit_ids: la liste des identificateurs d'autres unités qui peuvent être utilisées pour exprimer la valeur de **real_measure_type**.

NOTE 8 Lorsque la valeur d'une propriété dont le domaine est un **real_measure_type** est évaluée dans une unité soit définie au moyen de l'attribut **alternative_units**, soit identifiée au moyen de l'attribut **alternative_unit_ids**, sa valeur ne peut pas être représentée comme étant un réel simple. Elle sera représentée sous forme d'une paire (valeur, unité).

Propositions formelles:

WR1: l'un des deux attributs **unit** et **unit_id** doit exister.

WR2: Si les attributs **alternative_units** et **alternative_unit_ids** existent tous les deux, ils doivent avoir la même longueur.

WR3: chaque **dic_unit** dans l'attribut **alternative_units** doit avoir un attribut **string_representation**.

Propositions informelles:

IP1: Les **dic_unit_identifiers** utilisés dans les attributs **unit_id** et **alternative_unit_ids** doivent se résoudre en une **dic_unit** à partir d'un serveur ISO/TS 29002-20 existant.

IP2: lorsque les attributs **unit** et **unit_id** sont fournis tous les deux, ils doivent définir la même unité.

IP3: lorsque les attributs **alternative_units** et **alternative_unit_ids** sont fournis tous les deux, ils doivent définir la même liste d'unités, et ce, dans le même ordre.

IP4: lorsque l'attribut **alternative_unit_ids** est fourni, toutes les unités que l'attribut identifie doivent se résoudre en une entité **dic_unit** qui a un attribut **string_representation**.

5.10.3.10 Real_currency_type

L'entité **real_currency_type** définit les monnaies réelles.

EXPRESS specification:

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

Définitions des attributs:

currency: le code associé de la monnaie décrite conforme à l'ISO 4217. S'il est absent, le code de monnaie doit être échangé en même temps que les données (valeurs).

5.10.3.11 Rational_type

L'entité **rational_type** permet la prise en charge de valeurs des DET qui sont du type rationnel.

NOTE Dans l'ISO 13584-32, les valeurs rationnelles sont représentées par trois éléments XML entiers: la partie entière, le numérateur et le dénominateur. Dans l'ISO 13584-24:2003, les valeurs rationnelles sont représentées par une matrice de trois nombres entiers: la partie entière, le numérateur et le dénominateur.

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

L'entité **rational_measure_type** permet la prise en charge des valeurs des DET qui sont des mesures de type RATIONAL.

EXEMPLE Diamètre de vis: 4 1/8 pouces.

L'entité **rational_measure_type** spécifie une **unit**, ou un identificateur d'unité, dans laquelle sont exprimées les valeurs échangées comme "rational" (rationnelles). Elle peut également spécifier des unités de substitution, ou des identificateurs d'unités de substitution, qu'il est autorisé d'utiliser lorsque chaque valeur est explicitement associée à son unité.

NOTE 1 Soit un attribut **unit**, soit un attribut **unit_id** est obligatoire. S'ils sont fournis tous les deux, l'attribut **unit** prévaut.

NOTE 2 Lorsque les attributs **alternative_units** et **alternative_unit_ids** sont donnés tous les deux, ils ont tous les deux la même taille et l'attribut **alternative_units** prévaut.

NOTE 3 Les **dic_unit_identifier** utilisés dans les attributs **unit_id** et **alternative_unit_ids** sont des identificateurs d'unités qui peuvent se résoudre en une entité **dic_unit** à partir d'un serveur ISO/TS 29002-20.

NOTE 4 Chaque **ic_unit** définie dans l'attribut **alternative_units** et chaque **dic_unit** identifiée dans l'attribut **alternative_unit_ids** sont associées à un attribut **string_representation**, dont l'attribut **text_representation** peut être utilisé pour caractériser l'unité de remplacement utilisée au niveau d'instance.

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
(*
```

Définitions des attributs:

unit: l'unité de référence par défaut associée à la valeur de **rational_measure_type**.

alternative_units: la liste d'autres unités qui peuvent être utilisées pour exprimer la valeur de **rational_measure_type**.

NOTE 5 L'ordre de la liste est utilisé pour assurer que les **alternative_units** et **alternative_unit_ids**, s'ils existent tous les deux, définissent la même unité dans le même ordre.

unit_id: l'identificateur d'unité de référence par défaut associée à la mesure décrite.

NOTE 6 L'attribut **unit** et l'attribut **unit_id** sont utilisés tous les deux pour coder l'attribut "Unit" pour des propriétés. Lorsqu'ils sont fournis tous les deux, **unit** prévaut.

NOTE 7 Si la valeur d'une propriété dont le domaine est une **rational_measure_type** est échangée comme étant un nombre rationnel simple, cela signifie que cette valeur est exprimée dans l'unité de mesure **unit** ou **unit_id**.

alternative_unit_ids: la liste des identificateurs d'autres unités qui peuvent être utilisées pour exprimer la valeur de **rational_measure_type**.

NOTE 8 Lorsque la valeur d'une propriété dont le domaine est une **rational_measure_type** est évaluée dans une unité soit définie au moyen de l'attribut **alternative_units**, soit identifiée au moyen de l'attribut **alternative_unit_ids**, sa valeur ne peut pas être représentée comme étant un rationnel simple. Il est nécessaire de la représenter sous forme d'une paire (valeur, unité).

Propositions formelles:

WR1: l'un des deux attributs **unit** et **unit**_id doit exister.

WR2: si les attributs **alternative_units** et **alternative_unit_ids** existent tous les deux, ils doivent avoir la même longueur.

WR3: chaque **dic_unit** dans **alternative_units** doit avoir un attribut **string_representation**.

Propositions informelles:

IP1: Les **dic_unit_identifier**s utilisés dans les attributs **unit_id** et **alternative_unit_ids** doivent se résoudre en une entité **dic_unit** à partir d'un serveur ISO/TS 29002-20 existant.

IP2: lorsque les attributs **unit** et **unit_id** sont fournis tous les deux, ils doivent définir la même unité.

IP3: lorsque les attributs **alternative_units** et **alternative_unit_ids** sont fournis tous les deux, ils doivent définir la même liste d'unités, et ce, dans le même ordre.

IP4: lorsque l'attribut **alternative_unit_ids** est fourni, toutes les unités que l'attribut identifie doivent se résoudre en une entité **dic_unit** qui a un attribut **string_representation**.

5.10.3.13 boolean_type

L'entité **boolean_type** permet la prise en charge de valeurs des DET qui sont du type BOOLEAN (booléen).

EXPRESS specification:

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

5.10.3.14 String_type

L'entité **string_type** permet la prise en charge de valeurs des DET qui sont du type STRING (chaîne).

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

L'entité **translatable_string_type** permet la prise en charge de valeurs pour les DET qui sont du type STRING, mais qui sont supposées être représentés comme des chaînes différentes dans des langues différentes.

NOTE 1 Les valeurs de telles propriétés ne peuvent pas être utilisées pour l'identification de produit.

NOTE 2 Les valeurs de telles propriétés peuvent être soit une simple **string_value** lorsque **global_language_assignment** définit une langue courante, soit une **translated_string_value** lorsque chaque valeur de chaîne est associée à une langue.

NOTE 3 Deux valeurs de la même propriété dont l'entité **data_type** est **translatable_string_type** peuvent seulement être comparées pour vérifier leur égalité si la propriété correspondante comme **source_language** est définie comme partie de ses **administrative_data** et si ces valeurs sont disponibles dans cette **source_language**. Il n'est pas supposé que dans des langues différentes de cette **source_language**, la même signification soit représentée par la même chaîne.

EXPRESS specification:

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

5.10.3.16 Non_translatable_string_type

L'entité **non_translatable_string_type** permet la prise en charge des valeurs pour les DET qui sont du type STRING, mais qui sont représentées de la même façon dans n'importe quelle langue.

NOTE Les valeurs de telles propriétés peuvent être utilisées pour l'identification de produit.

EXPRESS specification:

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

5.10.3.17 URI_type

L'entité **URI_type** permet la prise en charge des valeurs des DET qui sont du type STRING, mais contiennent un URI[4] (Identificateur uniforme de ressource).

NOTE Une **URI_type** permet en particulier de fournir une URL[5] (Localisateur uniforme de ressource).

---

[4] URI = *Uniform Resource Identifier.*

[5] URL = *Uniform Ressource Locator*.

EXPRESS specification:

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

5.10.3.18 Date_time_data_type

L'entité date_time_data_type permet la prise en charge des valeurs des DET qui sont du type STRING, mais contient un instant spécifique spécifié conformément à une représentation particulière conforme à l'ISO 8601.

NOTE 1 Seul un sous-ensemble des représentations lexicales autorisées par l'ISO 8601 est autorisé pour les valeurs de **date_time_data_type**. Celui-ci est spécifié par IP1.

NOTE 2 La restriction ci-dessus des représentations de l'ISO 8601 est la même que celle définie par le Schéma XML.

EXPRESS specification:

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

Propositions informelles:

IP1: la valeur d'une propriété dont le type de données est **date_time_data_type** doit être conforme à la représentation lexicale suivante, qui est un sous-ensemble des représentations lexicales autorisées par l'ISO 8601. Cette représentation lexicale est le format étendu CCYY-MM-DDThh:mm:ss de l'ISO 8601 où "CC" représente l'ordre du siècle (l'ordre du premier siècle étant "00"), "YY" l'année, "MM" le mois et "DD" le jour, précédé d'un signe "-" facultatif en début pour indiquer un nombre négatif. Si le signe est omis, "+" est admis implicitement. La lettre "T" est le séparateur date/heure et "hh", "mm", "ss" représentent respectivement les heures, les minutes et les secondes. Des chiffres supplémentaires peuvent être utilisés pour augmenter la précision des secondes fractionnaires si cela est souhaité, c'est-à-dire que le format ss.ss... avec n'importe quel nombre de chiffres après le point séparateur est pris en charge. La partie des secondes fractionnaires est facultative; les autres parties de la forme lexicale ne sont pas facultatives. Pour prendre en charge des valeurs d'années supérieures à 9999, des chiffres supplémentaires peuvent être ajoutés à la gauche de la représentation. Des zéros en tête sont requis si la valeur de l'année possède moins de quatre chiffres; autrement, ils sont interdits. L'an 0000 est interdit. Le champ CCYY doit avoir au moins quatre chiffres, les champs MM, DD, SS, hh, mm et ss exactement deux chiffres chacun (en ne comptant pas les secondes fractionnaires); des zéros en tête doivent être utilisés si le champ risque d'avoir trop peu de chiffres. Cette représentation peut être immédiatement suivie d'un "Z" pour indiquer le Temps Universel Coordonné (TUC) ou, pour indiquer le fuseau horaire, c'est-à-dire la différence entre l'heure locale et le Temps Universel Coordonné, immédiatement suivie d'un signe, + ou -, suivi de la différence par rapport au TUC représentée sous la forme hh:mm (note: la partie des minutes est requise). Voir l'ISO 8601 pour les détails concernant les valeurs légales dans les divers champs. Si le fuseau horaire est inclus, les heures et aussi les minutes doivent être présentes.

EXEMPLE Pour indiquer 1:20 de l'après-midi du 31 mai 1999 pour l'Heure Normale de l'Est qui est en retard de 5 h sur le Temps Universel Coordonné (TUC), on écrira: `1999-05-31T13:20:00-05:00`.

5.10.3.19 Date_data_type

L'entité **date_data_type** permet la prise en charge des valeurs des DET qui sont du type STRING, mais contient une date calendaire spécifique spécifiée conformément à une représentation particulière conforme à l'ISO 8601.

NOTE 1 Seul un sous-ensemble des représentations lexicales autorisées par l'ISO 8601 est autorisé pour les valeurs de **date_data_type**. Celui-ci est spécifié par IP1.

NOTE 2 La restriction ci-dessus des représentations de l'ISO 8601 est la même que celle définie par le Schéma XML.

EXPRESS specification:

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

Propositions informelles:

IP1: la valeur d'une propriété dont le type de données est **date_data_type** doit être conforme à la représentation lexicale suivante, qui est un sous-ensemble des représentations lexicales autorisées par l'ISO 8601. La représentation lexicale pour **date_data_type** est la représentation lexicale réduite (tronquée à droite) pour **date_time_data_type**: CCYY-MM-DF. Aucune troncature à gauche n'est permise. A la suite un qualificateur facultatif de fuseau horaire est permis comme dans le cas de **date_time_data_type**. Pour prendre en charge les valeurs d'années situées à l'extérieur de la plage 0001 à 9999, des chiffres supplémentaires peuvent être ajoutés à la gauche de cette représentation et un signe "-" placé devant est autorisé.

EXEMPLE 1999-05-31 est la représentation **date_data_type** du: 31 mai 1999.

5.10.3.20 Time_data_type

L'entité **time_data_type** permet la prise en charge des valeurs des DET qui sont du type STRING, mais contient une heure spécifique spécifiée conformément à une représentation particulière conforme à l'ISO 8601. Une valeur de **time_data_type** représente un instant temporel qui se reproduit chaque jour.

NOTE 1 Seul un sous-ensemble des représentations lexicales autorisées par l'ISO 8601 est autorisé pour les valeurs de **time_data_type**. Celui-ci est spécifié par IP1.

NOTE 2 La restriction ci-dessus des représentations de l'ISO 8601 est la même que celle définie par le Schéma XML.

NOTE 3 Sachant que la représentation lexicale autorise un indicateur facultatif de fuseau horaire, les valeurs de **time_data_type** sont partiellement ordonnées, car il peut être impossible de déterminer l'ordre de deux valeurs dont l'une a un fuseau horaire et l'autre non.

EXPRESS specification:

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

Propositions informelles:

IP1: la valeur d'une propriété dont le type de données est **time_data_type** doit être conforme à la représentation lexicale suivante, qui est un sous-ensemble des représentations lexicales autorisées par l'ISO 8601. La représentation lexicale pour **time_data_type** est la représentation lexicale tronquée à gauche pour **date_time_data_type** hh:mm:ss.sss avec un indicateur facultatif de fuseau horaire suivant.

EXEMPLE 13:20:00-05:00 est la représentation **time_data_type** de: 1.20 de l'après-midi pour l'Heure Normale de l'Est qui est en retard de 5 h sur le Temps Universel Coordonné (TUC).

5.10.3.21 Non_quantitative_code_type

L'entité **non_quantitative_code_type** est un type énumération dans lequel des éléments de l'énumération sont représentés avec une valeur STRING (voir aussi ENTITY **non_quantitative_int_type** et la 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
(*
```

Définitions des attributs:

domain: l'ensemble des valeurs énumérées décrites dans **value_domain**.

Propositions formelles:

WR1: les valeurs associées à la liste de **domain.its_values** ne doivent contenir que des éléments de type **value_code_type**.

5.10.3.22 Complex_type

L'entité **complex_type** permet la prise en charge de la définition des types dont les valeurs sont représentées comme étant des instances EXPRESS.

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

L'entité **level_type** est un type complexe indiquant que la valeur d'une propriété est constituée d'une ou au maximum de quatre valeurs de mesures réelles ou entières, chacune étant qualifiée par un indicateur particulier spécifiant la signification de la valeur.

NOTE 1 Les valeurs d'instances d'un **level_type** contiennent des valeurs pour et seulement pour les indicateurs spécifiées dans l'attribut **levels**. Lorsque certaines de ces valeurs ne sont pas disponibles, elles sont représentées par des **null_value**.

EXEMPLE Si **level_type** spécifie que seules les valeurs *minimum* (minimale) et *typical* (type) doivent être fournies comme "integer" (entières), toute instance contient des valeurs entières (ou **null_value**) seulement pour les niveaux *minimum* et *typical* de la valeur d'instance de **level_type**.

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
(*
```

Définitions des attributs:

levels: la liste des indicateurs uniques qui spécifie quelles valeurs qualifiées doivent être associées à la propriété.

value_type: le type de données des valeurs qualifiées de **level_type**.

Propositions formelles:

WR1: le **SELF.value_type** doit être du type **int_measure_type** ou du type **real_measure_type**.

WR2: l'ordre du premier et du deuxième **level** (niveau), si les deux existent, doit suivre l'ordre d'énumération du type **level**.

WR3: l'ordre du deuxième et du troisième **level** (niveau), si les deux existent, doit suivre l'ordre d'énumération du type **level**.

WR4: l'ordre du troisième et du quatrième **level** (niveau), si les deux existent, doit suivre l'ordre d'énumération du type **level**.

5.10.3.24 Level

L'entité **level** spécifie les divers indicateurs qui peuvent être utilisés pour qualifier une valeur d'un **level_type**.

Ces indicateurs sont comme suit:

– minimum (minimum): valeur la plus basse spécifiée d'une grandeur, établie pour un ensemble spécifié de conditions de fonctionnement pour lesquelles un composant, un dispositif ou un matériel est exploitable et s'exécute conformément aux exigences spécifiées;

– nominal (nominale): valeur d'une grandeur utilisée pour désigner et identifier un composant, un dispositif, un matériel ou un système;

– typical (typique): valeur d'une grandeur généralement rencontrée, utilisée à des fins de spécification, établie pour un ensemble spécifié de conditions de fonctionnement d'un composant, dispositif, matériel ou système;

– maximum (maximale): valeur la plus élevée spécifiée d'une grandeur, établie pour un ensemble spécifié de conditions de fonctionnement pour lesquelles un composant, un dispositif ou un matériel est exploitable et s'exécute conformément aux exigences spécifiées.

NOTE 1 La valeur nominale est généralement une valeur arrondie.

EXEMPLE Une batterie de voiture de 12 V (nominale) contient 6 éléments ayant une tension typique d'environ 2,2 V chacun, ce qui donne une tension typique pour la batterie d'environ 13,5 V. En charge, la tension peut atteindre une valeur maximale d'environ 14,5 V, mais elle est considérée comme étant complètement déchargée lorsque la tension chute en dessous d'une valeur minimale de 12,5 V.

NOTE 2 Les valeurs qui sont données pour une propriété de valeur de type niveau sont spécifiées dans le dictionnaire.

NOTE 3 Il est conseillé de restreindre l'utilisation du type niveau aux DET qui sont applicables dans des domaines pour lesquels l'indication de valeurs multiples pour une seule et même caractéristique est reconnue comme une pratique courante et demandée, comme cela se vérifie dans l'industrie des composants électroniques.

EXPRESS specification:

```
*)
TYPE level = ENUMERATION OF(
     min,     (* la valeur minimale de la grandeur physique *)
     nom,     (* la valeur nominale de la grandeur physique *)
     typ,     (* la valeur typique de la grandeur physique *)
     max);    (* la valeur minimale de la grandeur physique *)
END_TYPE; -- level
(*
```

5.10.3.25 Class_reference_type

L'entité **class_reference_type** permet la prise en charge des valeurs des DET qui sont représentées comme étant des instances d'une **class**. Elle est utilisée, en particulier, pour la description d'assemblages ou pour décrire le matériau dont est constitué un composant (ou une partie de composant).

EXPRESS specification:

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

```
(*
```

Définitions des attributs:

domain: l'entité **class_BSU** renvoyant à la **class** représentant le type décrit.

5.10.3.26 Entity_instance_type

L'entité **entity_instance_type** permet la prise en charge des valeurs des DET qui sont représentées comme étant des instances de certains types de données d'entité EXPRESS. Un attribut **type_name** permet de spécifier quels types de données sont autorisés. Ces types de données sont des chaînes contenues dans un ensemble. Cet attribut, conjointement à la fonction EXPRESS TYPEOF appliquée à la valeur, permet une vérification poussée des types et le polymorphisme. Cette entité sera sous-typée pour certains types de données dont l'emploi est autorisé dans le schéma du dictionnaire.

NOTE Lorsqu'une entité EXPRESS est la valeur d'un DET donné dont le type de données est une entité **entity_instance_type**, il est possible de vérifier le typage correct en appliquant la fonction EXPRESS TYPEOF sur cette valeur de DET et de comparer les résultats de cette application de TYPEOF aux chaînes contenues dans l'attribut d'ensemble **type_name**.

EXPRESS specification:

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

Définitions des attributs:

type_name: l'ensemble des chaînes qui décrivent, dans le format de la fonction EXPRESS TYPEOF, les noms des types de données d'une entité EXPRESS qui doivent appartenir au résultat de la fonction EXPRESS TYPEOF lorsqu'elle est appliquée à une valeur qui référence l'entité présente comme son type de données.

5.10.3.27 Placement_type

L'entité **placement_type** permet la prise en charge des valeurs des DET qui sont des instances du type de données des entités **placement**.

NOTE 1 Les entités **placement** sont importées de l'ISO 10303-42. Conformément à l'ISO 10303-42, une instance de **placement** peut seulement exister si elle est reliée à une instance de **geometric_representation_context** dans une certaine instance de **representation**. Par conséquent, si certaines propriétés de classe ont des instances de **placement** comme valeurs, cette classe contient un **geometric_representation_context** (qui définit le contexte de ces placements) et une **representation** (qui rassemble ces **placements** avec leur contexte). Les entités **geometric_representation_context** et **representation** n'étant pas importées dans la présente partie de la CEI 61360, les entités **placement** ne peuvent pas être utilisées lorsque seuls des schémas de l'ISO 13584-42 sont utilisés. Ces entités sont introduites comme ressources pour les autres parties de l'ISO 13584.

NOTE 2 Les entités **placement** sont utilisées en particulier dans l'ISO 13584-32 (OntoML) et dans l'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
(*
```

Propositions formelles:

WR1: la chaîne "GEOMETRY_SCHEMA.PLACEMENT" doit être contenue dans l'ensemble défini par l'attribut **SELF\entity_instance_type.type_name**.

5.10.3.28 Axis1_placement_type

L'entité **axis1_placement_type** permet la prise en charge des valeurs des DET qui sont des instances du type de données des entités **axis1_placement** (voir l'ISO 10303-42 pour les détails).

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
(*
```

Propositions formelles:

WR1: la chaîne "GEOMETRY_SCHEMA.AXIS1_PLACEMENT" doit être contenue dans l'ensemble défini par l'attribut **SELF\entity_instance_type.type_name**.

5.10.3.29 Axis2_placement_2d_type

L'entité **axis2_placement_2d_type** permet la prise en charge des valeurs des DET qui sont des instances du type de données des entités **axis2_placement_2d** (voir l'ISO 10303-42 pour les détails).

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
(*
```

Propositions formelles:

WR1: la chaîne "GEOMETRY_SCHEMA.AXIS2_PLACEMENT_2D" doit être contenue dans l'ensemble défini par l'attribut **SELF\entity_instance_type.type_name**.

5.10.3.30 Axis2_placement_3d_type

L'entité **axis2_placement_3d_type** permet la prise en charge des valeurs des DET qui sont des instances du type de données des entités **axis2_placement_3d** (voir l'ISO 10303-42 pour les détails).

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
(*
```

Propositions formelles:

WR1: la chaîne "GEOMETRY_SCHEMA.AXIS2_PLACEMENT_3D" doit être contenue dans l'ensemble défini par l'attribut **SELF\entity_instance_type.type_name**.

5.10.3.31 Named_type

L'entité **named_type** permet la prise en charge de la référence à d'autres types par le biais du mécanisme des BSU.

EXPRESS specification:

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

Définitions des attributs:

referred_type: la BSU identifiant **data_type** à laquelle la présente entité se réfère.

5.10.4 Valeurs

Le présent article contient des définitions pour des types d'élément de données non quantitatifs (voir les entités **non_quantitative_int_type** et **non_quantitative_code_type**).

La Figure 12 présente, sous la forme d'un modèle de planification, les grandes lignes des principales données associées à des types d'élément de données non quantitatif.

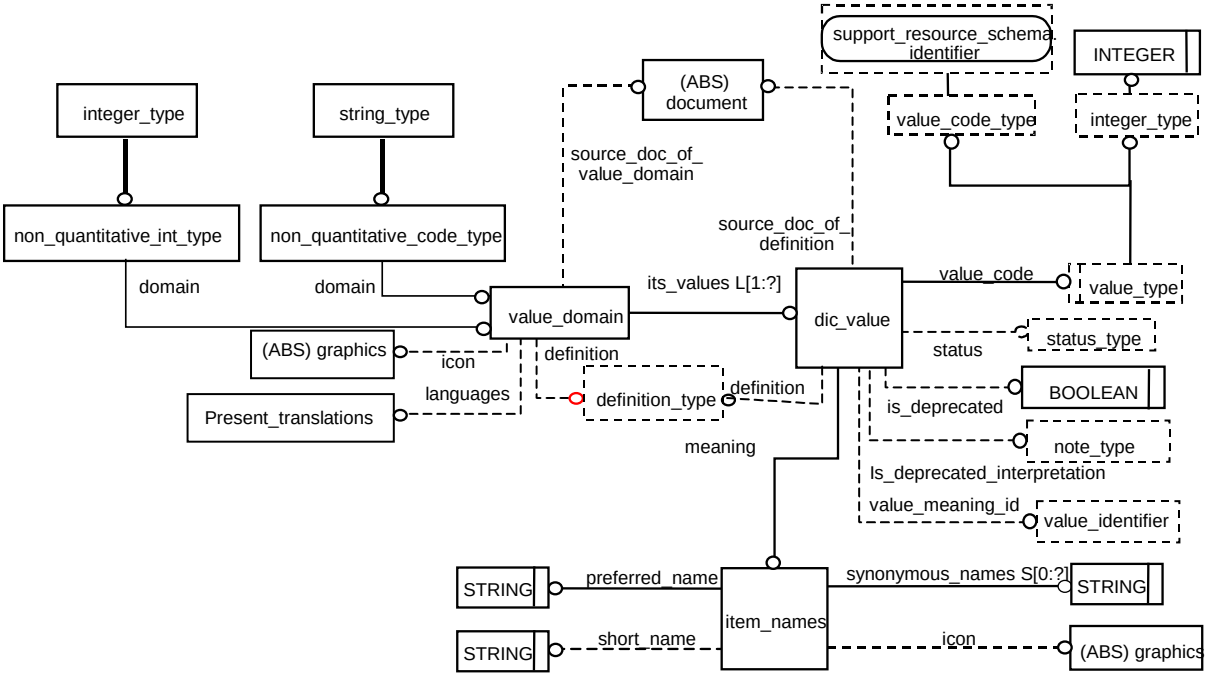


Figure 12 – Vue d'ensemble des types d'élément de données non quantitatif

5.10.5 Détail de structure

5.10.5.1 Value_domain

L'entité **value_domain** décrit l'ensemble des valeurs autorisées pour un type d'élément de données non quantitatif.

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
(*
```

Définitions des attributs:

its_values: la liste d'énumération des valeurs contenues dans le domaine décrit.

source_doc_of_value_domain: le document décrivant le domaine associé à l'entité **value_domain** décrite.

languages: les langues facultatives dans lesquelles les traductions sont assurées.

terms: la liste des **item_names** pour permettre la prise en charge du lien CEI 61360 avec le dictionnaire des termes.

definition: le texte facultatif décrivant **value_domain**.

icon: une **icon** (icône) facultative qui représente graphiquement la description associée à **value_domain**.

Propositions formelles:

WR1: Si les significations de valeurs sont données dans plus d'une langue, l'ensemble des langues utilisées doit être le même que celui de l'ensemble des valeurs.

WR2: les codes de valeurs doivent être uniques au sein de ce type de données.

WR3: Si aucune **langue** n'est fournie, les significations des valeurs ne doivent avoir aucune langue assignée.

WR4: Si aucune **langue** n'est fournie, la définition des valeurs ne doit avoir aucune langue assignée.

5.10.5.2 Value_type

Chaque valeur d'un élément de données non quantitatif est associée à un code qui caractérise la valeur. Un **value_type** peut être soit un INTEGER, soit un **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

L'entité **dic_value** est l'une des valeurs de **value_domain**.

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
(*
```

Définitions des attributs:

value_code: le code associé à la valeur décrite. Il peut être soit un **value_code_type**, soit un **integer_type**.

meaning: la signification associée à cette valeur. Elle est fournie par des noms.

source_doc_of_definition: le document source facultatif dans lequel la valeur est définie.

definition: le texte facultatif décrivant **dic_value**.

status: un **status_type** qui définit l'état du cycle de vie de **dic_value**.

NOTE 1 Les valeurs admissibles d'un **status_type** ne sont pas normalisées. Elles sont définies pour chaque dictionnaire particulier par son fournisseur d'informations.

EXEMPLE 1 Un ensemble de valeurs admissibles pour le statut des articles proposés à la normalisation à une agence de maintenance de normes ISO est défini dans les directives ISO.

EXEMPLE 2 Un ensemble de valeurs admissibles pour le statut d'articles dans la norme de la base de données CEI est défini dans les directives CEI.

NOTE 2 Pour les entités **dic_value** qui ne sont pas encore distribuées pour insertion, la représentation de projets d'entités **dic_value** pourrait s'avérer utile.

EXEMPLE 3 Pour des besoins d'expérimentation avant validation.

NOTE 3 Si l'attribut **status** n'est pas fourni pour une **dic_value** et si l'utilisation de cette **dic_value** n'est pas déconseillée comme indiqué par un éventuel attribut **is_deprecated**, **dic_value** a le même statut de normalisation que le dictionnaire au complet. En particulier, si le dictionnaire est normalisé, cette **dic_value** est partie intégrante de l'édition courante de la norme.

is_deprecated: un Boolean qui spécifie, lorsqu'il est "true", que **dic_value** ne doit plus être utilisée.

NOTE 4 Lorsque l'attribut **is_deprecated** n'a aucune valeur, **dic_value** n'est pas déconseillée.

NOTE 5 Les entités **dic_value** déconseillées sont laissées dans les entités **value_domain** pour des raisons de compatibilité ascendante.

is_deprecated_interpretation: spécifie la justification de la dépréciation et comment il convient d'interpréter les valeurs de l'instance de l'élément déconseillé.

value_meaning_id: un **dic_value_identifier** qui est un identificateur global de **dic_value**, quelle que soit **value_domain** dans laquelle elle est incluse.

NOTE 6 Cet identificateur permet de réutiliser la même définition de **dic_value** dans divers domaines.

Proposition formelle:

WR1: lorsque **is_deprecated** existe, **is_deprecated_interpretation** doit exister.

Proposition informelle:

IP1: les valeurs de l'instance de l'élément **is_deprecated_interpretation** doivent être définies au moment où la décision de dépréciation a été prise.

5.10.5.4 Administrative_data

Une entité **administrative_data** enregistre des informations relatives au cycle de vie d'un élément du dictionnaire.

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
(*
```

Définitions des attributs:

status: un **status_type** qui définit l'état du cycle de vie de l'élément du dictionnaire.

NOTE 1 Les valeurs admissibles d'un **status_type** ne sont pas normalisées. Elles sont définies pour chaque dictionnaire particulier par son fournisseur d'informations.

EXEMPLE 1 Un ensemble de valeurs admissibles pour le statut des articles proposés à la normalisation à une agence de maintenance de normes ISO est défini dans les directives ISO.

EXEMPLE 2 Un ensemble de valeurs admissibles pour le statut d'articles dans la norme de la base de données CEI est défini dans les directives CEI.

NOTE 2 Pour les entités **dictionary_element** qui ne sont pas encore distribuées pour insertion, la représentation de projets d'entités **dictionary_element** pourrait s'avérer utile.

EXEMPLE 3 Pour des besoins d'expérimentation avant validation.

NOTE 3 Si l'attribut **status** n'est pas fourni et si l'utilisation de cette entité **dictionary_element** n'est pas déconseillée comme indiqué par un éventuel attribut **is_deprecated**, **dictionary_element** a le même statut de normalisation que le dictionnaire au complet. En particulier, si le dictionnaire est normalisé, cette entité **dictionary_element** est partie intégrante de l'édition courante de la norme.

translation: description des traducteurs responsables dans les diverses langues.

source_language: la langue dans laquelle **dictionary_element** était initialement définie et qui fournit la signification de référence en cas de discordance de traduction.

NOTE 4 Un dictionnaire peut contenir des entités **dictionary_element** dont les attributs **source_language** sont différents, par exemple parce qu'elles avaient été importées de dictionnaires différents. Il est de la responsabilité du fournisseur de données du dictionnaire de s'assurer que les informations relatives à ces divers éléments sont fournies dans la même langue ou dans les mêmes langues.

administrated_element: l'entité **dictionary_element** dont les données du cycle de vie sont enregistrées dans **administrative_data**.

Propositions formelles:

WR1: les langues de traduction associées à une **administrative_data** sont uniques.

WR2: l'attribut **source_language** est absent dans les langues de traduction associées à une **administrative_data**.

5.10.5.5 Translation_data

Une entité **translation_data** enregistre des informations relatives aux possibles traductions d'un élément du dictionnaire.

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
(*
```

Définitions des attributs:

language: la langue dans laquelle l'élément du dictionnaire est traduit.

NOTE 1 En cas de discordance entre la définition initiale d'un élément du dictionnaire et certaines de ses traductions, la signification réelle de l'élément du dictionnaire est celle de la langue de définition source.

responsible_translator: l'organisation responsable de la traduction dans l'élément langue.

translation_revision: le numéro de révision de la traduction correspondante.

NOTE 2 Le changement de **version** ou le changement de **revision** d'un élément du dictionnaire ne nécessite pas toujours un changement de leurs traductions. S'il n'y a pas de changement de traduction dû à un changement de **version** ou à un changement de **revision** d'un élément du dictionnaire, il convient que l'attribut **translation_revision** correspondant ne change pas. Cependant, tout changement d'une traduction impliquera le changement de l'attribut **translation_revision** correspondant.

date_of_current_translation: la date de la dernière révision de la traduction correspondante

belongs_to: l'entité **administrative_data** qui référence l'enregistrement de **translation_data**.

5.10.6 Extension pour les définitions d'unités de l'ISO 10303-41

5.10.6.1 Généralités

Le présent article définit les ressources pour la description d'unités dans un dictionnaire. Il étend les ressources définies dans l'ISO 10303-41.

5.10.6.2 Non_si_unit

L'entité **non_si_unit** étend le modèle d'unités de l'ISO 10303-41 pour permettre la prise en charge de la représentation des unités non SI qui ne dépendent pas du contexte ou qui ne sont pas basées sur une conversion (voir l'ISO 10303-41 pour les détails).

EXPRESS specification:

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

Définitions des attributs:

name: l'étiquette (**label**) utilisée pour désigner l'unité décrite.

5.10.6.3 Règle assert_ONEOF

La règle **assert_ONEOF** affirme qu'ONEOF est valide entre les sous-types suivants de **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

La représentation de base des unités est donnée dans une forme structurée conformément à l'ISO 10303-41. Mais comme l'un des objectifs de la consignation d'unités dans le dictionnaire est la présentation à l'utilisateur, la représentation structurée seule ne suffit pas. Elle doit être complétée par une représentation sous forme de chaînes.

Les présentes définitions permettent diverses possibilités:

– la fonction **string_for_unit** (voir 5.12) peut être utilisée. Pour une représentation donnée structurée d'une unité, elle retourne une chaîne comme représentation correspondant à celle utilisée dans l'Annexe B de la CEI 61360-1:2009;

– une chaîne comme représentation peut être donnée sous la forme d'un texte clair (entité **mathematical_string**, attribut **text_representation**);

– une représentation MathML peut être donnée pour permettre la prise en charge d'une présentation évoluée de l'unité comportant des indices et des exposants, etc. (entité **mathematical_string**, attribut **MathML_representation**).

L'entité **dic_unit** décrit une unité devant être consignée dans un dictionnaire.

EXPRESS specification:

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

Définitions des attributs:

structured_representation: représentation structurée, issue de l'ISO 10303-41, comprenant l'extension définie en 5.10.6.

string_representation: la fonction **string_for_unit** peut être utilisée pour calculer une chaîne comme représentation à partir de l'attribut **structured_representation**, pour le cas où aucun attribut **string_representation** ne serait présent.

NOTE L'attribut **structured_representation** de **dic_unit** est utilisé pour coder l'attribut d'une propriété "Unit".

5.11 Types de base et définitions des entités

5.11.1 Définitions des types de base

Le présent paragraphe contient les types de base et les définitions des entités qui ont été utilisées dans la partie principale du modèle. La section suivante contient les types de bases et les définitions des entités, classés par l'ordre alphabétique.

5.11.2 Détail de structure

5.11.2.1 Class_code_type

Le type **class_code_type** identifie les valeurs autorisées pour un code de classe.

EXPRESS specification:

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

Propositions formelles:

WR1: la longueur des valeurs correspondant au type **class_code_type** doit être inférieure ou égale à la longueur de **class_code_len** (à savoir 35).

5.11.2.2 Code_type

Le type **code_type** identifie les valeurs autorisées pour un type de code utilisé pour une identification.

NOTE Si le code est également destiné à être échangé en utilisant l'ISO/TS 29002-5, il est recommandé de satisfaire aux exigences définies par cette norme. Pour un code, seuls les "caractères sûrs" sont autorisés. Les caractères sûrs comprennent: lettres majuscules, chiffres, deux-points, points ou trait de soulignement. En outre, le caractère moins, à savoir '-', est autorisé pour des besoins particuliers.

EXPRESS specification:

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

Propositions formelles:

WR1: le caractère '#' ne doit pas être contenu dans une valeur de **code_type**. Le caractère '#' est utilisé pour concaténer des identificateurs, (voir: CONSTANT **sep_id**), ou un code et une version (voir: CONSTANT **sep_cv**).

WR2: les espaces sont interdits, afin de prévenir les problèmes de blancs en tête et en queue lors de la concaténation de codes.

WR3: un type **code_type** ne doit pas être une chaîne vide.

5.11.2.3 Currency_code

5.11.2.3.1 Généralités

Le type **currency_code** identifie les valeurs autorisées pour un code de monnaie. Ces valeurs sont définies conformément à l'ISO 4217.

EXEMPLE Les valeurs sont: "CHF" pour les francs suisses, "CNY" pour le yuan renminbi (chinois), "JPY" pour le yen (japonais), "SUR" pour le rouble soviétique, "USD" pour les dollars américains, "EUR" pour l'euro.

EXPRESS specification:

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

Propositions formelles:

WR1: la longueur d'une valeur de **currency_code** doit être égale à 3.

5.11.2.3.2 Data_type_code_type

Le type **data_type_code_type** identifie les valeurs autorisées pour un code de type de donnés.

EXPRESS specification:

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

```
(*
```

Propositions formelles:

WR1: la longueur d'une valeur de **data_type_code_type** doit être égale à la valeur **data_type_code_len** (à savoir 35).

5.11.2.3.3 Date_type

Le type **date_type** identifie les valeurs autorisées pour une date. Ces valeurs sont définies conformément à l'ISO 8601.

EXEMPLE "1994-03-21".

EXPRESS specification:

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

5.11.2.4 Definition_type

Le type **définition_type** identifie les valeurs autorisées pour une définition.

EXPRESS specification:

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

5.11.2.5 DET_classification_type

Le type **DET_classification_type** identifie les valeurs autorisées pour une classification de DET. Ces valeurs sont utilisées pour la classification DET conformément à l'ISO 80000/CEI 80000 (anciennement ISO 31).

EXPRESS specification:

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

Propositions formelles:

WR1: la longueur d'une valeur de **DET_classification_type** doit être égale à la valeur d'un **DET_classification_len** (à savoir 3).

5.11.2.6 Note_type

Le type **note_type** identifie les valeurs autorisées pour une note.

EXPRESS specification:

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

5.11.2.7 Pref_name_type

Le type **pref_name_type** identifie les valeurs autorisées pour un nom préférentiel.

EXPRESS specification:

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

Propositions formelles:

WR1: la longueur d'une valeur de **pref_name_type** ne doit pas dépasser la longueur de **pref_name_len** (à savoir 255).

5.11.2.8 Property_code_type

Le type **property_code_type** identifie les valeurs autorisées pour un code de propriété.

EXPRESS specification:

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

Propositions formelles:

WR1: la longueur d'une valeur de **property_code_type** ne doit pas dépasser la longueur de **property_code_len** (à savoir 35).

5.11.2.9 Remark_type

Le type **remark_type** identifie les valeurs autorisées pour une remarque.

EXPRESS specification:

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

5.11.2.10 Hierarchical_position_type

Le type **prefhierarchical_position_type** identifie les valeurs autorisées pour une position hiérarchique.

EXPRESS specification:

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

NOTE La représentation d'une position hiérarchique dans un type **hierarchical_position_type** est basée sur certaines conventions de codage. Cette convention de codage n'est pas définie par la présente partie de la CEI 61360.

5.11.2.11 Revision_type

Le type **revision_type** identifie les valeurs autorisées pour une révision.

NOTE Lorsqu'une nouvelle version est publiée, sa valeur de révision est mise à '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
(*
```

Propositions formelles:

WR1: la longueur d'une valeur de **revision_type** ne doit pas dépasser la longueur de **revision_len** (à savoir 3).

WR2: le **revision_type** doit contenir des chiffres uniquement et le nombre entier qu'il représente doit être un nombre entier naturel.

5.11.2.12 Short_name_type

Le type **short_name_type** identifie les valeurs autorisées pour un nom court.

EXPRESS specification:

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

Propositions formelles:

WR1: la longueur d'une valeur de **short_name_type** ne doit pas dépasser la longueur de **short_name_len** (à savoir 30).

5.11.2.13 Supplier_code_type

Le type **supplier_code_type** identifie les valeurs autorisées pour un code fournisseur.

NOTE Si le code fournisseur est également destiné à être échangé en utilisant l'ISO/TS 29002-5, les diverses parties du code fournisseur telles que définies par l'ISO 6523 (ICD, OI, OPI, OPIS et AI) sont séparées par '-'.

EXPRESS specification:

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

Propositions formelles:

WR1: la longueur d'une valeur de **supplier_code_type** ne doit pas dépasser la longueur de **supplier_code_len** (à savoir 149).

5.11.2.14 Syn_name_type

Le type **syn_name_type** identifie les valeurs autorisées pour un nom synonyme.

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
(*
```

Propositions formelles:

WR1: la longueur d'une valeur de **syn_name_type** ne doit pas dépasser la longueur de **syn_name_len** (à savoir 255).

5.11.2.15 Keyword_type

Le type **keyword_type** identifie les valeurs autorisées pour un mot-clé.

EXPRESS specification:

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

5.11.2.16 ISO_29002_IRDI_type

Le type **ISO_29002_IRDI_type** est un identificateur global qui identifie un article administré dans un registre. La structure de cet identificateur obéit à la syntaxe des identificateurs définie dans l'ISO/TS 29002-5.

NOTE 1 Un type **ISO_29002_IRDI_type** peut être utilisé pour n'importe quelle sorte d'informations envisagées dans l'ISO/TS 29002-5 et qui peuvent être associées à un identificateur IRDI. Trois cas spéciaux sont identifiés ci-dessous, car ils sont spécifiquement utilisés dans le schéma **ISO13584_IEC61360_dictionary_schema**: **constraint_identifier**, **dic_unit_identifier** et **dic_value_identifier.**

EXPRESS specification:

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

Propositions formelles:

WR1: comme spécifié dans l'ISO/TS 29002-5, la longueur de l'identificateur ne doit pas être supérieure à 290.

Propositions informelles:

IP1: l'identificateur doit satisfaire aux exigences spécifiées dans l'ISO/TS 29002-5 pour un "international registration data identifier" (IRDI "identificateur international des données d'enregistrement").

NOTE 2 Conformément à l'ISO/TS 29002-5, un IRDI consiste soit en une chaîne, qui ne contient pas le caractère ''#', pour identifier une organisation, soit en trois sous-chaînes, qui ne contiennent pas le caractère '#' et qui sont séparées par le caractère '#', pour identifier n'importe quel autre article administré.

NOTE 3 Au cas où l'IRDI ne serait pas utilisé pour identifier une organisation:

– la première sous-chaîne, appelée "Registration Authority Identifier" (RAI «identificateur d'autorité d'enregistrement»), identifie l'organisation qui est responsable de l'administration de l'article administré;

– la deuxième sous-chaîne, appelée le "Data Identifier" (DI «identificateur de données»), contient à la fois une catégorisation de l'article administré, représentée par deux caractères suivis du signe moins ('-') comme défini dans l'ISO/TS 29002-5 (par exemple: classe, propriété, unité) et l'identificateur assigné à l'article administré par le RAI;

– la troisième sous-chaîne correspond à "Version Identifier" (VI «identificateur de version) de l'IRDI.

5.11.2.17 Constraint_identifier

Le **constraint_identifier** est un identificateur de type **ISO_29002_IRDI_type** qui fournit un identificateur global à une contrainte représentée comme étant une entité **constraint**. La structure de cet identificateur obéit à la syntaxe des identificateurs définie dans l'ISO/TS 29002-5.

NOTE Un **constraint_identifier** peut être associé à un service de résolution conforme à l'ISO/TS 29002-20. Ce service serait capable de retourner une définition formelle de la contrainte identifiée par le **constraint_identifier** conforme au modèle EXPRESS de **constraint** dans la syntaxe définie par l'ISO 13584-32 (OntoML) et éventuellement dans la syntaxe de l'ISO 10303-21.

EXPRESS specification:

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

Propositions informelles:

IP1: la partie de l'identificateur après le deuxième caractère '#', c'est-à-dire le Data Identifier, doit commencer par '04-' pour identifier une contrainte comme spécifié dans l'ISO/TS 29002-5.

5.11.2.18 Dic_unit_identifier

Le **dic_unit_identifier** est un identificateur de type **ISO_29002_IRDI_type** qui identifie une unité dont la représentation de **dic_unit** doit être téléchargeable à partir du serveur ISO/TS 29002-20. La structure de cet identificateur obéit à la syntaxe des identificateurs définie dans l'ISO/TS 29002-5.

EXPRESS specification:

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

Propositions informelles:

IP1: un **dic_unit_identifier** doit être associé à un service de résolution conforme à l'ISO/TS 29002, et ce service doit être capable de retourner une définition formelle de l'unité identifiée par le **dic_unit_identifier** conforme au modèle EXPRESS **dic_unit** dans la syntaxe définie dans l'ISO 13584-32 (OntoML), et éventuellement dans la syntaxe de l'ISO 10303-21.

IP2: la partie de l'identificateur après le deuxième caractère '#', c'est-à-dire le Data Identifier, doit commencer par '05-' pour identifier une unité comme spécifié dans l'ISO/TS 29002-5.

NOTE Un **dic_unit_identifier** constitue un International Registration Data Identifier (IRDI) tel que défini par l'ISO/CEI 11179-5.

5.11.2.19 Dic_value_identifier

Le **dic_value_identifier** est un identificateur de type **ISO_29002_IRDI_type** qui fournit un identificateur global à une valeur de propriété représentée comme étant une entité **dic_value**. La structure de cet identificateur obéit à la syntaxe des identificateurs définie dans l'ISO/TS 29002-5.

NOTE 1 Le fait d'assigner un **dic_value_identifier** permet de réutiliser la même définition de **dic_value** dans plusieurs entités **value_domain**.

NOTE 2 Un **dic_value_identifier** peut être associé à un service de résolution conforme à l'ISO/TS 29002. Ce service serait capable de retourner une définition formelle de la valeur identifiée par le **dic_value_identifier** conforme au modèle EXPRESS de **dic_value** dans la syntaxe définie par l'ISO 13584-32 (OntoML) et éventuellement dans la syntaxe de l'ISO 10303-21.

EXPRESS specification:

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

Proposition informelle:

IP1: la partie de l'identificateur après le deuxième caractère '#', c'est-à-dire le Data Identifier, doit commencer par '07-' pour identifier une valeur de propriété comme spécifié dans l'ISO/TS 29002-5.

NOTE 3 Un **dic_value_identifier** constitue un International Registration Data Identifier (IRDI) tel que défini par l'ISO/CEI 11179-5.

5.11.2.20 Value_code_type

Le type **value_code_type** identifie les valeurs autorisées pour un code de valeur.

EXPRESS specification:

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

Propositions formelles:

WR1: la longueur d'une valeur de **value_code_type** ne doit pas dépasser la longueur de **value_code_len** (à savoir 35).

5.11.2.21 Value_format_type

Le type **value_format_type** identifie les valeurs autorisées pour un format de valeur. Ces valeurs sont définies conformément à l'Annexe D.

EXPRESS specification:

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

Propositions formelles:

WR1: la longueur d'une valeur de **value_format_type** ne doit pas dépasser la longueur de **value_format_len** (à savoir 80).

5.11.2.22 Version_type

Le type version_type identifie les valeurs autorisées pour une 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
(*
```

Propositions formelles:

WR1: la longueur d'une valeur de **version_type** ne doit pas dépasser la longueur de **version_len** (à savoir 10).

WR2: le type **version_type** doit contenir des chiffres uniquement.

5.11.2.23 Status_type

Le type **status_type** identifie les valeurs autorisées pour un statut. Les valeurs admissibles d'un **status_type** ne sont pas normalisées. Elles doivent être définies pour chaque dictionnaire particulier par le fournisseur des données du dictionnaire.

EXEMPLE 1 Un ensemble de valeurs admissibles pour le statut des articles proposés à la normalisation à une agence de maintenance de normes ISO est défini dans les directives ISO.

EXEMPLE 2 Un ensemble de valeurs admissibles pour le statut d'articles dans la norme base de données CEI est défini dans les directives CEI.

NOTE Un statut peut être associé à **dictionary_element** et/ou à **dic_value**.

EXPRESS specification:

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

5.11.2.24 Dictionary_code_type

Le type **dictionary_code_type** est **code_type** qui identifie un dictionnaire.

EXPRESS specification:

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

Propositions formelles:

WR1: la longueur d'une valeur de **dictionary_code_type** ne doit pas dépasser la longueur de **dictionary_code_len** (à savoir 131).

5.11.3 Définitions de base des entités

5.11.3.1 Généralités

Le présent paragraphe contient les définitions de base des entités, classées par ordre alphabétique.

5.11.3.2 Dates

L'entité **dates** décrit les trois dates respectivement associées à la première description stable, à la version courante et à la révision courante d'une description donnée.

NOTE Pour toutes les règles de gestion d'un dictionnaire particulier, il est de la responsabilité du fournisseur d'informations du dictionnaire de choisir quel instant correspond à la première description stable d'un article.

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
(*
```

Définitions des attributs:

date_of_original_definition: la date associée à la première version stable d'un article.

date_of_current_version: la date associée à la présente version.

date_of_current_revision: la date associée à la dernière révision.

5.11.3.3 Document

L'entité **document** est une ressource abstraite qui représente un document. Le schéma du dictionnaire permet seulement la prise en charge de l'identification des documents pour les échanges (voir ci-dessous). L'entité **document** peut également être sous-typée avec des entités mettant en œuvre un moyen d'échanger des données documentaires.

EXEMPLE Par référence à un fichier externe et à l'exacte spécification du format du fichier.

EXPRESS specification:

```
*)
ENTITY document
ABSTRACT SUPERTYPE;
END_ENTITY; -- document
(*
```

5.11.3.4 Graphics

L'entité **graphics** (graphiques) est une ressource abstraite devant être sous-typée avec des entités mettant en œuvre un moyen d'échanger des données graphiques.

EXEMPLE Par référence à un fichier externe et à l'exacte spécification du format du fichier.

EXPRESS specification:

```
*)
ENTITY graphics
ABSTRACT SUPERTYPE;
END_ENTITY; -- graphics
(*
```

5.11.3.5 External_graphics

L'entité **external_graphics** (graphiques externes) permet la prise en charge de l'échange de données graphiques au moyen de fichiers externes référencés par une entité **graphic_files** (fichiers graphiques).

EXPRESS specification:

```
*)
ENTITY external_graphics
SUBTYPE OF (graphics);
     representation: graphic_files;
END_ENTITY; -- external_graphics
(*
```

Définitions des attributs:

representation: représentation d'un graphique au moyen de fichiers externes.

5.11.3.6 Graphic_files

Une **graphic_files** est un **external_item** dont le contenu est une image.

NOTE 1 Une entité **external_item**, définie dans l'ISO 13584-24:2003, est un article dont le contenu peut être fourni sous forme d'un ou plusieurs fichiers externes de la bibliothèque. Elle se réfère à **external_file_protocol** qui spécifie comment il convient que le(s) fichier(s) externe(s) de la bibliothèque soi(en)t traité(s) et à **external_content** qui spécifie le(s) fichier(s) externe(s) de la bibliothèque qui représente(nt) son contenu.

NOTE 2 Les deux entités **external_file_protocol** et **external_content** sont définies dans l'ISO 13584-24:2003.

NOTE 3 Seules les entités **external_content** qui sont constituées de fichiers **http_file** et seules les entités **external_file_protocol** qui sont des **http_protocol** sont référencées par le schéma **ISO13584_IEC61360_dictionary_schema** et peuvent être utilisées dans le contexte de la présente partie de la CEI 61360.

NOTE 4 Les fichiers d'une entité **graphic_files** peuvent dépendre de la langue; cela est spécifié par les sous-types suivants de l'entité external_content: **not_translatable_external_content**, **not_translated_external_content** et **translated_external_content**.

NOTE 5 **http_file**, **http_protocol**, **not_translatable_external_content**, **not_translated_external_content** et **translated_external_content** sont définis dans l'ISO 13584-24:2003 et référencés par le schéma **ISO13584_IEC61360_dictionary_schema**.

EXPRESS specification:

```
*)
ENTITY graphic_files
SUBTYPE OF (external_item);
END_ENTITY; -- graphic_files
(*
```

5.11.3.7 Identified_document

L'entité **identified_document** décrit un document identifié par son étiquette.

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
(*
```

Définitions des attributs:

document_identifier: l'étiquette du document décrit.

Propositions formelles:

WR1: la longueur d'une valeur d'attribut de type **document_identifier** ne doit pas dépasser la longueur de **source_doc_len** (à savoir 255).

5.11.3.8 Item_names

L'entité **item_names** identifie les noms qui peuvent être associés à une description donnée. Elle énonce le nom préférentiel, l'ensemble de noms synonymes, le nom court et les langues de l'attribut **languages** dans lesquelles les différents noms sont donnés. Elle peut être associée à une icône.

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
    (*
```

Définitions des attributs:

preferred_name: le nom qui est préférentiel à l'emploi.

synonymous_names: l'ensemble des noms synonymes.

short_name: l'abréviation du nom préférentiel.

languages: la liste facultative des langues dans lesquelles les différents noms sont donnés.

icon: une icône facultative qui représente graphiquement la description associée à **item_names**.

Propositions formelles:

WR1: si des noms préférentiels et des noms courts sont donnés dans plus d'une langue, tous les attributs **languages** des **translated_label** doivent contenir l'instance **present_translations** comme dans l'attribut "languages" de cette instance **item_names**.

WR2: si des noms synonymes sont donnés dans plus d'une langue, seules les langues indiquées dans l'instance de **present_translations** dans l'attribut **languages** de cette instance de **item_names** peuvent être utilisées.

WR3: si aucun attribut **languages** n'est fourni, **preferred_name**, **short_name** et **synonymous_names** ne doivent pas être traduits.

NOTE 1 Les attributs **preferred_name**, **synonymous_names** et **short_name** sont utilisés pour coder respectivement les attributs "Preferred name", "Synonymous name" et "Short name" pour des propriétés et des classes.

NOTE 2 L'attribut **languages** est utilisé pour définir la séquence de traductions (si cela est demandé pour des attributs).

5.11.3.9 Label_with_language

L'entité **label_with_language** fournit des ressources pour associer une étiquette à une langue.

EXPRESS specification:

```
*)
ENTITY label_with_language;
     l: label;
     language: language_code;
END_ENTITY; -- label_with_language
(*
```

Définitions des attributs:

l: l'étiquette associée à une langue.

language: le code de la langue étiquetée.

5.11.3.10 Mathematical_string

L'entité **mathematical_string** fournit des ressources définissant une représentation pour des chaînes mathématiques. Elle permet également une représentation au format MathML.

EXPRESS specification:

```
*)
ENTITY mathematical_string;
     text_representation: text;
     MathML_representation: OPTIONAL text;
END_ENTITY; -- mathematical_string
(*
```

Définitions des attributs:

text_representation: forme "linéaire" d'une chaîne mathématique, utilisant l'ISO 843, s'il y a lieu.

MathML_representation: texte MathML, balisé conformément à la DTD (définition de type de document)[6] XML pour MathML. Le texte MathML doit être controlé afin qu'il soit traité comme une seule chaîne au cours de l'échange (voir l'ISO 10303-21).

5.12 Définitions des fonctions

5.12.1 Généralités

Le présent paragraphe contient des fonctions qui sont référencées dans les Articles "WHERE" pour affirmer la cohérence des données ou qui fournissent des ressources pour le développement d'applications.

---

[6] DTD = *Document Type Definition*.

5.12.2 Fonction acyclic_superclass_relationship

La fonction **acyclic_superclass_relationship** vérifie qu'il n'y pas de cycle dans la relation de superclasse. Par la cardinalité de l'attribut **its_superclass** dans la classe ENTITY, il est assuré qu'il existe un arbre d'héritage, mais pas de graphe acyclique. Ainsi, cette fonction doit simplement vérifier qu'aucune instance de classe ne se réfère, dans l'attribut **its_superclass**, à une autre qui soit essentiellement une sous-classe.

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);
     (* incorrect: la courante déclare une sous-classe comme étant
sa superclasse *)
     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 Fonction check_syn_length

La fonction **check_syn_length** vérifie que la longueur de **s** ne dépasse pas la longueur indiquée par **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 Fonction codes_are_unique

La fonction **codes_are_unique** retourne TRUE si les attributs **value_code** sont uniques au sein de la liste des valeurs.

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 Fonction definition_available_implies

La fonction **definition_available_implies** vérifie si la définition correspondant au paramètre **BSU** existe ou non. Ensuite, si la définition existe, le paramètre **expression** est retourné.

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 Fonction is_subclass

La fonction **is_subclass** retourne TRUE si **sub** est soit **super**, soit une sous-classe de **super**. Si certains attributs **dictionary_definition** de classe ne sont pas disponibles, la fonction retourne 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
          (* fin de chaîne atteinte, n'a pas satisfait à super
jusqu'ici *)
          RETURN(FALSE);
     END_IF;

     IF SIZEOF(sub.its_superclass.definition) = 1
     THEN
     (* définition disponible *)
          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 Fonction string_for_derived_unit

La fonction **string_for_derived_unit** retourne une représentation STRING de la **derived_unit** (conformément à l'ISO 10303-41) passée comme paramètre. Tout d'abord, les éléments de l'unité dérivée sont séparés selon le signe de l'exposant. S'il existe des éléments des deux sortes, la notation '/' est utilisée pour séparer ceux ayant le signe positif de ceux ayant le signe négatif. S'il n'y a que des exposants négatifs, la notation u-e est utilisée. Un point '.' (code décimal 46 selon l'ISO/CEI 8859-1, voir l'ISO 10303-21) est utilisé pour séparer les éléments individuels.

EXPRESS specification:

```
*)
FUNCTION string_for_derived_unit(u: derived_unit): STRING;

     FUNCTION string_for_derived_unit_element(
          u: derived_unit_element; neg_exp: BOOLEAN
          (* imprimer les exposants négatifs avec la puissance -1
*)): STRING;
          (* retourne une représentation STRING de
               derived_unit_element (conformément à l'ISO 10303-41)
               passée comme paramètre *)

     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;

(* séparer les éléments d'unités selon le signe des exposants: *)
pos := QUERY(ue <* u.elements | ue.exponent > 0);
neg := QUERY(ue <* u.elements | ue.exponent < 0);
us := '';
IF SIZEOF(pos) > 0 THEN
     (* il y a des éléments d'unités avec un signe positif *)
     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
```

```
            (* il y a des éléments d'unités avec signe négatif,
    utiliser '/'
                la 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
        (* seulement des signes négatifs, utiliser la notation u-e *)
        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 Fonction string_for_named_unit

La fonction **string_for_named_unit** retourne une représentation STRING de la **named_unit** (conformément à l'ISO 10303-41 et à l'extension dans 5.10.6.2) passée comme paramètre.

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 Fonction string_for_SI_unit

La fonction **string_for_SI_unit** retourne une représentation STRING de **si_unit** (conformément à l'ISO 10303-41) passée comme paramètre.

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 Fonction string_for_unit

La fonction **string_for_unit** retourne une représentation STRING de l'**unit** (conformément à l'ISO 10303-41) passée comme paramètre.

NOTE La fonction **string_for_unit** n'est pas appelée à partir du code EXPRESS. Il s'agit d'une fonction utilitaire permettant de calculer une chaîne comme représentation à partir de l'attribut **structured_representation** de **dic_unit**, lorsqu'aucun attribut **string_representation** n'est présent. Cette chaîne comme représentation correspond à celle utilisée dans l'Annexe B de la CEI 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 Fonction all_class_descriptions_reachable

La fonction **all_class_descriptions_reachable** vérifie si, oui ou non, les entités **dictionary_element** qui décrivent une classe, référencée par **class_BSU**, et toutes ses superclasses, peuvent être calculées dans l'arbre d'héritage défini par la hiérarchie des classes.

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 Fonction compute_known_visible_properties

La fonction **compute_known_visible_properties** calcule l'ensemble des propriétés qui sont visibles pour une classe donnée. Lorsqu'une définition n'est pas disponible, elle retourne seulement les propriétés visibles qui peuvent être calculées.

NOTE Lorsque l'attribut **dictionary_definition** d'une certaine classe est absent dans le même contexte d'échange (un contexte d'échange PLIB n'est jamais supposé être complet), la superclasse d'une classe peut ne pas être connue. Par conséquent, les propriétés définies comme étant visibles par cette superclasse ne peuvent pas être calculées par la fonction **compute_known_visible_properties**. C'est seulement pour le système de

réception que tous les attributs **dictionary_definition** des **BSU** sont disponibles. Par conséquent, pour le système de réception, la fonction **compute_known_visible_properties** calcule toutes les propriétés qui sont visibles à une classe pour la référencer (ou l'une quelconque de ses superclasses) par leur attribut **name_scope**.

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 Fonction compute_known_visible_data_types

La fonction **compute_known_visible_data_types** calcule l'ensemble des **data_type** qui sont visibles pour une classe donnée. Lorsqu'une définition n'est pas disponible, elle retourne seulement les entités **data_type** visibles qui peuvent être calculées.

NOTE Lorsque l'attribut **dictionary_definition** d'une certaine classe est absent dans le même contexte d'échange (un contexte d'échange PLIB n'est jamais supposé être complet), la superclasse d'une classe peut ne pas être connue. Par conséquent, les entités **data_type** définies comme étant visibles par cette superclasse ne peuvent pas être calculées par la fonction **compute_known_visible_data_types**. C'est seulement pour le système de réception que tous les attributs **dictionary_definition** des **BSU** sont disponibles. Par conséquent, sur le système de réception, la fonction **compute_known_visible_data_types** calcule toutes les **data_type** qui sont visibles à une classe pour la référencer (ou l'une quelconque de ses superclasses) par leur attribut **name_scope**.

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 Fonction compute_known_applicable_properties

La fonction **compute_known_applicable_properties** calcule l'ensemble des propriétés qui sont applicables pour une classe donnée. Lorsqu'une définition n'est pas disponible, elle retourne seulement les propriétés applicables qui peuvent être calculées.

NOTE Lorsque l'attribut **dictionary_definition** d'une certaine classe est absent dans le même contexte d'échange (un contexte d'échange PLIB n'est jamais supposé être complet), la superclasse d'une classe peut ne pas être connue. Par conséquent, les propriétés définies comme étant applicables par cette superclasse ne peuvent pas être calculées par la fonction **compute_known_applicable_properties**. C'est seulement pour le système de réception que tous les attributs **dictionary_definition** des **BSU** sont disponibles. Par conséquent, sur le système de réception, la fonction **compute_known_applicable_properties** calcule toutes les propriétés qui sont applicables à une classe pour être référencées par un attribut **described_by** ou être importées par le biais d'**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.refer
           enced_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 Fonction compute_known_applicable_data_types

La fonction **compute_known_applicable_data_types** calcule l'ensemble des **data_type** qui sont applicables pour une classe donnée. Lorsqu'une définition n'est pas disponible, elle retourne seulement les entités **data_type** applicables qui peuvent être calculées.

NOTE Lorsque l'attribut **dictionary_definition** d'une certaine classe est absent dans le même contexte d'échange (un contexte d'échange PLIB n'est jamais supposé être complet), la superclasse d'une classe peut ne pas être connue. Par conséquent, les entités **data_type** définies comme étant applicables par cette superclasse ne peuvent pas être calculées par la fonction **compute_known_applicable_data_types**. C'est seulement pour le système de réception que tous les attributs **dictionary_definition** des **BSU** sont disponibles. Par conséquent, sur le système de réception, la fonction **compute_known_applicable_data_types** calcule toutes les entités **data_type** qui sont applicables à une classe pour être référencées par un attribut **defined_types** ou être importées par le biais d'une **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.refer
             enced_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 Fonction list_to_set

La fonction **list_to_set** crée un SET à partir d'une LIST nommée **l**, le type d'élément pour le SET sera le même que celui présent dans la LIST d'origine.

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 Fonction check_properties_applicability

La fonction check_properties_applicability s'assure que seules les propriétés qui ne sont pas applicables pour une classe par héritage deviennent applicables à cette classe pour être référencées par l'attribut **described_by**.

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 Fonction check_datatypes_applicability

La fonction **check_datatypes_applicability** s'assure que seuls les datatypes qui ne sont pas applicables pour une classe par héritage deviennent applicables à cette classe pour être référencés par l'attribut **defined_types**.

EXPRESS specification:

```
*)
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 Fonction one_language_per_translation

La fonction **one_language_per_translation** s'assure que les langues de traduction dans **administrative_data** sont uniques.

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 Fonction allowed_values_integer_types

La fonction **allowed_values_integer_types** calcule l'ensemble des valeurs de type **integer_type** autorisées par une entité **non_quantitative_int_type**. Si le paramètre est indéterminé, elle retourne une valeur indéterminée.

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 Fonction is_class_valued_property

La fonction **is_class_valued_property** retourne TRUE si la propriété **prop** est définie comme étant une propriété de valeur d'une classe pour la classe **cl** au moyen d'un attribut **sub_class_properties** dans la classe **cl** ou dans l'une quelconque de ses superclasses. Si certains attributs **dictionary_definition** de classe ne sont pas disponibles pour calculer toutes les superclasses de la classe **cl**, la fonction retourne 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
             (* fin de chaîne atteinte, ne satisfait à super jusqu'ici
*)
                  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 Fonction class_value_assigned

La fonction **class_value_assigned** retourne l'ensemble des valeurs de la propriété **prop** qui ont été assignées à une classe **cl** au moyen d'un attribut **class_constant_values** dans la classe **cl** ou dans n'importe quelle superclasse de la classe **cl**. Si plusieurs valeurs sont assignées dans plusieurs superclasses, la fonction retourne l'ensemble de toutes les valeurs assignées. Si certains attributs **dictionary_definition** de la classe ne sont pas disponibles pour calculer toutes les superclasses de la classe **cl**, seules les valeurs calculées sont retournées.

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 Vue d'ensemble

Le schéma suivant fournit des ressources pour les chaînes autorisées en diverses langues. Il a été extrait du schéma du dictionnaire, car il pourrait être utilisé dans d'autres schémas conceptuels. Il repose largement sur le **support_resource_schema** issu de l'ISO 10303-41, et il peut être vu comme une extension à cela. Il permet l'utilisation d'une langue spécifique dans tout un contexte d'échange (fichier physique) sans la surcharge introduite lorsque plusieurs langues sont utilisées. Voir Figure 13 pour une description graphique.

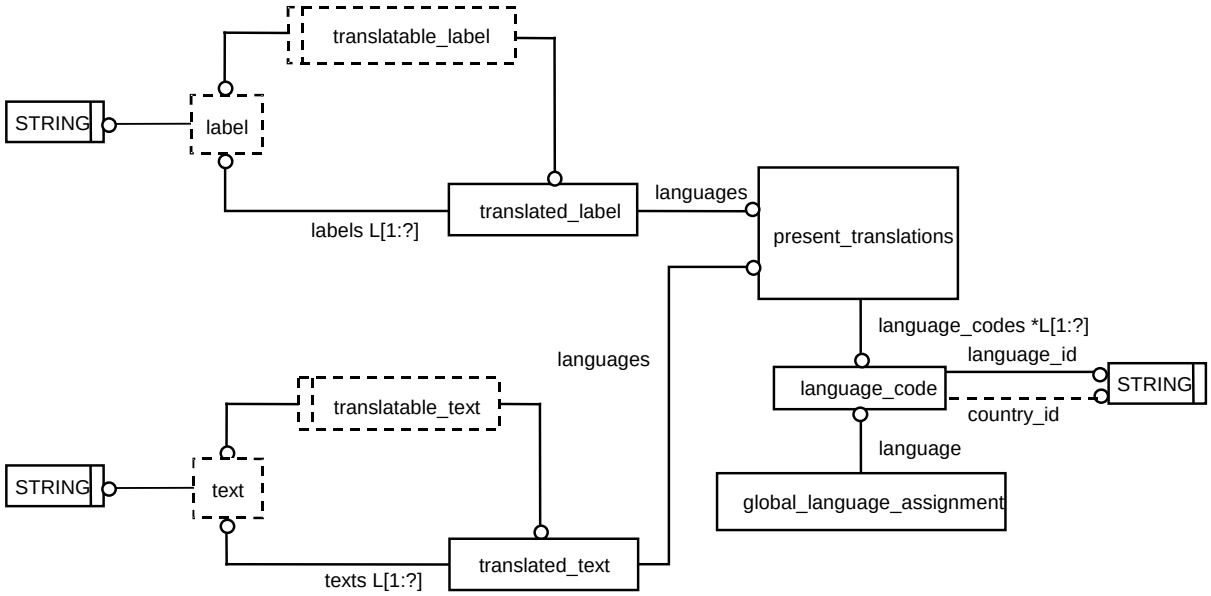


Figure 13 – ISO13584_IEC61360_language_resource_schema et support_resource_schema

EXPRESS specification:

```
*)
SCHEMA ISO13584_IEC61360_language_resource_schema;

REFERENCE FROM support_resource_schema(identifier, label, text);

(*
```

NOTE Le schéma **support_resource_schema** référencé ci-dessus peut être consulté dans l'ISO 10303-41.

6.2 Type ISO13584_IEC61360_language_resource_schema et définitions d'entités

6.2.1 Généralités

Le présent paragraphe contient le type EXPRESS et les définitions des entités dans le schéma **ISO13584_IEC61360_language_resource_schema**.

6.2.2 Language_code

L'entité **language_code** permet d'identifier un langage conformément à l'ISO 639-1. Elle contient deux codes:

– la langue telle que définie dans l'ISO 639-1 ou dans l'ISO 639-2, et, facultativement
– le code de pays, tel que défini dans l'ISO 3166-1, spécifiant le pays dans lequel la langue est parlée.

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
(*
```

Définitions des attributs:

language_id: le code qui spécifie la langue telle que définie par l'ISO 639-1 ou par l'ISO 639-2.

country_id: le code qui spécifie le pays dans lequel la langue est parlée, tel que défini par l'ISO 3166-1.

Propositions formelles:

WR1: la longueur d'une valeur **language_id** doit être égale à 2 ou à 3.

WR2: la longueur d'une valeur **currency_id** doit être égale à 2.

6.2.3 Global_language_assignment

L'entité **global_language_assignment** spécifie la langue pour les entités **translatable_label** et **translatable_text**, si **label** et **text** sont respectivement sélectionnés, (c'est-à-dire sans indication explicite de langue comme cela est réalisé dans **translated_label** et **translated_text)**.

EXPRESS specification:

```
*)
ENTITY global_language_assignment;
     language: language_code;
END_ENTITY; -- global_language_assignment
(*
```

Définitions des attributs:

language: le code de la langue assignée.

6.2.4 Present_translations

L'entité **present_translations** sert à indiquer les langues utilisées pour **translated_label** et **translated_text**.

EXPRESS specification:

```
*)
ENTITY present_translations;
     language_codes: LIST [1:?] OF UNIQUE language_code;
UNIQUE
     UR1: language_codes;
END_ENTITY; -- present_translations
(*
```

Définitions des attributs:

language_codes: la liste des codes de langue uniques correspondant à la langue dans laquelle une traduction est réalisée.

Proposition formelle:

UR1: pour chaque liste de **language_code**, il doit y voir une instance unique de **present_translations**.

6.2.5 Translatable_label

Un translatable_label définit un type de valeurs qui peuvent être des "label" (étiquettes) ou des "translated_labels" (étiquettes traduites).

EXPRESS specification:

```
*)
TYPE translatable_label = SELECT(label, translated_label);
END_TYPE; -- translatable_label
(*
```

6.2.6 Translated_label

L'entité **translated_label** définit les étiquettes qui sont traduites et les langues de traduction correspondantes.

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
(*
```

Définitions des attributs:

labels: la liste des **label**s (étiquettes) qui sont traduites.

languages: la liste des **languages** (langues) dans lesquelles chaque étiquette est traduite.

Propositions formelles:

WR1: le nombre de **label**s dans la liste **labels** doit être égal au nombre de langues fournies dans l'attribut **languages.language_codes**.

Propositions informelles:

IP1: le contenu de **labels[i]** est la langue identifiée par **languages.language_codes[i]**.

6.2.7 Translatable_text

Un **translatable_text** définit un type de valeurs qui peut être soit **text** (texte) soit **translated_text** (texte traduit).

EXPRESS specification:

```
*)
TYPE translatable_text = SELECT(text, translated_text);
END_TYPE; -- translatable_text
(*
```

6.2.8 Translated_text

L'entité **translated_text** définit les **text**s qui sont traduits et les **langues** de traduction correspondantes.

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
(*
```

Définitions des attributs:

texts: la liste des **text** (textes) qui sont traduits.

languages: la liste des langues dans lesquelles chaque text est traduit.

Propositions formelles:

WR1: le nombre de **text** dans la liste **texts** doit être égal au nombre de langues fournies dans l'attribut **languages.language_codes**.

Propositions informelles:

IP1: le contenu de **texts[i]** est la langue identifiée par **languages.language_codes[i]**.

6.3 Définitions des fonctions de l'ISO13584_IEC61360_language_resource_schema

6.3.1 Généralités

Le présent paragraphe contient une fonction qui est référencée dans les articles WHERE pour affirmer la cohérence des données.

6.3.2 Fonction check_label_length

La fonction **check_label_length** s'assure qu'aucune étiquette dans **l** ne dépasse la longueur indiquée par **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 (* l'argument l est une chaîne simple *)
     RETURN(LENGTH(l) <= l_length);
END_IF;
END_FUNCTION; -- check_label_length
(*
```

6.4 Définition des règles de l'ISO13584_IEC61360_language_resource_schema

La règle **single_language_assignment** affirme qu'une seule langue peut être assignée pour être utilisée dans **translatable_label** et **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 Généralités

Le présent article définit les exigences pour **class_constraint_schema**. La déclaration EXPRESS suivante introduit le bloc **ISO13584_IEC61360_class_constraint_schema** et identifie les références externes nécessaires.

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 Les schémas conceptuels référencés ci-dessus peuvent être consultés dans les documents suivants:

ISO13584_IEC61360_dictionary_schema (qui est dupliqué pour la commodité en 4.5 et après).	CEI 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 à l'ISO13584_IEC61360_class_constraint_schema type

Le schéma **ISO13584_IEC61360_class_constraint_schema** fournit des constructions EXPRESS permettant de redéfinir, par restriction, le domaine des valeurs d'une propriété donnée lorsqu'il est appliqué à une sous-classe de la classe de caractérisation dans laquelle la propriété était définie comme visible. Cette contrainte doit seulement expliciter une restriction du domaine de valeurs qui résulte déjà de la structure de classes.

EXEMPLE Dans l'ISO 13584-511, la classe *boulon/vis à filetage métrique* est une classe définie comme suit: "élément de fixation à filetage externe et à tête avec tige cylindrique, qui peut être partiellement ou complètement filetée et la tête peut être garnie d'une empreinte". Cette classe a, entre autres, deux propriétés appelées *type de tête* et *propriétés de tête*. Le domaine des valeurs de la propriété *type de tête* est un type de données non quantitatif qui inclut en particulier les valeurs suivantes: *hexagon_head* (tête hexagonale), *octagonal_head* (tête octogonale) et *round_head* (tête ronde). La propriété *propriétés de tête* est une caractéristique intrinsèque. Cela signifie qu'elle a un type de données *item_class* («classe d'articles»), dont le domaine est une classe *tête* qui définit toute sorte de tête. La classe *tête* a plusieurs sous-classes, y compris: *tête hexagonale*, associée à toutes les propriétés permettant de décrire une tête hexagonale (par exemple: *sur plats*), et *tête ronde*, associées à toutes les propriétés permettant de décrire une tête ronde (par exemple: *diamètre tête*).

La classe *boulon/vis à filetage métrique* a une sous-classe appelée *vis à tête hexagonale* définie comme suit: "élément de fixation à filetage externe métrique avec tête hexagonale filetée jusqu'à la tête". Cette classe hérite des propriétés *type de tête* et *tête*. À partir de la définition de la sous-classe *vis à tête hexagonale*, il est clair que la propriété *type de tête* pourrait seulement prendre la valeur *hexagon_head* (tête hexagonale) et que la propriété *propriétés de tête* pourrait seulement être une instance de la classe de caractéristique intrinsèque *tête hexagonale*. Mais ces contraintes sont implicites: elles sont juste énoncées de façon informelle dans la définition.

Ainsi, ces contraintes ne sont pas interprétables par un ordinateur. Les contraintes définies dans l'**ISO13584_IEC61360_class_constraint_schema** permettraient d'expliciter ces deux contraintes par association avec la classe *vis à tête hexagonale*: (1) une **enumeration_constraint** pour la propriété *type de tête* (autorisant

seulement le code *hexagon_head*) et (2) une **subclass_constraint** *pour la propriété propriétés de tête* (autorisant seulement la classe de caractéristique intrinsèque *tête hexagonale*).

Les contraintes sont héritées. Lorsqu'une propriété dont le domaine de valeurs a déjà été limité dans une classe C par une contrainte a besoin d'être davantage limitée dans une sous-classe C par une autre contrainte, les deux contraintes s'appliquent ensemble. Ainsi, le domaine réel de valeurs dans la sous-classe C est l'intersection des deux domaines définis par les deux contraintes. Le mécanisme proposé est semblable au fonctionnement de redéfinition du mécanisme des types disponible dans le langage EXPRESS.

Ce schéma permet d'exprimer les contraintes qui peuvent s'appliquer à des types de données issus du système type de l'**ISO13584_IEC61360_dictionary_schema**. Des règles sont utilisées dans les entités qui référencent une contrainte pour assurer que chaque contrainte peut s'appliquer au type de données auquel elle est connexe.

7.3 Définitions d'entités de l'ISO13584_IEC61360_class_constraint_schema

7.3.1 Généralités

Le présent article définit les entités dans l'ISO13584_IEC61360_class_constraint_schema.

7.3.2 Constraint

L'entité **constraint** permet de définir une contrainte.

EXPRESS specification:

```
*)
ENTITY constraint
ABSTRACT SUPERTYPE OF ( ONEOF (
                  property_constraint,
                  class_constraint));
     constraint_id: OPTIONAL constraint_identifier;
END_ENTITY; -- constraint
(*
```

Définitions des attributs:

constraint_id: le **constraint_identifier** qui identifie la contrainte.

7.3.3 Property_constraint

L'entité **property_constraint** est une contrainte qui limite l'ensemble admissible d'instances par la seule restriction du domaine des valeurs de l'une de ses propriétés.

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
(*
```

Définitions des attributs:

constrained_property: l'entité **property_BSU** pour laquelle la contrainte s'applique.

7.3.4 Class_constraint

L'entité **class_constraint** est une contrainte qui limite l'ensemble admissible d'instances d'une classe en contraignant plusieurs propriétés ou par plusieurs contraintes globales.

EXPRESS specification:

```
*)
ENTITY class_constraint
ABSTRACT SUPERTYPE OF (configuration_control_constraint)
SUBTYPE OF (constraint);
END_ENTITY; -- class_constraint
(*
```

7.3.5 Configuration_control_constraint

L'entité **configuration_control_constraint** permet de limiter l'ensemble des instances, appelées "instances référencées", qu'une instance particulière, appelée "instance de référence", peut référencer, directement ou indirectement, au moyen d'une chaîne de propriétés. L'instance de référence est toute instance d'une classe qui référence **configuration_control_constraint** au moyen de son attribut **constraints** ou hérite d'une classe qui le fait. L'entité **configuration_control_constraint** définit un attribut **precondition** (précondition) facultatif qui spécifie la condition imposée sur l'instance de référence pour que la restriction s'applique. Elle définit un attribut **postcondition** qui spécifie les ensembles admissibles de valeurs pour certaines propriétés de la classe d'instances référencées.

EXEMPLE Un *assemblage boulonné* est constitué de l'ensemble suivant d'éléments de fixation: un *élément de fixation à filetage externe*, un nombre quelconque de *rondelles* et un ou plusieurs *écrous*. Il existe différentes sortes de filetages, notamment *filetage de vis autotaraudeuse, filetage de vis à bois, filetage métrique externe, filetage métrique interne, filetage interne impérial* et *filetage externe impérial*. Supposons que l'on souhaite décrire un *assemblage boulonné métrique*. Il est nécessaire de s'assurer que, quelle que soit la structure précise de l'assemblage, l'*élément de fixation à filetage externe* et aussi tous les *écrous* impliqués dans l'assemblage ont un filetage métrique. Cela peut être réalisé en spécifiant dans la classe *assemblage boulonné métrique* l'entité **configuration_control_constraint** qui assure que tout élément de fixation décrit dans l'ISO 13584-511 référencé par une instance quelconque de cette classe, ou de n'importe laquelle de ses sous-classes, appartienne à des classes soit qui n'ont pas de valeurs pour la propriété *type of filetage* (par exemple *rondelle*), soit dont les valeurs appartiennent à l'ensemble: {*filetage externe métrique, filetage interne métrique*}.

NOTE 1 Les deux attributs **precondition** et **postcondition** peuvent seulement restreindre les propriétés dont le type de données est **non_quantitative_code_type**. Ces propriétés peuvent avoir une valeur qui leur est assignée soit au niveau de l'instance, soit au niveau de la classe si elles sont déclarées comme étant des propriétés valuées d'une classe, à savoir **sub_class_properties** dans une **class**.

NOTE 2 Il convient que les propriétés référencées dans l'attribut **precondition** soient applicables à la classe qui référence **configuration_control_constraint**.

NOTE 3 Dans **configuration_control_constraint**, les entités **filter** (filtre) servent aussi bien à représenter la précondition imposée sur l'instance de référence qu'à exprimer des contraintes imposées sur des instances référencées.

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
```

```
(*
```

Définitions des attributs:

precondition: les entités **filter** qui doivent être valides sur l'instance de référence pour que la restriction s'applique.

NOTE 4 Si l'ensemble des filtres est vide, la restriction s'applique à toute instance de référence.

postcondition: les entités **filter** qui doivent être valides sur une instance référencée pour être admissible comme référence.

7.3.6 Filter

L'entité **filter** est une entité **enumeration_constraint** qui limite le domaine admissible d'une propriété dont le type de données est soit **non_quantitative_code_type**, soit **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
(*
```

Définitions des attributs:

referenced_property: la propriété dont le domaine des valeurs est limité par **filter**.

domain: l'entité **enumeration_constraint** qui limite le domaine des valeurs de la propriété référencée.

Propositions formelles:

WR1: le type de données de **referenced_property** doit être soit **non_quantitative_code_type**, soit **non_quantitative_int_type**.

WR2: l'attribut **domain** doit définir un domaine de valeurs qui peuvent limiter le domaine initial des valeurs de la propriété.

7.3.7 Integrity_constraint

L'entité **integrity_constraint** est une contrainte de propriété particulière qui permet d'expliciter le fait que pour une certaine classe particulière, en raison de la définition de la classe, et de toutes ses sous-classes, seule une restriction du domaine des valeurs spécifié par un type de données est autorisée pour une propriété.

EXEMPLE Dans le dictionnaire de référence défini pour les éléments de fixation dans l'ISO 13584-511, un/une *boulon/vis à filetage métrique* a une propriété *propriétés de tête* que peut prendre, comme valeur, un membre de n'importe quelle sous-classe de la classe de caractéristiques intrinsèques *tête*. Si le/la *boulon/vis à filetage métrique* est également un membre de la sous-classe *vis à tête hexagonale*, la propriété *propriétés de tête* peut seulement être membre de la classe de caractéristiques intrinsèques *tête hexagonale*, autrement le/la *boulon/vis à filetage métrique* ne peut pas être membre de la sous-classe *vis à tête hexagonale*.

NOTE Dans l'exemple ci-dessus, la contrainte d'intégrité ne change nullement la signification de la propriété *propriétés de tête* héritée de *boulon/vis à filetage métrique* dans *vis à tête hexagonale*. Elle explicite seulement le fait que dans le contexte de la sous-classe *vis à tête hexagonale*, seul un sous-ensemble des valeurs admissibles pour cette propriété dans le contexte de la classe *boulon/vis à filetage métrique* reste autorisé.

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
(*
```

Définitions des attributs:

redefined_domain: la contrainte qui s'applique sur le domaine des valeurs de la propriété contrainte.

Propositions formelles:

WR1: **redefined_domain** doit définir un domaine des valeurs qui limite le domaine initial des valeurs de la propriété.

7.3.8 Context_restriction_constraint

L'entité **context_restriction_constraint** est une **property_constraint** qui limite le domaine admissible des valeurs pour les paramètres de contexte dont dépend une propriété dépendant d'un contexte.

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
    (*
```

Définitions des attributs:

context_parameter_constraints: la contrainte qui s'applique sur le domaine de valeurs des paramètres de contexte.

Propositions formelles:

WR1: l'ensemble des propriétés dont le domaine est contraint par la propriété **context_parameter_constraints** doit être constitué des paramètres de contexte dont dépend la propriété contrainte.

WR2: toutes les **context_parameter_constraints** doivent être des entités **integrity_constraint**.

WR3: la propriété **constrained_property** doit être une propriété dépendant du contexte **dependent_P_DET**.

7.3.9 Domain_constraint

Une entité **domain_constraint** définit une contrainte qui limite le domaine des valeurs d'un type de données.

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

Une entité **subclass_constraint** limite le domaine de valeurs de **class_reference_type** à une ou plusieurs sous-classes de la classe qui définit son domaine initial.

EXPRESS specification:

```
*)
ENTITY subclass_constraint
SUBTYPE OF(domain_constraint);
     subclasses: SET [1:?] OF class_BSU;
END_ENTITY; -- subclass_constraint
(*
```

Définitions des attributs:

subclasses: les entités **class_BSU** qui redéfinissent la classe à laquelle la valeur de la propriété **constrained_property** doit appartenir.

7.3.11 Entity_subtype_constraint

Une entité **entity_subtype_constraint** limite le domaine de **entity_instance_type** à un sous-type de l'ENTITY qui définit son domaine initial.

EXPRESS specification:

```
*)
ENTITY entity_subtype_constraint
SUBTYPE OF(domain_constraint);
     subtype_names: SET[1:?] OF STRING;
END_ENTITY; -- entity_subtype_constraint
(*
```

Définitions des attributs:

subtype_names: l'ensemble des chaînes qui décrivent, dans le format de la fonction EXPRESS TYPEOF, les noms des types de données de l'entité EXPRESS qui doivent appartenir au résultat de la fonction EXPRESS TYPEOF lorsqu'elle est appliquée à une valeur qui référence la propriété redéfinie **constrained_property**.

7.3.12 Enumeration_constraint

Une entité **enumeration_constraint** limite le domaine des valeurs d'un type de données à une liste de valeurs définies dans l'extension. L'ordre défini par la liste est l'ordre recommandé pour des besoins de présentation. Une description particulière peut facultativement être associée à chaque valeur d'un sous-ensemble au moyen d'un **non_quantitative_int_type**, dont la $i^{ème}$ valeur décrit la signification de la $i^{ème}$ valeur du sous-ensemble.

Pour les sous-types de **number_type** qui sont associés à **dic_unit** et à des unités de substitution, et éventuellement à un **dic_unit_identifier** et des identificateurs d'unités de substitution, la contrainte s'applique à la valeur correspondant à **dic_unit**, ou au seul **dic_unit_identifier**. S'ils existent tous les deux, ils correspondent à la même unité.

Pour les sous-types de **number_type** qui sont associés à une monnaie, la contrainte s'applique à la monnaie spécifiée dans leur définition des types de données. Si aucune monnaie n'est spécifiée dans la définition des types de données, la contrainte ne doit pas être utilisée.

Pour les valeurs qui appartiennent à **translatable_string_type**, la contrainte s'applique à la chaîne qui est dans la langue source dans laquelle le domaine de la propriété était défini.

Cette langue source peut être définie dans l'attribut **source_language** de **administrative_data** de la propriété. Si cet attribut n'existe pas, cette langue source est supposée être connue de l'utilisateur du dictionnaire.

Si une autre **enumeration_constraint** est appliquée sur une propriété déjà associée à une **enumeration_constraint** dans une certaine superclasse, les deux contraintes s'appliquent. Ainsi, l'ensemble admissible des valeurs est l'intersection des deux sous-ensembles. Quant à l'ordre de présentation, et l'éventuelle signification associée à chaque valeur, seules les significations définies dans l'**enumeration_constraint** inférieure s'appliquent.

EXEMPLE 1 Si, dans la classe C1, cette propriété est associée à une **enumeration_constraint** dont l'attribut **subset** (sous-ensemble) est {1, 3, 5, 7}, alors dans la classe C1, et n'importe laquelle de ses sous-classes, la propriété P1 peut seulement prendre l'une des quatre valeurs suivantes: 1, 3, 5 ou 7.

EXEMPLE 2 Si le type de données de la propriété P1 est LIST [1:4] OF INTEGER, et si dans la classe C1, cette propriété est associée à une **enumeration_constraint** dont l'attribut **subset** est { {1}, {3, 5}, {7}, {1, 3, 7} }, alors dans la classe C1 et n'importe laquelle de ses sous-classes, la propriété P1 peut seulement prendre l'une des quatre valeurs suivantes: {1} ou {3, 5} ou {7} ou {1, 3, 7}.

EXPRESS specification:

```
*)
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
(*
```

Définitions des attributs:

subset: la liste décrivant le sous-ensemble des valeurs qui sont admissibles comme valeurs possibles de la propriété identifiée par **constrained_property**.

value_meaning: l'ensemble des entités **dic_value** qui définissent la signification de chaque valeur appartenant au **subset**.

Propositions formelles:

WR1: si **value_meaning non_quantitative_int_type** existe, l'ensemble des **value_code** de ses **dic_value** doit se situer dans la plage 1.. SIZE_OF(subset).

7.3.13 Range_constraint

Une entité **range_constraint** limite le domaine des valeurs d'un type ordonné à un sous-ensemble de valeurs définies par une plage.

NOTE 1 Les chaînes ne sont pas considérées comme étant des types ordonnés et ne peuvent pas être contraintes par **range_constraint**.

Pour les sous-types de **number_type** qui sont associés à **dic_unit** et à des unités de remplacement, et éventuellement à un **dic_unit_identifier** et des identificateurs d'unités de remplacement, la contrainte s'applique à la valeur correspondant à **dic_unit**, ou au seul **dic_unit_identifier**. S'ils existent tous les deux, ils correspondent à la même unité.

Pour les sous-types de **number_type** qui sont associés à une monnaie, la contrainte s'applique à la monnaie spécifiée dans leur définition des types de données. Si aucune monnaie n'est spécifiée dans la définition des types de données, la contrainte ne doit pas être utilisée.

NOTE 2 Pour **non_quantitative_int_type**, la contrainte s'applique à l'attribut **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
(*
```

Définitions des attributs:

min_value: le nombre définissant la borne inférieure de la plage des valeurs; une valeur inexistante signifie absence de borne inférieure.

max_value: le nombre définissant la borne supérieure de la plage de valeurs; une valeur inexistante signifie absence de borne supérieure.

min_inclusive: spécifie si **min_value** appartient à la plage ou non; une valeur inexistante signifie qu'il n'y a pas de borne inférieure.

max_inclusive: spécifie si **max_value** appartient à la plage ou non; une valeur inexistante signifie qu'il n'y a pas de borne supérieure.

Propositions formelles:

WR1: **min_value** doit être inférieure ou égale à **max_value**.

WR2: **min_value** et **max_value** doivent avoir les mêmes types de données.

WR3: si **min_value** a une valeur, **min_inclusive** doit aussi avoir une valeur.

WR4: si **max_value** a une valeur, **max_inclusive** doit aussi avoir une valeur.

7.3.14 String_size_constraint

Une entité **string_size_constraint** limite la longueur des valeurs STRING admissibles pour un type STRING, ou n'importe lequel de ses sous-types.

NOTE 1 Un domaine de valeurs de propriétés string_type est soit un string_type, soit un non_translatable_string_type, soit un translatable_string_type, soit un URI_type, soit un non_quantitative_code_type, soit un date_data_type, soit un time_data_type, soit un date_time_data_type.

NOTE 2 Pour **non_quantitative_code_type**, la contrainte s'applique au code.

Pour les valeurs qui appartiennent à **translatable_string_type**, la contrainte s'applique à la chaîne qui est dans la langue source dans laquelle le domaine de la propriété était défini. Cette langue source peut être définie dans l'attribut **source_language** de **administrative_data** de la propriété. Si cet attribut n'existe pas, cette langue source est supposée être connue de l'utilisateur du dictionnaire.

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
(*
```

Définitions des attributs:

min_length: la longueur minimale pour les chaînes qui sont admissibles comme valeurs pour la propriété identifiée par la propriété **constrained_property**.

max_length: la longueur maximale pour les chaînes qui sont admissibles comme valeurs pour la propriété identifiée par la propriété **constrained_property**.

NOTE 3 Si la valeur **min_length** n'existe pas, 0 est sous-entendu. Si la valeur **max_length** n'existe pas, "non borné" est sous-entendu.

Propositions formelles:

WR1: **min_length** et **max_length** définissent les bornes correctes.

7.3.15 String_pattern_constraint

Une entité **string_pattern_constraint** limite le domaine des valeurs d'un **string_type**, ou de n'importe quel de ses sous-types, à des valeurs de chaîne qui correspondent à un profil particulier de caractères génériques ("pattern"). La syntaxe **pattern** est définie par une expression rationnelle XML et les algorithmes de concordance associés qui sont définis par le Schéma XML Partie 2: Recommandation pour les datatypes.

NOTE 1 Un domaine de valeurs de propriétés **string_type** est soit un **string_type**, soit un **non_translatable_string_type**, soit un **translatable_string_type**, soit un **URI_type**, soit un **non_quantitative_code_type**, soit un **date_data_type**, soit un **time_data_type**, soit un **date_time_data_type**.

Dans le cas de string_type, de non_translatable_string_type, de URI_type, de non_quantitative_code_type, de date_data_type, de time_data_type ou de date_time_data_type, la contrainte s'applique à la chaîne (unique) qui est la valeur du type de données. Pour non_quantitative_code_type, la contrainte s'applique au code.

Pour les valeurs qui appartiennent à des entités translatable_string_type, la contrainte s'applique à la chaîne qui est dans la langue source dans laquelle le domaine de la propriété était défini. Cette langue source peut être définie dans l'attribut source_language de l'entité administrative_data de la propriété; si cet attribut n'existe pas, cette langue source est supposée être connue de l'utilisateur du dictionnaire.

NOTE 2 Dans le cas de **non_quantitative_code_type**, de **date_data_type**, de **time_data_type** ou de **date_time_data_type**, il convient que le **pattern** se conforme aux propositions informelles définies dans les types de données correspondants.

EXPRESS specification:

```
*)
ENTITY string_pattern_constraint
SUBTYPE OF (domain_constraint);
     pattern: STRING;
END_ENTITY; -- string_pattern_constraint
(*
```

Définition d'attributs:

pattern: le profil des valeurs de chaîne qui sont admissibles comme valeurs de la propriété identifiée par la propriété contrainte.

Proposition informelle:

IP1: la syntaxe **pattern** doit se conformer à la syntaxe d'expression rationnelle XML et aux algorithmes de concordance associés qui sont définis par le Schéma XML Partie 2: Recommandation pour les datatypes.

EXEMPLE Le profil du Schéma XML qui correspond à l'expression SQL SIMILAR “[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]” est “[0-9]{4}\-[0-9]{2}\-[0-9]{2}”. Il permet de faire concorder des chaînes à “2009-05-31”.

7.3.16 Cardinality_constraint

Une entité **cardinality_constraint** limite la cardinalité d'un type de données agrégé.

NOTE 1 La plage de cardinalités qui en résulte est l'intersection des plages de cardinalités préexistantes et de celle définie par **cardinality_constraint**.

NOTE 2 Les contraintes **cardinality_constraint** sont interdites sur le type **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
(*
```

Définitions des attributs:

bound_1: la borne inférieure de la cardinalité.

bound_2: la borne supérieure de la cardinalité.

NOTE 3 Si **bound_1** n'existe pas, la cardinalité minimale est 0. Lorsque **bound_2** n'existe pas, il n'y a aucune contrainte sur la cardinalité maximale.

Propositions formelles:

WR1: **bound_1** et **bound_2** définissent les bornes correctes.

7.4 Définitions des types de l'ISO13584_IEC61360_class_constraint_schema

7.4.1 Généralités

Le présent paragraphe définit le type dans l'ISO13584_IEC61360_class_constraint_schema.

7.4.2 Constraint_or_constraint_id

Le constraint_or_constraint_id est soit une constraint (contrainte), soit un constraint_identifier (identificateur de contrainte).

EXPRESS specification:

```
*)
TYPE constraint_or_constraint_id =
     SELECT (constraint, constraint_identifier);
END_TYPE; -- constraint_or_constraint_id
(*
```

7.5 Définition de fonctions de l'ISO13584_IEC61360_class_constraint_schema

7.5.1 Généralités

Le présent paragraphe définit les fonctions dans l'**ISO13584_IEC61360_class_constraint_schema**.

7.5.2 Fonction integer_values_in_range

La fonction **integer_values_in_range** calcule les valeurs entières qui appartiennent à une plage entière définie par sa borne inférieure et sa borne supérieure. Elle retourne "indéterminé" (?) lorsque l'une ou l'autre bornes est indéterminée.

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 Fonction correct_precondition

La fonction **correct_precondition** s'assure que la précondition de l'entité **configuration_control_constraint** définie par **cons** utilise seulement des propriétés qui sont

applicables à la classe **cl**. Elle retourne un résultat logique qui est UNKNOWN lorsque l'ensemble complet des propriétés applicables de la classe ne peut pas être calculé.

EXPRESS specification:

```
*)
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 Fonction correct_constraint_type

La fonction **correct_constraint_type** s'assure que le **domain_constraint** défini par **cons** est compatible avec **data_type** définie par **typ**. Elle retourne un élément logique qui est UNKNOWN lorsque **domain_constraint** définie par **cons** n'est pas l'un des sous-types définis dans le schéma **ISO13584_IEC61360_class_constraint_schema**.

EXPRESS specification:

```
*)
FUNCTION correct_constraint_type(
     cons: domain_constraint; typ:data_type): LOGICAL;

(*cas de contrainte de sous-classe*)

IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
     +'SUBCLASS_CONSTRAINT') IN TYPEOF(cons)
THEN
     (*le type de données doit être class_reference_type*)
     IF NOT

     ('ISO13584_IEC61360_DICTIONARY_SCHEMA.CLASS_REFERENCE_TYPE'
          IN TYPEOF (typ))
     THEN RETURN(FALSE);
     END_IF;

     (*les cons.subclasses doivent consister en sous-classes pour
la classe
     qui a défini le domaine initial de typ.*)
     IF NOT (QUERY (sc <* cons.subclasses |
```

```
                definition_available_implies
            (sc,definition_available_implies
            (typ\class_reference_type.domain,is_subclass(sc.definitio
        n[1]
            ,       typ\class_reference_type.domain.definition[1])))=
        false)
            = [])
        THEN RETURN(FALSE);
        END_IF;

        RETURN (TRUE);
    END_IF;
    (*cas de contrainte de sous-type d'entité*)

    IF (('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
        +'ENTITY_SUBTYPE_CONSTRAINT') IN TYPEOF (CONS))
    THEN

    (* le type de données est class_reference_type*)
        IF NOT (('ISO13584_IEC61360_DICTIONARY_SCHEMA'
        +'.ENTITY_INSTANCE_TYPE') IN TYPEOF (typ))
        THEN RETURN(FALSE);
        END_IF;
    (*  le  subtype_name  doit  définir  un  sous-type  pour  l'entité
    entity_instance_type du contraint *)
        IF NOT (cons\entity_subtype_constraint.subtype_names
            >= typ\entity_instance_type.type_name)
        THEN RETURN(FALSE);
        END_IF;

    RETURN (TRUE);
    END_IF;

    (*cas d'enumeration_constraint *)

    IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
        +'.ENUMERATION_CONSTRAINT') IN TYPEOF (CONS)
    THEN

    (*  toutes  les  valeurs  appartenant  au  sous-ensemble  de  valeurs
    doivent être compatibles avec le type de données typ. *)
        IF (QUERY (val<*cons.subset |
            NOT compatible_data_type_and_value ( typ, val))<> [])
        THEN RETURN(FALSE);
        END_IF;

    RETURN (TRUE);
    END_IF;


    (*cas de range_constraint *)

    IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA.RANGE_CONSTRAINT'
          IN TYPEOF (CONS))
```

```
THEN

(*si le data type est un integer_type, alors min_value et max_value
doivent être des INTEGER.*)
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.INTEGER_TYPE'
          IN TYPEOF (typ)) AND
          NOT ('INTEGER' IN TYPEOF (cons.min_value))
     THEN RETURN(FALSE);
     END_IF;

(*si le type de données est un rational_type, min_value et
max_value doivent être rationnelles.*)
     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;

(*si le type de données est un real_type, min_value et max_value
doivent être REAL.*)
     IF ('ISO13584_IEC61360_DICTIONARY_SCHEMA.REAL_TYPE'
          IN    TYPEOF    (typ))    AND    NOT    ('REAL'    IN    TYPEOF
(cons.min_value))
     THEN RETURN(FALSE);
     END_IF;

(*toutes les valeurs de la plage doivent appartenir aux valeurs
admissibles définies par le 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;

(*cas d'entité string_size_constraint*)

IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
     +'.STRING_SIZE_CONSTRAINT') IN TYPEOF (CONS)
THEN

(* le type de données doit être un string_type ou n'importe lequel
de ses sous-types. *)
IF NOT    ('ISO13584_IEC61360_DICTIONARY_SCHEMA.STRING_TYPE'
          IN TYPEOF (typ))
THEN RETURN(FALSE);
END_IF;

RETURN (TRUE);
END_IF;
```

```
(*cas d'entité string_pattern_constraint *)

IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
     +'.STRING_PATTERN_CONSTRAINT') IN TYPEOF (CONS)
THEN

(* le type de données doit être un string_type ou n'importe lequel
de ses sous-types. *)
     IF NOT ('ISO13584_IEC61360_DICTIONARY_SCHEMA.STRING_TYPE'
          IN TYPEOF (typ))
     THEN RETURN(FALSE);
     END_IF;
RETURN (TRUE);
END_IF;

(*cas d'entité cardinality_constraint *)

IF ('ISO13584_IEC61360_CLASS_CONSTRAINT_SCHEMA'
     +'.CARDINALITY_CONSTRAINT') IN TYPEOF (CONS)
THEN

(* le type de données doit être un type agrégé mais pas une matrice
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 Fonction compatible_data_type_and_value

La fonction **compatible_data_type_and_value** vérifie si une valeur **val** d'une **primitive_value** est de type compatible avec le type défini par un type **dom**. Elle retourne un LOGICAL qui est TRUE lorsqu'ils sont compatibles et FALSE lorsqu'ils ne le sont pas. Cette fonction retourne UNKNOWN si le type de données de **val** est un **uncontrolled_instance_value** (voir ISO 13584-24:2003) ou lorsque le type n'est pas l'un des types définis dans le schéma **ISO13584_instance_resource_schema**.

NOTE La valeur de **val** peut ou peut ne pas exister.

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;

(* les déclarations express suivantes traitent des types simples.
*)

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;

    (* les déclarations express suivantes traitent des types complexes.
    *)

    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;


(* les déclarations express suivantes traitent des types agrégés.*)

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 Définition de règles de l'ISO13584_IEC61360_class_constraint_schema

7.6.1 Généralités

Le présent paragraphe définit la règle dans l'ISO13584_IEC61360_class_constraint_schema.

7.6.2 Unique_constraint_id

La règle **unique_constraint_id** affirme que deux **constraint_identifier** associés à deux **constraint** différentes ont des valeurs différentes.

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 Vue d'ensemble

Le présent article définit l'exigence pour l'**ISO13584_IEC61360_item_class_case_of_schema**. La déclaration EXPRESS suivante introduit le bloc **ISO13584_IEC61360_item_class_case_of_schema** et identifie les références externes nécessaires.

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 Les schémas conceptuels référencés ci-dessus peuvent être consultés dans les documents suivants:

ISO13584_IEC61360_dictionary_schema	CEI 61360-2
(qui est dupliqué pour la commodité dans ce document).	
ISO13584_IEC61360_class_constraint_schema	CEI 61360-2
(qui est dupliqué pour la commodité dans ce document).	
ISO13584_extended_dictionary_schema	ISO 13584-24:2003

8.2 Introduction à l'ISO13584_IEC61360_item_class_case_of_schema

Pour des raisons de modularité, le modèle complet de dictionnaire commun ISO/CEI est divisé en plusieurs documents. Le modèle noyau pour ontologies de produits est défini dans la CEI 61360-2 et dupliqué dans la présente partie de la CEI 61360. Les ressources pour étendre ce modèle, y compris la représentation d'instance, la représentation de document, le modèle fonctionnel, les vues fonctionnelles et la représentation des tables, sont définies dans l'ISO 13584-24:2003. Les divers niveaux normalisés de mise en œuvre du modèle complet ISO/CEI, appelés "classes de conformité", sont définis dans l'ISO 13584-25, et dupliqués à des fins informatives dans la CEI 61360-5. Le premier niveau correspond précisément au contenu de la présente partie de la CEI 61360 plus la ressource pour des valeurs structurées en agrégat définies dans l'ISO 13584-25. D'autres classes de conformité incluent de plus en plus de ressources issues de l'ISO 13584-24:2003.

Définir une classe d'articles comme étant une case-of d'une autre classe d'articles est de plus en plus utilisé par des applications basées sur le modèle de dictionnaire commun ISO13584/CEI61360. En outre, la présente édition de la présente partie de la CEI 61360 a

nécessité un changement du modèle d'information de ce concept. Par conséquent, il a été décidé de déplacer l'entité EXPRESS correspondante, appelée **item_class_case_of**, et sa superclasse, appelée **a_priori_semantic_relationship**, de l'ISO 13584-24:2003 vers la présente partie de la CEI 61360. Ces entités sont incluses dans un nouveau schéma, appelé **ISO13584_IEC61360_item_class_case_of_schema**. Les normes ISO 13584-24:2003 et ISO 13584-25 seront mises à jour en conséquence au moyen d'un corrigendum technique.

8.3 Définitions d'entités de l'ISO13584_IEC61360_item_class_case_of_schema

8.3.1 Relation sémantique *a priori*

Une **a_priori_semantic_relationship** est une **class** abstraite qui est définie sur la base d'autres classes et qui peut importer des propriétés, des types de données, des tableaux et des documents contenus dans ces classes. Elle importe aussi toutes les **contrainte**s qui limitaient le domaine des propriétés importées dans les classes à partir desquelles elles sont importées. Cette ressource abstraite est destinée à être sous-typée par des classes. Lorsqu'une classe spécialise une **a_priori_semantic_relationship,** les propriétés, types de données, tableaux ou documents dont les définitions sont importées par héritage d'une **a_priori_semantic_relationship** deviennent applicables à la classe qui les importe. En particulier, les propriétés et les types de données qui sont ainsi importés sont autorisés à être utilisés pour décrire des instances de classes. De plus, le fait, pour un produit, d'avoir un aspect qui corresponde à chaque propriété importée est un critère nécessaire pour être membre de la classe.

NOTE 1 Toutes les propriétés et tous les types de données importés deviennent directement applicables sans être visibles. Ainsi, ils ne sont pas retournés par la fonction **compute_known_visible_properties** ou **compute_known_visible_data_types**.

NOTE 2 La relation d'héritage est un exemple notoire de relation sémantique entre des classes modélisées selon le paradigme orienté objet. Toutes les propriétés et autres ressources définies dans une classe s'appliquent habituellement de façon implicite à toutes ses sous-classes. Cette relation est utilisée dans la série ISO 13584 lorsque toutes les propriétés, tous les types de données, tous les tableaux ou tous les documents visibles (respectivement applicables) à certaines classes sont implicitement visibles (respectivement applicables) à toutes ses sous-classes. Comme d'habitude, dans l'ISO 13584, cet héritage est implicite (c'est-à-dire: non déclaré au moyen d'une **a_priori_semantic_relationship**) et global (c'est-à-dire: toutes les propriétés et tous les types de données sont hérités par toutes ses sous-classes). Une **a_priori_semantic_relationship** permet de définir d'autres relations sémantiques qui sont utiles dans le domaine d'application de l'ISO 13584, et en particulier, la relation case-of qui permet une importation explicite et partielle de propriétés, et d'autres ressources définies dans une classe.

EXPRESS specification:

```
*)
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
  (*
```

Définitions des attributs:

referenced_classes: la/les classe(s) à partir de laquelle/desquelles les propriétés, types de données, tableaux ou documents sont importés.

NOTE 3 La classe à partir de laquelle des propriétés, types de données, tableaux ou documents sont importés ne peut pas être déduite de l'identification des propriétés, types de données, tableaux ou documents importés, car ils peuvent être importés d'une classe où ils sont hérités ou importés. Par exemple, dans la CEI 61360-DB, "input-voltage" (tension d'entrée) est une propriété visible au niveau de la racine de la classification CEI. Si une classe fournisseur importe la propriété "input-voltage" d'une classe "transistor" CEI, cela signifie que (1) la classe fournisseur définit un transistor, et (2) ces transistors sont décrits au moyen de la propriété "input voltage".

referenced_properties: les propriétés dont les définitions sont importées au moyen de l'entitié **a_priori_semantic_relationship**.

NOTE 4 L'ordre de la liste définit l'ordre par défaut pour afficher des propriétés importées au cours de l'accès utilisateur à divers sous-types de **a_priori_semantic_relationship**.

referenced_data_types: les types de données dont les définitions sont importées par le biais de l'entité **a_priori_semantic_relationship**.

referenced_tables: les tableaux dont les définitions sont importées par le biais de l'entité **a_priori_semantic_relationship**.

NOTE 5 Les ressources et les règles détaillées sur l'utilisation de tableaux sont définies dans l'ISO 13584-24:2003. Elles ne sont utilisées ni dans la présente partie de la CEI 61360 ni dans les modèles intégrés documentés dans l'ISO 13584-32 (OntoML) et l'ISO 13584-25.

referenced_documents: les documents dont les définitions sont importées par le biais de l'entité **a_priori_semantic_relationship**.

NOTE 6 Les ressources et les règles détaillées sur l'utilisation des documents sont définies dans l'ISO 13584-24:2003. Elles sont utilisées dans les modèles intégrés documentés dans l'ISO 13584-32 (OntoML) et l'ISO 13584-25.

referenced_constraints: les **property_constraint** qui s'appliquent aux diverses propriétés importées.

NOTE 7 Contrairement aux entités référencées, les contraintes referenced_constraints ne peuvent pas être sélectionnées lorsque **a_priori_semantic_relationship** est conçue. Ces contraintes sont toutes les contraintes qui

affectent un nombre quelconque de propriétés définies dans l'attribut **referenced_properties** dans n'importe quelle classe de l'attribut **referenced_classes**.

Propositions formelles:

WR1: toutes les **referenced_constraints** qui ne sont pas IRDI doivent être des **property_constraint**.

WR2: toutes les **referenced_constraints** doivent contraindre les propriétés qui sont importées par le biais de l'attribut **referenced_properties**.

WR3: toutes les **property_constraint** qui contraignent l'une des propriétés **referenced_properties** dans n'importe laquelle des classes **referenced_classes** doivent être importées par le biais de l'attribut **referenced_constraints**.

WR4: les propriétés importées définies par l'attribut **referenced_properties** doivent être visibles ou applicables pour l'une des classes appartenant à l'attribut **referenced_classes**.

WR5: les types importés définis par l'attribut **referenced_data_types** doivent être visibles ou applicables pour l'une des classes appartenant à l'attribut **referenced_classes**.

Propositions informelles:

IP1: toutes les **constraint** qui sont représentées par des **constraint_identifier** dans l'ensemble de **referenced_constraints** doivent correspondre aux **property_constraint** qui contiennent l'une des propriétés **referenced_properties** dans l'une des classes **referenced_classes**. Cette contrainte ne doit pas être représentée, dans le même ensemble **referenced_constraints**, tant comme **property_constraint** que comme **constraint_identifier**.

NOTE 8 Une contrainte représentée comme une **property_constraint** dans l'une des classes **referenced_classes** peut être représentée dans l'ensemble **referenced_constraints** soit comme une **property_constraint**, soit comme une **constraint_identifier.**

IP2: toutes les contraintes qui sont représentées par un **constraint_identifier** dans l'une des classes **referenced_classes**, mais dont la **constraint** correspondante est une **property_constraint** qui contraint l'une des propriétés importées par le biais de l'attribut **referenced_properties**, doivent être représentées par leur **constraint_identifier** dans l'ensemble **referenced_constraints**.

NOTE 9 Ces deux règles informelles assurent que l'ensemble **referenced_constraints** de contraintes est l'union des ensembles de **property_constraint** définis dans les diverses classes **referenced_classes** dont la **constrained_property** appartient à l'ensemble **referenced_properties**, même lorsque le contexte d'échange ne contient pas les définitions de toutes les classes impliquées dans **a_priori_semantic_relationship** et lorsque certaines **constraint** sont seulement représentées par leurs **constraint_identifier**.

8.3.2 Item_class_case_of

Une **item_class_case_of** est la description d'une classe d'articles qui est définie comme étant une "is-case-of " de quelque autre(s) classe(s) d'articles.

NOTE 1 Une entité **item_class_case_of** définit une relation sémantique *a priori*.

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
    (*
```

Définitions des attributs:

is_case_of: la/les **item_class** dont la présente **item_class** est "is-case-of".

imported_properties: la liste des propriétés qui sont importées de(s) **item_class** dont l'**item_class** définie est "is-case-of".

imported_types: l'ensemble des types de données qui sont importés de(s) **item_class** dont l'**item_class** définie est "is-case-of".

imported_tables: l'ensemble des **table_BSU** qui sont importés de(s) **item_class** dont l'**item_class** définie est "is-case-of".

imported_documents: l'ensemble des **document_BSU** qui sont importés de(s) **item_class** dont l'**item_class** définie est "is-case-of".

imported_constraints: l'ensemble de **property_constraint** ou de **constraint_id** qui sont importés de(s) **item_class** dont l'**item_class** définie est "is-case-of".

NOTE 2 Contrairement à d'autres entités importées, les contraintes **imported_constraints** ne peuvent pas être sélectionnées lorsque **item_class_case_of** est conçue. Ces contraintes sont toutes les contraintes qui limitent les domaines d'un nombre quelconque de propriétés définies dans l'**imported_properties** dans les classes de l'attribut **is_case_of** à partir desquelles elles sont importées. Ceci est spécifié dans une règle WHERE d'une **a_priori_semantic_relationship**.

Propositions formelles:

WR1: la superclasse de **item_class_case_of** doit être une **item_class**.

WR2: une **item_class_case_of** doit être une/des **item_class** "case-of".

WR3: **sub_class_properties** doit appartenir soit à la liste **described_by**, soit à la liste **imported_properties**.

WR4: toutes les propriétés de valeurs d'une classe déclarées au moyen de **sub_class_properties** qui sont des **imported_properties** doivent être des propriétés de valeur d'une classe dans toutes les classes **is-case-of** où elles sont applicables.

WR5: les valeurs assignées à une propriété importée au moyen de l'affectation de **class_constant_value** ne doivent pas être différentes de la valeur possible assignée à la même propriété dans les classes référencées.

WR6: toutes les **imported_properties** qui sont des propriétés de valeurs d'une classe dans une classe issue de l'ensemble **is_case_of** de classes, doivent être de propriétés de valeur d'une classe dans la classe courante.

WR7: toutes les **imported_properties** qui ont une valeur **class_constant_value** qui leur est assignée dans une classe issue de l'ensemble **is_case_of** de classes, doivent avoir la même **class_constant_value** assignée dans la classe courante.

8.4 Définitions de fonctions de l'ISO13584_IEC61360_item_class_case_of_schema

8.4.1 Généralités

Le présent paragraphe contient des fonctions qui sont référencées dans les articles WHERE pour affirmer la cohérence des données ou qui fournissent des ressources pour le développement d'applications.

8.4.2 Fonction compute_known_property_constraints

La fonction **compute_known_property_constraints** calcule l'ensemble des **property_constraint** qui s'applique aux propriétés d'un ensemble de classes. Les contraintes représentées par leurs identificateurs ne sont pas retournées. Lorsque la définition de certaines classes n'est pas disponible, la fonction retourne seulement les entités **property_constraint** qui peuvent être calculées.

NOTE Lorsque l'attribut **dictionary_definition** d'une classe n'est pas disponible dans le même contexte d'échange (un contexte d'échange PLIB n'est jamais supposé être complet), sa propre superclasse peut ne pas être connue. Par conséquent, les contraintes définies par cette superclasse ne peuvent pas être calculées par la fonction **compute_known_property_constraints**. Au contraire, lorsque toutes les superclasses "is-a" d'une classe sont disponibles dans le même contexte d'échange, toutes les contraintes qui s'appliquent à cette classe peuvent être calculées par une seule analyse traverse de son arbre d'héritage " is-a", même lorsque certaines de ces superclasses importent des propriétés au moyen de **a_priori_semantic_relationship** telles que **item_class_case_of**.

EXPRESS specification:

```
*)
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 Fonction compute_known_referenced_property_constraints

La fonction **compute_known_referenced_property_constraints** calcule toutes les entités **property_constraints** qu'il convient qu' **ap a_priori_semantic_relationship** ait importées en calculant toutes les contraintes qui s'appliquent à une propriété qui est importée par le biais de l'attribut **referenced_properties** de **ap**, et qui sont définies ou héritées dans n'importe quelle classe de **referenced_classes** de **ap** dont **dictionary_definition** de **classe** est disponible dans le même contexte d'échange.

NOTE 1 Dans **a_priori_semantic_relationship**, il convient que toutes les **property_constraint** définies ou héritées dans les classes référencées par leur attribut **referenced_classes** qui s'appliquent à une propriété qui est importée par le biais de l'attribut **referenced_properties** de **a_priori_semantic_relationship** soient importées par le biais de l'attribut **referenced_constraints**.

NOTE 2 Lorsque l'attribut **dictionary_definition** d'une classe appartenant à l'attribut **referenced_classes** de **ap** n'est pas disponible dans le même contexte d'échange que **ap** (un contexte d'échange PLIB n'est jamais supposé être complet), les contraintes appartenant à cette classe ne peuvent pas être calculées. Ainsi, le résultat de la fonction **compute_known_referenced_property_constraints** peut seulement être un sous-ensemble des contraintes qu'il convient d'importer par **ap**.

NOTE 3 Lorsque les attributs **dictionary_definition** de toutes les classes **referenced_classes** de **ap** sont disponibles dans le même contexte d'échange, et lorsque aucune contrainte n'est représentée par un seul **constraint_identifier**, la fonction **compute_known_referenced_property_constraints** retourne exactement toutes les contraintes qu'il convient d'importer par **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);
          --  toutes  les  property_constraints  existant  dans  les
diverses
          -- classes issues de cl qui peuvent être calculées dans
le présent
          -- contexte d'échange.
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 Fonction superclass_of_item_is_item

La fonction **superclass_of_item_is_item** s'assure que la superclasse d'une **item_class cl**, si elle existe, est une **item_class.**

Si la **classe** associée à **class_BSU** ne peut pas être calculée, la fonction retourne 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 Fonction check_is_case_of_referenced_classes_definition

La fonction **check_is_case_of_referenced_classes_definition** retourne TRUE si l'ensemble **item_class_case_of is_case_of** de définition(s) du dictionnaire des classes référencées est de type compatible avec l'instance donnée de **item_class_case_of** de **cl**. Autrement, elle retourne 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 Définitions de règle de l'ISO13584_IEC61360_item_class_case_of_schema

8.5.1 Généralités

Le présent paragraphe définit la règle dans l'**ISO13584_IEC61360_item_class_case_of_schema**.

8.5.2 Règle imported_properties_are_visible_or_applicable_rule

La règle **imported_properties_are_visible_or_applicable_rule** s'assure que lorsqu'une propriété est importée par une classe au moyen de **a_priori_semantic_relationship**, cette propriété est visible ou applicable pour la classe à partir de laquelle elle est importée.

NOTE Les propriétés applicables comprennent les propriétés importées par le biais d'une relation sémantique. Cette règle permet d'importer des propriétés issues d'une classe dans laquelle elles étaient déjà importées.

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 Règle imported_data_types_are_visible_or_applicable_rule

La règle **imported_data_types_are_visible_or_applicable_rule** s'assure que lorsqu'un type de données est importé par une classe au moyen d'**a_priori_semantic_relationship**, ce type de données est visible ou applicable pour la classe à partir de laquelle elle est importée.

NOTE Les types de données applicables comprennent les types de données importés par le biais d'une relation sémantique. Cette règle permet d'importer des types de données issus d'une classe dans laquelle ils étaient déjà importés.

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 Règle allowed_named_type_usage_rule

La règle **allowed_named_type_usage_rule** est relative à l'utilisation d'un type nommé. Elle énonce que seuls des types qui sont applicables à une classe peuvent être utilisés pour spécifier le domaine des propriétés déclarées par une classe, par le biais de son attribut **described_by**.

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
    (*
```

Annexe A
(informative)

Exemple de fichier physique

A.1 Objectif

La présente annexe donne un certain nombre de fragments d'un fichier physique pour échanger les données de la CEI 61360-DB. Il vise à montrer l'utilisation du modèle EXPRESS à l'Article 5 L"ISO13584_IEC61360_dictionary_schema" et l'ISO 10303-21 ensemble pour échanger des données correspondantes.

A.2 En-tête de fichier

```
*/
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('Exemple de fichier physique'), '2;1');
FILE_NAME('example.spf', '2007-07-18', ('IEC SC3D WG2'), (),
'Version 1', '', '');
FILE_SCHEMA(('example_schema'));
ENDSEC;
DATA;
/*
```

A.3 Données fournisseur

```
*/
#1=SUPPLIER_BSU('112/2///61360_4_1', *); /*conformément à l'ISO 13584-
26*/
#2=SUPPLIER_ELEMENT(#1, #3, '01', $, $, $, #4, #5);
#3=DATES('1994-09-16', '1994-09-16', $);
#4=ORGANIZATION('CEI', 'Agence de maintenance CEI', 'L'Agence de
maintenance de la CEI');
#5=ADDRESS('à déterminer', $, $, $, $, $, $, $, $, $, $, $);
#10=SUPPLIER_BSU('112/3///_00', *); /* ICS ISO/CEI */
/*
```

A.4 Données de la classe-racine

La classe-racine AAA000 de la CEI fournit une portée des noms correspondants à la CEI 61360-DB. Elle couvre deux arbres, un pour les matériaux, un pour les composants. Elle est définie comme une **item_class**.

```
*/
#90=CLASS_BSU('OO', '001', #10);
#100=CLASS_BSU('AAA000', '001', #1);
#101=ITEM_CLASS(#100, #3, '01', $, $, $, #102, TEXT('La classe-racine
CEI qui donne une portée de noms correspondant à la CEI 61360-DB. Elle
couvre deux arbres, un pour les matériaux, un pour les composants'),
$, $, $, #90, (#110), (), (), $, (), (#110), (), $, $, $);
```

```
#102=ITEM_NAMES(LABEL('Classe-racine CEI'), (), LABEL('Racine CEI'),
$, $);
#110=PROPERTY_BSU('AAE000', '001', #100);
#111=NON_DEPENDENT_P_DET(#110, #3, '01', $, $,$, #112, TEXT('le type
d'arbre: matériau ou composant'), $, $, $, $, (), $, $, #113, $);
#112=ITEM_NAMES(LABEL('type d'arbre'), (), LABEL('tree type'), $, $);
#113=NON_QUANTITATIVE_CODE_TYPE((), 'A..8', #114);
#114=VALUE_DOMAIN((#120,#122), $, $, (), $, $);
#120=DIC_VALUE(VALUE_CODE_TYPE('MATERIAU'), #121, $, $, $, $, $, $);
#121=ITEM_NAMES(LABEL('arbre de matériaux'), (), LABEL('arbre de
mat'), $, $);
#122=DIC_VALUE(VALUE_CODE_TYPE('COMPONS'), #123, $, $, $, $, $, $);
#123=ITEM_NAMES(LABEL('arbre de composant'), (), LABEL('arbre de
comp'), $, $);
/*
```

A.5 Données des matériaux

```
*/
#200=CLASS_BSU('AAA218', '001', #1);
#201=ITEM_CLASS(#200, #3, '01', $, $, $, #202, TEXT('classe-racine de
l'arbre de matériaux'), $, $, $, #100, (#210,#230), (), (), $, (),
(#210), (#205), $, 'MATERIAL', $);
#202=ITEM_NAMES(LABEL('classe-racine de matériaux'), (), LABEL('racine
de matériaux'), $, $);
#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 du
type de matériau'), $, $, $, $, (), $, 'A57', #213, $);
#212=ITEM_NAMES(LABEL('type de matériau'), (), LABEL('type de
matériau'), $, $);
#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('acoustique'), (), LABEL('acoustique'), $, $);
#222=DIC_VALUE(VALUE_CODE_TYPE('MG'), #223, $, $, $, $, $, $);
#223=ITEM_NAMES(LABEL('magnétique'), (), LABEL('magnétique'), $, $);
#224=DIC_VALUE(VALUE_CODE_TYPE('OP'), #225, $, $, $, $, $, $);
#225=ITEM_NAMES(LABEL('optique'), (), LABEL('optique'), $, $);
#226=DIC_VALUE(VALUE_CODE_TYPE('TH'), #227, $, $, $, $, $, $);
#227=ITEM_NAMES(LABEL('thermoélectrique'), (), LABEL('th-électrique'),
$, $);
#230=PROPERTY_BSU('AAF286', '005', #100);
#231=NON_DEPENDENT_P_DET(#230, #3, '01', $, $,$, #232, TEXT('La masse
volumique nominale (en kg/m**3) d'un matériau.'), $, $, $, #233, (),
$, 'K02', #234, $);
#232=ITEM_NAMES(LABEL('masse volumique'), (), LABEL('masse
volumique'), $, $);
#233=MATHEMATICAL_STRING('$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 Données des composants

```
*/
#300=CLASS_BSU('EEE000', '001', #1);
#301=ITEM_CLASS(#300, #3, '01', $, $, $, #302, TEXT('classe-racine de l'arbre de composants'), $, $, $, #100, (#310,#330,#350), (), (), $, (), (#310), (#305), $, 'COMPONS', $);
#302=ITEM_NAMES(LABEL('classe-racine de composants'), (), LABEL('racine de composants'), $, $);
#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 de la principale classe fonctionnelle à laquelle un composant appartient'), $, $, $, $, (), $, 'A52', #313, $);
#312=ITEM_NAMES(LABEL('principale classe de composant'), (), LABEL('principale classe'), $, $);
#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 (électrique/ électronique)'), (), LABEL('EE'), $, $);
#322=DIC_VALUE(VALUE_CODE_TYPE('EM'), #323, $, $, $, $, $, $);
#323=ITEM_NAMES(LABEL('électromécanique'), (), LABEL('électroméc'), $, $);
#324=DIC_VALUE(VALUE_CODE_TYPE('ME'), #325, $, $, $, $, $, $);
#325=ITEM_NAMES(LABEL('mécanique'), (), LABEL('mécanique'), $, $);
#326=DIC_VALUE(VALUE_CODE_TYPE('MP'), #327, $, $, $, $, $, $);
#327=ITEM_NAMES(LABEL('pièce magnétique'), (), LABEL('magnétique'), $, $);
#330=PROPERTY_BSU('AAF267', '005', #100);
#331=NON_DEPENDENT_P_DET(#330, #3, '01', $, $,$, #332, TEXT('La distance nominale (en m) entre l'intérieur de deux bandes utilisées pour les produits bandés avec conducteurs axiaux'), $, $, $, #333, (), $, 'T03', #334, $);
#332=ITEM_NAMES(LABEL('espacement de bande interne'), (), LABEL('espacement de bande interne'), $, $);
#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('La valeur
telle que spécifiée par le niveau (miNoMax) du diamètre extérieur (en
m) d'un composant avec un corps de section circulaire'), $, $, $,
#353, (), $, 'T03', #354, $);
#352=ITEM_NAMES(LABEL('diamètre extérieur'), (), LABEL('diamètre
ext'), $, $);
#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 Données des composants électriques/électroniques

```
*/
#400=CLASS_BSU('EEE001', '001', #1);
#401=ITEM_CLASS(#400, #3, '01', $, $,$, #402, TEXT('composants
électriques / électroniques'), $, $, $, #300, (#410,#470), (), (), $,
(), (#410), (#405), $, 'EE', $);
#402=ITEM_NAMES(LABEL('Composants EE'), (), LABEL('Composants EE'), $,
$);
#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 de
la catégorie à laquelle appartient le composant
électrique/électronique.'), $, $, $, $, (), $, 'A52', #413, $);
#412=ITEM_NAMES(LABEL('composant de catégorie EE'), (), LABEL('comp
catég EE'), $, $);
#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('amplificateur'), (), LABEL('amplificateur'), $,
$);
#422=DIC_VALUE(VALUE_CODE_TYPE('ANT'), #423, $, $, $, $, $, $);
#423=ITEM_NAMES(LABEL('antenne (aerial)'), (), LABEL('antenne (aer)'),
$, $);
#424=DIC_VALUE(VALUE_CODE_TYPE('BAT'), #425, $, $, $, $, $, $);
#425=ITEM_NAMES(LABEL('batterie'), (), LABEL('batterie'), $, $);
#426=DIC_VALUE(VALUE_CODE_TYPE('CAP'), #427, $, $, $, $, $, $);
#427=ITEM_NAMES(LABEL('condensateur'), (), LABEL('condensateur'), $,
$);
#428=DIC_VALUE(VALUE_CODE_TYPE('CND'), #429, $, $, $, $, $, $);
#429=ITEM_NAMES(LABEL('conducteur'), (), LABEL('conducteur'), $, $);
#430=DIC_VALUE(VALUE_CODE_TYPE('DEL'), #431, $, $, $, $, $, $);
#431=ITEM_NAMES(LABEL('ligne de retard'), (), LABEL('ligne de
retard'), $, $);
```

```
#432=DIC_VALUE(VALUE_CODE_TYPE('DID'), #433, $, $, $, $, $, $);
#433=ITEM_NAMES(LABEL('dispositif diode'), (), LABEL('dispositif
diode'), $, $);
#434=DIC_VALUE(VALUE_CODE_TYPE('FIL'), #435, $, $, $, $, $, $);
#435=ITEM_NAMES(LABEL('filtre'), (), LABEL('filtre'), $, $);
#436=DIC_VALUE(VALUE_CODE_TYPE('IC'), #437, $, $, $, $, $, $);
#437=ITEM_NAMES(LABEL('circuit intégré'), (), LABEL('IC'), $, $);
#438=DIC_VALUE(VALUE_CODE_TYPE('IND'), #439, $, $, $, $, $, $);
#439=ITEM_NAMES(LABEL('inducteur'), (), LABEL('inducteur'), $, $);
#440=DIC_VALUE(VALUE_CODE_TYPE('LAM'), #441, $, $, $, $, $, $);
#441=ITEM_NAMES(LABEL('lampe'), (), LABEL('lampe'), $, $);
#470=PROPERTY_BSU('AAE754', '005', #100);
#471=NON_DEPENDENT_P_DET(#470, #3, '01', $, $, $,#472, TEXT('Le nombre
de bornes électriques d'un composant électrique/électronique ou
électromécanique'), $, $, $, #473, (), $, 'Q56', #474, $);
#472=ITEM_NAMES(LABEL('nombre de bornes'), (LABEL('nombre de
broches')), LABEL('nr de bornes'), $, $);
#473=MATHEMATICAL_STRING('N_term', 'N<sub>term</sub>');
#474=INT_TYPE((), 'NR1..4');

ENDSEC;
END-ISO-10303-21;
```

Annexe B
(informative)

Diagramme EXPRESS-G

La présente annexe contient les diagrammes EXPRESS-G pour les Articles 6 à 8. EXPRESS-G est défini dans l'Annexe A de l'ISO 10303-11:2004.

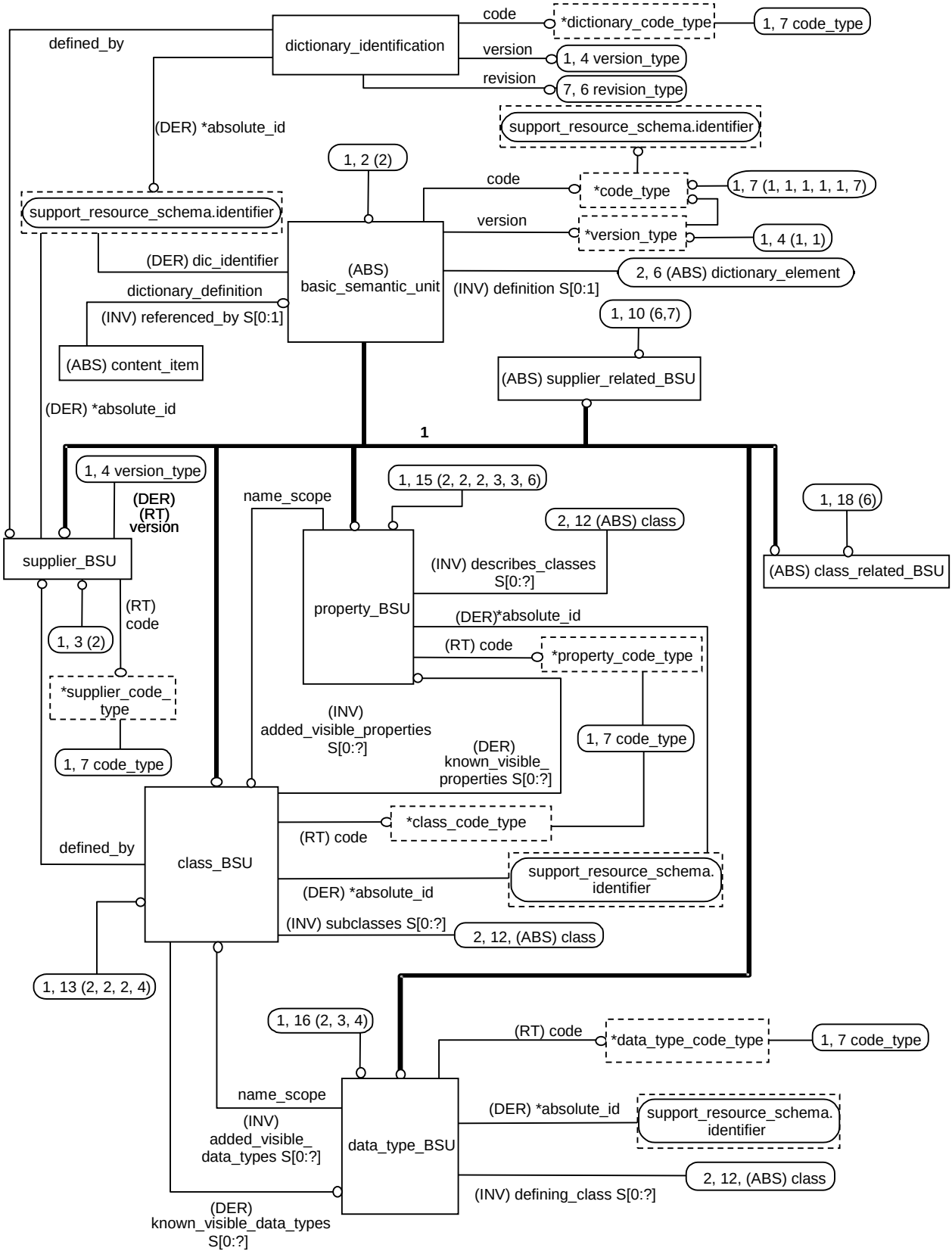


Figure B.1 – ISO13584_IEC61360_dictionary_schema – Diagramme EXPRESS-G 1 sur 7

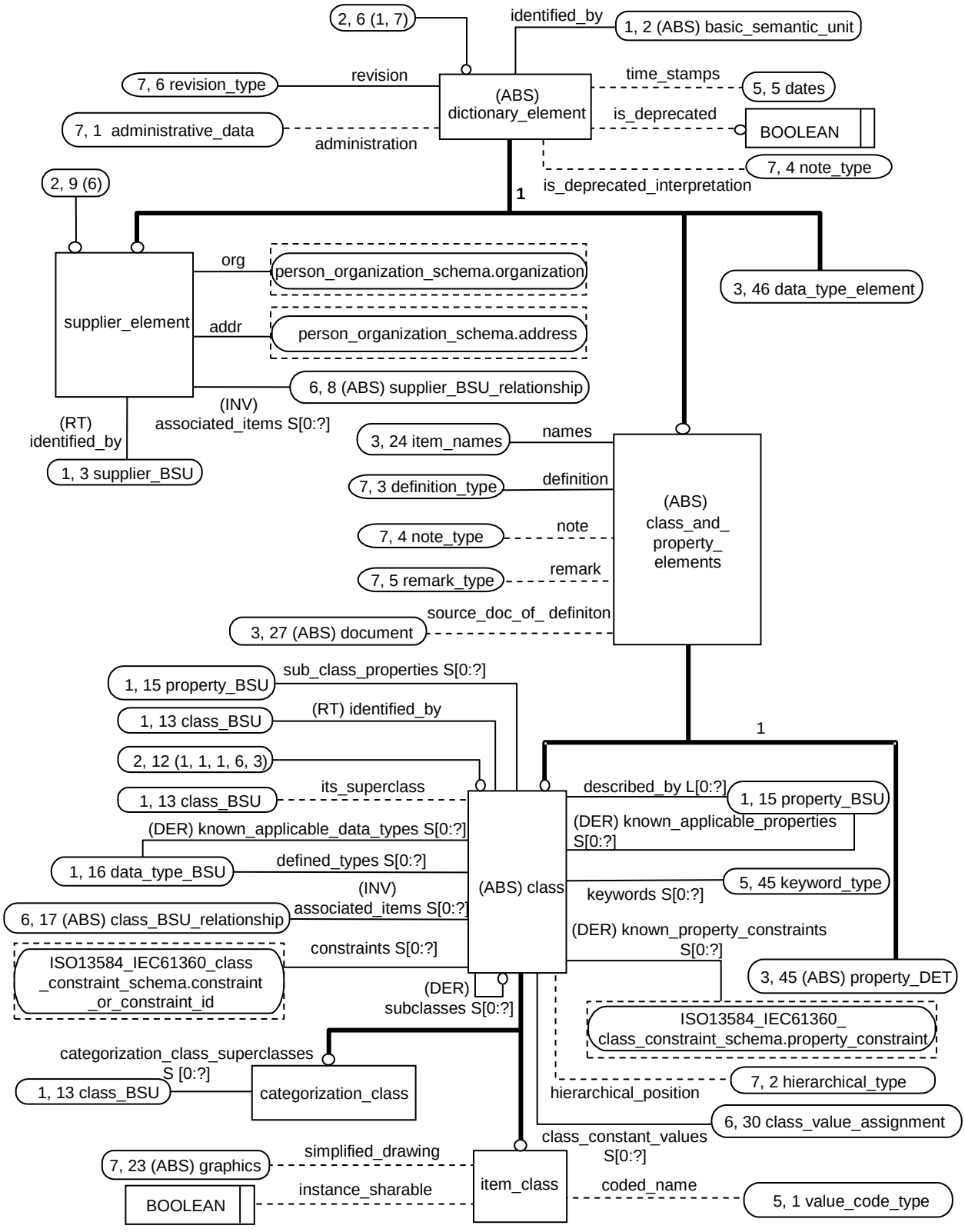


Figure B.2 – ISO13584_IEC61360_dictionary_schema – Diagramme EXPRESS-G 2 sur 7

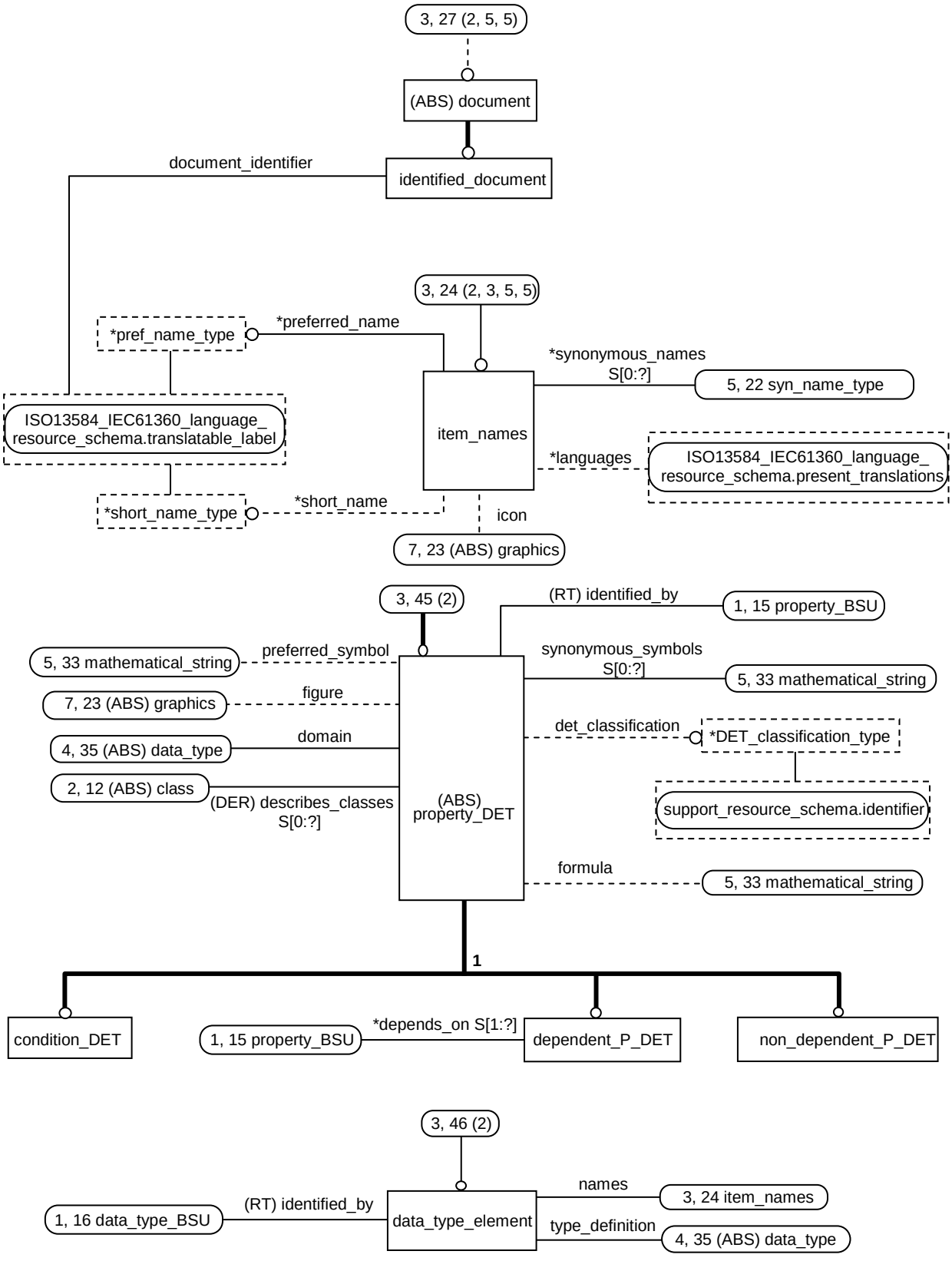


Figure B.3 – ISO13584_IEC61360_dictionary_schema – Diagramme EXPRESS-G 3 sur 7

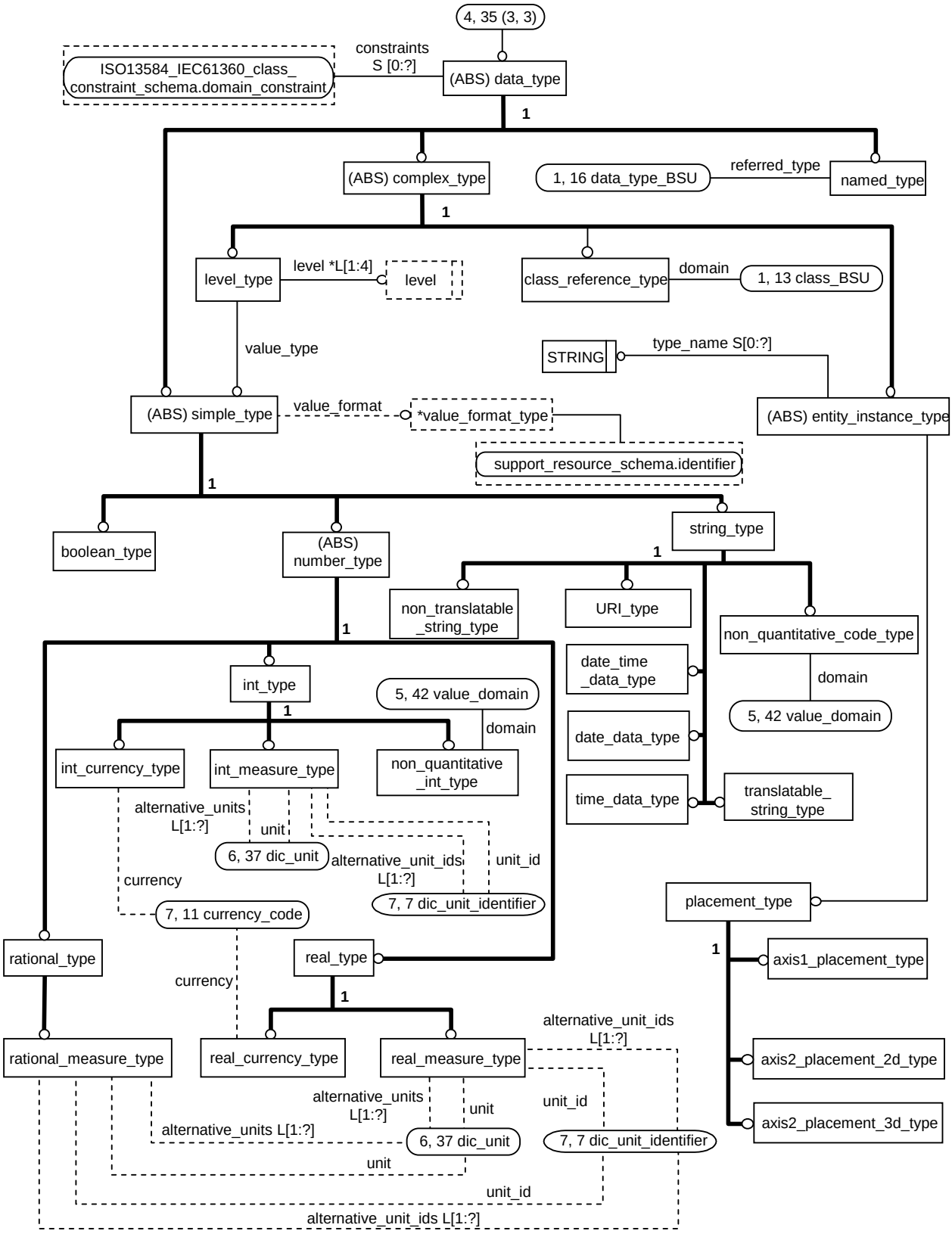


Figure B.4 – ISO13584_IEC61360_dictionary_schema – Diagramme EXPRESS-G 4 sur 7

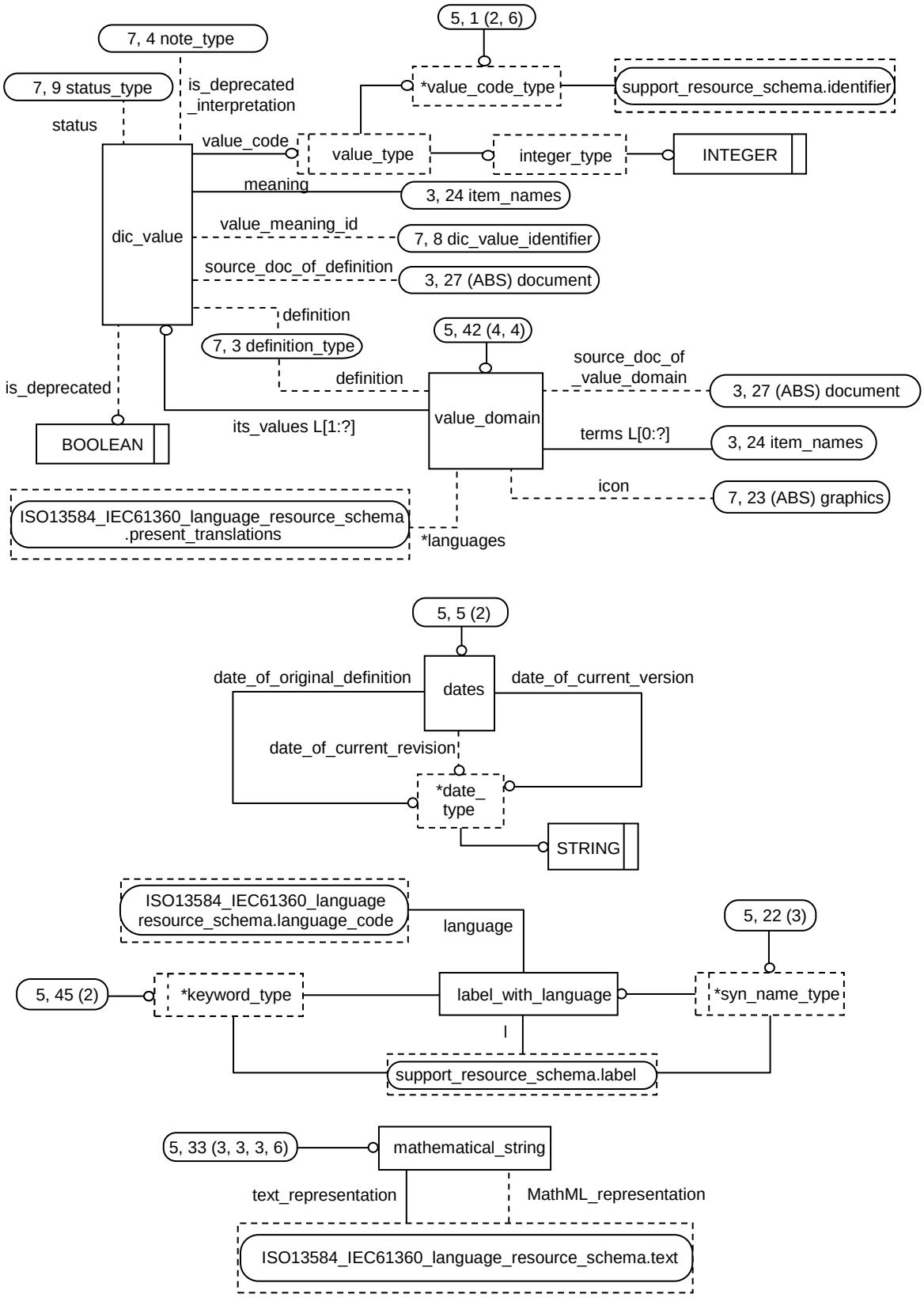


Figure B.5 – ISO13584_IEC61360_dictionary_schema – Diagramme EXPRESS-G 5 sur 7

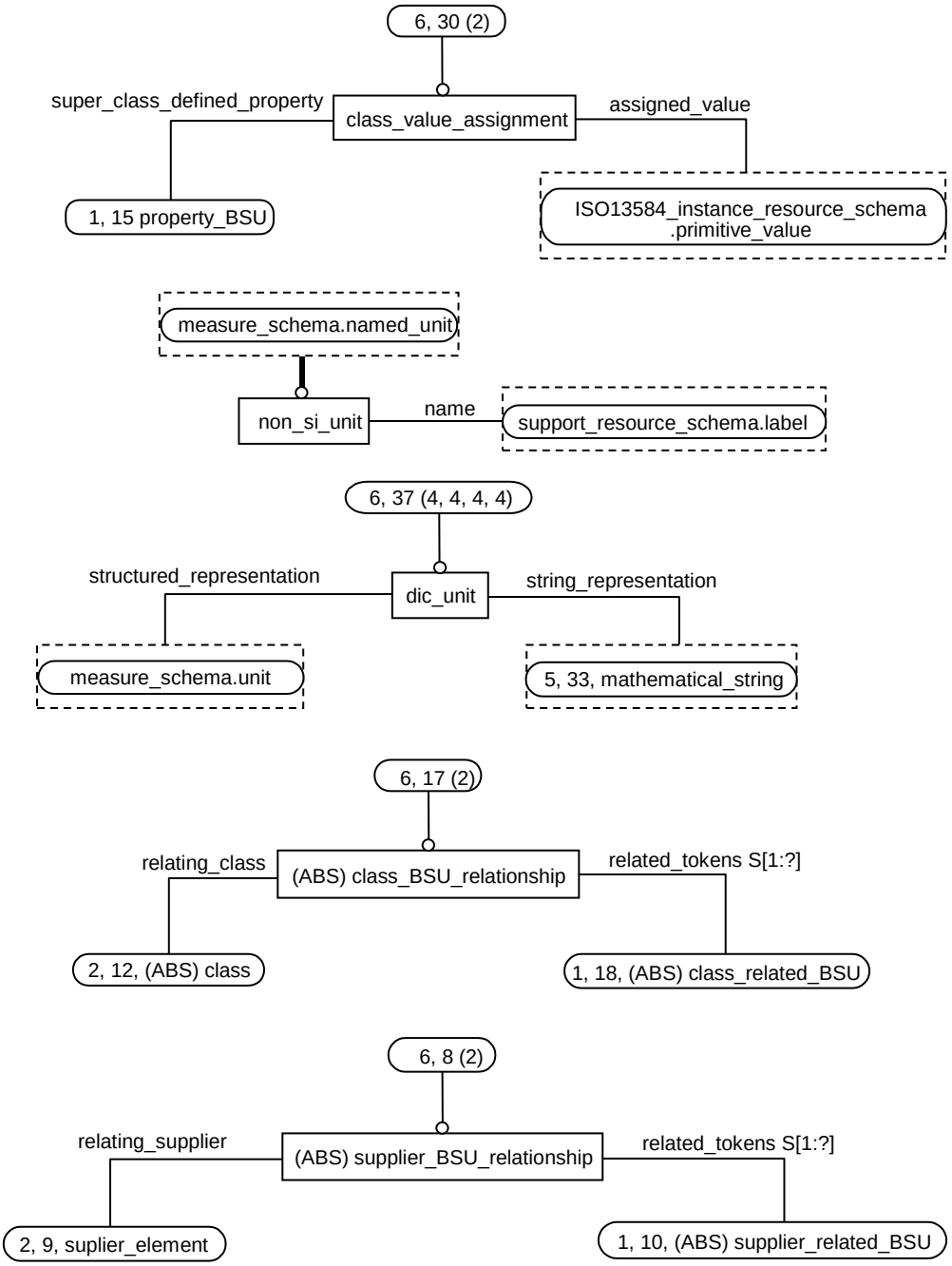


Figure B.6 – ISO13584_IEC61360_dictionary_schema – Diagramme EXPRESS-G 6 sur 7

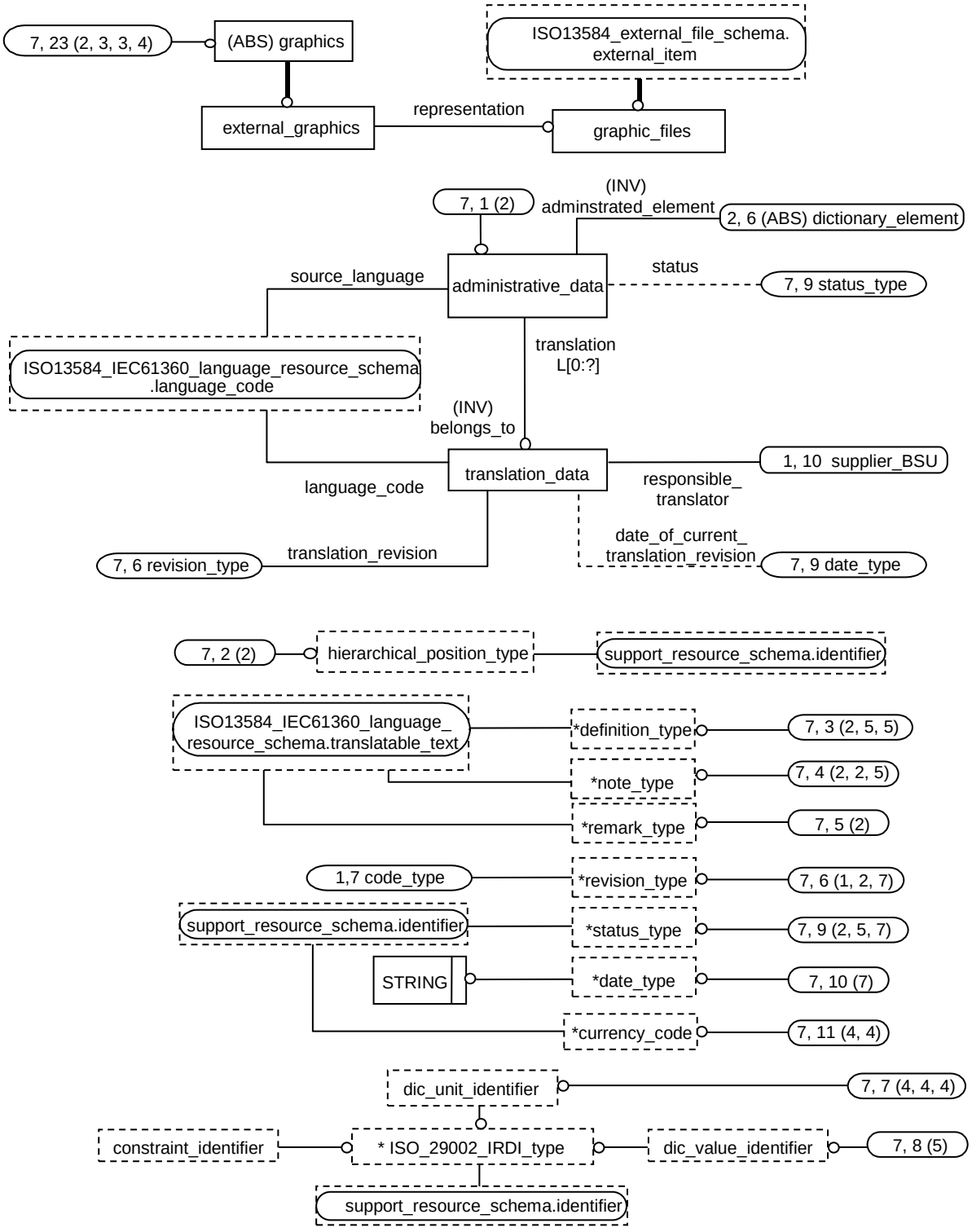


Figure B.7 – ISO13584_IEC61360_dictionary_schema – Diagramme EXPRESS-G 7 sur 7

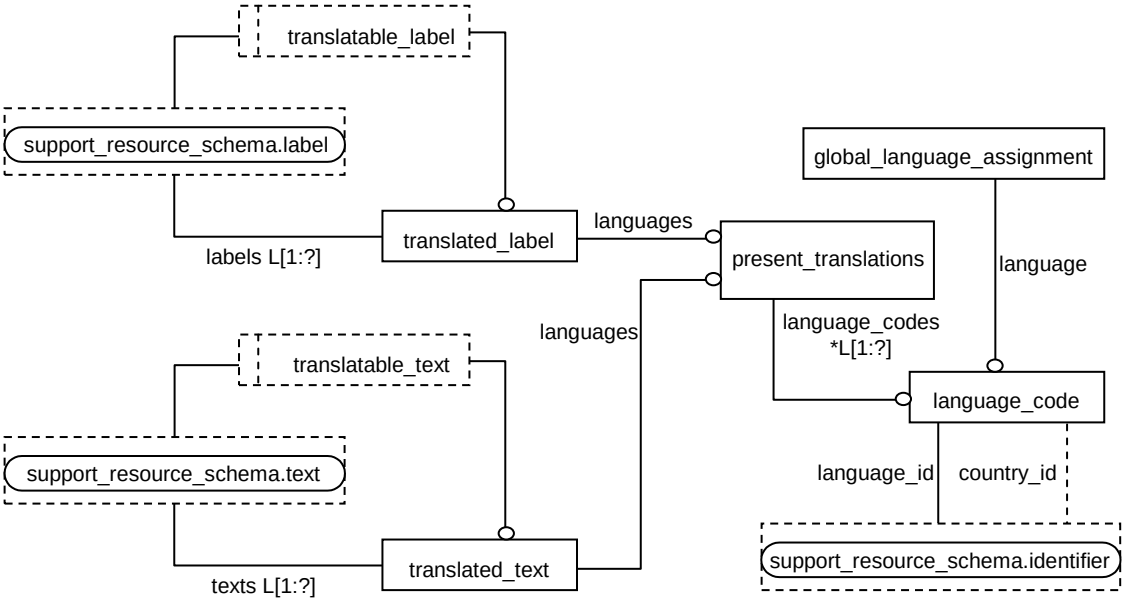


Figure B.8 – ISO13584_IEC61360_language_resource_schema – Diagramme EXPRESS-G 1 sur 1

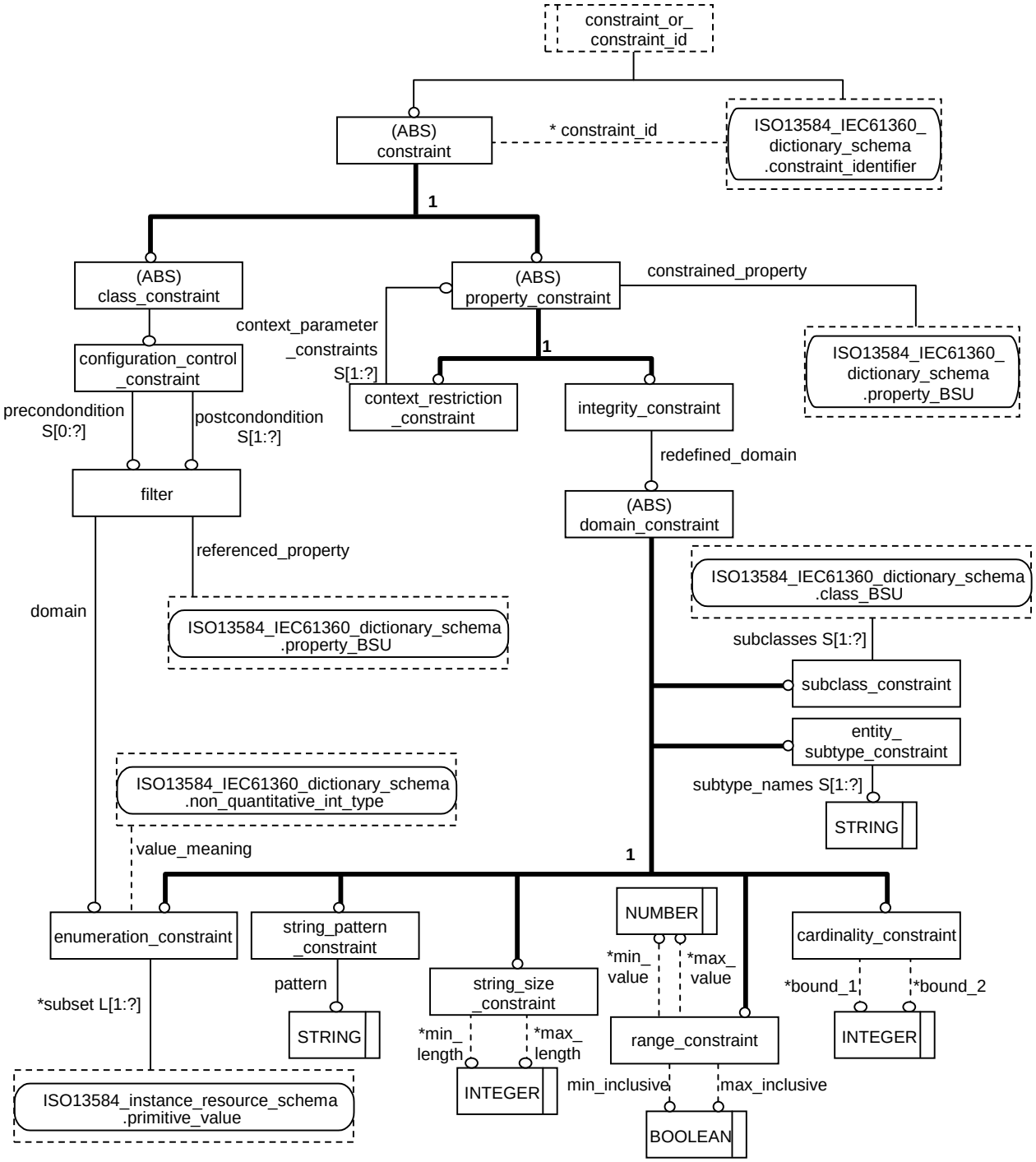


Figure B.9 – ISO13584_IEC61360_constraint_schema – Diagramme EXPRESS-G 1 sur 1

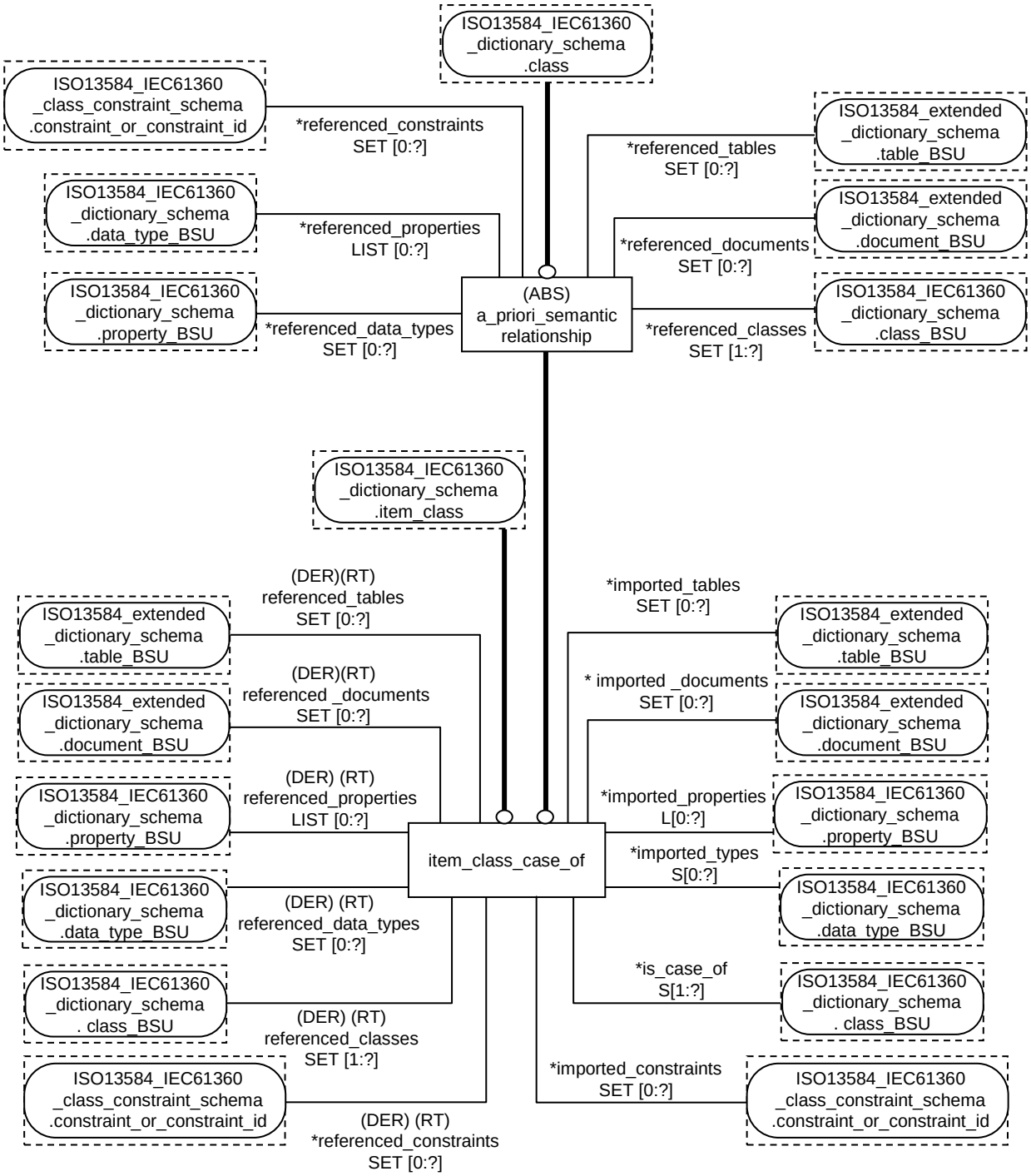


Figure B.10 – ISO13584_IEC61360_item_class_case_of_schema – Diagramme EXPRESS-G 1 sur 1

Annexe C
(informative)

Dictionnaires partiels

Le modèle de données EXPRESS publié dans la présente partie de la CEI 61360 et dupliqué pour la commodité dans l'ISO 13584-42. Il permet de décrire des dictionnaires composés de classes, de propriétés et de types de données et il permet la prise en charge de leur identification unique par le biais du mécanisme des unités sémantiques de base (BSU). Ce modèle permet de décrire des hiérarchies de classes structurées selon une structure arborescente utilisant un mécanisme d'héritage simple. Avec ce modèle de données, seuls les dictionnaires uniques et autonomes étaient traités.

L'utilisation de ce modèle de données conduit à plusieurs dictionnaires différents. Lors du processus de construction d'un dictionnaire, il peut arriver que des concepteurs exigent de référencer une classe particulière, une propriété particulière ou un type de données particulier déjà défini(e) dans un autre dictionnaire. La possibilité d'importer des propriétés et des types de données dans le dictionnaire en cours de conception est offerte par la relation "case-of" qui peut être utilisée avec le modèle complet de dictionnaire commun ISO13584/CEI61360, documenté tant dans l'ISO 13584-25 que dans la CEI 61360-5. En fait, cette relation permet d'importer des propriétés et des types de données définies extérieurement donnant la possibilité de prendre en charge des dictionnaires partiels et d'éviter la duplication entre dictionnaires, chaque dictionnaire définissant sa propre structure de classes.

Le modèle complet de dictionnaire commun ISO13584/CEI61360 propose deux mécanismes:

– l'entité EXPRESS **a_priori_case_of_semantic_relationship** permet d'utiliser directement les propriétés ou les types de données défini(e)s dans un ou plusieurs dictionnaires externes sans les décrire de nouveau;
– l'entité EXPRESS **a_posteriori_case_of_relationship** permet, une fois les propriétés ou les types de données déjà défini(e)s dans un dictionnaire en cours de conception, de les mettre en correspondance avec des propriétés ou types de données correspondant(e)s défini(e)s dans un dictionnaire externe.

Ces mécanismes permettent de concevoir des dictionnaires qui se réfèrent à des éléments de données qui sont définis dans d'autres dictionnaires sans affecter leur signification sémantique. En outre, les relations "case-of" sont enregistrées dans les dictionnaires qui utilisent cette fonctionnalité, permettant la prise en charge de l'intégration automatique de dictionnaires basés sur les mêmes dictionnaires normalisés.

Ce mécanisme est également recommandé lors de la conception d'un dictionnaire utilisateur final. En règle générale, un dictionnaire utilisateur final n'a pas besoin de toute la structure de classes définie dans les dictionnaires normalisés, tout en souhaitant pouvoir échanger de l'information avec d'autres utilisateurs dont les dictionnaires sont basés sur le(s) même(s) dictionnaire(s) normalisé(s). Si l'utilisateur final définit sa propre hiérarchie, mais met en correspondance chacune de ses classes, par la relation "case-of", avec la classe normalisée correspondante et s'il/elle importe toutes les propriétés normalisées existantes qui s'avèrent utiles pour son contexte, tout en ajoutant des propriétés spécifiques à un utilisateur, l'utilisateur personnalise son propre dictionnaire tout en pouvant échanger des informations normalisées avec d'autres utilisateurs. Cette approche de la conception de dictionnaires utilisateur final est celle recommandée dans la présente norme.

Annexe D
(normative)

Spécification du format des valeurs

D.1 Généralités

La présente partie de la CEI 61360 et l'ISO 13584-42 fournissent une syntaxe particulière pour spécifier les formats autorisés pour les valeurs de chaîne et les valeurs numériques qui peuvent être associées à une propriété.

EXEMPLE 1 Le format NR1 3 permet de spécifier que seules des valeurs entières constituées de trois chiffres exactement sont autorisées.

NOTE 1 Aucun format de valeur n'est défini pour un quelconque autre **data_type**, y compris le **boolean_type**.

NOTE 2 Dans la présente partie de la CEI 61360, définir le format des valeurs de propriété n'est pas obligatoire.

La syntaxe des formats autorisés est définie par la forme de Backus-Naur étendue (EBNF pour «Extended Backus-Naur Form») définie dans l'ISO/CEI 14977.

EXEMPLE 2 La syntaxe du format NR1 3 est constituée des lettres 'NR1' ' ' '3'.

La signification de chaque syntaxe, à savoir les caractères qui peuvent être utilisés pour représenter une valeur, ne peut pas être définie à l'aide de la forme EBNF. Donc, la signification de chaque partie du format concernant les caractères autorisés pour représenter la valeur est spécifiée séparément pour chaque partie du format.

EXEMPLE 3 La syntaxe du format NR1 3 a la signification suivante: *NR1* signifie que seule une valeur entière (integer) peut être représentée. L'espace signifie qu'un nombre fixe de caractères est spécifié par le format. *3* signifie que trois chiffres exactement sont requis.

D.2 Notation

Le Tableau D.1 résume le sous-ensemble du métalangage syntaxique EBNF de l'ISO/CEI 14977 utilisé par la présente partie de la CEI 61360 pour spécifier le format des valeurs des propriétés.

En utilisant ces notations, la syntaxe du sous-ensemble du métalangage EBNF utilisée par la présente partie de la CEI 61360 pour spécifier le format de valeurs des propriétés est récapitulée par la grammaire suivante (le caractère, la lettre et le chiffre du meta-identifier («méta-identificateur») ne sont pas détaillés):

```
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 = ;
```

Le signe '=' indique une règle de syntaxe. Le meta-identifier sur la gauche peut être réécrit par la combinaison des éléments situés à droite. Tous les espaces éventuels apparaissant entre les éléments sont sans signification, sauf s'ils apparaissent au sein d'un `terminal`. Une règle de syntaxe se termine par un point-virgule ';'.

Tableau D.1 – Métalangage syntaxique EBNF de l'ISO/CEI 14977

Représentation	Noms de caractère ISO/CEI 10646-1	Symbole de métalangage et rôle
‘ ‘	apostrophe	Premier symbole de citation: représente les bornes du langage. Le texte ne doit pas contenir d'apostrophe. Exemple: 'Hello'
" "	guillemets de citation	Second symbole de citation: représente les bornes du langage. Le texte ne doit pas contenir de guillemets. Exemple: "Voiture de Jean"
( )	parenthèse ouvrante, parenthèse fermante	Début / fin de symboles de groupe. Le contenu est considéré comme étant un seul symbole.
[ ]	crochet ouvrant, crochet fermant	Début / fin de symboles d'option. Le contenu peut ou peut ne pas être présent.
{ }	accolade ouvrante, accolade fermante	Début / fin de symboles de répétition. Le contenu peut être présent 0 fois à n fois.
-	trait d'union-moins	Symbole d'exception
,	virgule	symbole de concaténation
=	signe égal	Symbole de définition. Règle de syntaxe: définit le symbole de gauche par la formule située à droite.
|	trait vertical	Symbole séparateur d'alternatives
;	point-virgule	Symbole de fin Fin d'une règle de syntaxe.

L'utilisation d'un méta-identificateur dans une liste de définition désigne un symbole non borné qui apparaît sur la gauche d'une autre règle de syntaxe. Un méta-identificateur se compose de lettres ou de chiffres, le premier caractère étant une lettre. Si un terme contient à la fois un `primary` («primaire») précédant un signe moins, et un `primary` qui suit un signe moins, seule la séquence de symboles qui sont représentés par le premier `primary` et qui ne sont pas représentés par le second `primary` sont représentés par le terme.

EXEMPLE 1 Notation:

"'", caractère – "'", "'"

signifie n'importe quel caractère sauf le caractère apostrophe, inséré entre deux caractères apostrophes.

Le `terminal` désigne un symbole qui ne peut pas être étendu davantage par une règle syntaxique, et qui apparaîtra dans le résultat final. Deux façons sont permises pour représenter un `terminal`: soit un ensemble de caractères sans apostrophe, inséré entre

deux apostrophes, soit un ensemble de caractères sans guillemets, inséré entre deux guillemets.

EXEMPLE 2 Supposons que nous voulons décrire, par une telle grammaire, le prix d'un produit en €. Un tel prix est un nombre positif avec pas plus de 2 chiffres dans la partie des centimes. Nous introduisons trois méta-identificateurs associés à trois règles de syntaxe:

```
chiffre = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
centimes = [ '.', chiffre [, chiffre] ];
euros = chiffre {, chiffre} centimes;
```

Avec ces règles de syntaxe: 012, 4323.3, 3.56 sont des exemples de représentations légitimes des Euros. 12.,.10 sont des exemples de représentations illégitimes des euros.

D.3 Types de format de valeurs de données

La grammaire définie la présente annexe définit huit différents types de formats de valeurs: quatre formats de valeurs quantitatives et cinq formats de valeurs non quantitatives.

Dans le prochain article, nous définissons les méta-identificateurs qui sont utilisés pour spécifier ces formats. À l'Article D.5, nous définissons la règle de syntaxe pour les quatre méta-identificateurs qui représentent les quatre formats de valeurs quantitatives, accompagnés de leur signification au niveau de la valeur. À l'Article D.6 nous définissons les méta-identificateurs pour les cinq formats de valeurs non quantitatives, accompagnés de leur signification au niveau de la valeur.

D.4 Méta-identificateur utilisé pour définir les formats

Les méta-identificateurs utilisés dans la grammaire qui définissent les divers formats de valeurs sont les suivants:

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: séparateur entre partie entière et partie fractionnaire de nombres au format NR2 ou NR3.

leadingDigit: premier chiffre d'un nombre comprenant un ou plusieurs chiffres.

trailingDigit: un des chiffres qui se combinent pour former des nombres, sauf le premier.

NOTE Si un nombre comprend un seul chiffre, aucun `trailingDigit` n'est présent.

D.5 Formats de valeurs quantitatives

D.5.1 Généralités

Les quatre règles de syntaxe du format de valeurs quantitatives et leurs significations pour la représentation des valeurs sont définies dans les quatre paragraphes suivants. Elles sont autorisées à l'emploi pour des propriétés ayant les types de données suivants:

– **number_type** ou n'importe lequel de ses sous-types;
– **level_type** dont le **value_type** est soit **real_measure_type**, soit **int_measure_type**;
– **list_type**, **set_type**, **bag_type**, **array_type** ou **set_with_subset_constraint_type** dont le **value_type** est **number_type** ou n'importe lequel de ses sous-types.

NOTE 1 **list_type**, **set_type**, **bag_type**, **array_type** ou **set_with_subset_constraint_type** sont définis dans l'ISO 13584-25.

NOTE 2 Pour **non_quantitative_int_type**, le format de valeurs s'applique au code.

NOTE 3 Il convient que la valeur de cet attribut soit compatible avec le type de données de la propriété: il convient qu'elle ne change pas ce type de données, sinon il convient de l'ignorer.

EXEMPLE Le format de valeurs NR2 n'est pas compatible avec **int_type**, car il convient que les valeurs entières n'aient pas de partie fractionnaire.

D.5.2 Format de valeurs NR1

La syntaxe des valeurs NR1 spécifie le format d'une valeur d'une propriété entière.

Règle de syntaxe:

```
NR1Value = 'NR1', ((signedNumber, variableLengthIndicator) |
(signedNumber, space) | variableLengthIndicator | space),
lengthOfNumber;
```

La signification des composants du format de valeurs NR1 pour la représentation de valeurs est comme suit:

– `'NR1'`: la valeur doit être un nombre entier.

NOTE 1 Il convient que les valeurs des nombres NR1 ne contiennent aucun espace.

– lengthOfNumber: nombre de chiffres de la valeur.

NOTE 2 S'il est précédé d'un `variableLengthIndicator`, le nombre effectif de chiffres peut être inférieur.

– `signedNumber`: si `signedNumber` est présent, le nombre correspondant doit avoir une valeur positive, négative ou nulle (zéro). En cas de valeurs positives, un signe '+' peut être présent. Les valeurs négatives doivent être précédées d'un signe '-'. La valeur zéro ne doit pas être précédée d'un signe '-'.

– `variableLengthIndicator`: si `variableLengthIndicator` est présent, le nombre correspondant doit contenir un nombre de chiffres qui est inférieur ou égal à sa spécification de longueur, c'est-à-dire, à `lengthOfNumber`.

D.5.3 Format des valeurs NR2

La syntaxe des valeurs NR2 spécifie le format d'une valeur de propriété réelle qui n'a pas besoin d'un exposant.

Règle de syntaxe:

```
NR2Value = 'NR2', ((signedNumber, variableLengthIndicator) |
(signedNumber, space) | variableLengthIndicator | space),
lengthOfIntegralPart, decimalMark, lengthOfFractionalPart;
```

La signification des composants du format de valeurs NR2 pour la représentation de valeurs est comme suit:

– `'NR2'`: la valeur doit être un réel.

NOTE 1 Il convient que les valeurs des nombres NR2 ne contiennent aucun espace.

– `lengthOfFractionalPart`: nombre de chiffres de la partie fractionnaire du nombre.

NOTE 2 S'il est précédé d'un `variableLengthIndicator`, le nombre effectif de chiffres de la partie fractionnaire peut être inférieur.

NOTE 3 `lengthOfFractionalPart` spécifie de façon implicite l'exactitude recommandée de la valeur. L'exactitude effective du nombre à partir duquel cette valeur a été dérivée peut avoir été plus élevée que la valeur exprimée ici.

– `lengthOfIntegralPart`: nombre de chiffres de la partie entière du nombre.

NOTE 4 S'il est précédé d'un `variableLengthIndicator`, le nombre effectif de chiffres de la partie entière peut être inférieur.

– `signedNumber`: si `signedNumber` est présent, le nombre correspondant doit avoir une valeur positive, négative ou nulle (zéro). En cas de valeurs positives, un signe '+' peut être présent. Les valeurs négatives doivent être précédées d'un signe '-'. La valeur zéro ne doit pas être précédée d'un signe '-'.

– `variableLengthIndicator`: si `variableLengthIndicator` est présent, soit la partie entière, soit la partie fractionnaire du nombre, soit les deux parties, doi(ven)t contenir un nombre de chiffres qui est inférieur ou égal à sa spécification de longueur correspondante, c'est-à-dire, à `lengthOfIntegralPart` ou à `lengthOfFractionalPart`. Au moins un chiffre doit être présent dans le nombre.

D.5.4 Format de valeurs NR3

La syntaxe des valeurs NR3 spécifie le format d'une valeur d'une propriété réelle qui est représentée avec un exposant.

Règle de syntaxe:

```
NR3Value = 'NR3', ((signedNumber, variableLengthIndicator) |
(signedNumber, space) | variableLengthIndicator | space),
```

lengthOfIntegralPart, decimalMark, lengthOfFractionalPart, exponentIndicator, [signedExponent], lengthOfExponent;

La signification des composants du format de valeurs NR3 pour la représentation de valeurs est comme suit:

– 'NR3': la valeur doit être un réel avec un exposant de base 10.

NOTE 1 Il convient qu'il y ait au moins un chiffre et le signe décimal dans la mantisse. L'exposant doit contenir au moins un chiffre, aussi.

NOTE 2 Les valeurs des nombres NR3 ne doivent contenir aucun espace.

– exponentIndicator: séparateur entre mantisse et exposant dans les nombres au format NR3.

– lengthOfExponent: nombre de chiffres de l'exposant.

NOTE 3 S'il est précédé d'un variableLengthIndicator, le nombre effectif de chiffres de l'exposant peut être inférieur.

NOTE 4 Les signes existants éventuellement ou un signe décimal ne sont pas comptés par lengthOfNumber, lengthOfIntegralPart, lengthOfFractionalPart ou lengthOfExponent.

– lengthOfFractionalPart: nombre de chiffres de la partie fractionnaire de la mantisse.

NOTE 5 S'il est précédé d'un variableLengthIndicator, le nombre effectif de chiffres de la partie fractionnaire peut être inférieur.

NOTE 6 lengthOfFractionalPart spécifie de façon implicite l'exactitude recommandée d'une valeur. L'exactitude effective du nombre à partir duquel cette valeur a été dérivée peut avoir été plus élevée que la valeur exprimée ici.

– lengthOfIntegralPart: nombre de chiffres de la partie entière de la mantisse.

NOTE 7 S'il est précédé d'un variableLengthIndicator, le nombre effectif de chiffres de la partie entière peut être inférieur.

– signedExponent: si signedExponent est présent, l'exposant correspondant doit avoir une valeur positive, négative ou nulle (zéro). En cas de valeurs positives, un signe '+' peut être présent. Les valeurs négatives doivent être précédées d'un signe '-'. La valeur zéro ne doit pas être précédée d'un signe '-'.

– variableLengthIndicator: si variableLengthIndicator est présent, le nombre ou exposant correspondant doit contenir un nombre de chiffres qui est inférieur ou égal à sa spécification de longueur correspondante, c'est-à-dire, à lengthOfIntegralPart, à lengthOfFractionalPart ou à lengthOfExponent. Au moins un chiffre doit être présent dans la mantisse et dans l'exposant.

D.5.5 Format de valeurs NR4

La syntaxe de valeurs NR4 spécifie le format d'une valeur de propriété rationnelle qui est représentée avec une partie entière, et éventuellement une partie fractionnaire avec un dénominateur et un numérateur.

Règle de syntaxe:

NR4Value = 'NR4', ((signedNumber, variableLengthIndicator) | (signedNumber, space) | variableLengthIndicator | space), lengthOfIntegerPart, numeratorIndicator, lengthOfNumerator, denominatorIndicator, lengthOfDenominator;

La signification des composants du format de valeurs NR4 pour la représentation de valeurs est comme suit:

– 'NR4': la valeur doit être un nombre rationnel représenté soit comme un nombre entier, soit comme une fraction constituée d'un numérateur et d'un dénominateur, soit comme un entier et une fraction.

EXEMPLE 12 ½ et 12 ¾ sont des valeurs qui peuvent être représentées dans le format NR4.

NOTE 1 Il doit y avoir au moins un chiffre soit dans la partie entière, soit à la fois dans la partie numérateur et dans la partie dénominateur. Si une partie d'une fraction contient un chiffre, l'autre partie doit aussi contenir un certain nombre de chiffres. Toutes les trois parties peuvent aussi contenir des chiffres.

NOTE 2 Les valeurs des nombres NR4 ne doivent contenir aucun espace.

– numeratorIndicator: séparateur entre la description de la partie entière et la description de la partie fraction dans les formats NR4.

– lengthOfNumerator: nombre de chiffres du numérateur.

NOTE 3 S'il est précédé d'un variableLengthIndicator, le nombre effectif de chiffres du numérateur peut être inférieur.

NOTE 4 Si la valeur du nombre rationnel est complètement représentée par sa partie entière, ni le numérateur de la fraction ni son dénominateur ne doivent être représentés.

– denominatorIndicator: séparateur entre la description de la partie numérateur et la description de la partie dénominateur dans les formats NR4.

– lengthOfDenominator: nombre de chiffres du dénominateur.

NOTE 5 S'il est précédé d'un variableLengthIndicator, le nombre effectif de chiffres du dénominateur peut être inférieur.

NOTE 6 Si la valeur du nombre rationnel est complètement représentée par sa partie entière, ni le numérateur de la fraction ni son dénominateur ne doivent être représentés.

– lengthOfIntegerPart: nombre de chiffres de la partie entière du nombre rationnel.

NOTE 7 S'il est précédé d'un variableLengthIndicator, le nombre effectif de chiffres de la partie entière peut être inférieur.

– variableLengthIndicator: si variableLengthIndicator est présent, les trois parties du nombre rationnel doivent contenir un nombre de chiffres qui est inférieur ou égal à sa spécification de longueur correspondante, c'est-à-dire, à lengthOfIntegralPart, à lengthOfNumerator ou à lengthOfDenominator. Au moins un chiffre doit être présent soit dans la partie entière, soit dans les deux parties de la fraction.

D.6 Formats de valeurs non quantitatives

D.6.1 Généralités

Les règles de syntaxe du format des valeurs non quantitatives et leurs significations sont définies dans les cinq paragraphes suivants. Elles sont autorisées à l'emploi pour des propriétés ayant les types de données suivants:

– **string_type** ou importe lequel de ses sous-types;

– **list_type**, **set_type**, **bag_type**, **array_type** ou **set_with_subset_constraint_type** dont le **value_type** est **string_type** ou n'importe lequel de ses sous-types.

NOTE 1 **list_type**, **set_type**, **bag_type**, **array_type** ou **set_with_subset_constraint_type** sont définis dans l'ISO 13584-25.

NOTE 2 Pour **translatable_string_type**, le format de valeurs s'applique à n'importe quelle représentation spécifique à un langage de la chaîne de caractères.

NOTE 3 Pour **non_quantitative_code_type**, le format de valeurs s'applique au code.

Les valeurs non quantitatives sont représentées par des chaînes qui comprennent des caractères. La longueur d'une chaîne peut être spécifiée soit en spécifiant directement la

limite supérieure du nombre de caractères contenus, soit en spécifiant que le nombre total de caractères peut être tout multiple entier de la longueur spécifiée.

Règle de syntaxe:

factor = leadingDigit, {trailingDigit};

numberOfCharacters = (leadingDigit, {trailingDigit})|('(nx', factor,')');

La signification des composants factor («facteur») est comme suit

– `factor`: lorsque `factor` est présent, `numberOfCharacters` doit être un multiple entier de la valeur donnée dans `factor`. `Factor` ne doit pas contenir la valeur zéro.
– `numberOfCharacters`: détermine la quantité maximale des caractères contenus dan la chaîne.

D.6.2 Format de valeurs alphabétiques

Un “Alphabetic Value Format ((A) «Format de valeurs alphabétiques»)” définit le format de valeurs d'une chaîne de caractères qui contient des lettres alphabétiques. Donc, le contenu doit être pris dans les caractères de la rangée 00, cellule 20, cellule 40 à 7E, ou cellule C0 à FF, du Basic Multilingual Plane ((BMP) «Plan multilingue de base») (Plan 00 du Groupe 00) de l'ISO/CEI 10646-1.

NOTE 4 En raison des problèmes potentiels d'interprétation du contenu des valeurs au sein de composants d'un seul système ou de plusieurs systèmes, la recommandation est que, lorsque cela est possible, il convient de limiter les caractères utilisés au jeu G0 de l'ISO/CEI 10646-1 et/ou de la rangée 00 colonnes 002 à 007 de l'ISO/CEI 10646-1.

NOTE 5 Pour les autres langues supportées par les données de traduction, les caractères ou idéogrammes pertinents du jeu correspondant de caractères spécifique à une langue sont disponibles tels que définis par la norme Unicode. Dans la plupart des cas, il n'y aura pas de relation du type 1:1 (biunivoque) entre les caractères de la langue source aux caractères de la langue cible.

EXEMPLE Idéogrammes CJC (Chinois – japonais – coréen).

Règle de syntaxe:

AValue = 'A', (space | variableLengthIndicator), numberOfCharacters;

La signification des composants du format de valeurs A pour la représentation de valeurs est comme suit:

– 'A': la valeur doit être une chaîne, ou plusieurs sous-chaînes, de lettres alphabétiques.
– variableLengthIndicator: si variableLengthIndicator est présent, la chaîne peut contenir moins de caractères qu'il n'en est indiqué par numberOfCharacters. La chaîne doit contenir au moins un caractère.

D.6.3 Format de valeurs en caractères mélangés

Un format “Mixed value format («format de valeurs mélangées»(M))” définit le format de valeurs d'une chaîne qui peut contenir n'importe quel caractère spécifié à l'Article D.7.

NOTE Pour les autres langues supportées par les données de traduction, les caractères ou idéogrammes pertinents du jeu correspondant de caractères spécifique à une langue sont disponibles tels que définis par la norme Unicode.

EXEMPLE Caractères CJC (Chinois – japonais – coréen).

Règle de syntaxe:

MValue = 'M', (space | variableLengthIndicator), numberOfCharacters;

La signification des composants du format de valeurs M pour la représentation de valeurs est comme suit:

– `'M'`: la valeur doit être une chaîne, ou plusieurs sous-chaînes.

– `variableLengthIndicator`: si `variableLengthIndicator` est présent, la chaîne peut contenir moins de caractères qu'il n'en est indiqué par `numberOfCharacters`. La chaîne doit contenir au moins un caractère.

D.6.4 Format de valeurs numériques

Un “Number value format ((N) «format de valeurs numériques»)” définit le format de valeurs d'une chaîne de caractères qui contient des chiffres numériques seulement. Donc, le contenu doit être pris dans les caractères de la rangée 00, cellule 2B, cellule 2D, cellule 30 à 39, ou cellule 45, du Basic Multilingual Plane ((BMP) «plan multilingue de base» (Plan 00 du Groupe 00) de l'ISO/CEI 10646-1.

NOTE Pour les autres langues supportées par les données de traduction, les caractères ou idéogrammes pertinents du jeu correspondant de caractères spécifiques à une langue sont disponibles tels que définis par la norme Unicode.

EXEMPLE Le Tableau D.2 montre la transposition des chiffres européens “0” à “9” en chiffres arabes.

Tableau D.2 – Transposition des chiffres de style européen en chiffres arabes

Chiffres européens	9	8	7	6	5	4	3	2	1	0
Chiffres arabes	٩	٨	٧	٦	٥	٤	٣	٢	١	٠

Règle de syntaxe:

NValue = 'N', (space | variableLengthIndicator), numberOfCharacters;

La signification des composants du format de valeurs N pour la représentation des valeurs est comme suit:

– `'N'`: la valeur doit être une chaîne, ou plusieurs sous-chaînes, de chiffres numériques.

– `variableLengthIndicator`: si `variableLengthIndicator` est présent, la chaîne peut contenir moins de caractères qu'il n'en est indiqué par `numberOfCharacters`. La chaîne doit contenir au moins un caractère.

D.6.5 Format de valeurs en caractères numériques ou alphabétiques mélangés

Un “Mixed alphabetic or numeric characters value format («Format de valeurs en caractères numériques ou alphabétiques mélangés» (X))” définit le format des valeurs d'une chaîne qui contient des caractères alphanumériques, à savoir toute combinaison de caractères issus d'un format de valeurs A ou d'un format de valeurs N.

NOTE Pour les autres langues supportées par les données de traduction, les caractères ou idéogrammes pertinents du jeu correspondant de caractères spécifique à une langue sont disponibles tels que définis par la norme Unicode.

Règle de syntaxe:

XValue = 'X', (space | variableLengthIndicator), numberOfCharacters;

La signification des composants du format de valeurs X pour la représentation de valeurs est comme suit:

– `'X'`: la valeur doit être une chaîne, ou plusieurs sous-chaînes, de caractères alphanumériques, à savoir toute combinaison de caractères alphabétiques et numériques;

– `variableLengthIndicator`: si `variableLengthIndicator` est présent, la chaîne peut contenir moins de caractères qu'il n'en est indiqué par `numberOfCharacters`. La chaîne doit contenir au moins un caractère.

D.6.6 Format de valeurs binaires

Un “Binary Value Format (Format de valeurs binaires (B))” définit le format de valeurs d'une chaîne qui contient des caractères binaires, à savoir “0” ou “1”. Donc, le contenu doit être pris dans les caractères de la rangée 00, cellule 30 ou 31, du Basic Multilingual Plane ((BMP) «Plan multilingue de base») (Plan 00 du Groupe 00) de l'ISO/CEI 10646-1.

NOTE Pour les autres langues supportées par les données de traduction, les caractères ou idéogrammes pertinents du jeu correspondant de caractères spécifique à une langue sont disponibles tels que définis par la norme Unicode.

Règle de syntaxe:

```
BValue = 'B', (space | variableLengthIndicator), numberOfCharacters;
```

La signification des composants du format de valeurs B pour la représentation de valeurs est comme suit:

– `'B'`: la valeur doit être une chaîne, ou plusieurs sous-chaînes, consistant en valeurs binaires, à savoir les caractères “0” ou ”1” ou leurs séquences.

– `variableLengthIndicator`: si `variableLengthIndicator` est présent, la chaîne peut contenir moins de caractères qu'il n'en est indiqué par `numberOfCharacters`. La chaîne doit contenir au moins un caractère.

D.7 Exemples de valeurs

Le Tableau D.3 ci-dessous contient certains exemples pour les valeurs qui peuvent être contenues dans des nombres et des chaînes caractérisés par le plan de formats de valeurs ci-dessus.

Tableau D.3 – Exemples de valeurs numériques

Format de la valeur	Exemples de valeurs possibles
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

Format de la valeur	Exemples de valeurs possibles
NR4S..3N2D2	-1 ½ ; 12; +123 ¾; ¼ ;
A 19	Mon nom est 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;

Caractères issus de l'ISO/CEI 10646-1

Les caractères suivants doivent être utilisés pour les besoins du format Mixed value format (M) (voir D.5.3):

– tous les caractères à partir de la rangée 00 du plan multilingue de base (Basic Multilingual Plane – BMP) (Plan 00 du Groupe 00) de l'ISO/CEI 10646-1;

– les caractères des autres rangées du Basic Multilingual Plane (plan multilingue de base) de l'ISO/CEI 10646-1 tels qu'ils sont énumérés au Tableau D.4;

– pour les autres langues supportées par les données traduites, le jeu complet de caractères est disponible tel que défini par la norme Unicode.

NOTE 1 En raison des problèmes potentiels d'interprétation du contenu des valeurs au sein des composants d'un seul système ou de plusieurs systèmes, la recommandation est que, lorsque cela est possible, il convient de limiter les caractères utilisés au jeu G0 de l'ISO/CEI 10646-1 et/ou de la rangée 00 colonnes 002 à 007 de l'ISO/CEI 10646-1.

NOTE 2 La norme Unicode est publiée par The Unicode Consortium, P.O. Box 391476, Mountain View, CA 94039-1476, U.S.A., www.unicode.org.

Tableau D.4 – Caractères issus d'autres rangées du Basic Multilingual Plane (plan multilingue de base) de l'ISO/CEI 10646-1

Caractère	Nom	Rangée	Cellule
	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

Tableau D.4 *(suite)*

∴	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
∇	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

Tableau D.4 *(suite)*

Caractère	Nom	Rangée	Cellule
⊾	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

Tableau D.4 *(suite)*

Caractère	Nom	Rangée	Cellule
⃗	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

Tableau D.4 *(suite)*

Caractère	Nom	Rangée	Cellule
‰	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

Bibliographie

[1] CEI 60027 (toutes les parties), *Symboles littéraux à utiliser en électrotechnique*

[2] CEI 60748(toutes les parties), *Dispositifs à semiconducteurs – Circuits intégrés*

[3] CEI 61360-5, *Types normalisés d'éléments de données avec plan de classification pour composants électriques – Partie 5: Extensions au schéma du dictionnaire EXPRESS*

[4] CEI 80000 (toutes les parties)[3], *Grandeurs et unités*

[5] ISO/CEI 8824-1, *Technologies de l'information – Notation de syntaxe abstraite numéro un (ASN.1): Spécification de la notation de base* (disponible en anglais seulement)

[6] ISO/CEI 11179-5, *Technologies de l'information – Registres de métadonnées (RM) – Partie 5: Principes de dénomination et d'identification* (disponible en anglais seulement)

[7] ISO/CEI, *International Classification of Standard (ICS)*

[8] ISO 31 (toutes les parties), *Grandeurs et unités*

[9] ISO 639-1, *Codes pour la représentation des noms de langue – Partie 1: Code alpha-2*

[10] ISO 639-2, *Codes pour la représentation des noms de langue – Partie 2: Code alpha-3*

[11] ISO 704, *Travail terminologique – Principes et méthodes*

[12] ISO 843, *Information et documentation – Conversion des caractères grecs en caractères latins*

[13] ISO 1087-1, *Travaux terminologiques – Vocabulaire – Partie 1: Théorie et application*

[14] ISO 6523, *Échange de données – Structure pour l'identification des organisations*

[15] ISO 10303-1:1994, *Systèmes d'automatisation industrielle et intégration – Représentation et échange de données de produits – Partie 1: Aperçu et principes fondamentaux* (disponible en anglais seulement)

[16] ISO 10303-21, *Systèmes d'automatisation industrielle et intégration – Représentation et échange de données de produits – Partie 21: Méthodes de mise en application: Encodage en texte clair des fichiers d'échange* (disponible en anglais seulement)

[17] ISO 10303-42:2000, *Systèmes d'automatisation industrielle et intégration – Représentation et échange de données de produits – Partie 42: Ressource générique intégrée: Représentation géométrique et topologique* (disponible en anglais seulement)[7]

[18] ISO 13584-1:2001, *Systèmes d'automatisation industrielle et intégration – Bibliothèque de composants – Partie 1: Aperçu et principes fondamentaux* (disponible en anglais seulement)

---

7 Une nouvelle édition de l'ISO 10303-42 a été publiée en 2003.

[19] ISO 13584-24:2003, *Systèmes d'automatisation industrielle et intégration – Bibliothèque de composants – Partie 24: Ressource logique : Modèle logique de fournisseur* (disponible en anglais seulement)

[20] ISO 13584-25, *Systèmes d'automatisation industrielle et intégration – Bibliothèque de composants – Partie 25: Ressource logique: Modèle logique de fournisseur avec des valeurs d'ensemble et un contenu explicite* (disponible en anglais seulement)

[21] ISO 13584-32, *Systèmes d'automatisation industrielle et intégration – Bibliothèque de composants – Partie 32: Resources d'implémentation: OntoML: Langage de marquage ontologique* (disponible en anglais seulement)

[22] ISO 13584-511, *Systèmes d'automatisation industrielle et intégration – Bibliothèque de composants – Partie 511: Systèmes mécaniques et composants pour utilisation générale – Dictionnaire de référence pour éléments de fixation* (disponible en anglais seulement)

[23] ISO/TS 23768-1[8], *Roulements – Bibliothèque de composants – Partie 1: Dictionnaire de référence des roulements*

[24] ISO/TS 29002-5, *Systèmes d'automatisation industrielle et intégration – Échange de données caractéristiques – Partie 5: Schéma d'identification* (disponible en anglais seulement)

[25] ISO/TS 29002-20, *Systèmes d'automatisation industrielle et intégration – Échange de données caractéristiques – Partie 20: Services de résolution d'un dictionnaire de concept* (disponible en anglais seulement)

[26] ISO 80000 (toutes les parties)[9], *Grandeurs et unités*

[27] XML Schema Part 2: Datatypes. Second Edition. World Wide Web Consortium Recommendation 28-October-2004
Disponible sur <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
Disponible sur <http://www.w3.org/TR/MathML2/>

_____

8 A publier.

9 La série de parties ISO 31 a été annulée et remplacée par la série de parties ISO 80000/CEI 80000.

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
PO Box 131
CH-1211 Geneva 20
Switzerland

Tel: + 41 22 919 02 11
Fax: + 41 22 919 03 00
info@iec.ch
www.iec.ch