Creating and using reference data with ISO15926

Matthew West, Shell IT International, UK

1. Introduction

ISO 15926 specifies a data model [1], a reference data library, XML exchange templates, and a Façade integration environment to support the lifecycle integration of engineering data. A key element of this is the reference data library (RDL), which extends the semantics of the data model and is used in the definition of the templates. The data model, the **lifecycle_integration_schema**, is defined in EXPRESS [2] and incorporates the meta-model for the RDL and so the RDL can be extended by adding concepts to it that conform to the ISO 15926 data model.

The requirements for reference data come at different levels:

1. **Dictionary:** The concept has a name and definition only, and can be used consistently to refer to the concept defined by multiple users.
2. **Thesaurus:** In addition, the concept is defined in terms of a subclass/superclass hierarchy, and its own class membership, so that a term can be found and selected from similar terms, and concepts can be managed more effectively.
3. **Ontology:** In addition, properties, relationships and rules are defined, that can be inherited through the subclass/superclass relationships and provide the foundations for template definition, constraint checking, and other intelligent processing.

In this paper I take the reader through these stages of development, with particular emphasis on good practice, and how to populate the ISO 15926 data model in developing reference data.

2. Some preliminaries

The **lifecycle_integration_schema** requires that some distinctions are made. There is a supreme supertype, **thing**, but it is an abstract supertype, which means that it cannot be instantiated (have members) without one of its subtypes also being instantiated. The immediate subtypes are shown in Fig. 2.1. For the most part the choice will be between a **class** and an **individual**, where an **individual** is something that exists in space-time, and a **class** is a set of **thing** (refer to ISO 15926-2:2003 for full definitions). An **individual** is unusual in a Reference Data Library, but there are some notable exceptions like countries and companies that might be found there. However, for simplicity, in this paper I will assume that a **class**, is being created. It should also be noted that there will be many instances of **relationship**, **class_of_relationship**, and **multidimensional_object** involved in describing a **class** or an **individual**.

In addition to the information that can be held about the concepts themselves, the data model makes provision for holding limited information about the computer records used to represent the concepts in the Reference Data Library. The record management data is specified as attribute values that may be given for each entity instance. The scope of that information is as follows:

- for records that originate in another system, the date and time when this copy of the record was created in the current system;
- the date and time on which this record was first created in its originating system;
- the person, organisation, or system that first created this record in the originating system;
- the date and time that this record was logically deleted;
- for logically deleted records, the reason why the record was logically deleted, where logical deletion means that whilst the record is still available in the system as a matter of historical record, it is now recognised as always being an invalid statement.

These attributes are also shown in Fig. 2.1.

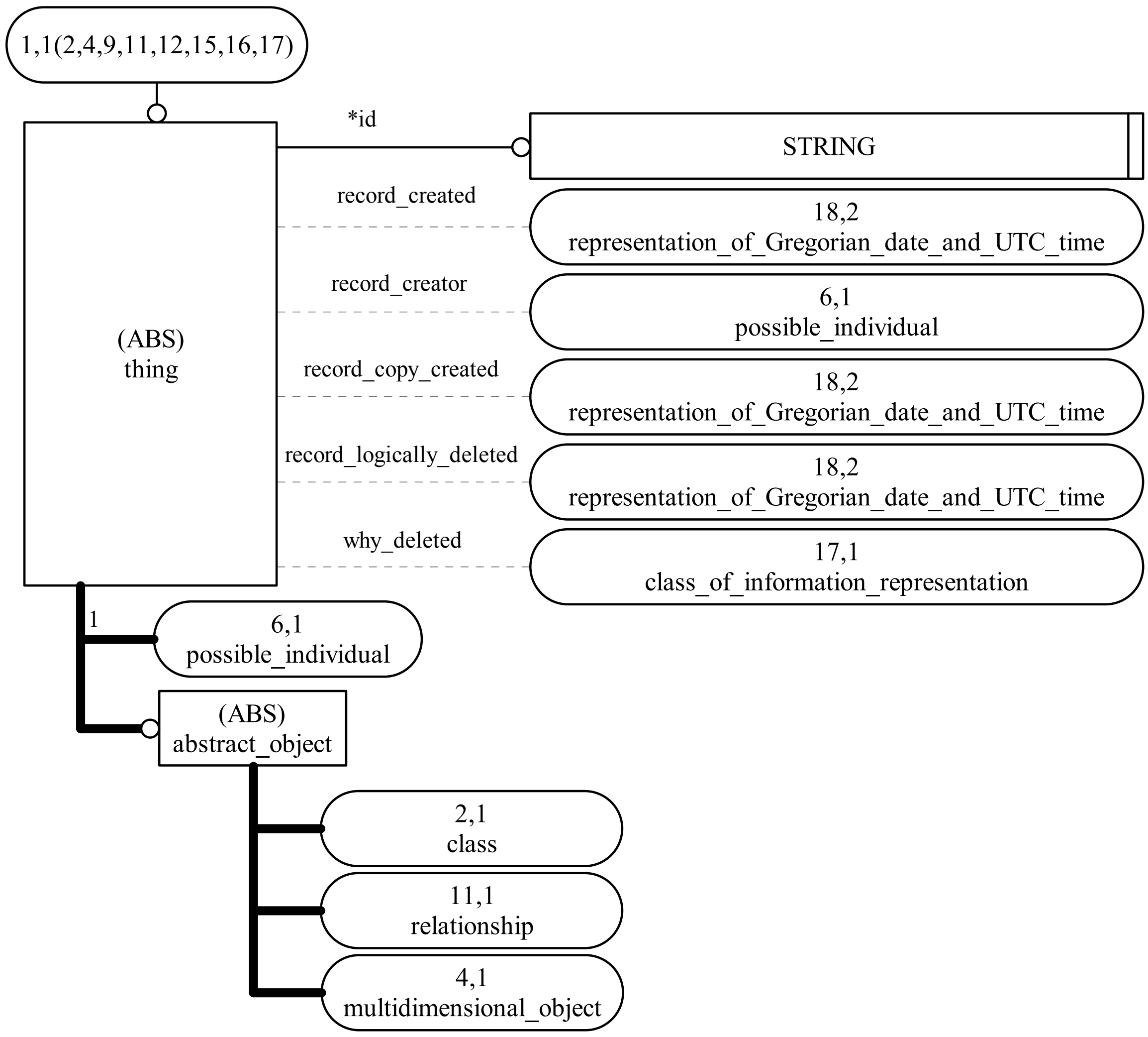


Fig. 2.1 Thing and some of its subtypes and attributes - taken from ISO 15926-2:2003

The model also specifies a unique **id** attribute. This is not for a name of the concept, but for an arbitrary identifier that is managed to be unique. Although this is specified in STRING format, using a number (in string format) is recommended.

All objects in an ISO 15926 Reference Data Library have these attributes; this includes the **EXPRESS_string** objects that hold the names and definitions, and the **relationship** and **class_of_relationship** objects between things.

3. Creating dictionary level reference data

The two key elements of a dictionary level concept are a name and a definition. Since these are critical items for reference data, it is worth a few words on good practice.

The name for a concept should be unambiguous, i.e. the meaning of the name should be clear even when it is taken out of context. Hence “plant” would be a poor name, and either “process plant”, “manufacturing plant” or “herbaceous plant” should be preferred.

The definition should have a number of parts. The most important of these is the core definition. This should be a single sentence and take the form: “A [broader term] that [distinguishing features]. With the first “A” removed this should be a suitable replacement for the name where the name appears. In addition to this, explanatory notes may be added, and examples should always be added. As an example, here is a definition for actual individual, adapted from ISO 15926-2:

> A possible individual that is a part of the space-time continuum that we inhabit, i.e. it exists in the present, past, or future of our universe, as opposed to some imagined universe.
> NOTE The things we plan can usually only be assumed to be part of some possible universe, until they come about.
> EXAMPLE 1 The Eiffel Tower is an actual individual.
> EXAMPLE 2 The fictional character, Sherlock Holmes, is a possible individual who is not an actual individual.

It is important to remember that for a particular concept there may be many names and definitions, both synonyms in one language, or names and definitions in different languages.

The parts of the **lifecycle_integration_schema** that are involved in giving names and definitions to a **class** are illustrated in Fig. 3.1, and Fig. 3.2.

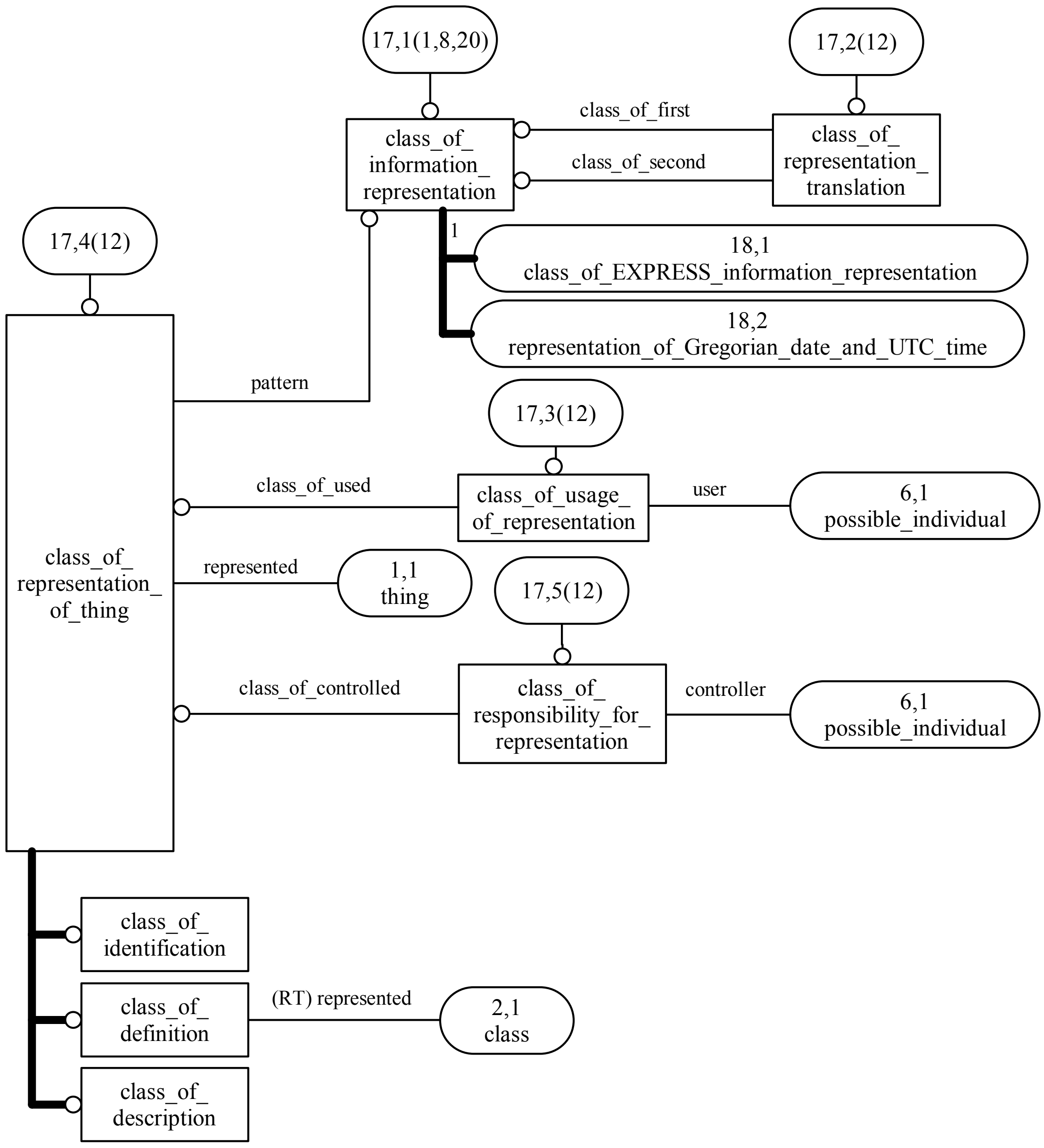

17,1(1,8,20)
17,2(12)
class_of_first
class_of_second
class_of_ information_ representation
class_of_ representation_ translation
1
18,1 class_of_EXPRESS_information_representation
18,2 representation_of_Gregorian_date_and_UTC_time
17,4(12)
pattern
17,3(12)
class_of_used
class_of_usage_ of_representation
user
6,1 possible_individual
class_of_ representation_ of_thing
represented
1,1 thing
17,5(12)
class_of_controlled
class_of_ responsibility_for_ representation
controller
6,1 possible_individual
class_of_ identification
(RT) represented
class_of_ definition
2,1 class
class_of_ description

Fig. 3.1 Identification and definition of thing – taken from ISO 15926-2:2003

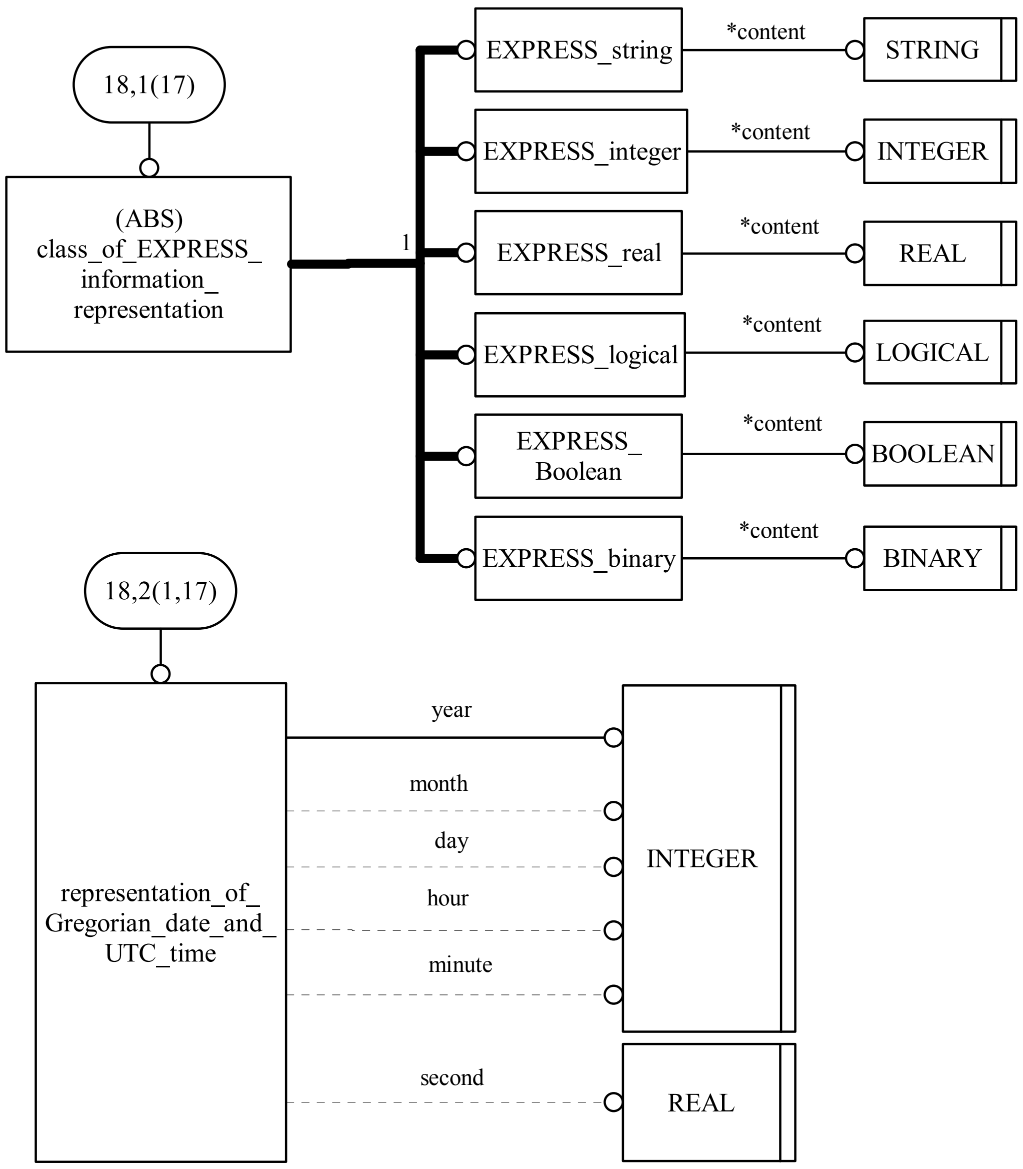


Fig. 3.2 Information representation – taken from ISO 15926-2:2003

To give something a name is to create a **class_of_identification** between an **EXPRESS_string** that holds the name and the **thing** (we are assuming a **class**) that has the name.

A point of interest here is that this is a **class_of_relationship**, rather than a **relationship**. This is because we are saying that any occurrence of the name is an identification of the **class** rather than pointing to the name on a particular piece of paper (say) and saying that is an **identification** of the **class**. These identification instances can also be recorded in the **lifecycle_integration_schema** but are generally less interesting, and certainly not appropriate to a Reference Data Library.

Providing the definition of a concept is done in a similar way. This time a **class_of_definition** is created between the **class** and the **EXPRESS_string**.

This dictionary level of information about a **thing** already gives value by allowing agreed names and definitions to be shared. However, there is little by way of computer intelligence that can be applied.

4. Creating thesaurus level reference data

There are two elements to extending an ISO15926 Reference Data Library to the thesaurus level:

1. Identify the entity types in the **lifecycle_integration_schema** that the concepts are instances of,
2. Placing the **class** in the subtype/supertype hierarchy of the Reference Data Library.

A key distinction here is between being an instance and being a subtype. For a subtype, all instances of the subtype are also instances of the supertype. However, this does not apply to instances. So for example, pump is an instance of **class_of_functional_object**, however, a particular pump (instance of pump) is clearly not an instance of **class_of_functional_object**. On the other hand, centrifugal pump is a subtype of pump, and any thing that is a centrifugal pump is also a pump.

4.1 Identifying entity type membership

The **lifecycle_integration_schema** is organised as a subtype/supertype hierarchy, so identifying the entity types that a **thing** is an instance of is, in principle, just a matter of walking through the subtype/supertype structure, at each level considering which of the subtypes (if any) the **thing** is an instance of. The important thing to note is that in many cases the subtypes are ANDOR subtypes, so that a **thing** can be an instance of more than one of the subtypes. The secret to accurate placement is being conscientious in considering all the valid possibilities, taking due note of the entity type definitions.

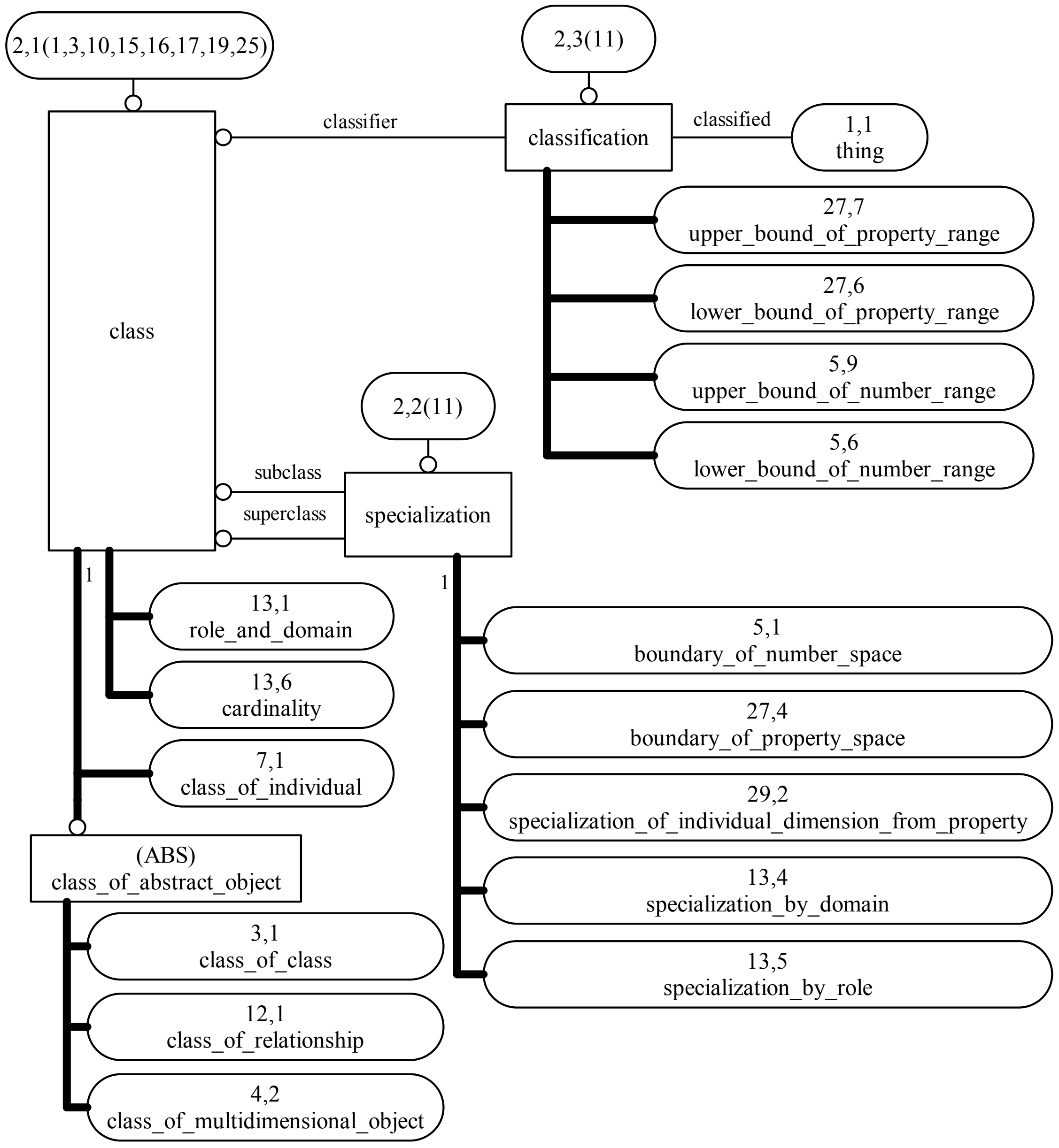
4.2 Placement within the subtype/supertype hierarchy

The ISO 15926 Reference Data Library will have the entity types of the **lifecycle_integration_schema** as its root. All classes in the Reference Data Library will be within a single subtype/supertype hierarchy, so the principles of placing a class within that hierarchy are similar to those for identifying the entity type it is an instance of, except that here of course we are identifying which classes it is a subtype of.

In placing a class in the subtype/supertype hierarchy, you may also discover that some existing subtypes of the supertype may be subtypes of the new class. This can lead to some redundant records from the previous state of the database which can be logically deleted.

This just leaves creating the records to reflect the changes. The part of the **lifecycle_integration_schema** that is used to do that is shown in Fig. 4.1 below. Instances of **specialization** are created for each subtype/supertype pair.

Adding the thesaurus level information turns the Reference Data Library from a list of defined concepts into an organised and much more usable, manageable, and scaleable. However, it is still not enough to enable exchange templates to be defined. For this you need the full ontology level, so that you can say things in a way that is not just human interpretable, but computer interpretable.



2,1(1,3,10,15,16,17,19,25)
2,3(11)
class
classifier
classification
classified
1,1 thing
27,7 upper_bound_of_property_range
27,6 lower_bound_of_property_range
5,9 upper_bound_of_number_range
5,6 lower_bound_of_number_range
2,2(11)
subclass
superclass
specialization
1
13,1 role_and_domain
13,6 cardinality
7,1 class_of_individual
(ABS) class_of_abstract_object
3,1 class_of_class
12,1 class_of_relationship
4,2 class_of_multidimensional_object
5,1 boundary_of_number_space
27,4 boundary_of_property_space
29,2 specialization_of_individual_dimension_from_property
13,4 specialization_by_domain
13,5 specialization_by_role

Fig. 4.1 Class, classification and specialization – taken from ISO 15926-2:2003

5. Creating ontology level reference data

Ontology level reference data enables computer support for questions like “What parts does a centrifugal pump have?” and “What properties does a pipe have?” This is achieved by holding instances of **class_of_relationship** that enable these questions to be answered.

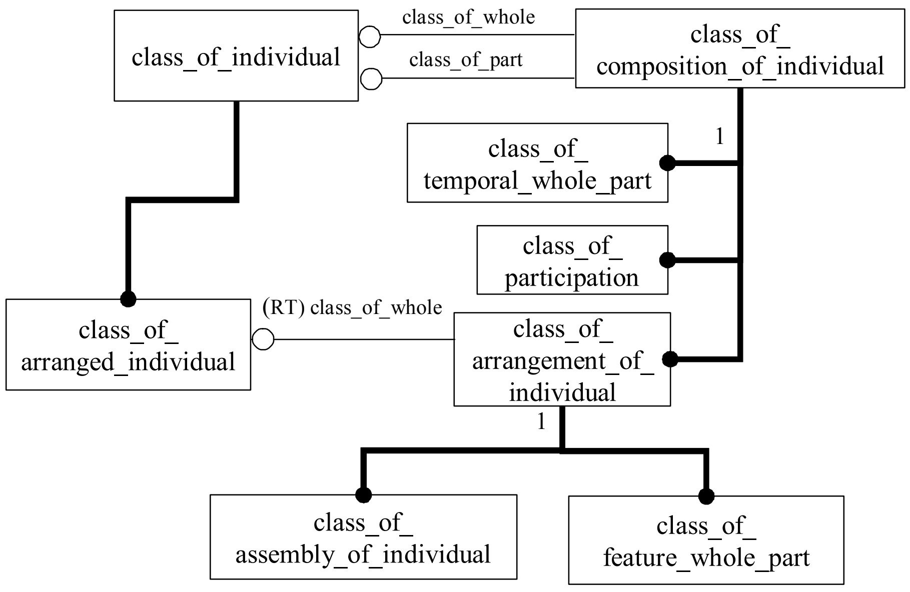
In the **lifecycle_integration_schema** there are two ways this can be done:

- There are a basic set of classes of relationships provided within the schema,
- The schema provides a meta-model so that additional classes of relationship can be added to the Reference Data Library and then used.

Here, due to lack of space, we will restrict ourselves to a few classes of relationship within the **lifecycle_integration_schema**.

5.1 Parts list

One of the things one may wish to do is to identify the parts of a type of equipment like a centrifugal pump. The part of the **lifecycle_integration_schema** required for this is illustrated in Fig. 5.1.



class_of_individual
class_of_whole
class_of_part
class_of_ composition_of_individual
1
class_of_ temporal_whole_part
class_of_ participation
class_of_ arranged_individual
(RT) class_of_whole
class_of_ arrangement_of_ individual
1
class_of_ assembly_of_individual
class_of_ feature_whole_part

Fig. 5.1 The data model for a parts list.

As an example consider the **class,** centrifugal pump with a parts list that includes an impeller. The **class** of shaft and the **class** of impeller have to be created in their own right, and then an instance of **class_of_assembly_of_individual** is created between these classes. Additional detail can be added, such as how many impellers (maximum and minimum) a centrifugal pump can have, however the description of this is beyond the space available here.

5.2 Recognised properties

In the **lifecycle_integration_schema**, properties are considered to be classes, so when a pipe has a temperature of 25 C, you say that (a temporal part of) the pipe is a member (**classification**) of the **property** that maps to 25 on the Celsius scale. So if we want to say that pipes in general can have a temperature, then what we want to say is that a member of the pipe class is classified by a member of the temperature class.

Now in the **lifecycle_integration_schema** temperature is an instance of **class_of_property_space**, pipe is an instance of **class_of_inanimate_physical_object** and that a member of the pipe class is a member of a temperature class is indicated by a **class_of_classification**. This is illustrated in Fig. 5.2 below.

The kinds of information that can be defined in this ontology level of Reference Data Library are what are required to support the computer interpretable definition of data sheets that can be supported by the XML templates defined in ISO 15926-7. The advantage of the templates is that they simplify the understanding needed to perform day-to-day exchanges, since the complexity is dealt with in the definition of the template, leaving a relatively straightforward task for the user. However, the template definition in turn grounds the data exchanged in the data model and Reference Data Library of ISO15926.

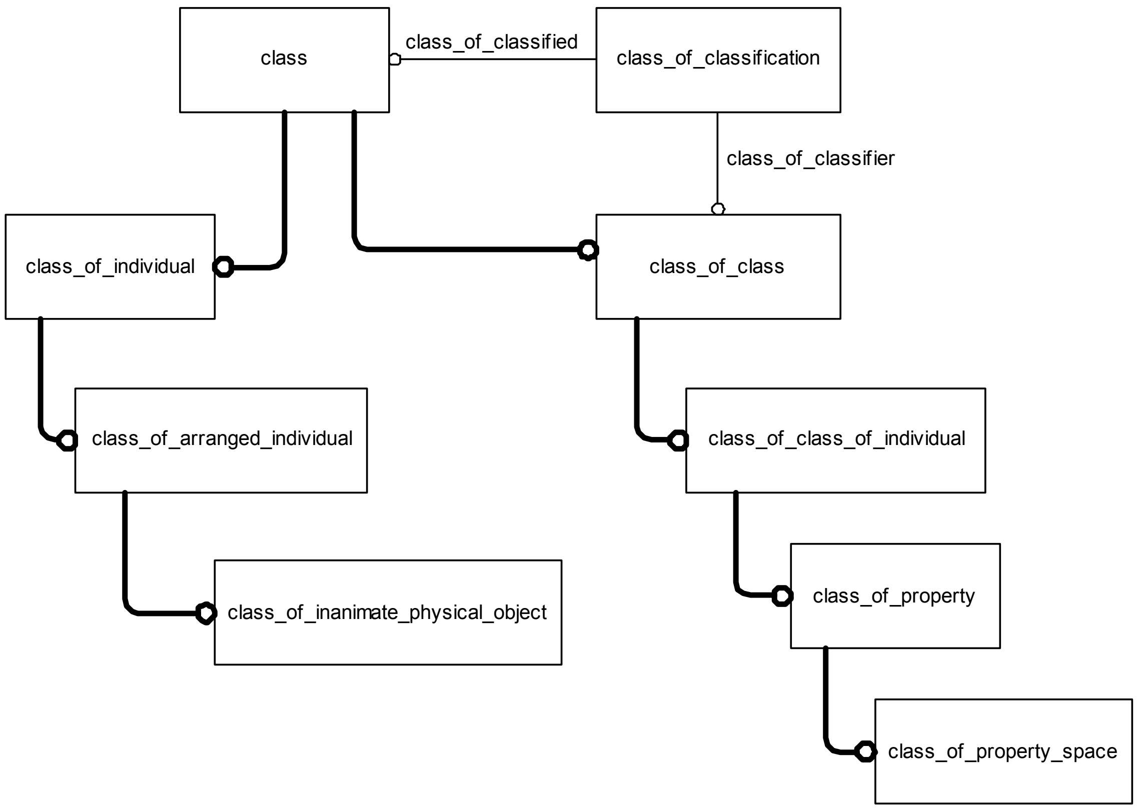


Fig. 5.2 Assigning classes of properties to classes of inanimate physical object – extract from ISO 15926-2:2003

6. Conclusions and further work

This brief paper has illustrated the way in which ISO 15926 can be used to define and use a Reference Data Library to different levels, and points out the advantages to be gained as the quality of Reference Data is improved.

The data model presented here has already reached International Standard status. The Reference Data Library and definition of XML Exchange Templates have been submitted as Technical Specifications.

Work is now needed to develop and extend both the Reference Data Library and a supporting set of XML exchange templates for industrial use (see http://www.infowebml.ws .

References

[1] ISO 15926-2:2003, *Industrial automation systems and integration — Integration of life-cycle data for process plants including oil and gas production facilities — Part 2: Data model.*

[2] ISO 10303-11:1994 - Description methods: The EXPRESS language reference manual.