ISO 15926

the Lingua Franca for

global interoperability

a gentle introduction

by Hans Teijgeler - ISO 15926 coauthor

Table of Contents

Table of Figures

INTRODUCTION

Ever heard of medical systems that cannot exchange information? And systems of police, fire fighters, and ambulances not being able to exchange information?

The process industries have that same problem. During the life cycle of any process plant, from conceptual design, via engineering and construction, to operations and maintenance, during the decades that a plant exists, information between the disciplines of all parties involved has to be exchanged. That used to be a problem and it still is.

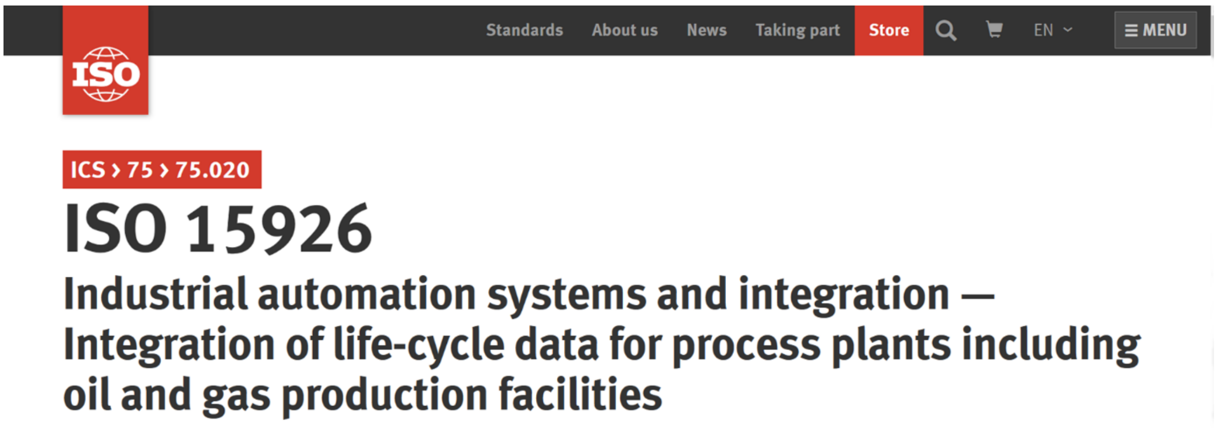
In 1991 the European Union launched an ESPRIT project called ProcessBase with the task to develop a data model for life-cycle information. From there on representatives of plant owners, EPC contractors and equipment manufacturers kept developing that model. It reached ISO Standard status in 2003. Also other 'parts' of that standard ISO 15926 have been developed.

The parts that will be explained in this paper are:

- Part 2 – Data Model
- Part 4 – Reference Data
- Part 7 – Templates
- Part 8 – Implementation in RDF
- Part 9 – Triple Store (still in development)
- Part 10 - Conformance Testing (still in development)

There are other parts, but these are not in the scope of this paper.

The overall title of ISO 15926 is:



ISO 15926

Industrial automation systems and integration — Integration of life-cycle data for process plants including oil and gas production facilities

Figure 1 – Title of ISO 15926

This integration of life-cycle information is doable for a software supplier, but to do that on a global scale and for all software that is used in the process industries appeared to be a task of Herculean proportions. It took us nearly 30 years to master that task.

It is now up to the plant owner/operators to take the lead in the application of ISO 15926.

WHY ISO 15926 ?

In today's world we often see that people from all over the world can communicate in meetings and on the Internet. This is possible despite the fact that many of them speak another language back home. In those meetings and on the Internet they use a common language, be it English, Spanish, Chinese, or any other major language.

Computer systems also speak their own language internally, and as such can only communicate with other systems by means of interfaces. The number of interfaces grows exponentially with the number of systems. If you have two systems you have two interfaces. But if you have ten systems you already have, as a maximum, 90 interfaces.

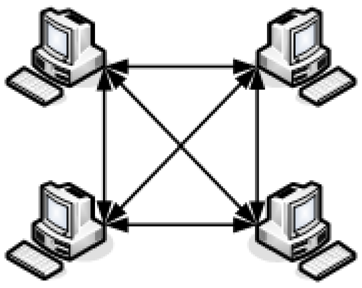


Figure 2 - Point-to-point interfaces

This is, at present, the common situation. The cost for setting up and maintaining all these interfaces account for a substantial part of all IT costs.

Through the years this problem hasn't been solved. Sure, we see an abundance of interface tools, but they don't really help to solve that problem, because they keep being, for the most, application-to-application or an 'island of automation'.

So, the solution is that we teach all our systems one common language.

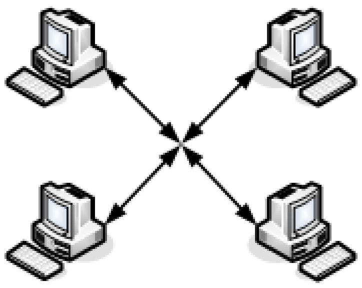


Figure 3 - One interface only via a hub

This language can be anything, if there wouldn't be more requirements.

However, the goal of integrating life-cycle information for the facilities meant that we had to cope with the following problems:

- the number of different application programs is very large;
- the terminology for the various phases in that life cycle are widely different;
- we need a stable standard that will be there for decades.

In this paper the ISO 15926 solutions to these problems are covered.

ISO 15926 AND THE SEMANTIC WEB

Let us peek into the end result as a kind of sign post:

- all things, that 'own' data, are 'declared', so typed and identified, and are on the internet (with due security measures in place) ;
- all relationships are, one by one (for the most), turned into things that are typed and identified; these things are called 'templates' ;
- all technical and operational information of a facility can be mapped to templates, with a so-called 'effective date-time' and, when required, other meta data such as status, access rights, etc. ;
- the basis for the integration of this information are functional diagrams such as P&IDs, that provide the topological data ;
- all information is turned into RDF N-triples and stored in triple stores ;
- each triple is a set of three internet addresses that are subsequently for:
  - 'subject' (the thing about which the triple gives information) ;
  - 'predicate' (the relation) ;
  - 'object' (the thing the predicate points at) ;

  e.g. Vessel V-145 hasAsPart Nozzle N3 ;
- each template forms an encapsulated set of triples, representing data and meta data;
- it is possible to create 'federations' of triple stores, even at individual login level, so that all the information that is required is accessible on a need-to-know basis ;
- triple stores, alone or as a federation, can be queried, using the SPARQL query language ;
- no information is replaced, as in a customary data base, it 'sediments' and can always be mined, even years later for a performance analysis or revamp ;
- operational data in the form of measured values can be stored in times series, e.g. per shift ;
- information, that was valid at a given point in time, can be fetched by looking for the latest date-time of each thing that is in the query result before that point in time – the actual point in time is the default for the latest information ;
- An application with an ISO 15926 compliant export adapter maps its output data to the ISO 15926 format, validate it, and uploads it to a triple store ;
- An application with an ISO 15926 compliant import adapter launches a query to a triple store or federation of triple stores, imports the query result, and maps it to its internal format. Since everything is standardized the set-up of those queries is predictable.

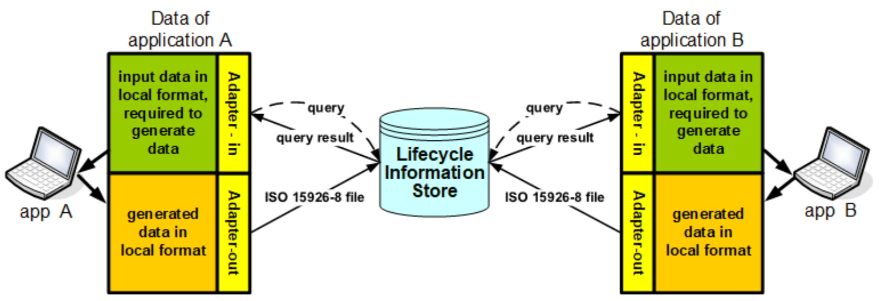

Data of application A
input data in local format, required to generate data
Adapter - in
generated data in local format
Adapter-out
app A
query
query result
ISO 15926-8 file
Lifecycle Information Store
query
query result
ISO 15926-8 file
Data of application B
Adapter - in
input data in local format, required to generate data
Adapter-out
generated data in local format
app B

Figure 4 – ISO 15926 in one picture

OVERVIEW

First of all this: We live in a globalized world and the applications that are or will be used during the life of a plant are located around the globe. All of them need input data and all of them produce data that may, at some point in time, be an input to one or more other applications.

ISO 15926 is exclusively meant to

- provide global interoperability between all applications ;
- integrate all technical and operational information of a facility during its entire life.

The fact that information is integrated supports above interoperability, but also archiving and knowledge mining.

In a nutshell ISO 15926 is set up as shown in Fig. 5.

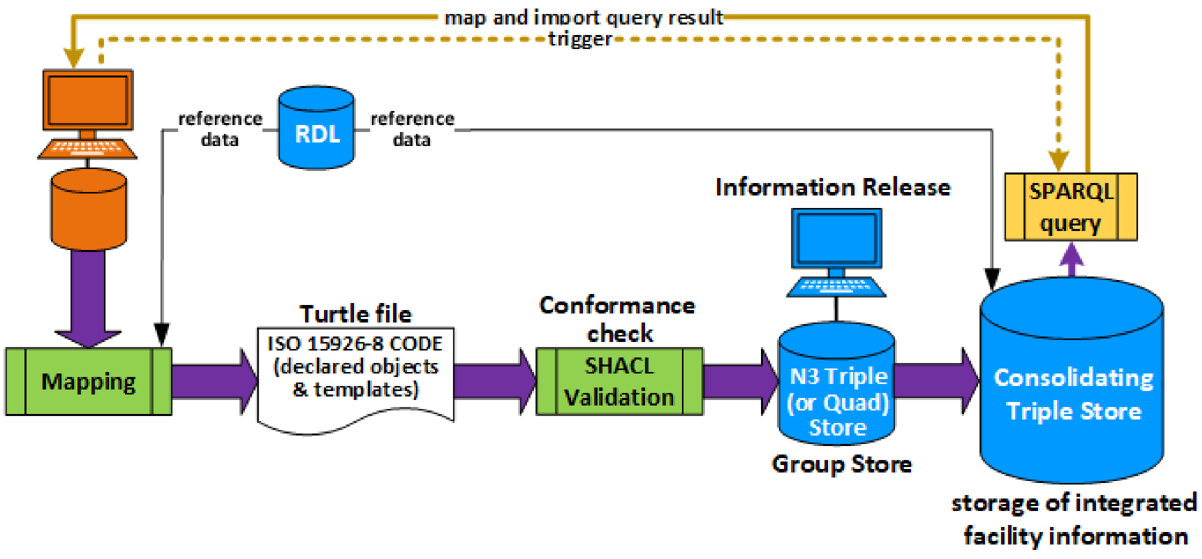

map and import query result
trigger
reference data
RDL
reference data
Information Release
SPARQL query
Turtle file
ISO 15926-8 CODE (declared objects & templates)
Conformance check
SHACL Validation
Mapping
N3 Triple (or Quad) Store
Group Store
Consolidating Triple Store
storage of integrated facility information

Figure 5 – Overview

Each application gets an ISO 15026 export adapter that maps data from the internal format, naming convention and logic to an exchange file according ISO 15926-8.

The contents of that file are validated using the W3C SHACL standard, or a similar technology, and stored in a triple store of the discipline involved.

When the data can be shared with parties outside the discipline the latter will release them to a consolidating triple store, e.g. for a project or a plant.

Each application gets an import adapter that can launch queries (SPARQL or other) to that consolidating triple store and map the query results to data in the internal format, naming convention and logic.

Information can be handed over to another triple store, for example from a project store to a plant store. Hand-over means transfer of custody to the owner of the target triple store.

The components of the above overview, including the ones not shown, will be discussed in the following chapters.

PART 2 DATA MODEL

The data model defined in Part 2 is different from the usual data models. It is highly generic and has classes and relationships as autonomous concepts that are called 'entity types'. The model has 201 such entity types.

These entity types are interrelated into a taxonomy, a hierarchy of classes. The top of that hierarchy is shown in Fig. 6.

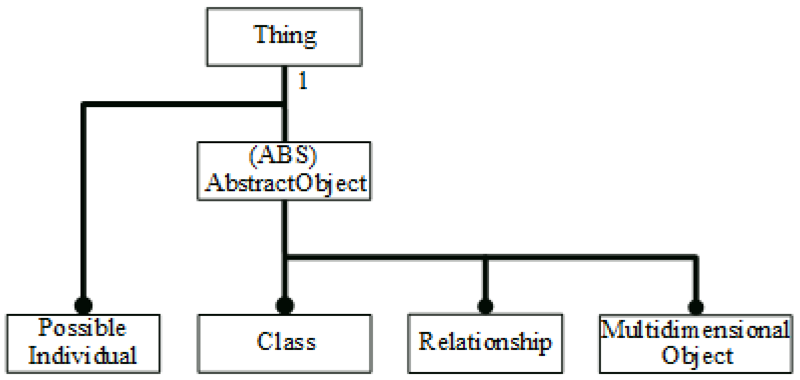


Figure 6 – Top of Part 2 taxonomy

In Fig. 6 all blocks are a 'class' in terms of general set theory.

Class is defined in Part 2 as "A class is an understanding of the nature of things that divides things into things that are members of the class and things that are not, **according to one or more criteria**. The identity of a class is its membership. No two classes have the same membership."

PossibleIndividual is a Thing that exists in space and time. This includes:
- things that actually exist, or have existed,
- things that are fictional or conjectured and possibly exist in the past, present or future.
Examples: The pump with serial number ABC123, Battersea Power Station, Shakespeare, and the starship "Enterprise" can be represented by instances of PossibleIndividual.

Relationship is an AbstractObject that indicates something that one thing has to do with another.
Examples: Classification, Specialization, CompositionOfIndividual, ConnectionOfIndividual, RelativeLocation, Definition, Description, Identification.

MultidimensionalObject is an AbstractObject that is an ordered list of Thing.

The relationship between Class and PossibleIndividual is shown in Fig. 7.

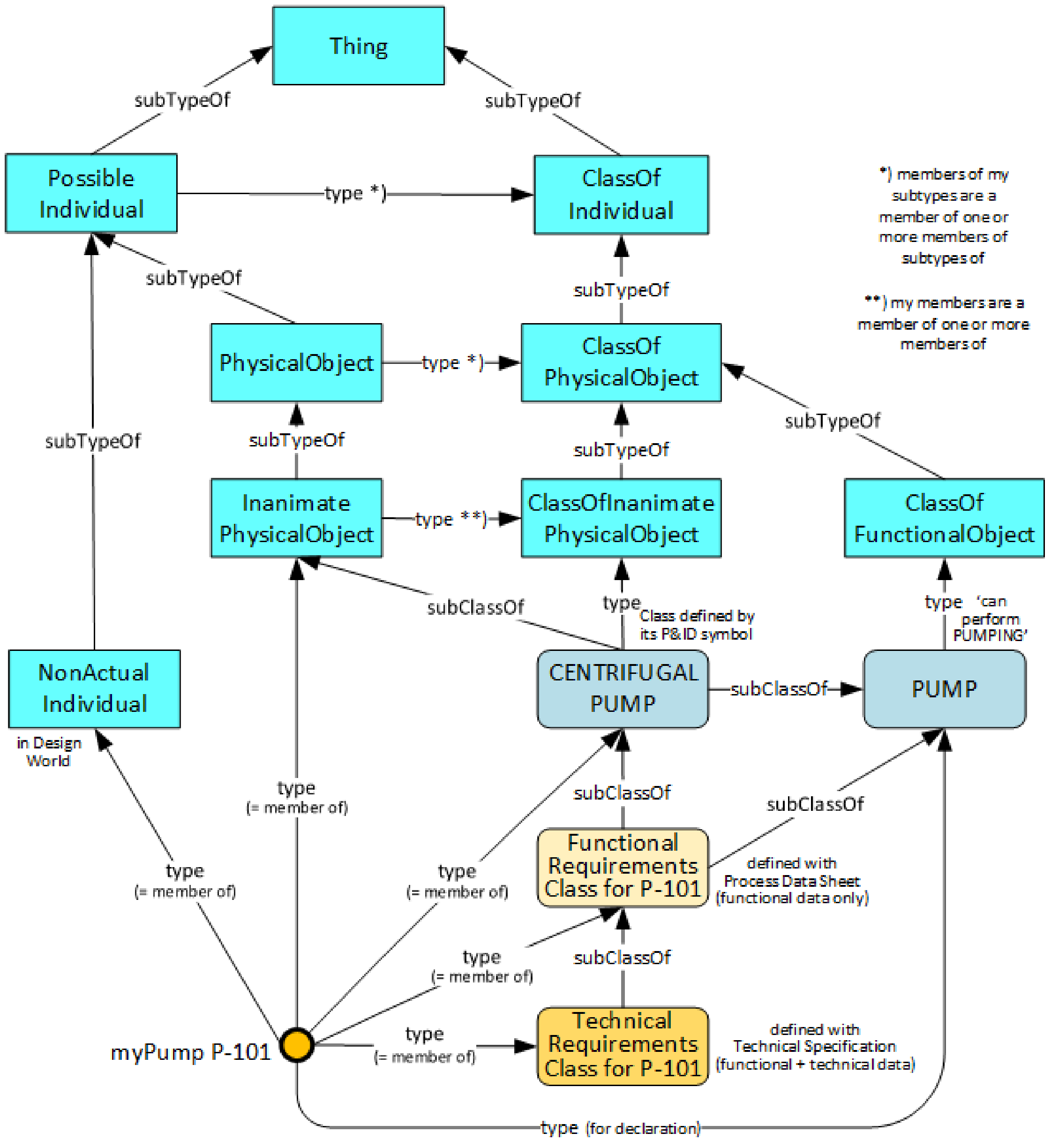


Figure 7 – Example of interrelationships between classes and individuals

This diagram shows how the Part 2 entity types (in blue) are related to each other, to Part 4 RDL classes (in grey), to a functional requirements class, to a technical requirements class defined by a specification, and finally to a member of InanimatePhysicalObject and of NonActualIndividual (see next page).

By convention we speak about subTypeOf in case of specialisations of entity types, and about subClassOf in other cases.

The 'type' arrows with one or two asterisks are not a simple 'is a member of' but somewhat more complex, as indicated in the diagram.

It is shown how the typing of an individual can progress over time, from declaration to presentation on a P&ID, to, finally, its technical requirements class on a specification.

POSSIBLE INDIVIDUALS

PossibleIndividual is a 'class' and is the top of a taxonomy of types of PossibleIndividual, as shown in Fig. 8 below.

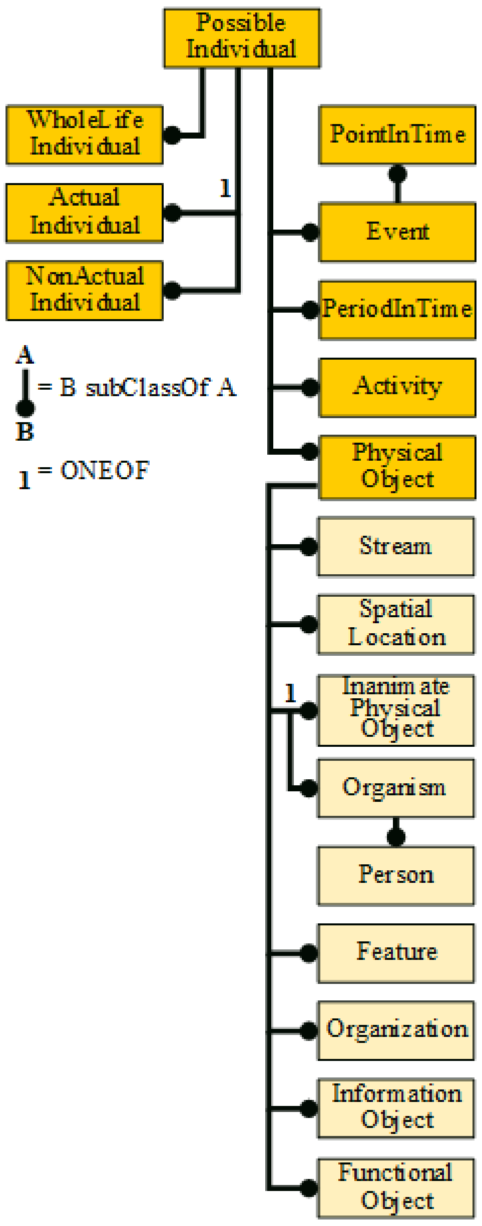


Figure 8 - Subtypes of PossibleIndividual

Here are the definitions:

WholeLifeIndividual - see chapter TEMPORAL PARTS

ActualIndividual and **NonActualIndividual** – see chapter POSSIBLE WORLDS

Event - An Event is the temporal boundary of the existence of one or more members of PossibleIndividual.
Example - The connection of power to a pump is an Event that marks the beginning of an operational state of that pump.

PointInTime - A PointInTime is an Event with zero extent in time.

Activity - An Activity brings about change by causing the Event that marks the beginning or ending of a PossibleIndividual or a temporal part therof.

PhysicalObject - A PhysicalObject is a PossibleIndividual that is a distribution of matter and/or energy.
Stream - A Stream is a PhysicalObject that is material or energy moving along a path.

SpatialLocation - A SpatialLocation is a Physical_Object that has continuity of relative position.
Examples - license block, construction area, country, air corridor, maritime traffic zone, hazard control zone, points, lines, planes.

InanimatePhysicalObject – An InanimatePhysicalObject is a PhysicalObject that is not living.

Organism – An Organism is a PhysicalObject that is living.

Person – A Person is an Organism of the species Homo Sapiens.

Feature - A Feature is a contiguous, non-separable part of some PossibleIndividual and it has an incompletely defined boundary with the rest of that PossibleIndividual.
Example – a flange of a cast iron valve body

Organization – An **Organization** is a PossibleIndividual that is composed of temporal parts of people and other assets, and is organised with a particular purpose.

InformationObject – An **InformationObject** is a PhysicalObject that is a carrier of information.

FunctionalObject – A **FunctionalObject** is a PhysicalObject that only has a function or purpose.

CLASSES AND INDIVIDUALS

In Fig. 9 below the subtypes of PossibleIndividual are related to subtypes of ClassOfIndividual. For the definitions see http://15926.org/topics/data-model/index.htm .

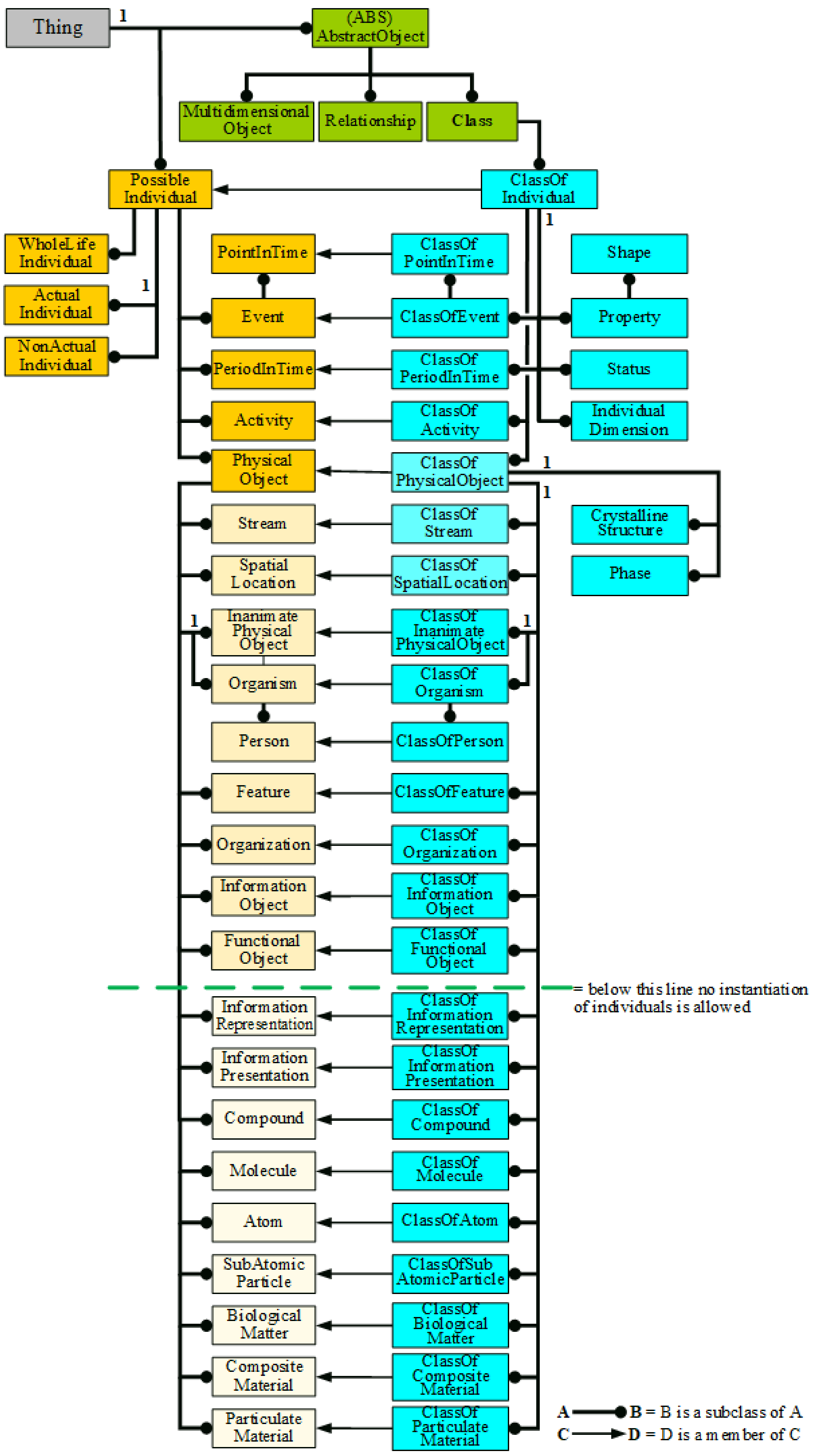


Figure 9 - PossibleIndividuals with their Classes

RELATION, RELATIONSHIP AND CLASS OF RELATIONSHIP

A Relationship is a link between two things, representing information.

Usually (not always) an instance of Relationship relates two instances of subtypes of PossibleIndividual, and an instance of ClassOfRelationship usually relates two instances of subtypes of ClassOfIndividual.

There is a difference between 'relation' and 'relationship' in ISO 15926. This is shown in Fig. 10.

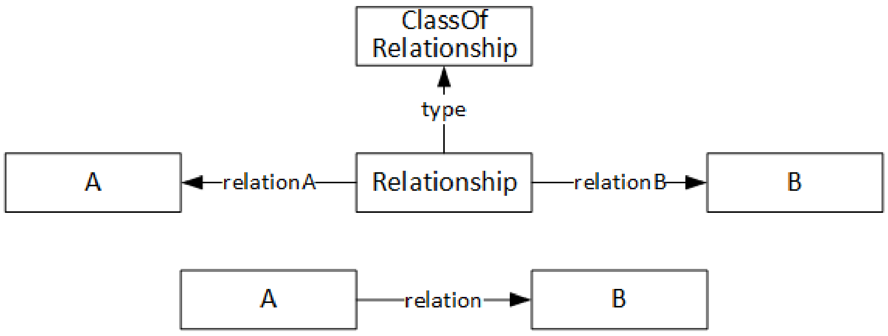


Figure 10 - Relation vs Relationship

The following subtypes of Relationship are defined below:

Approval - An Approval is a Relationship that indicates that a Relationship has been approved by a PossibleIndividual that is an APPROVER.

CauseOfEvent - A CauseOfEvent is a Relationship that indicates that the caused Event is caused by the causer Activity.

Classification - A Classification is a Relationship that indicates that the classified Thing is a member of the classifier Class; Classification is not transitive.

ComparisonOfProperty - A ComparisonOfProperty is a Relationship that indicates the magnitude of one Property is greater than that of another.

CompositionOfIndividual - A CompositionOfIndividual is a Relationship that indicates that the part PossibleIndividual is a part of the whole PossibleIndividual. A simple composition is indicated, unless a subtype is instantiated too; CompositionOfIndividual is transitive.

ConnectionOfIndividual - A ConnectionOfIndividual is a Relationship that indicates that matter, energy, or both can be transferred between the members of PossibleIndividual that are connected, either directly or indirectly.

FunctionalMapping - A FunctionalMapping is a Relationship that indicates that the input gave the result as determined by the classifying ClassOfFunctionalMapping.

IndirectProperty - An IndirectProperty is a Relationship between a Property and a PossibleIndividual. The nature of the IndirectProperty is defined by its Classification by a ClassOfIndirectProperty. A property is indirect when it does not directly apply to the PossibleIndividual it applies to, but is derived from some process.

IndividualUsedInConnection - An IndividualUsedInConnection is a Relationship that indicates that a PossibleIndividual is used in a ConnectionOfIndividual.

IntendedRoleAndDomain - An IntendedRoleAndDomain is a Relationship that indicates the RoleAndDomain some temporal part of the PossibleIndividual is intended to take with respect to some Activity.

InvolvementByReference - An InvolvementByReference is a Relationship that indicates that a Thing is referred to in an Activity.

LifecycleStage - A LifecycleStage is a Relationship that indicates the interest that a PossibleIndividual has in some PossibleIndividual.

PossibleRoleAndDomain - A PossibleRoleAndDomain is a Relationship that indicates that a player PossibleIndividual can possibly play the played RoleAndDomain.

Recognition - A Recognition is a Relationship that indicates that a Thing is recognized through an Activity.

RelativeLocation - A RelativeLocation is a Relationship that indicates that the position of one PossibleIndividual is relative to another.

RepresentationOfThing - A RepresentationOfThing is a Relationship that indicates that a PossibleIndividual is a sign for a Thing.

ResponsibilityForRepresentation - A ResponsibilityForRepresentation is a Relationship that indicates that the controller PossibleIndividual administers the controlled RepresentationOfThing.

Specialization - A Specialization is a Relationship that indicates that all members of the subclass are members of the superclass. Specialization is transitive.

TemporalSequence - A TemporalSequence is a Relationship that indicates that one PossibleIndividual precedes another in a temporal sense.

UsageOfRepresentation - A UsageOfRepresentation is a Relationship that indicates that the RepresentationOfThing is used by the PossibleIndividual. Usage does not imply responsibility.

and the 'Catch All' :

OtherRelationship - An OtherRelationship is a Relationship that is not a member of any of the other explicit subtypes of Relationship. The meaning of an OtherRelationship is specified by a Classification by an instance of ClassOfRelationshipWithSignature.

An OtherRelationship can be modeled, see for example the template ProductManufacturedBy in http://15926.org/templatespecs/IN-OTHRL-500.xml

The definition of subtypes of ClassOfRelationship often goes like this:

A **ClassOfXXX** is a **ClassOfRelationship** whose members are instances of **XXX**

which is not overly interesting.

ONTOLOGIES

When discussing the role of instances of subtypes of ClassOfRelationship we must discuss the overloaded term 'Ontology' first. Overloaded because there are many definitions.

Here is a paraphrased definition by Tom Gruber:

1. *An ontology defines a set of representational primitives with which to model a domain of knowledge or discourse.*
2. *These representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members).*
3. *The definitions of the representational primitives include information about their meaning and constraints on their logically consistent application.*
4. *An ontology can be viewed as a level of abstraction of data models, intended for modeling knowledge about individuals, their attributes, and their relationships to other individuals.*

Basically we define at Class level what shall apply at PossibleIndividual level, which in turn defines what shall apply at the level of instances of PossibleIndividual.

Assume, for example, the following very simple construct at class level and its implementation at instance level:

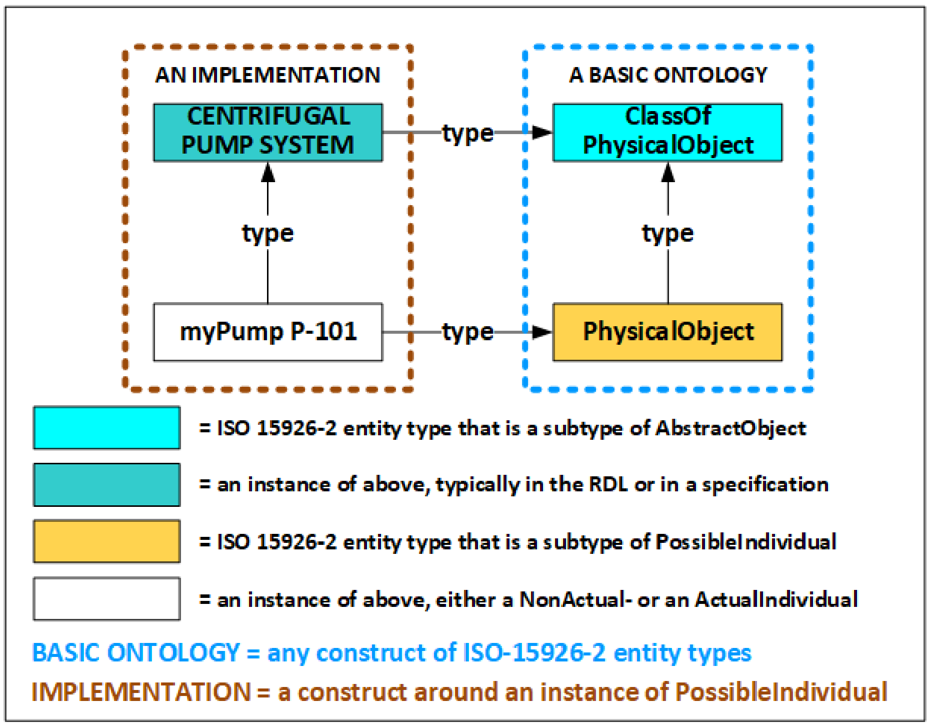

AN IMPLEMENTATION
CENTRIFUGAL PUMP SYSTEM
type
myPump P-101
type
type
A BASIC ONTOLOGY
ClassOf PhysicalObject
type
PhysicalObject
= ISO 15926-2 entity type that is a subtype of AbstractObject
= an instance of above, typically in the RDL or in a specification
= ISO 15926-2 entity type that is a subtype of PossibleIndividual
= an instance of above, either a NonActual- or an ActualIndividual
BASIC ONTOLOGY = any construct of ISO-15926-2 entity types
IMPLEMENTATION = a construct around an instance of PossibleIndividual

Figure 11 – Basic Ontology and Implementation

Ontologies in ISO 15926 can involve more than above one level as is shown in Fig. 12.

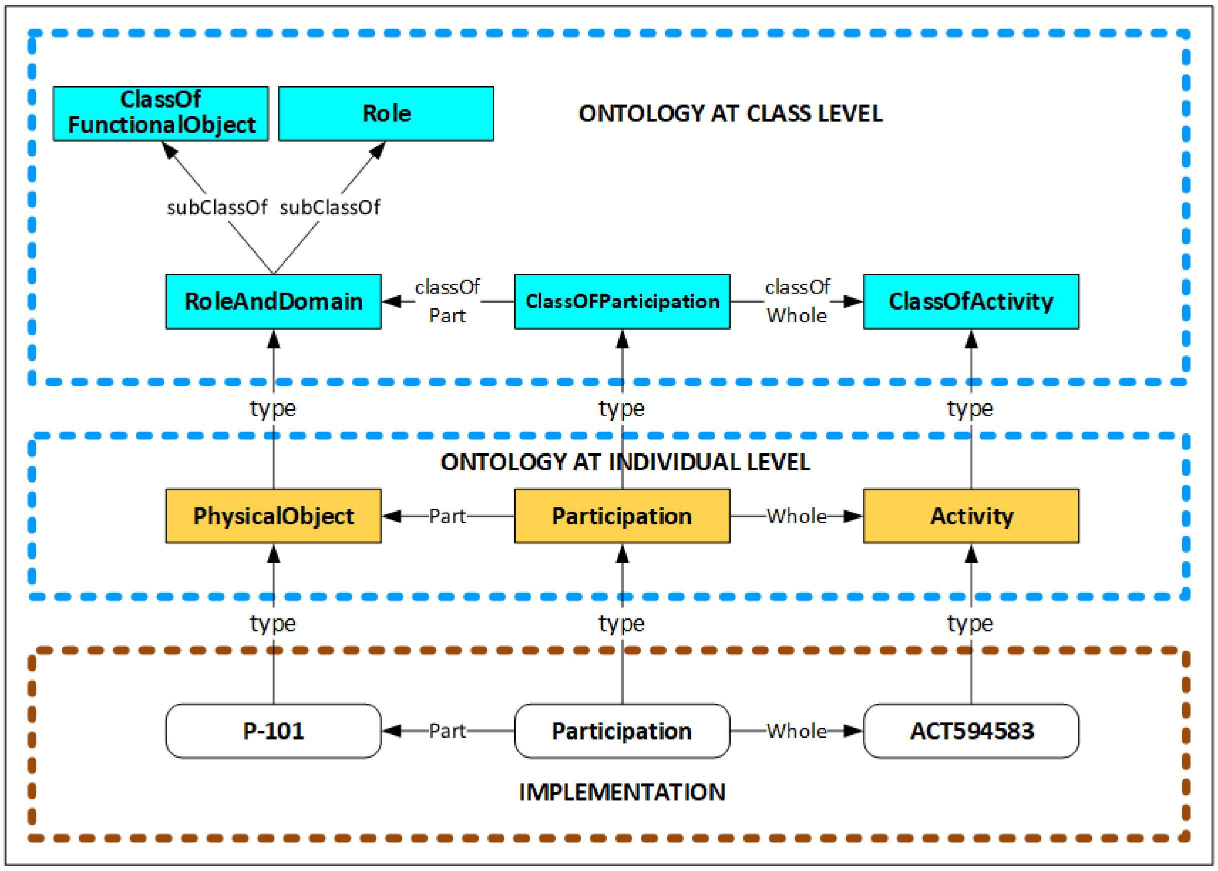


Figure 12 – Two-level ontology and implementation

This can be specialized, for example by replacing ClassOfFunctionalObject with its instance PUMP SYSTEM, Role with PERFORMER, and ClassOfActivity with PUMPING. Then P-101 shall be a member of PUMP SYSTEM and ACT594583 shall be a member of PUMPING.

A nearly infinite number of other, Part 2 compliant, combinations is possible.

In order to avoid miscommunications due to different opinions about the scope of these combinations we have developed a set of 235 (and counting) ISO 15926-7 compliant 'Templates'. These are ontologies that represent one particular information element which has, in general, the same information content as an attribute in a data base.

In the next chapter the concept of templates is further detailed.

PART 7 TEMPLATES

Information is a set of relationships between two or more objects that are known by, or made known to, the recipient. For example, when you hear that John Doe and Mary Bloggs have been married, that is non-information unless you know these people. So, in computer terms, we must "declare" the objects before we can represent any information about them.

Declaring a thing involves the following:

- give an identifier to the thing; the use of a UUID is recommended ;
- type the thing with the applicable Part 2 entity types ;
    - in case of a PossibleIndividual type with the applicable 'essential type' ;
    - in case of a Class define of which 'essential type' it is a subclass ;
- give the thing a label, e.g. "P-101" ;
- state during which 'lifecycle activity' this declaration was done, e.g. 'plant design' ;
- state as of which dateTime the declarations becomes effective (valid) .

NOTES

1. A 'UUID' is a Universally Unique Identifier. The use of it avoids duplicate identifiers, which is good to have in the context of life-cycle information, collected for decades by many parties. UUID allocation does not require any coordination.
2. 'Essential Type' is the applicable instance of ClassOfFunctionalObject, such as PUMP in Fig. 7. It tells what the thing is in essence. Showing a class that is lower in the hierarchy, such as CENTRIFUGAL PUMP, is risky, because when later it appears to be a POSITIVE DISPLACEMENT PUMP, a new pump would have to be declared and all functional information for that CENTRIFUGAL PUMP would be lost.

Now we take the 'ONTOLOGY AT INDIVIDUAL LEVEL' from Fig. 12 and turn that into a template, called ParticipationOfIndividualInActivity:

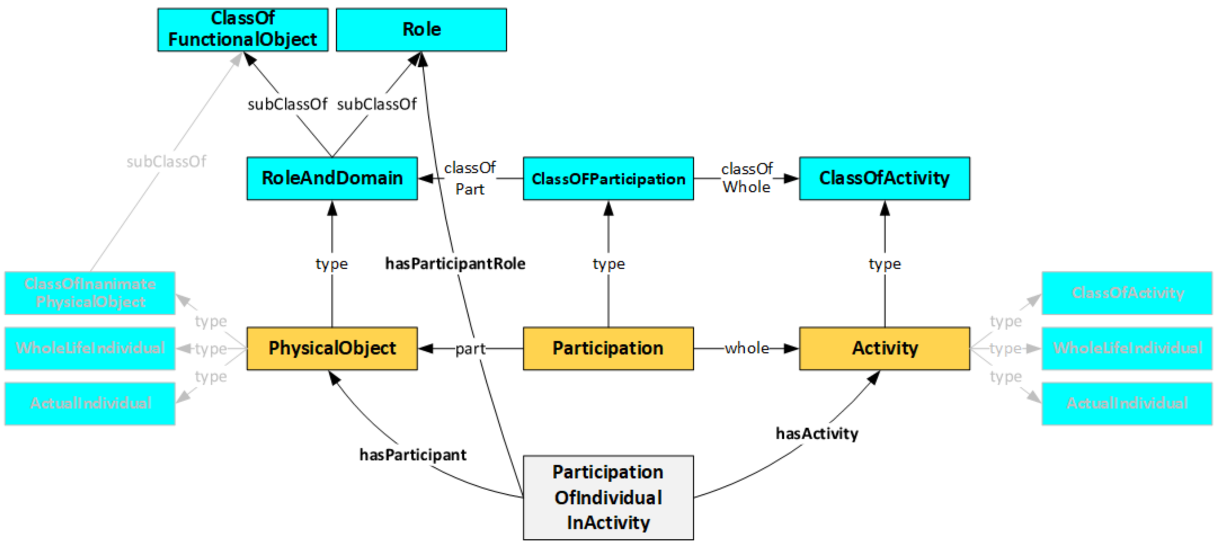


Figure 13 – Template for Fig. 12

We added the grey box, which is called 'template'. A template is an instance of the entity types: ClassOfInformationRepresentation and MultidimensionalObject (see here).

A template has two or more relations that point at the things of which the value is variable. Together these relations are called the 'signature' of the template, because when the template is instantiated and the relations are pointing to instances of the shown entity types that combination is unique, like a signature.

The signature of this template is part of the following code that defines the template in generic, Part 2 entity, terms:

```
:ParticipationOfIndividualInActivity
    rdf:type lci:Template ;
    rdfs:isDefinedBy <http://data.15926.org/owltpl/ParticipationOfIndividualInActivity ;
    rdfs:label "[EssentialType] individual [hasParticipant] participates in Activity [hasActivity] in a [hasParticipantRole] Role"@en ;
    tpl:hasActivity dm:Activity ;
    tpl:hasParticipant dm:PhysicalObject ;
    tpl:hasParticipantRole dm:Role ;
    meta:hasLifecycleActivity dm:ClassOfActivity ;
    meta:valEffectiveDate "xsd:dateTime"^^xsd:dateTime .
```

Assume that we use the same data as in Fig. 12, that is that PUMP SYSTEM P-101/AN4519159 participates in Activity ACT594583 in the Role of PERFORMER. Then an instance of above template is being instantiated with the relations (dubbed 'roles') being populated with these data:

```
:73622c8a-2fde-4f13-98d8-add6708587f8
    rdf:type tpl:ParticipationOfIndividualInActivity ;
    rdfs:label "[PUMP SYSTEM] individual [P-101] participates in Activity [Pumping waste water to sewer] in a [PERFORMER] Role"@en ;
    tpl:hasActivity :ACT594583 ;
    tpl:hasParticipant :501b174f-9ab4-4b1d-a64e-0c143c91b7e2 ;
    tpl:hasParticipantRole rdl:RDS222365 ;
    meta:hasLifecycleActivity rdl:RDS2229992 ;
    meta:valEffectiveDate "2020-04-07T13:36:00Z"^^xsd:dateTime .
```

This paper being a "gentle introduction" calls for an extensive explanation of above code, which is being given below:

The prefixes : rdf: rdfs: tpl: rdl: xsd: and meta: are short forms for an https: address where the term behind such a prefix is fully defined ;

The prefix : is the 'base' i.e. the address of the server where the template must be stored.

The terms behind the tpl: prefix are the template and its three roles in its 'signature' ;

The format is such that when it has been parsed it creates 'triples' such as:

```
:73622c8a-2fde-4f13-98d8-add6708587f8 rdf:type tpl:ParticipationOfIndividualInActivity .
:73622c8a-2fde-4f13-98d8-add6708587f8 rdfs:label "[PUMP SYSTEM] individual etc"@en .
```

etc. until the dot is found (here behind ^^xsd:dateTime .).

The label "[PUMP SYSTEM] individual [P-101] participates in Activity [Pumping waste water to sewer] in a [PERFORMER] Role"@en can be automatically generated using the labels and the 'essential types' of the things involved. The texts between [ ] are the variables.

In the template library at http://15926.org/15926_template_specs.php each template is fully defined.

For this template the graph of Fig. 14 is given.

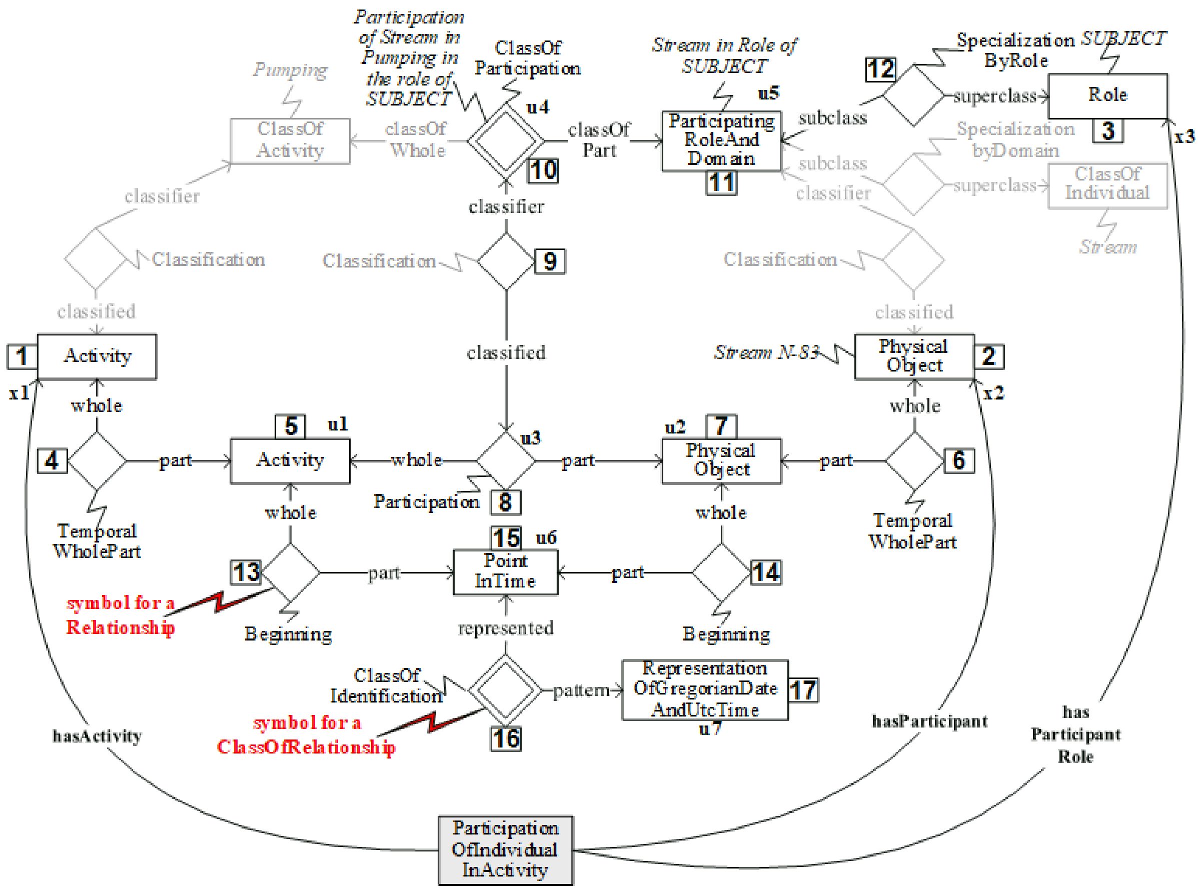


Figure 14 – Example template graph

When comparing this with Fig. 13 there are some extra things:

- The Activity and the PhysicalObject have temporal parts (see TEMPORAL PARTS)
- These temporal parts begin at a PointinTime
- Here Classification is used instead of rdf:type ; rdf:type is used only in declarations
- Here Specialization is used instead of rdfs:subClassOf ; the latter is used in declarations
- The texts in italics are example instantiations

It can be seen clearly that the ClassOfParticipation, being a part of the ontology at Class level, plays an indispensable role here as an ontology for this template. This template in its entirety is an ontology in its own right, as can be seen on the previous page. It prescribes of which entity type (or a subtype thereof) the 'role fillers' shall be an instance. And it precisely defines the semantics of the template instance by the construct in terms of Part 2 entity types.

The Classifications of the Activity and the PhysicalObject are defined in their declarations.

MULTIDIMENSIONAL OBJECTS

So far the MultidimensionalObject has only been mentioned once, for the fact that a Template is, next to being an instance of ClassOfInformationRepresentation, is an instance of MultidimensionalObject.

In fact a MultidimensionalObject is an N-ary relationship. In Part 2 it has some more relation types, but those are optional and in practice not very useful.

The important difference with an N-ary relation is that you can classify it and attribute meta data to it.

In order to give meaning to the relations of a MultidimensionalObject a classification with an instance of ClassOfMultidimensionalObject is required, see Fig. 15.

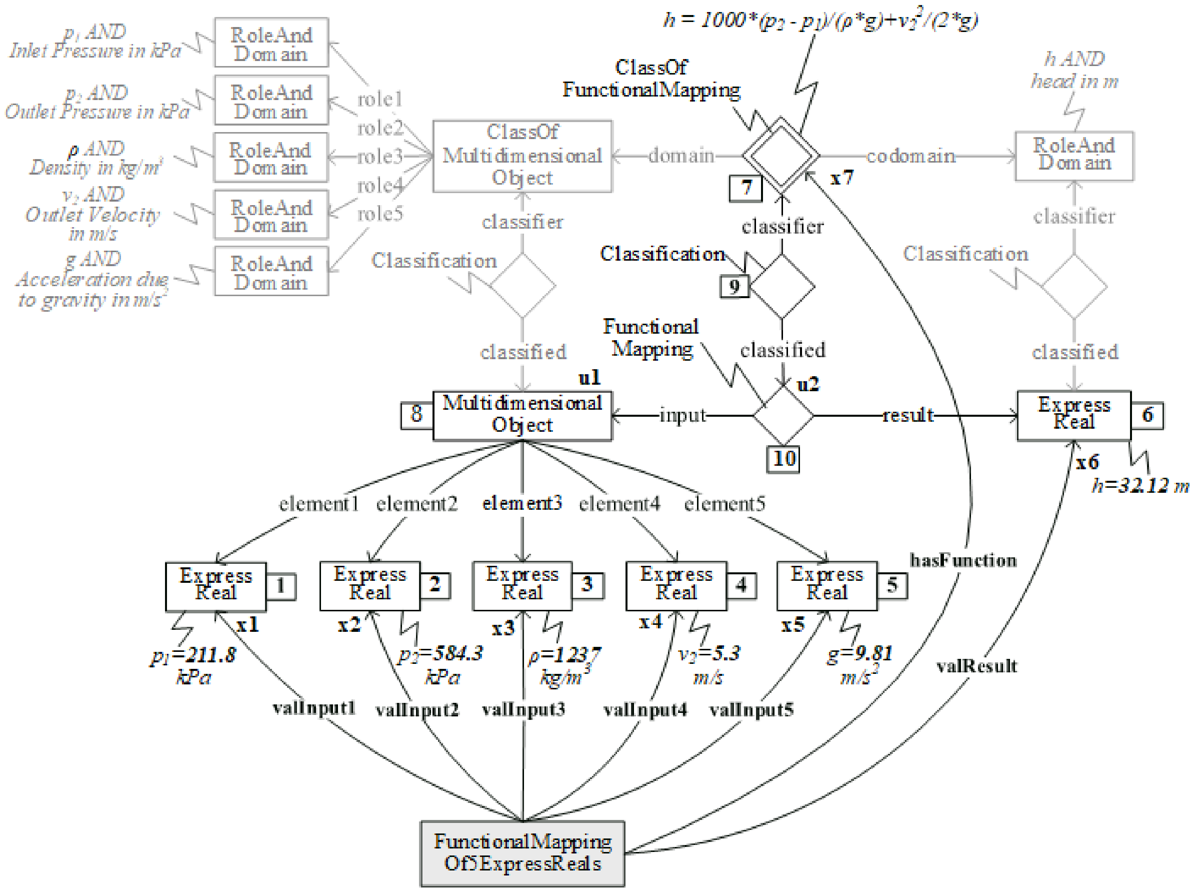


Figure 15 – An example of the use of MultidimensionalObject

This, of course, is a case that will not often be represented like this. But it shows the versatility of the combination of MultidimensionalObject and ClassOfMultidimensionalObject.

The formal definitions are:

A **MultidimensionalObject** is an AbstractObject that is an ordered list of Thing.
The significance of the MultidimensionalObject is determined by being a member of a ClassOfMultidimensionalObject that indicates the role played by each of its elements.

A **ClassOfMultidimensionalObject** is a Class whose members are instances of MultidimensionalObject. The role played by each position in the classified MultidimensionalObject is specified at the same position in the 'roles' attribute. (NOTE – it has some more optional attributes that are never being used).

Where the 'roles' and 'elements' are supposed to be LISTs, that appeared to be one of the impossibilities in the language of use (OWL, discussed later). As can be observed the 'roles' and the 'elements' get a serial number for implementation reasons (role1, role2, etc.).

The main use of MultidimensionalObject is in the Templates. A Template is an instance of ClassOfInformationRepresentation AND MultidimensionalObject (that is legal, since the three subtypes of AbstractObject are not related in a ONEOF mode) , see chapter TEMPLATES.

MultidimensionalObject has a few, occasionally used, subtypes:

- MultidimensionalNumber – e.g. [3.2, 5.4, 55.6]
- MultidimensionalNumberSpace – e.g. the three-dimensional Euclidean space R^3
- MultidimensionalProperty – e.g. a Q,H property pair on a pump curve
- MultidimensionalPropertySpace – e.g. entire pump curve flowrate vs differential head
- MultidimensionalScale – e.g. a temperature variation over time

TEMPORAL PARTS

One of the key concepts of ISO 15926 is that of "temporal parts", providing the means to place information about instances of PossibleIndividual in the time. In data modeler jargon this is called '4D' – three spatial dimensions plus one temporal dimension.

Here is a quote from Wikipedia:

Temporal parts are sometimes used to account for change. The problem of change is just that if an object x and an object y have different properties, then by Leibniz's Law, one ought to conclude that they are different. For example, if a person changes from having long hair to short hair, then the temporal-parts theorist can say that change is the difference between the temporal parts of a temporally extended object (the person). So, the person changes by having a temporal part with long hair, and a temporal part with short hair; the temporal parts are different, which is consistent with Leibniz's Law.

Temporal parts can be thought of as states of an individual. Take for instance the life of a pump, that roughly is subsequently in the following states:

- manufacturing:
  - assembly (starting the life of the WholeLifeIndividual);
  - testing;
- logistics;
- receipt at site, including handling and storage;
- installation and de-installation;
- commissioning;
- operations:
  - in service A;
  - in service B (etc.).

- maintenance:
    - in-line maintenance
    - shop maintenance
- performance analysis;
- demolition (ending the life of the WholeLifeIndividual).

These are the most important ones, and in the Plant Life-cycle Model (discussed later) this is clearly shown:

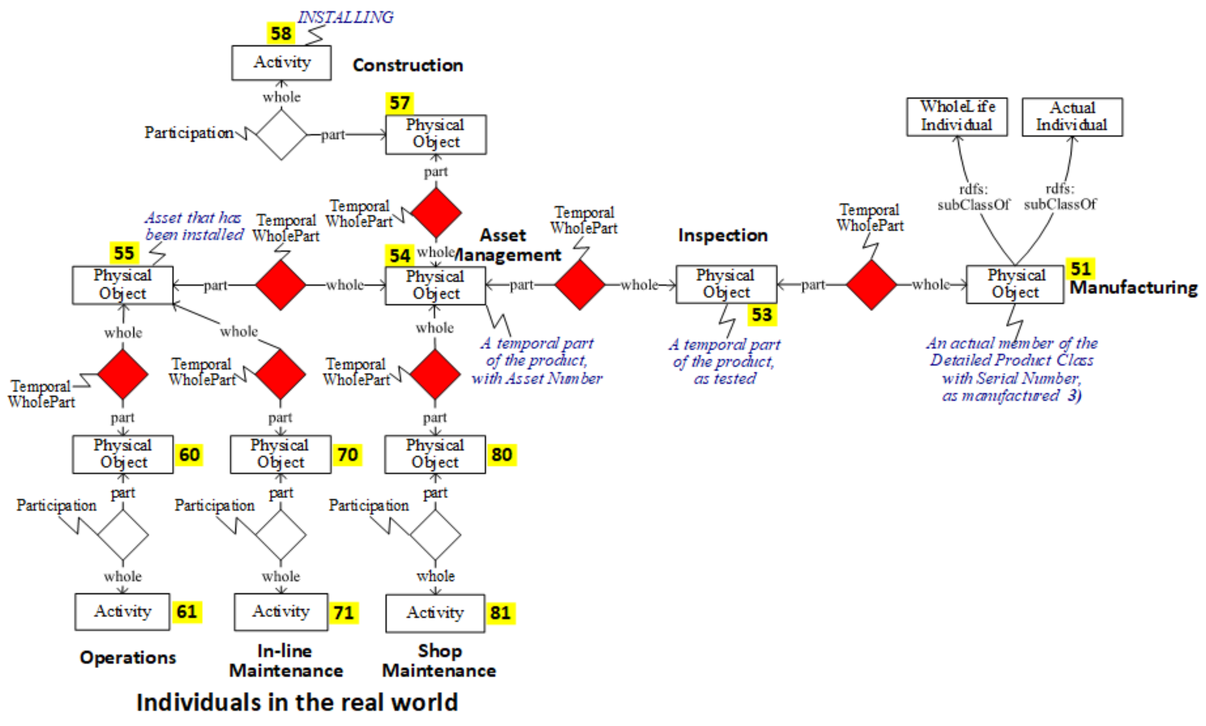


Figure 16 – Chain of temporal parts in the real world

In practice these temporal parts are being managed in different systems.

The temporal parts can have temporal parts and these can have temporal parts, etc., so a hierarchy. That hierarchy has one PossibleIndividual, typed as WholeLifeIndividual, at the top. This WholeLifeIndividual is dubbed the immutable 'anchor'. Each instance of a subtype of PossibleIndividual that is being declared is typed that way. As an example we take that pump P-101/AN4519159 that participated in that pumping activity:

```
:501b174f-9ab4-4b1d-a64e-0c143c91b7e2
    rdf:type lci:InanimatePhysicalObject,  dm:ActualIndividual,  dm:WholeLifeIndividual, rdl:RDS414081061 ;
    rdfs:label "P-101/AN4519159" ; # identifier of AssetNumber installed in function place P-101
    meta:hasLifecycleActivity rdl: RDS2229992 ; # PROCESS OPERATIONS
    meta:valEffectiveDate "2020-02-17T00:00:00Z"^^xsd:dateTime .
```

# and subsequent text are just annotations that do not belong to the code.

Lower in that hierarchy are new temporal parts for every state change in temperature, location, etc. For any implementation it would be totally impractical to declare each and every such temporal part PhysicalObjects. The solution for that can be seen in Fig. 14 where the temporal parts are in the template graph, for the semantic definition, but not in the template signature. For each recorded change in the semantics the 'effective date' of a representing (new) template is being recorded. This is how the 4th dimension is taken care of.

Engineering companies attribute most of their engineering data to instances of ClassOfInanimatePhysicalObject (see Fig. 7). Although Classes are deemed to be eternal, even for the ones that have not been invented yet, such as an efficient Fusion Reactor, we need to know when a given Class instance has become relevant/valid in a given context and when it is no longer relevant.

For that the data model has ClassOfTemporalWholePart, a subtype of ClassOfRelationship. Similar to temporal part PhysicalObjects we can create a hierarchy of Class instances with one 'anchor' at the top. Unfortunately the Part 2 data model does not have the concept of WholeLifeClass or the like (which would be a strange thing anyway because of the eternal nature of Classes).

Temporal part Individuals and Classes must be declared and then will be linked to their (class of) temporal whole with a template as shown in Fig. 17.

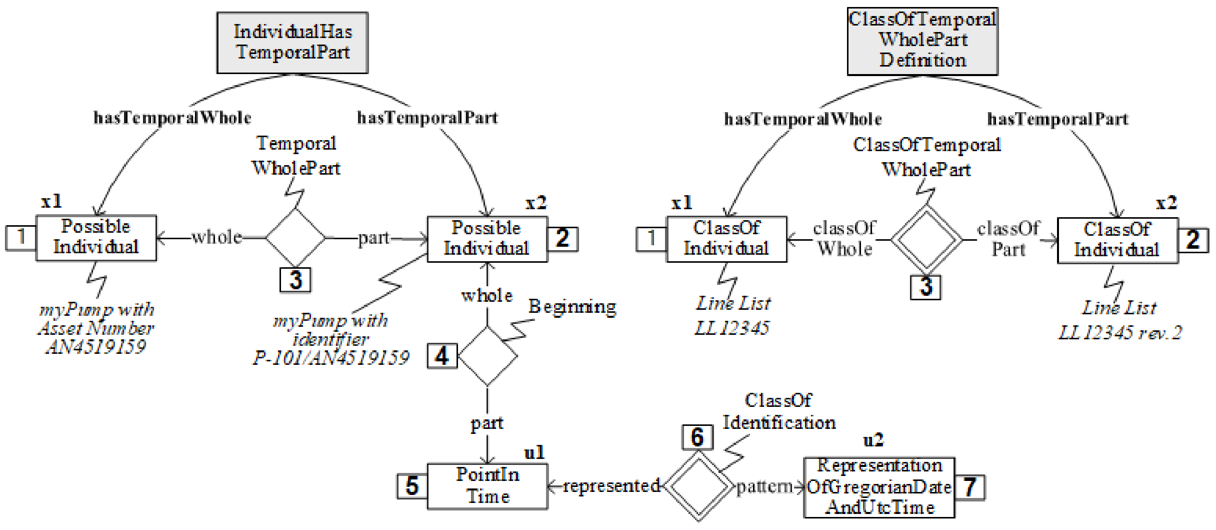


Figure 17 – Adding a (Class of) temporal part

Note that the Line List in Fig. 17 is an instance of ClassOfInformationObject, which is a subtype of ClassOfPhysicalObject, which is a subtype of ClassOfIndividual, so the template is applicable.

Also note that each revision of that Line List is a class of temporal part of the 'anchor' Class and not of the previous revision. In case you want to model that revision 2 came after rev. 1 then use the subtype of ClassOfRelationhip called ClassOfTemporalSequence (in a template).

The RepresentationOfGregorianDateAndUtcTime is not in the signature but in the effective date in order to be able to handle queries on templates for Individuals and Classes in the same manner.

MODELING

Unlike an entity-relationship model ISO 15926 doesn't have a hard-coded model, with the exception of the generic data model of Part 2 and the Part 7 templates.

Consider Parts 2 + 7 as grammar, defining the structure of sentences. That alone isn't enough, because the actual words in these standard sentences need to filled in. In each template specification is specified of which classes the 'role fillers' shall be a member. These are reference data from a Reference Data Library, or other plant items or parts thereof, persons, organizations, etc.

You can add declared things and templates at will, there is no hard-coded model that tells you not to do so.

But, in the light of interoperability, in a setting of integrated life-cycle information, it makes sense to do some old-fashioned modeling. This comes as ontologies at three levels:

1. top: a complete Plant Life-cycle Model ;
2. middle : object models - pre-defined relationships between reference classes ;
3. bottom: user-defined detailed object models .

Re 1. – Fig. 18 shows a high-level plant life-cycle model:

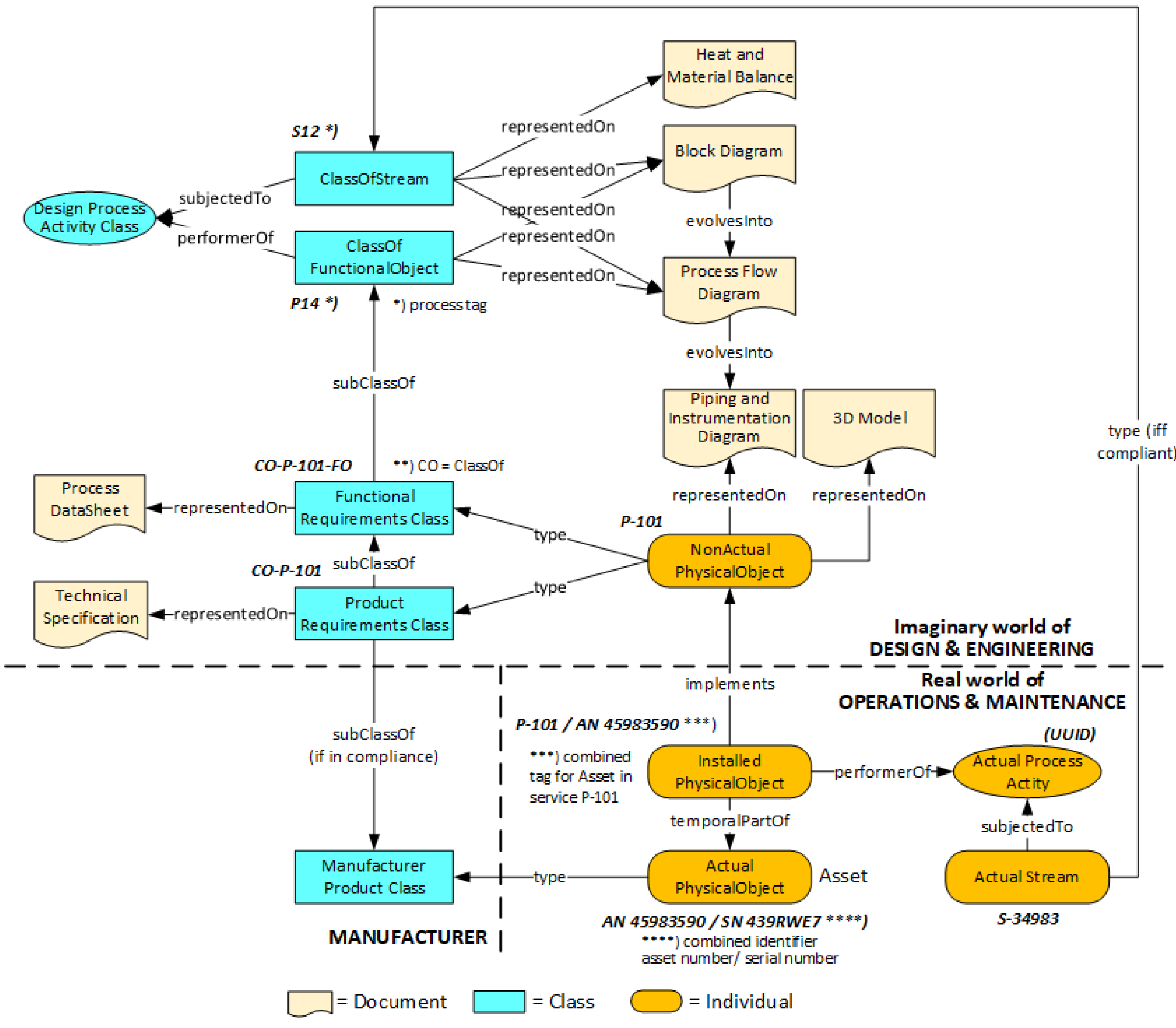


Figure 18 – High-level Plant Life-cycle Model

The full diagram is too large to be included here. Please visit
http://15926.org/topics/plant-lifecycle-model/index.htm#DataModel

Re 2. – Build Object Information Models (OIMs) for a start, with composition data and with recommended information sets.

Re 3. – Owner/operators may set up detailed models for proprietary design, and so can any EPC contractor or supplier. Manufacturers might be requested to come with OIMs for their equipment. Their OIM could also be detailed to the level of spare parts. When the industry want to adopt the Digital Twins paradigm, they better do that on the basis of ISO 15926.

MAPPING

All customary data stores and documents contain "implicit" data. This means that much contextual information is left out, because the software and the user "know" the context (or at least think they do). But other users and applications don't necessarily know that.

Where ISO 15926 strives for 'global interoperability' it is essential that during the mapping such implicit information is turned into explicit information as much as the business requires.

Implicit information is usually found in cases of:

- equipment and piping that contain a stream;
- compositional information, such as:
    - assembly of piping systems;
    - assembly of equipment systems (e.g. a pump system has a bare pump, drive(s), impeller(s), transmission, motor controls, etc.;
    - relationship between piping or equipment and the insulation or heat tracing;
    - relationship between piping or equipment and in-line/on-line instrumentation items;
- activity-based information (e.g. hydrostatic testing); etc

This often leads to conflated properties like 'normal operating outlet steam pressure auxiliary driver' (historic). There are an almost infinite number of such combined properties, making any system confronted with them totally unmanageable.

An important aspect is that conflated properties require human interpretation or some form of AI. But worse than that is that the property is not attributed to its owner, and when information about that owner is required that property is missed and can only be retrieved using elaborate rules.

Conflated properties can be avoided by precisely determining by which object the property is owned (or: of which object it describes a state). For example most functional data that are attributed to an equipment item actually are describing a state of the fluid handled by that equipment item.

Each mapping activity should therefore be started with writing a narrative that precisely describes the information that is being represented. In most cases that narrative reveals the true semantics of that information. This narrative also helps when asking for a second opinion and as a record for an audit trail, for example required when certifying an ISO 15926 Adapter. Incorrect mappings lead to miscommunication and in the industry that can turn out to be costly. That is why certification of ISO 15926 export and import adapters is essential.

The narrative for 'normal operating outlet steam pressure auxiliary driver' could be something like this in a kind of structured English, with the proper template as the outcome:

- A [PUMP SYSTEM] [class] [has] an [AUXILIARY DRIVER] [class] as [part] - ClassOfAssemblyDefinition
- that [AUXILIARY DRIVER] is a [STEAM TURBINE] – SpecializationOfClassOfIndividual
- that [AUXILIARY DRIVER] [has] an [OUTLET] for the [DRIVER STEAM] - ClassOfAssemblyDefinition
- the [DRIVER STEAM] is [contained by] that [OUTLET] - ClassOfContainmentDefinition
- the [DRIVER STEAM] [has] a [NORMAL OPERATING PRESSURE] - ClassOfIndividualHasIndirectPropertyWithValue

The [names] in capitals are labels of classes in the Reference Data Library, and the [words] in lower case are frequently used proxies for Part 2 instances of ClassOfRelationship.

Ideally this template selection will be generated with some form of AI.

As can be observed this calls for a number of template instances. Of course this makes it verbose, but this standard is for data sharing and integration and not for applications with split-second response times. And such combinations can, if so required, be implemented as one object.

OUTLINE OF MAPPING PROCESS

Export Mapping

1. Start with determining what the 'Object Of Interest' (OOI) is, and refer to the Plant Life-cycle Model to determine the applicable 'life-cycle activity' (e.g. PLANT DESIGN) ;
2. Determine whether the OOI is a PossibleIndividual or a ClassOfIndividual ;
    a. when an Individual whether a NonActualIndividual or ActualIndividual ;
    b. when an Individual of which EssentialType it is a member ;
    c. when a Class of which EssentialType it is a subclass .
3. Write a narrative, preferably in Structured English, in which a precise description of the semantics is given – when necessary go back to 1. ;
4. List in that narrative which Individual(s) and/or Class(es), other than the OOI, are required as 'role fillers' to fully represent the semantics of the narrative ;
5. Verify, by SPARQL query, whether or not the OOI already exists, if not declare it (NOTE - this may be subject to business rules about ownership of declarations) ;
6. Now the hardest part: select the applicable template and write the SPARQL code to fetch the 'role fillers' required by that template ;
7. From here on it depends which mapping software you use (e.g. the Paap-Rathling application) ;
8. The resulting ISO 15926-8 file in Turtle then is validated against the applicable SHACL 'Shape' of the selected template.
9. If the validation is successful: upload to the applicable triple store .

Import Mapping

1. Start with determining the data element for which a value has to be imported ;
2. Determine the OOI in the triple store, that owns that data element in the form of a template instance ;

3. Write a SPARQL query to fetch the value of the applicable 'role filler' in that template instance ;
4. Import that value with whatever input feature the application has .

MAPPING SOFTWARE

The mapping software developed by Onno Paap and Rowen Rathling works has a generic configuration/maintenance part and an application-specific mapping/transformation part.

This engine will be available in the public domain for further development.

Configurator

In the configurator a wizard leads the data engineer to a correct choice of the template that fits the semantics of the source information.

Mapper

Using an ETL script the data, required to fill the roles in the signature of that template, can be fetched from the source data store or app for which the mapper has been configured.

The user can select which of the shareable data shall be mapped and uploaded.

With SPARQL queries is checked in the federation of triple stores whether or not the OOI and related objects have already been declared, and if not they are now (provided that the applicable business rules permit such declarations to originate in that application).

The output is an ISO 15926-8 file that is validated against the applicable SHACL-SPARQL shape(s). The validated ISO 15936-8 file is uploaded to a triple store.

MAPPING SEQUENCE

It is recommended to first map the interrelationships between OOIs, then the components of each OOI (when and detailed as applicable), and then the properties of each of the OOIs and OOI components. In above software special constructs, called TIP-COMs, are available to quickly build an OOI component hierarchy.

PART 9 TRIPLE STORE

This part of ISO 15926 is still under development.

A provisional definition of its functionality is as follows:

Part 8 file import - rules for validation (well-formedness and other validation (duplicates, cardinalities, etc.) and the form of responses of success and failure) of imported Turtle files that conform to Part 8, logging of import transactions.

Data Caching – rules for managing information in Part 8 format that is “cached” (copied into a triple store that doesn’t “own” it), support for a ‘subscription’ service to enable a caching triple store to obtain relevant updates from the owning triple store, including actual data changes and the beginning of a new temporal part with new data values.

NOTE: Part 9 is not about how to do caching, but the rules in data updating to facilitate better caching.

Part 8 file export – rules for selection of data to be exported and generation of a corresponding transfer file that conforms to Part 8. Transfer of the file to the target recipient, or to a file system; logging of export transactions.

Data hand-over

- activation of the Part 8 file export function in the source triple store for the data to be handed over;
- invoking the Part 8 file import function in the target triple store;
- sending transfer file and confirming successful completion of transaction;
- logging and production of contractual transmittals of handover transactions.

SPARQL query processing – rules for the acceptance, processing and response to SPARQL queries against the information maintained by the triple store or a federation of triple stores.

Federation – support for creation of federations of triple stores; guidelines for distribution architecture, data sharing, and central security services (informative).

URI Dereferencing – rules for online availability of resources associated with URIs (dereference ability) and allocation of responsibility for de-referencing URIs in a federation of participating triple stores.

Security - Rules for the authentication of triple store clients; control of access to information maintained by the triple store, according to client privileges, including import, correction, query, and export; support for encryption of information during query, import and export transfers. Selective access rights per template instance.

PART 10 CONFORMANCE TESTING

This part of ISO 15926 is still under development.

A provisional definition of its functionality is as follows:

Well-formed RDF - Validate wellformedness of the Turtle under test.

Mapping quality - The test Authority shall set up a Part 9 façade with a browser that can display the results of any incoming Part 8 file. It can also store the information that the system under test must be able to fetch with a SPARQL query and map the results therefore to its internal format.

URI References of Role Fillers - Validate that all rdf:objects have been declared as per Part 8 C.8.1 'Declaration': "Where in OWL resources are normally not declared, this is mandatory in this part of ISO 15926. The only exception is that any temporal part, being the rdf:object of a template property, can not be explicitly declared.".

Entity type of Role Fillers - For each role of a template, the role fillers shall be of one or more Part 2 entity type or a subtype thereof, as defined in the applicable Template Specification.

Endpoints in federation - All façades on a project are federated. This federation can be queried to verify that all endpoints of all role fillers in the templates that are part of the exchange file are indeed in that federation.

Conformance with ontology - Any template for a PossibleIndividual can be validated against the applicable ontology, if available. This information comes in the form of templates for Classes.

Validation Standard - For the validation of the mapping results the W3C Recommendation "Shapes Constraint Language (SHACL)" shall be used.
This is possible because a Part 8 exchange file basically has two types of standard constructs: declared objects and template instances.

CERTIFICATION

The ISO 15926 Export Adapter of each application shall be certified, based on Part 10. In this way can be avoided that erroneous information is being uploaded and subsequently used by others for further 'downstream' life-cycle activities.

REFERENCE DATA

An important aspect of the interoperability that often is being touted as the most important one is that of 'reference data'. They are essential but so are other aspects, like grammar. With only a dictionary in a foreign language a fluent communication in that language is an illusion.

What exactly are 'reference data'? In fact every information that can be referred to falls under that category, and for the software it makes no difference. In ISO 15926-1 it is defined as follows: "*process plant life-cycle data that represents information about classes or individuals which are common to many process plants or of interest to many users.*".

This is a wide definition that includes data defined by:

- ISO 15926-4
- Standardization organizations, national and categorical
- Suppliers
- End users
- Projects

These are interconnected as shown in Fig. 19.

This is not yet an actual situation, although there are many RDL extensions, collected by the industry over the years as a result of a project-related need for such reference classes. The addresses of these extensions can be found at the bottom part of

http://15926.org/topics/reference-data/index.htm

Any class in an RDL extension shall be, directly or indirectly, be a subclass of an Part 4 Class.

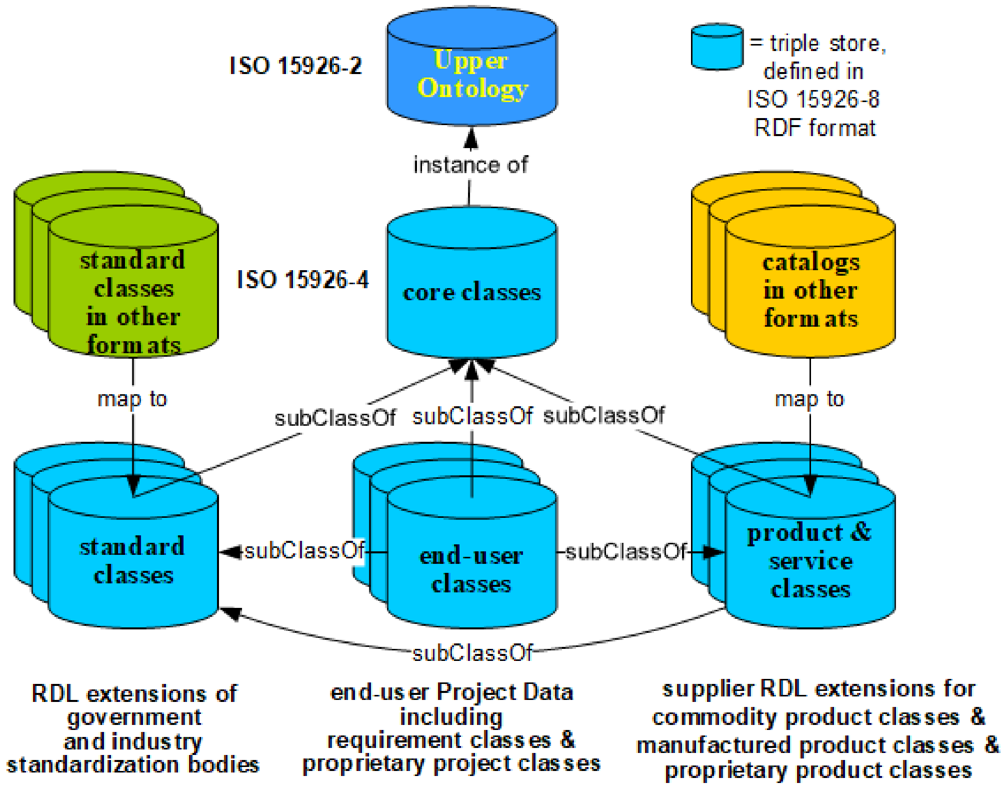

ISO 15926-2
Upper Ontology
= triple store, defined in ISO 15926-8 RDF format
instance of
standard classes in other formats
ISO 15926-4
core classes
catalogs in other formats
map to
subClassOf
subClassOf
subClassOf
map to
standard classes
subClassOf
end-user classes
subClassOf
product & service classes
subClassOf
RDL extensions of government and industry standardization bodies
end-user Project Data including requirement classes & proprietary project classes
supplier RDL extensions for commodity product classes & manufactured product classes & proprietary product classes

Figure 19 – Reference Data Library and its extensions

RELATED SUBJECTS

DOCUMENTS AND ISO 15926

The engineering world is a world of documents, many documents and these will, for the most, not be replaced with data. But they can be seamlessly integrated with the data.

This can be done by integrating the document meta data, stored in an EDMS (Electronic Document Management System), in the plant life-cycle information, as shown in Fig. 20.

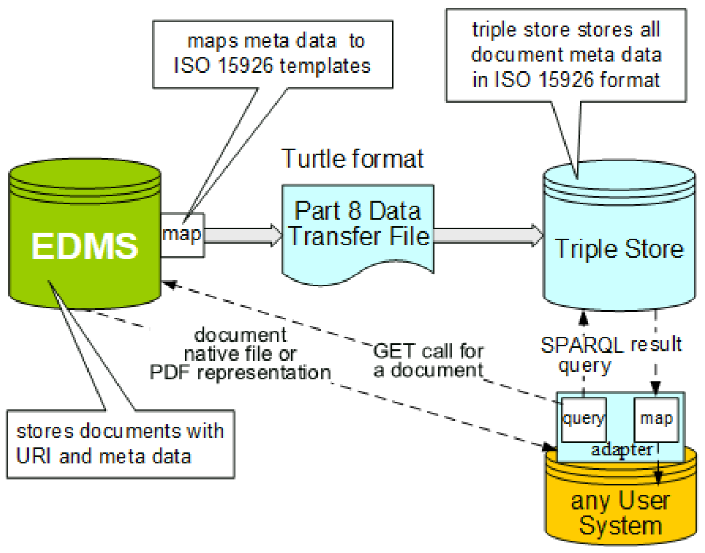

maps meta data to ISO 15926 templates
triple store stores all document meta data in ISO 15926 format
Turtle format
EDMS
map
Part 8 Data Transfer File
Triple Store
document native file or PDF representation
GET call for a document
SPARQL query
result
query
map
adapter
any User System
stores documents with URI and meta data

Figure 20 – Integration of documents in life-cycle information

WORKFLOW MANAGEMENT SYSTEM AND ISO 15926

In cases where a workflow management system is used the availability of up-to-date and integrated information is an asset that can be used to automatically import any available input data in the software that is used for each task. This can be done by launching a predefined SPARQL query at a point in time determined by that workflow system.

It is also conceivable that the system checks whether the query result is what is required, and if not send the 'upstream' task performer(s) a message, whereas the receiving task cannot be started.

POSSIBLE WORLDS

ISO 15926 does not support modal logic. Nevertheless there is, at times, a need for representing modalities. For this the concept of Possible Worlds, as described in http://15926.org/topics/possible-worlds/index.htm has being adopted.

Whenever a modality needs to be modeled it is placed in a possible world where it becomes a fact.

When an Individual is in the real world we type it with ActualIndividual and when in the Design World with NonActualIndividual.

PLANNING SYSTEM AND ISO 15926

Planning is about activities, and ISO 15926 has a complete set of activity-related templates. But these are planned activities that have a predicted 'effective date' in the future, so not really effective according the definition.

Referring to POSSIBLE WORLDS, for each planning a Planning World is created that is valid at the dateTime of the last (planned) completion date. All 'efffective' dates are then valid in that imaginary planning world.

Tip: Creating such a Planning World can be done by declaring a dedicated specialization of NonActualIndividual, such as PlannedIndividual20200416 for an Individual in the Planning World of April 16th 2020, the date of the new schedule.

Later a comparison can be made between the dates in the real world vs planning world.

CFIHOS AND ISO 15926

CFIHOS (Capital Facilities Information HandOver Specification) for process industries aims to standardize the specification of project information handover requirements for operators, contractors, and equipment manufacturers and suppliers. The target is the domain of the process industries, including the oil and gas, chemical and nuclear sectors.

The CFIHOS approach is based on a snapshot data transfer in batch mode. This also happens to be a part of the scope of ISO 15926. This CFIHOS approach may have been caused by the fact that the implementation of ISO 15926 took too long. This, in turn, was caused by the fact that the scope of ISO 15926 as an international interoperability standard for all technical and operational life-cycle information of a facility was a proverbial 'terra incognita'.

The advantage of the CFIHOS project is that, for the first time, the process industries, and in particular the plant owner/operators, tell us what is important to them. It would be far more productive when that aspect would prevail in an ISO 15926 implementation context rather that starting all over again.

An impediment for ISO 15926 is that it requires a paradigm shift to start using the modeling style and the technologies of the Semantic Web and Big Data. The latter also applies here when the life-cycle information of a plant has been (automatically) collected over the years.

OTHER PARTS OF ISO 15926

In Fig. 21 an overview of ISO 15926 is given. The parts missing are:

- Part 1 - Overview and fundamental principles
- Part 11 - Methodology for simplified industrial usage of reference data
- Part 13 - Integrated asset planning life-cycle

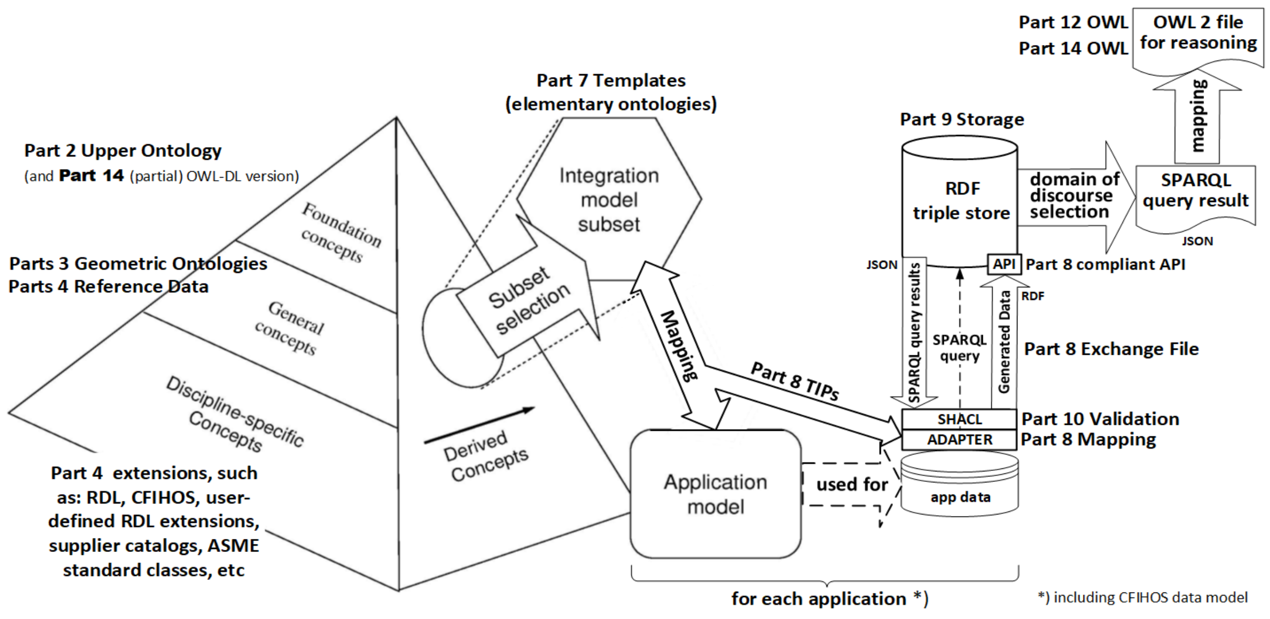


Figure 21 - Overview of ISO 15926