IOF Core Patterns

A compendium of Semantic data Patterns based on IOF Core

by

IOF Core working group

Authors: I. Surname
Editors: I. Surname
Proofread by: I. Surname1, I. Surname2

Release: 2025

Preface

A preface...

IOF Core working group
April 2025

Summary

summary of the compendium ...

Contents

Glossary

List all domain-specific, technical, other abbreviations and internally standardised terms in the glossary, e.g., construct - we mean terms in IOF.
DO NOT include ontology constructs including BFO or IOF in the glossary.

aliquam
tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

cras viverra
metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat.

donec nonummy
pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo.

integer sapien
est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus.

lorem ipsum
dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris.

1 Introduction

This chapter is for everything we need to inform in general before presenting the patterns.

Purpose and scope

Basic Formal Ontology (BFO)

IOF Core

How to use the patterns

Identify the correct patterns to use

Use in Data mapping

How to use the INSERT DATA mapping (adopt and adjust)

Use in Data validation

SPARQL and SHACL

Extension and customization

understand the general pattern Explore how different patterns can be combined.

Versioning and changes

Only add aspects that are main chapters. Each scenario will be a section with TOC entry only with the scenario name. Compile scenarios under the aspect chapters below.

2 Time

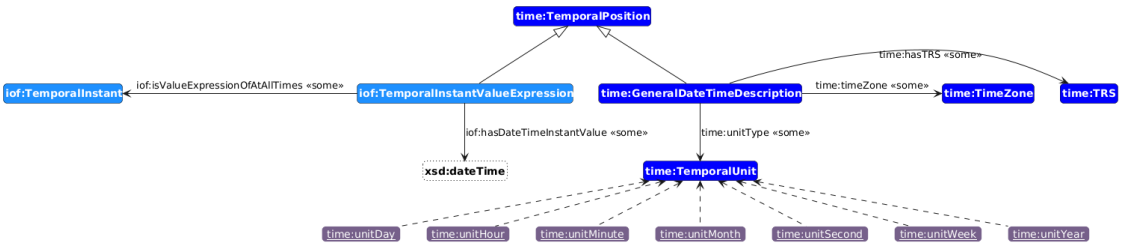
2.1. Clock time and calendar date

Created by: Arkopaul Sarkar
Modified by: Arkopaul Sarkar

Scenario Objective

This scenario illustrates how to associate clock time and calendar dates with instances of time. While the ability to store clock time and calendar dates allows users to perform various quantitative analyses on the processes and their durations, the wide variety of clock and calendar systems presents challenges in accurately representing dates and times in the correct format. We include use cases depicting both basic and advanced methods for representing date and time values of temporal instances.

General Pattern Description



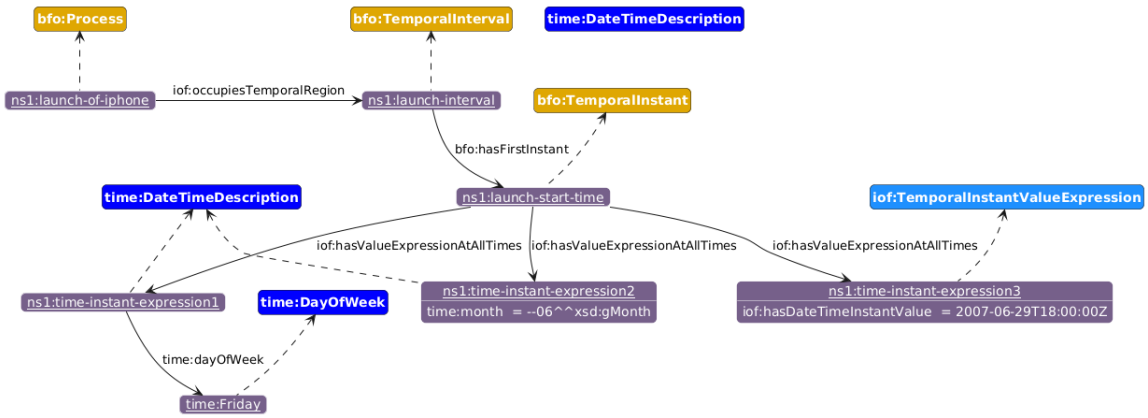
the calendar and clock time for a specific temporal instant can be expressed by associating different instances of the information class `TemporalInstantValueExpression` or OWL-Time class `time:GeneralDateTimeDescription` as both are subclass of `time:TemporalPosition` (`core:TemporalInstantValueExpression` is mapped as subclass of `time:TemporalPosition`.

Use Case: Launch of first iPhone

The original iPhone was first launched on June 29, 2007 during the Macworld Conference & Expo in San Francisco, California.

The date and time of the launch are captured in 1) XSD dateTime format, 2) in a specific time zone, and 3) using a custom date and time format. The process `launch-of-iphone` is not detailed further except the temporal interval it occupies. This instance of temporal interval is then connected to its first temporal instant, for which the calendar date and clock time are assigned.

Use-Case Pattern Description



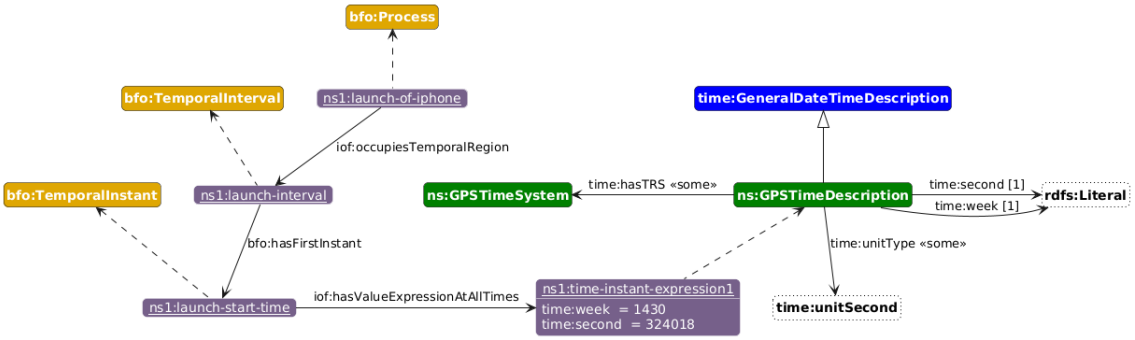
The launch date and time are expressed in three different ways.

`hasDateTimeInstantValue` can associate the date and time value in `xsd:dateTime` format. If the date and time values can be expressed in XSD format, this pattern does not require any reference to OWL Time ontology. Also, `xsd:dateTime` format already has provision for mentioning time zone (e.g., 2007-06-29T18:00:00-05:00 for a UTC-5 timezone). However, as XSD:dateTime datatype is the range of `hasDateTimeInstantValue`, no other XSD type or a different format can be expressed using this pattern.

Other components of a clock time

In the above pattern, the launch date and time of the iPhone in New York are expressed as 'day of the week' using `time:dayOfWeek` (OWL Time ontology provides the days of a week as instances of type `time:DayOfWeek`) and 'month', using `time:month` data property `xsd:gMonth` which links to an indexical value based on the order of months in the calendar of type `xsd:gMonth` [1]. Various combinations of data properties of `GeneralisedDateTimeDescription`, e.g., year, month, day, hour, minute, and second, can be used to express a clock time or calendar date, e.g., only date value in year, month and day. The corresponding time zone can be mentioned by linking an instance of `TimeZone` [2], using `time:timeZone` property.

Using custom clock time format



In the above pattern the launch date is expressed in 'GPS time'. As GPS time is the number of seconds since an epoch in 1980, encoded as the number of weeks and seconds into the week, the new class `GPSTimeDescription`, extended from `time:DateTimeDescription`, should contain values for data properties `time:second` and `time:week`. The example does not provide details of `GPSTimeSystem`, which is a `time:TemporalReferenceSystem`, and may refer to a suitable description of 'GPS time', e.g., A taxonomy of temporal reference systems is provided in ISO 19108:2002.

Data Mapping Description

```
INSERT DATA {
    ns1:launch-of-iphone a bfo:Process;
                         bfo:occupiesTemporalRegion ns1:launch-interval.
    ns1:launch-interval a bfo:TemporalInterval;
                         iof:hasFirstInstant ns1:launch-start-time.
    ns1:launch-start-time a bfo:TemporalInstant;
                         iof:hasValueExpressionAtAllTimes ns1:instant-expression-xsd;
                         iof:hasValueExpressionAtAllTimes ns1:instant-expression-month;
                         iof:hasValueExpressionAtAllTimes ns1:instant-expression-dow;
                         iof:hasValueExpressionAtAllTimes ns1:instant-expression-gps.
    ns1:instant-expression-xsd a iof:TemporalInstantValueExpression;
                         iof:hasDateTimeInstantValue "2007-06-29T18:00:00Z"^^xsd:dateTime.
    ns1:instant-expression-month a time:DateTimeDescription;
```

[1] https://www.w3.org/TR/xmlschema11-2/#gMonth

[2] For detailed guidance about working with time zones, see http://www.w3.org/TR/timezone/ .

```
                        time:month "--06"^^xsd:gMonth.
    ns1:instant-expression-dow a time:DateTimeDescription;
                        time:dayOfWeek time:Friday.
    ns1:instant-expression-gps a ns:GPSTimeDescription;
                        time:week "1430"^^xsd:decimal;
                        time:second "324018"^^xsd:decimal.
}
```

`ns:GPSTimeDescription` class has a temporal reference system as `ns:GPSTimeSystem`, which refer to the specification of GPS time format. Following the standard, a `GPSTimeDescription` has a

e

```
:GPSTimeDescription rdf:type owl:Class ;
    rdfs:subClassOf
    <http://www.w3.org/2006/time#GeneralDateTimeDescription> ,
    [ rdf:type owl:Restriction ;
        owl:onProperty <http://www.w3.org/2006/time#hasTRS> ;
        owl:hasValue :GPSTimeSystem
    ] ,
    [ rdf:type owl:Restriction ;
        owl:onProperty <http://www.w3.org/2006/time#unitType> ;
        owl:hasValue <http://www.w3.org/2006/time#unitSecond>
    ] .

:GPSTimeSystem rdf:type owl:NamedIndividual ,
    <http://www.w3.org/2006/time#TRS> ;
    AnnotationVocabulary:adaptedFrom "https://en.wikipedia.org/?title=GPS_time" .
```

Data Validation

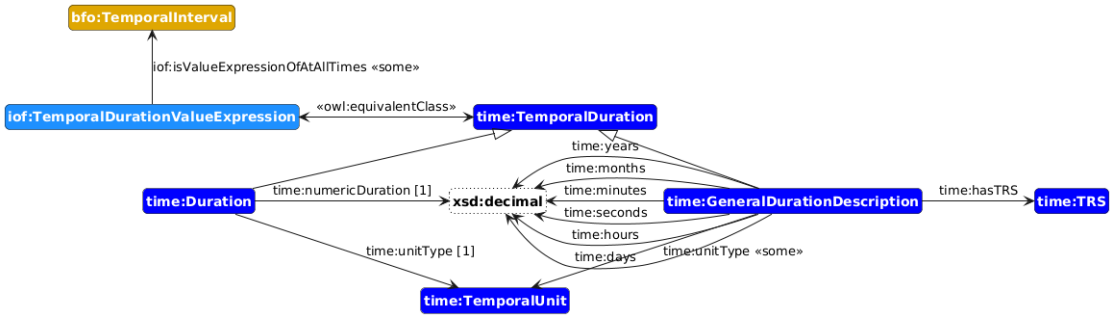
2.2. Duration in time units

Created by: Arkopaul Sarkar
Modified by: Arkopaul Sarkar

Scenario Objective

This scenario demonstrates how to represent time durations of time intervals. Duration values can be measured in different time units, such as years, hours, and ticks. In the following, we present the patterns for asserting duration values in standard and custom units.

General Pattern Description



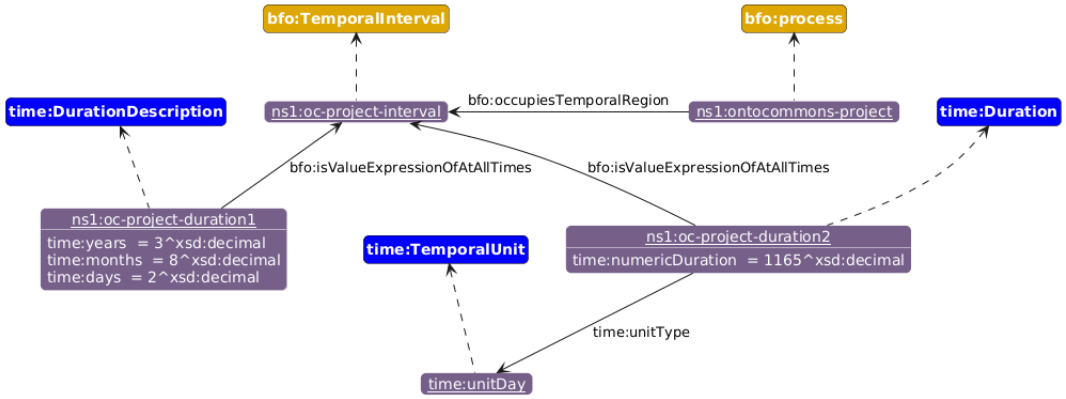
The duration of a temporal interval can be expressed by associating different instances `iof:TemporalDurationValueExp` or some subtype of OWL-Time class `time:TemporalDuration` to the instance of `bfo:TemporalInterval` using `iof:isValueExpressionOfAtAllTimes` as they are equivalent classes. An instance of `time:DurationDescriptio` which is a subclass of `time:TemporalDuration`, provides several data properties to assert the duration value in various units used in Gregorian Calendar, e.g., `time:years`, `time:months`, and `time:minutes`. Alternatively, a numeric value and corresponding duration type can be asserted using the instance of `time:Duration`. A custom duration description class can be defined by extending `time:GeneralDurationDescription` class with an appropriate temporal reference system.

Use Case: Duration of OntoCommons project

OntoCommons project started on Wednesday, 1 September 2021 and ended on Saturday, 9 November 2024. It ran for 1165 days from the start to the end date, not including the end date.

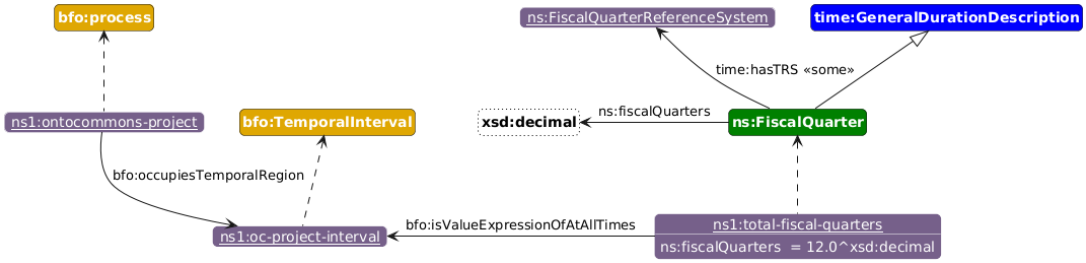
The duration of the temporal interval (`ns1:oc-project-interval`), that the project (`ns1:ontocommons-project`) ran for, can be expressed in typical duration units from Gregorian calendar, e.g., year, month, day, hour, minute, and second, using OWL Time classes. OWL Time also provides mechanisms for creating custom duration descriptions.

Use-Case Pattern Description



The duration value can be expressed by linking an instance of `time:DurationDescription` or `time:Duration`. `DurationDescription` provides dedicated data properties in various units, allowing users to express the duration value in mixed units, e.g., the project duration is 3 years 8 months and 2 days. `time:Duration` allows expression of the duration value in a single unit (various common unit instances are already provided by OWL Time), e.g., 1165 days.

Using custom clock time format



In the above pattern the launch date is expressed in 'Fiscal Quarters'. The new duration description class `DurationDescriptionInFiscalQuater`, extended from `time:GeneralDurationDescription`, with restriction that the number of fiscal quarter can be asserted in decimal using only `ns:fiscalQuarters` data property. The example does not provide details of `FiscalQuarterReferenceSystem`, which is a `time:TemporalReferenceSystem`, and may refer to a suitable description of 'fiscal quarter'.

Data Mapping Description

```
INSERT DATA {
    ns1:ontocommons-project a bfo:Process ;
                            bfo:occupiesTemporalRegion ns1:oc-project-interval .

    ns1:oc-project-interval a bfo:TemporalInterval .

    ns1:oc-project-duration1 a time:DurationDescription ;
                             iof:isValueExpressionOfAtAllTimes ns1:oc-project-interval ;
                             time:years "3"^^xsd:decimal ;
                             time:months "8"^^xsd:decimal ;
                             time:days "2"^^xsd:decimal .

    ns1:oc-project-duration2 a time:Duration ;
                             iof:isValueExpressionOfAtAllTimes ns1:oc-project-interval ;
                             time:numericDuration "1165"^^xsd:decimal ;
                             time:unitType time:unitDay .
```

```
    ns1:oc-project-duration3 a ns:FiscalQuarter ;
                             iof:isValueExpressionOfAtAllTimes ns1:oc-project-interval ;
                             ns:fiscalQuarters "12"^^xsd:decimal .
}
```

`ns:DurationDescriptionInFiscalQuarter` class has a temporal reference system as `ns:FiscalQuarterReferenceSystem`, which refers to the specification of fiscal quarter duration format. The class restrict the usage of other data properties from `time:GeneralDurationDescription` and allows exactly one decimal value to be linked with data property `ns:fiscalQuarters`

```
:DurationDescriptionInFiscalQuarter rdf:type owl:Class ;
    rdfs:subClassOf
        <http://www.w3.org/2006/time#DurationDescription> ,

        [ rdf:type owl:Restriction ;
          owl:onProperty <http://www.w3.org/2006/time#hasTRS> ;
          owl:allValuesFrom :FiscalQuarterReferenceSystem
        ] ,

        [ rdf:type owl:Restriction ;
          owl:onProperty :fiscalQuarters ;
          owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
          owl:onDataRange xsd:decimal
        ] ,

        [ rdf:type owl:Restriction ;
          owl:onProperty <http://www.w3.org/2006/time#days> ;
          owl:cardinality "0"^^xsd:nonNegativeInteger
        ] ,

        [ rdf:type owl:Restriction ;
          owl:onProperty <http://www.w3.org/2006/time#hours> ;
          owl:cardinality "0"^^xsd:nonNegativeInteger
        ] ,

        [ rdf:type owl:Restriction ;
          owl:onProperty <http://www.w3.org/2006/time#minutes> ;
          owl:cardinality "0"^^xsd:nonNegativeInteger
        ] ,

        [ rdf:type owl:Restriction ;
          owl:onProperty <http://www.w3.org/2006/time#months> ;
          owl:cardinality "0"^^xsd:nonNegativeInteger
        ] ,

        [ rdf:type owl:Restriction ;
          owl:onProperty <http://www.w3.org/2006/time#seconds> ;
          owl:cardinality "0"^^xsd:nonNegativeInteger
        ] ,

        [ rdf:type owl:Restriction ;
          owl:onProperty <http://www.w3.org/2006/time#weeks> ;
          owl:cardinality "0"^^xsd:nonNegativeInteger
        ] ,

        [ rdf:type owl:Restriction ;
          owl:onProperty <http://www.w3.org/2006/time#years> ;
```

```
          owl:cardinality "0"^^xsd:nonNegativeInteger
        ] .

:FiscalQuarterReferenceSystem rdf:type owl:Class ;
    rdfs:subClassOf <http://www.w3.org/2006/time#TRS> .
```

Data Validation

3 Objects, artifacts and their materials

3.1. object vs artifact

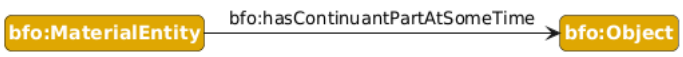
3.2. What an Object is Made of

Created by: Perawit Charoenwut
Modified by:

Scenario Objective

This scenario illustrates how to represent what materials an object is made of using the IOF/BFO ontology framework. It focuses on representing what components make up an object.

General Pattern Description

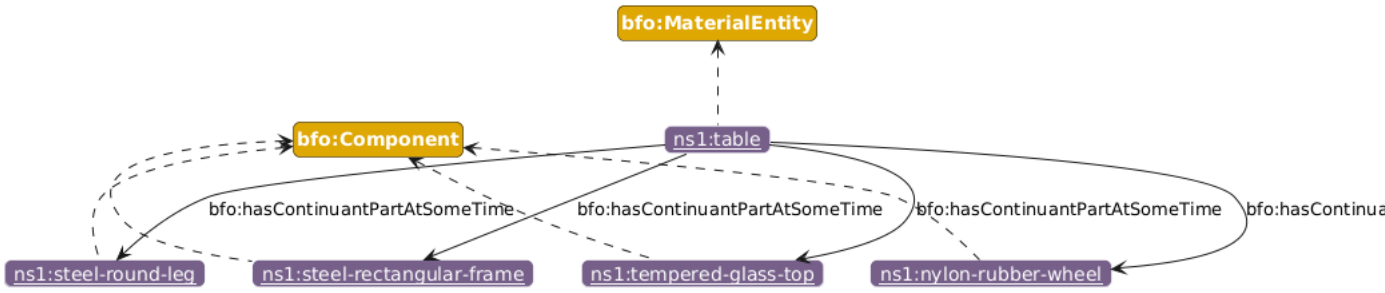


A material entity (bfo:MaterialEntity) may be composed of multiple components (iof:Component). The relationships between a material entity and its components are represented using bfo:hasContinuantPartAtSomeTime.

Use Case: Glass Top Steel Table

A modern table with a tempered glass top and stainless steel frame demonstrates how different materials compose different parts of an object.

Use Case Pattern Description



The table (ns1:table) is represented as a bfo:MaterialEntity that has four main components:

- A tempered glass top (ns1:glass-top)
- A stainless steel frame (ns1:steel-frame)component for the frame
- Steel legs (ns1:steel-leg)
- Nylon wheels (ns1:nylon-wheel)

Use-Case Example Data

Component ID	Product ID	Component Type	Material	Quality Type	Quality Value	Unit
TOP001	TABLE001	Table Top	Tempered Glass	Thickness	10	mm
TOP001	TABLE001	Table Top	Tempered Glass	Area	0.8	m²
FRAME001	TABLE001	Table Frame	Steel	Length	100	cm
FRAME002	TABLE001	Table Frame	Steel	Length	80	cm
LEG001	TABLE001	Table Leg	Steel	Height	75	cm
LEG001	TABLE001	Table Leg	Steel	diameter	5	cm
WHEEL001	TABLE001	Wheel	Nylon	Diameter	50	mm

Data Mapping

Data Validation

3.3. material vs object

3.4. material vs material constitution

3.5. object vs fiat objects

3.6. part of an object

4 Assembly component

4.1. components of an assembly

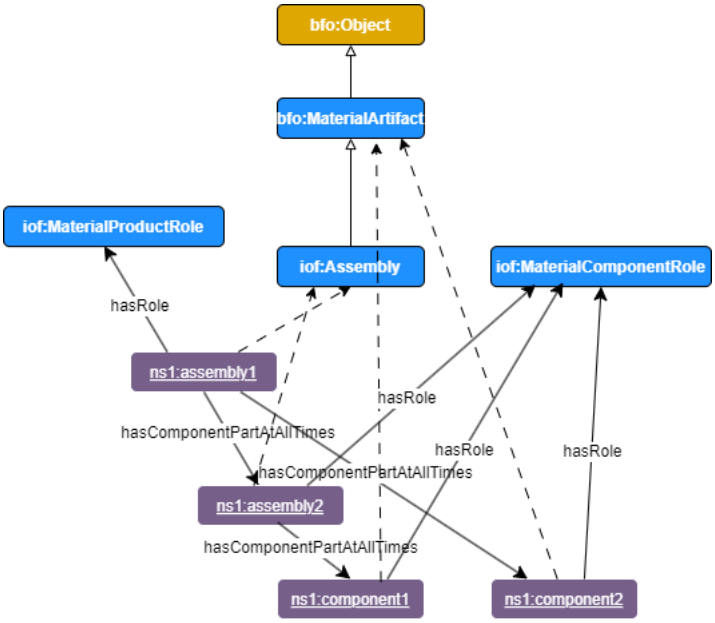


Figure 4.1: The general pattern for the use of component-related object properties

4.2. Assembly component

Created by: Dušan Šormaz
Modified by:

address all times/ sometimes and also transitivity of these two pairs of relationships.

Scenario Objective

From Arko's email, 1/7/25 I want to mention that however we are showing the basic use of "component part of" relation for expressing parthood, we should also show the use of "at some time" vs. "at all times" under this scenario. Also, the nuanced semantics of "component of at all times" vs. "has component at all times" (explain why they are not inverses to each other) is to be examined. DNS We also need to show multiple levels of assemblies, at least three. Should we consider assembly process(es)?

General Pattern Description

This scenario shows uses of `iof:componentPartOf`, `bfo:continuantPartofAtSomeTime`, and `bfo:continuantPartofAt` object properties. The general pattern is shown in fig. 15. The figure shows that an artifact in the product role may be assembly when it consists of components that are assembled to form a functional unit. In general, an assembled product consists of several levels of components, some of which may be assemblies, and some just simple mechanical parts.Both of those categories play the role of a MaterialComponentRole. The relations between an assembly and its component may be of two kinds: a) permanent, such as for welded (or similar processes) assemblies, in which case we need to use the `iof:hasComponentAtAllTimes` relation, or b) temporal, with an option of later disassembly (for example, for repair or replacement), usually assembled with some kind of fasteners (bolts, nuts, etc.), in which case we need to use the `iof:hasComponentAtSomeTime` relation.

example car is salvaged and engine is kept for us in the other car. another example is in welding it is permanent, you have to throw away both welded parts. use example of the car to illustrate reasoning and its difference in AllTimes and SomeTimes

Examples

A car engine is often showcased in parts: pistons, crankshaft, and cylinders.

An insulated wall comprises drywall, insulation foam, and outer sheathing.

Sewing patches of silk, cotton and leather make an item of clothing.

4.3. Embedded components

Reinforcement Bars in Concrete Structures

coating on a metal surface - this is not assembly

printed circuit on board

Embedded sensors

4.4. being components of different assemblies at different times

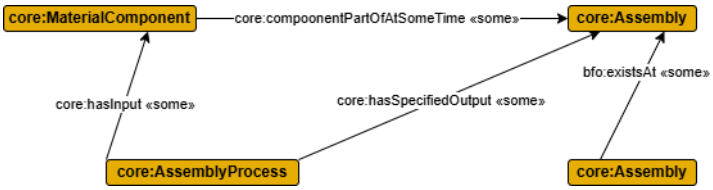
Created by: Ali Hasanzadeh
Modified by:

Scenario Objective

This scenario illustrates how to represent the change in an object's association with different assemblies over time using the IOF/BFO ontology framework. It focuses on:

- Highlighting the use of temporal regions and temporal intervals to associate an object's presence in specific assemblies over time.
- Representing the object's changing relationships to multiple assemblies through different part-hood properties.
- Using physical connections of components as an outcome some process assembly to denote a material artifacts function(s).

General Pattern Description



A material component may be a component of different assemblies at different times. In this scenario, the `core:MaterialComponent` would be related to multiple `core:Assembly` via the `core:componentPartOfAtSomeTime` property. As this connection cannot be true between a `core:MaterialComponent` and different `core:Assembly` artifacts at once, we would denote this reality by pairing the `bfo:TemporalRegion` of each `core:AssemblyProcess` with the `core:MaterialComponent`.

Use-Case Pattern Description

A PLA joint can be part of a modular shelf used for storing books and later be repurposed in another modular shelf used for storing tools. The particular example is chosen to emphasize the difference between assemblies and machines as well. In this scenario, the joint is `core:componentPartOfAtSomeTime` multiple shelves (`core:Assembly`). In other words, this `core:Component` is `core:componentPartOfAtSomeTime` one `core:Assembly` shelf at some `bfo:TemporalRegion` as the result of some `core:AssemblyProcess` and `core:componentPartOfAtSomeTime` of another `core:Assembly` shelf at some other `bfo:TemporalRegion` as the result of a different `core:AssemblyProcess`.

4.5. being components of different assemblies at the same times

door is a component of both room.

4.6. Temporary components

Each stage of the rocket is only part of the whole vehicle for a specific phase of the journey

4.7. Components as an identity of the assembly

Changing the battery of a wall clock or the tire of a car does not change the wall clock or the car.
But changing the CPU changes the laptop.

5 Person and agent

Created by: Perawit Charoenwut
Modified by:

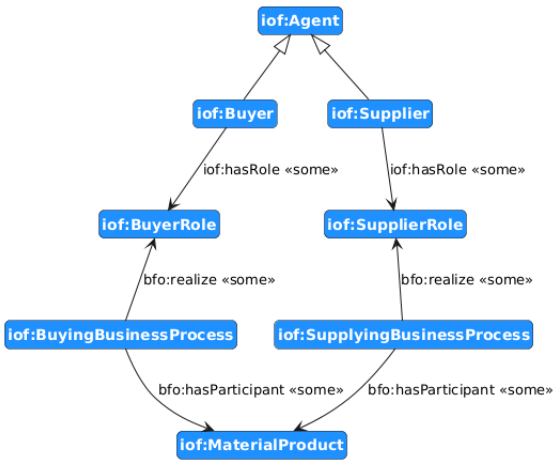
Scenario Objective

This scenario aims to demonstrate how to model relationships between different types of agents in commercial transactions using the IOF Core patterns. The pattern models how different entities (such as persons and organizations) can take on specific roles (such as buyer and supplier) within complementary business processes that involve material products. This pattern is fundamental for representing any transaction where one party acquires a product from another, making it clear which agents are involved and what their specific roles are within the transaction context.

General Pattern Description

The pattern models the relationship between buyers, suppliers, and material products through their roles and participation in business processes. It demonstrates how:

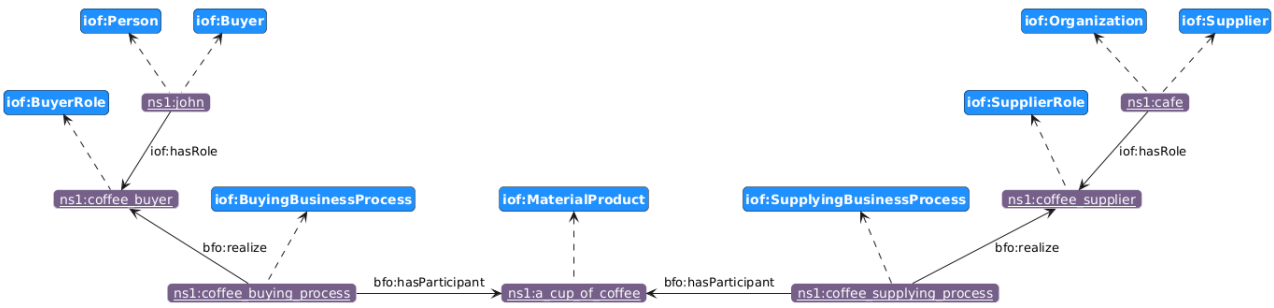
An agent classified as a Buyer has a BuyerRole that participates in a BuyingBusinessProcess An agent classified as a Supplier has a SupplierRole that participates in a SupplyingBusinessProcess Both business processes have the same MaterialProduct as a participant



Use Case: Coffee Shop Transaction

Use-Case Pattern Description

This use case demonstrates how a buyer (John), purchases a cup of coffee from a cafe.



Use-Case Example Data

Data Mapping

Data Validation

Agent as an organisational agent

Agent as a group of agents

Agent as an EngineeredSystem

6 System and organization

6.1. Group of agent vs organized group of agents

6.2. type of group of agent

Mix of human and machine

6.3. Different types of organisations

Created by: Perawit Charoenwut
Modified by:

Scenario Objective

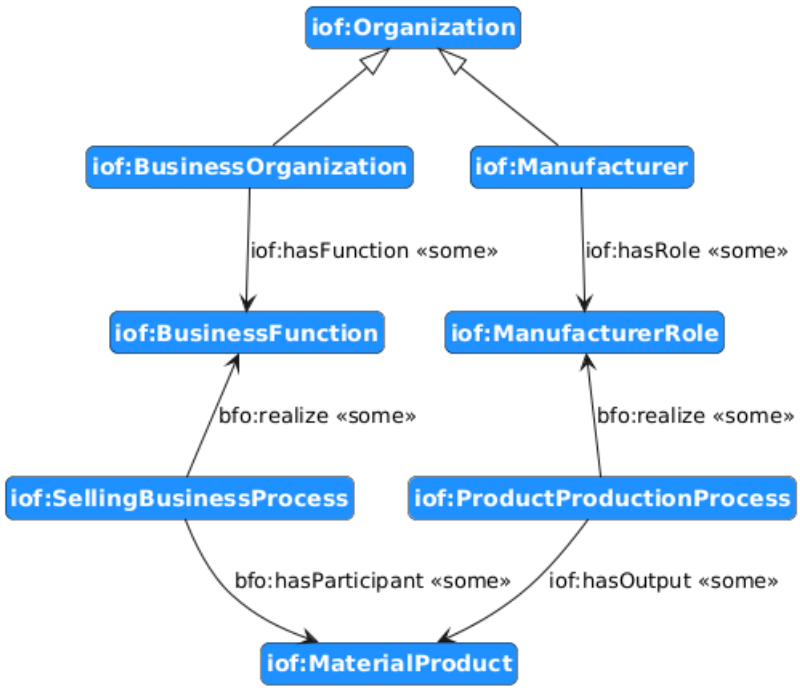
This scenario illustrates how to represent a complete company structure with manufacturing and business divisions using the IOF core ontology. It focuses on:

- Showing how business and manufacturing units coexist within one company
- Distinguishing between business and manufacturing activities
- Capturing different organizational roles and functions

General Pattern Description

A company can contain both business organizations and manufacturers as part of its structure. An Organization can simultaneously be:

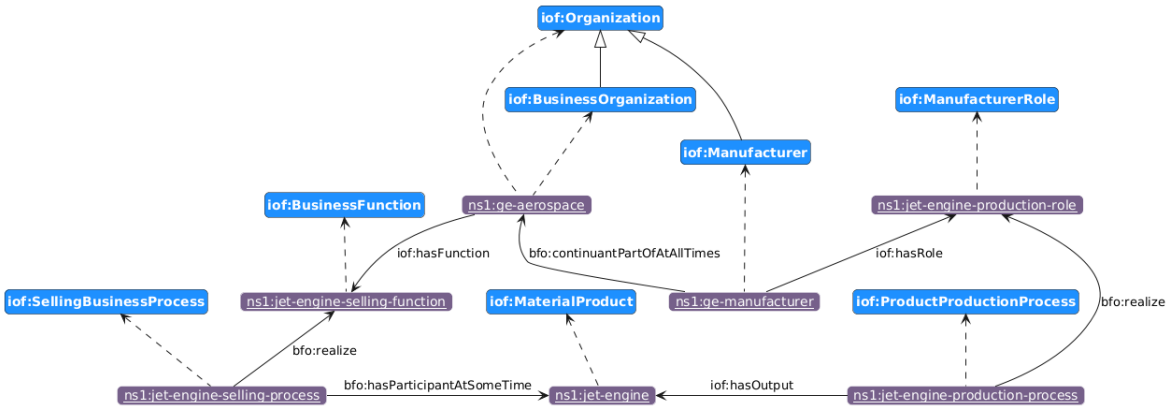
1. A BusinessOrganization that has a BusinessFunction, which is realized through a SellingBusinessProcess that sells the MaterialProduct
2. A Manufacturer that has a ManufacturerRole, which is realized through a ProductProductionProcess and has the MaterialProduct as its output.



Use Case:

This use case demonstrates how a company like GE operates as both a manufacturer and a business organization using the same material products.

Use-Case Pattern Description



GE Aerospace (`ns1:ge-aerospace`) operates as both an organization (`iof:Organization`) and a business organization (`iof:BusinessOrganization`) with respect to jet engines (`ns1:jet-engine`). Through its manufacturing division (`ns1:ge-manufacturer`), which is `bfo:continuantPartOfAtAllTimes` of GE Aerospace, it bears a manufacturer role (`ns1:jet-engine-production-role`) which is `bfo:realize` through a production process (`ns1:jet-engine-production-process`), producing jet engines as `iof:hasOutput`. The organization also has a business function (`ns1:jet-engine-selling-function`) which is `bfo:realize` through a selling process (`ns1:jet-engine-selling-process`), where the same jet engines are sold (`bfo:hasParticipantAtSomeTime`) in the selling process.

Use-Case Example Data

Entity	Function/Role
GE Aerospace	Sells products and has Manufacturer
GE Manufacturer	Manufactures products
Jet engine	Product

Data Mapping

```
INSERT DATA {
    ns1:ge-aerospace a iof:Organization, iof:BusinessOrganization;
        iof:hasFunction ns1:jet-engine-selling-function.

    ns1:ge-manufacturer a iof:Manufacturer;
        bfo:continuantPartOfAtAllTimes ns1:ge-aerospace;
        iof:hasRole ns1:jet-engine-production-role.

    ns1:jet-engine-selling-function a iof:BusinessFunction.
    ns1:jet-engine-selling-process a iof:SellingBusinessProcess;
        bfo:realize ns1:jet-engine-selling-function;
        bfo:hasParticipantAtSomeTime ns1:jet-engine.

    ns1:jet-engine-production-role a iof:ManufacturerRole.
    ns1:jet-engine-production-process a iof:ProductProductionProcess;
        bfo:realize ns1:jet-engine-production-role;
        iof:hasOutput ns1:jet-engine.

    ns1:jet-engine a iof:MaterialProduct.

    iof:BusinessOrganization rdfs:subClassOf iof:Organization.
    iof:Manufacturer rdfs:subClassOf iof:Organization.
```

```
}
```

Data Validation

6.4. Group of agent vs group of persons

6.5. System and subsystem

Mix of hardware and software

6.6. Engineered systems

7 Changes

7.1. Change of object's location over time

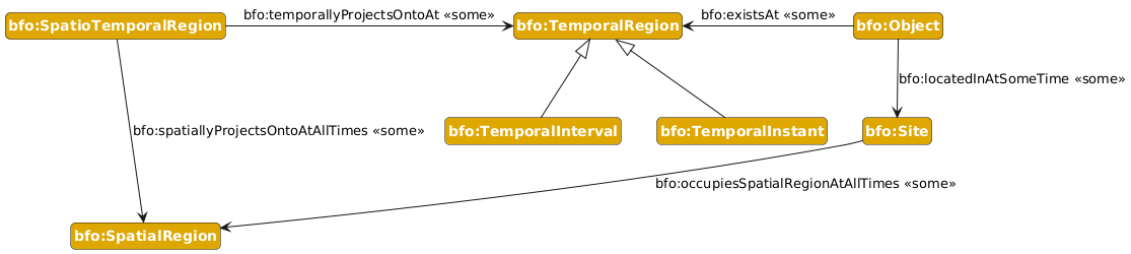
Created by: Arkopaul Sarkar
Modified by: Arkopaul Sarkar

Scenario Objective

This scenario illustrates how to represent the change in an object's physical location over time using the IOF/BFO ontology framework. It focuses on:

- Emphasizing physical locations as sites and their association with spatial regions.
- Capturing actual movements of objects between existing locations (does not express any future plan or schedule of movement).
- **Temporal focus:** Associating specific temporal instants with the object's presence at particular locations.

General Pattern Description



An object may be located in different places at different times. In this scenario, the object can be related to multiple locations as `bfo:Site` by the `bfo:locatedInAtSomeTime` property. Similarly, the object can be related to multiple instances of `bfo:TemporalRegion`, each being either a `bfo:TemporalInterval` or a `bfo:TemporalInstant` by the `bfo:existsAt` property. As the object cannot be at two different locations at the same time, the times and the locations need to be paired up to denote at which location the object was at what time. Each pairing can be done by a different `bfo:SpatioTemporalRegion`. Following the 4D ontology of BFO, the object's spatial and temporal co-occupation can be represented by a `bfo:SpationTemporalRegion`, which `bfo:temporallyProjectsOnto` the temporal region the object exists at and `bfo:spatiallyProjectsOntoAtAllTimes` the `bfo:SpatialRegion` the object occupies (`occupiesSpatialRegionAtAllTimes`).

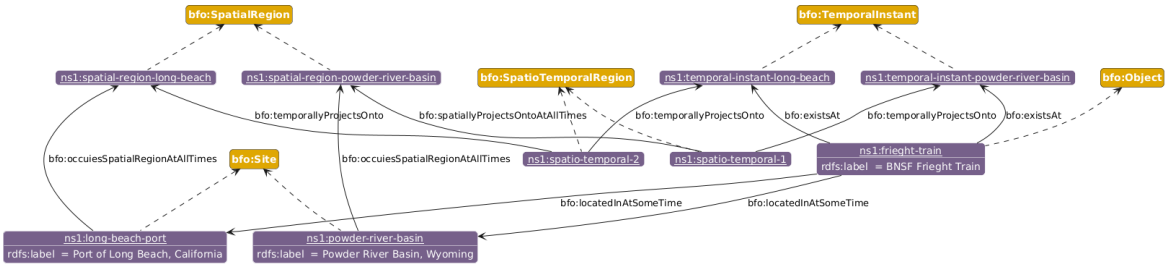
Note that the pattern shows that it is the object's location (a `bfo:Site`) that `occupiesSpatialRegionAtAllTimes`. However, an object also occupies the same spatial region that its location occupies. Therefore, this pattern can also be used by directly relating the object to a spatial region by `occupiesSpatialRegionAtAllTimes`. The choice depends on the kind of location, which is detailed in **??**.

Note further that a `SpatioTemporalRegion` cannot project onto two or more different instances of `bfo:SpatialRegion` while projecting on a single `bfo:TemporalRegion`, as an object cannot be at two different locations at the same time. But it can be in the same location at different times. Therefore, a `SpatioTemporalRegion` can project onto two or more different instances of `bfo:TemporalRegion` while projecting onto a single `bfo:SpatialRegion`. However, the data mapping becomes easier if one pair of temporal and spatial regions is related to each spatio-temporal region. If the object is in the same location at different times, two spatio-temporal regions can relate each temporal region to the same spatial region. Observe that these two spatio-temporal regions are simply parts of the spatio-temporal region that projects onto two temporal regions. Therefore, they capture the same semantics while making data mapping easier.

Use Case: Freight Train Location Change

A freight train operated by BNSF Railway hauls coal from Powder River Basin, Wyoming, to the Port of Long Beach in California. It was located at Powder River Basin station at 11:00 AM on Wednesday and then at Long Beach station at midnight on Thursday.

Use-Case Pattern Description



- The train `ns1:freight-train` is located at (`bfo:locatedInAtSomeTime`) two different locations: `ns1:powder-river-basin` and `ns1:long-beach-port`, both of which are instances of `bfo:site`.
- Two separate instances of `bfo:SpatialRegion`: `ns1:spatial-region-powder-river-basin` and `ns1:spatial-region-long-beach`.
- The sites are linked to the spatial regions by `occupiesSpatialRegionAtAllTimes`.
- Temporal instants are linked to the sites, but the actual clock times are omitted for brevity.

Use-Case Example Data

Train Number	Source	Destination	Departure Time	Arrival Time
31087	Powder River Basin	Long Beach	11:00 AM	00:00 PM
60123	Long Beach	Powder River Basin	8:00 AM	1:00 PM
31087	Powder River Basin	Long Beach	10:00 AM	4:00 PM
60123	Long Beach	Powder River Basin	8:00 AM	1:00 PM

Data Mapping Description

```
INSERT DATA {
    ns1:long-beach a bfo:Site;
                    bfo:occupiesSpatialRegionAtAllTimes ns1:spatial-region-long-beach.
    ns1:powder-river-basin a bfo:Site;
                    bfo:occupiesSpatialRegionAtAllTimes ns1:spatial-region-powder-river-basin.
    ns1:spatial-region-long-beach a bfo:SpatialRegion.
    ns1:spatial-region-powder-river-basin a bfo:SpatialRegion.
    ns1:temporal-instant-departure a bfo:TemporalInstant.
    ns1:temporal-instant-arrival a bfo:TemporalInstant.
    ns1:spatio-temporal-1 a bfo:SpatioTemporalRegion;
                    bfo:temporallyProjectsOnto ns1:temporal-instant-powder-river-basin;
                    bfo:temporallyProjectsOnto ns1:temporal-instant-long-beach;
                    bfo:spatiallyProjectsOntoAtAllTimes ns1:spatial-region-long-beach;
                    bfo:spatiallyProjectsOntoAtAllTimes ns1:spatial-region-powder-river-basin.
    ns1:freight-train a bfo:Object;
                    rdfs:label "BNSF Freight Train";
                    bfo:locatedInAtSomeTime ns1:powder-river-basin;
                    bfo:locatedInAtSomeTime ns1:long-beach-port;
                    bfo:existsAt ns1:temporal-instant-powder-river-basin;
                    bfo:existsAt ns1:temporal-instant-long-beach.
```

```
}
```

```
INSERT {
    _:source a bfo:Site;
                    iof:hasValueExpressionAtAllTimes ?location1;
                    bfo:occupiesSpatialRegionAtAllTimes _:spatial-region-source.
    _:destination a bfo:Site;
                    iof:hasValueExpressionAtAllTimes ?location2;
                    bfo:occupiesSpatialRegionAtAllTimes _:spatial-region-destination.
    _:spatial-region-source a bfo:SpatialRegion.
    _:spatial-region-destination a bfo:SpatialRegion.
    _:temporal-instant-departure a bfo:TemporalInstant;
                    iof:hasValueExpressionAtAllTimes ?departure.
    _:temporal-instant-arrival a bfo:TemporalInstant;
                    iof:hasValueExpressionAtAllTimes ?arrival.
    _:spatio-temporal-1 a bfo:SpatioTemporalRegion;
                    bfo:temporallyProjectsOnto _:temporal-instant-departure;
                    bfo:spatiallyProjectsOntoAtAllTimes _:spatial-region-source.
    _:spatio-temporal-2 a bfo:SpatioTemporalRegion;
                    bfo:temporallyProjectsOnto _:temporal-instant-arrival;
                    bfo:spatiallyProjectsOntoAtAllTimes _:spatial-region-destination.
    ?freight-train a bfo:Object;
                    rdfs:label "BNSF Freight Train";
                    bfo:locatedInAtSomeTime _:source;
                    bfo:locatedInAtSomeTime _:destination;
                    bfo:existsAt _:temporal-instant-departure;
                    bfo:existsAt _:temporal-instant-arrival.
}
WHERE{
    BIND(COLUMN("Train Number") as ?freight-train).
    BIND(COLUMN("Source") as ?location1).
    BIND(COLUMN("Destination") as ?location2).
    BIND(COLUMN("Departure Time") as ?departure).
    BIND(COLUMN("Arrival Time") as ?arrival).
}
```

Data Validation

```
# Shape for bfo:SpatioTemporalRegion
ns1:SpatioTemporalRegionShape a sh:NodeShape ;
    sh:targetClass bfo:SpatioTemporalRegion ;
    sh:property [
        sh:path bfo:temporallyProjectsOnto ;
        sh:class ns1:TemporalInstant ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path bfo:spatiallyProjectsOntoAtAllTimes ;
        sh:class bfo:SpatialRegion ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] .

# Shape for the object
```

```
ns1:ObjectShape a sh:NodeShape ;
    sh:targetClass bfo:Object ;
    sh:property [
        sh:path bfo:locatedInAtSomeTime ;
        sh:or (
            [ sh:class bfo:Site ] ;
            [ sh:class bfo:SpatialRegion ]
        ) ;
        sh:minCount 1 ;
    ] ;
    sh:property [
        sh:path bfo:existsAt ;
        sh:class ns1:TemporalInstant ;
        sh:minCount 1 ;
    ] .

# Shape for bfo:Site (if ns1:ObjectShape has bfo:Site as the target of path bfo:locatedInAtSomeTime)
ns1:SiteShape a sh:NodeShape ;
    sh:targetClass bfo:Site ;
    sh:property [
        sh:path bfo:occupiesSpatialRegionAtAllTimes ;
        sh:class bfo:SpatialRegion ;
        sh:minCount 1 ;
    ] .

Q. 1: How many
```

8 Process and Event

8.1. Simple Process Sequence

Created by: Jim Logan
Modified by: Jim Logan

Scenario Objective

This scenario demonstrates how to represent a sequence of time-stamped events using the **BFO/IOF ontology**. Several use cases are given, where each subsequent use case augments the previous one, culminating in a use case that demonstrates how to determine the type of an overall process automatically from a sequence of sub-processes.

Note: I'm unsure what to say about the purpose of doing this!

General Pattern Description

This pattern creates instances of bfo:process, each of which core:*occupiesTemporalRegion* one bfo:temporalInterval with single start and end instances of bfo:TemporalInstant that each core:*hasValueExpressionAtAllTimes* some core:TemporalInstantValueExpression, according to the earlier pattern in 2.1.

TBD: Provide a figure containing the pattern independent of the use case (e.g., just measurements in general as opposed to measuring a particular thing such as pH or length). This diagram is generally a simple class-relation diagram. Please see examples of class-relation diagram here.

Describe the general pattern, including any background why such types and relations are used. Also mention any alternative or shortcuts. Please provide a background of the types and relations in the context of IOF (please prefer logical arguments to didactic arguments).

Use Case: Simple Process Sequence

This first use case is the simplest. It uses the example data, shown below, to create corresponding instances in the IOF theory of what must have existed in the world to justify the data.

Describe a concrete use case for the pattern. (Multiple use cases can be described for one pattern. If multiple, please use distinct use case titles and repeat all subsubsections for each use case.)

Use-Case Pattern Description

This use case follows the pattern in 2.1. Each timestamp in a row that is marked "Start" creates several instances at once and then wires them together as follows: a bfo:Process that core:*occupiesTemporalRegion* a bfo:TemporalInstant that bfo:*hasFirstInstant* a bfo:TemporalInstant that iof:hasValueExpressionAtAllTimes a time:DateTimeDescription.

Note that in this first use case, the EventType is used to create a generic bfo:process. A subsequent use case will create an instance of a more specific type of process that corresponds to the EventType.

Each timestamp in a row that is marked "Stop" augments what was already created for a "Start" row with several more instances, and then wires them together as follows: the already-created bfo:TemporalRegion bfo:*hasLastInstant* a bfo:TemporalInstant that iof:hasValueExpressionAtAllTimes a time:DateTimeDescription.

Provide a diagram which matches the use-case pattern description.
This diagram is generally an object diagram. Please see examples of object diagram here.

Describe how the general pattern is applied to the use case. It is important to highlight within the description how the use-case-specific concepts align with the general pattern (e.g., SubClassOf Class for general pattern).

Use-Case Example Data

The following table shows CSV data resulting from a fictitious machine that makes drip coffee. Each row contains the date and time at which an instance of a type of process started or stopped.

Timestamp	EventType	StartStop
2025-02-21T07:08:00	HeatWater	Start
2025-02-21T07:08:02	HeatWater	Stop
2025-02-21T07:08:02	SoakCoffee	Start
2025-02-21T07:08:04	SoakCoffee	Stop
2025-02-21T07:08:04	HeatCarafe	Start

Table 8.1: An Example Event Log for Making Drip Coffee

Data Mapping

The following SPARQL converts the CSV file into RDF triples. Its WHERE clause connects to a service that makes each row in the CSV file available, it binds each CSV field to a variable, constructs an IRI from a universally unique identifier (UUID). Its INSERT clause then uses those variables from a row in the CSV file to insert a pattern of three triples into a repository. The predicates of those triples exist to connect the IRI (bound to the variable "?id") to the three fields of each row.

```
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix ex: <http://example.com/schema/>
BASE <http://example.com/base/>
INSERT {
  ?id
    ex:timestamp ?Timestamp ;
    ex:eventType ?EventType ;
    ex:startStop ?StartStop .
}
WHERE {
  SERVICE <http://localhost:7333/repositories/ontorefine:2117727886747> {
    BIND(xsd:dateTime(?c_Timestamp) as ?Timestamp) .
    BIND( ?c_EventType AS ?EventType ) .
    BIND( ?c_StartStop AS ?StartStop ) .
    BIND(IRI(CONCAT("https://ontogenesis-solutions.com/ontologies/examples/instances/",
        STRUUID())) AS ?id)
  }
}
```

After the content of the CSV file is inserted into the triple store, the next SPARQL query pairs up the start and end of each type of event and then properly instantiates classes and properties from the IOF Core ontology. The syntax here demonstrates what may be a more familiar variable notation for some practitioners.

Note that the last row of the CSV file (shown in 8.1) has no "Stop" row because the drip coffee maker continues to heat the carafe until it is turned off, which, in this fictitious model, also disconnects power from the internal computer producing the event file. The SPARQL query handles this using an "OPTIONAL" clause. Also note that the UUID from every row that contained "Start" is used as a prefix for *all* of the related instance IRIs, which keeps them together when listed and provides a sensible, human-readable suffix.

Given a pattern match in the "WHERE" clause, all of the variables are ready for the "CONSTRUCT" clause.

```
prefix ex: <http://example.com/schema/>
prefix dc: <http://purl.org/dc/elements/1.1/>
prefix bfo: <http://purl.obolibrary.org/obo/>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix xml: <http://www.w3.org/XML/1998/namespace>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix inst: <https://ontogenesis-solutions.com/ontologies/examples/instances/>
```

```
prefix classes: <https://ontogenesis-solutions.com/ontologies/examples/classes/>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix skos: <http://www.w3.org/2004/02/skos/core#>
prefix terms: <http://purl.org/dc/terms/>
prefix iof-av: <https://spec.industrialontologies.org/ontology/core/meta/AnnotationVocabulary
    />
prefix iof-core: <https://spec.industrialontologies.org/ontology/core/Core/>

CONSTRUCT {
    $pid
        rdf:type
            $bfoProcess ,
            $class;
        $bfoOccupiesTemporalRegion $interval .


    $interval
        rdf:type
            $bfoTemporalInterval ;
        $bfoHasFirstInstant $firstInstant ;
        $bfoHasLastInstant $lastInstant .

    $firstInstant
        rdf:type
            $bfoTemporalInstant ;
        iof-core:hasValueExpressionAtAllTimes $firstVE .

    $lastInstant
        rdf:type
            $bfoTemporalInstant ;
        iof-core:hasValueExpressionAtAllTimes $lastVE .

    $firstVE
        rdf:type
            iof-core:TemporalInstantValueExpression ;
        iof-core:hasDateTimeInstantValue $startTime .

    $lastVE
           rdf:type
            iof-core:TemporalInstantValueExpression ;
        iof-core:hasDateTimeInstantValue $endTime .
}

WHERE {
    $startId
        ex:startStop "Start";
        ex:eventType $eventType;
        ex:timestamp $startTime.

    OPTIONAL {
        $endId
            ex:startStop "Stop";
            ex:eventType $eventType;
            ex:timestamp $endTime.
    }

    BIND(bfo:BFO_0000203 AS $bfoTemporalInstant)
    BIND(bfo:BFO_0000015 AS $bfoProcess)
    BIND(bfo:BFO_0000199 AS $bfoOccupiesTemporalRegion)
    BIND(bfo:BFO_0000202 AS $bfoTemporalInterval)
    BIND(bfo:BFO_0000222 AS $bfoHasFirstInstant)
    BIND(bfo:BFO_0000224 AS $bfoHasLastInstant)

    BIND(IRI(CONCAT(STR($startId), "/process")) AS $pid)
    BIND(IRI(CONCAT(STR($startId), "/interval")) AS $interval)
    BIND(IRI(CONCAT(STR($startId), "/firstInstant")) AS $firstInstant)
    BIND(IRI(CONCAT(STR($startId), "/lastInstant")) AS $lastInstant)
    BIND(IRI(CONCAT(STR($startId), "/firstVE")) AS $firstVE)
    BIND(IRI(CONCAT(STR($startId), "/lastVE")) AS $lastVE)

    BIND(IRI(CONCAT("https://ontogenesis-solutions.com/ontologies/examples/classes/",
```

```
        $eventType)) AS $class)

}
ORDER BY $startTime
```

Data Validation

Data validation can be performed in two ways: accessing interesting facts using SPARQL or validating whether the entire data conforms to the ontology using SHACL. It is preferable to provide both. Provide the SPARQL query in the code block along with the result of the query.

Use Case: Typed Process Sequence

This use case extends the use case from section 8.1 with specializations of bfo:process corresponding to the EventType column in Table 8.1.

Describe a concrete use case for the pattern. (Multiple use cases can be described for one pattern. If multiple, please use distinct use case titles and repeat all subsubsections for each use case.)

Use-Case Pattern Description

Provide a diagram which matches the use-case pattern description.
This diagram is generally an object diagram. Please see examples of object diagram here.

Describe how the general pattern is applied to the use case. It is important to highlight within the description how the use-case-specific concepts align with the general pattern (e.g., SubClassOf Class for general pattern).

Use-Case Example Data

The following table shows CSV data resulting from a fictitious machine that makes drip coffee. Each row contains the date and time at which a type of event started or stopped.

Data Mapping

Describe how the data was mapped to RDF. Provide an INSERT DATA/ INSERT SPARQL for mapping. Use INSERT only when the use case example data needs further manipulation.
For INSERT DATA SPARQL, use only 2/3 records/transactions with named class and individuals.
For INSERT SPARQL, declare column names by '{ }' to the variable. For INSERT query, For both, do not use blank nodes.

Data Validation

Data validation can be performed in two ways: accessing interesting facts using SPARQL or validating whether the entire data conforms to the ontology using SHACL. It is preferable to provide both. Provide the SPARQL query in the code block along with the result of the query.

Process Design

Here discuss how to represent a SysML / UML activity diagram in an IOF Ontology.

Process Design and Conformance

Here discuss how to classify time stamped events automatically as making coffee.

8.2. Algorithm execution

Created by: Milos Drobnjakovic
Modified by: Milos Drobnjakovic

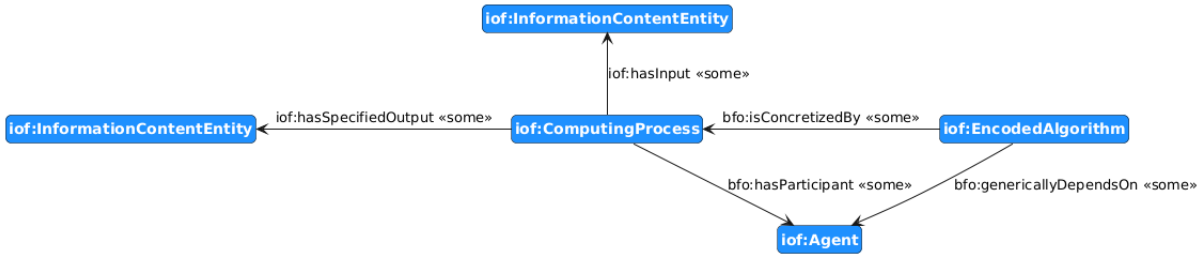
Scenario Objective

This scenario demonstrates how to represent the execution of algorithms within the context of the **BFO/IOF ontology**. It specifically addresses cases where algorithm executions generate **Information Content Entities (ICEs)** as outputs. Algorithms that do not produce outputs are beyond the scope of this pattern.

Key Points

- This pattern is intended for algorithms integrated into a software system.
- Only algorithm executions resulting in ICEs are covered; executions without outputs are excluded.

General Pattern Description

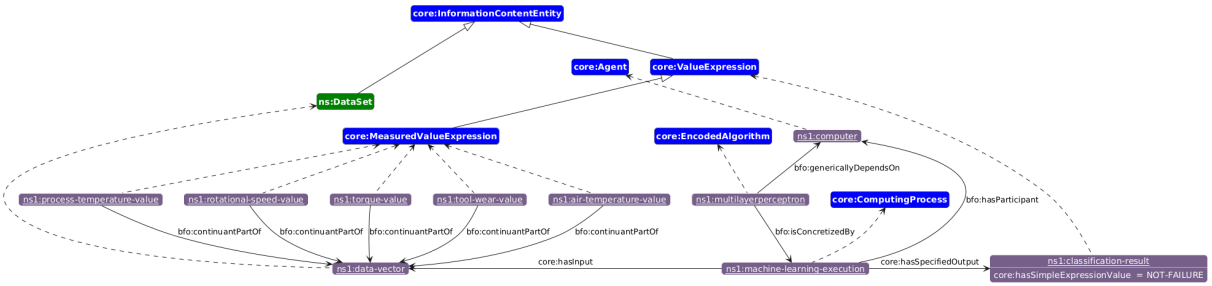


...

Use Case: Drill failure prediction

A multilayer perceptron (MLP – a type of a neural network) is used to predict drill failure based on five measured parameters: ‘tool wear’, ‘air temperature’, ‘process temperature’, ‘torque’,’rotational speed’. The model uses the given parameters and outputs either ‘FAIL’ or ‘NOT FAIL’.

Use-Case Pattern Description



This use case illustrates the execution of a machine learning model, `mutltilayerperceptron`, on a computer system. The model is an instance of `Encoded Algorithm` and is `concretized` through a `Computing Process` representing its execution.

Actors and Dependencies

- The `mutltilayerperceptron generically depends on` a `computer` (an instance of `Agent`) that `participates in` the `machine-learning-execution` (an instance of `ComputingProcess`).
- The `machine-learning-execution` relies on (`hasInput`) measured values associated with the drill and the drilling process as input data (represented as various instances of `MeasuredValueExpression`).

Process and Output

- The `machine-learning-execution` produces (`has-specified-output`) a `classification-result`, an instance of `ValueExpression`.
- The `classification-result` can take one of two possible values, represented by the `hasSimpleExpressionValue` property:
  1. `NOT-FAILURE`
  2. `FAILURE`

Diagram Context

The diagram below demonstrates a specific example where the `classification-result` has the value `NOT-FAILURE`. For simplicity: The diagram below demonstrates a specific example where the `classification-result` has the value `NOT-FAILURE`. For simplicity:

- The actual measurement values and their units are excluded from the diagram.
- Associations between measured values and entities such as “drill attributes” or the “drilling process” are not shown.

Further Guidance

For details on connecting measured values with their respective entities, please refer to the detailed guide.

Use-Case Example Data

For this use case a publicly available dataset was used: Predictive Maintenance

The dataset is a single CSV table consisting of the columns: ‘product ID’, ‘type’, ‘air temperature’, ‘process temperature’, ’rotational speed’, ‘torque’, ‘tool wear’, ‘target’, and ‘failure type’.

Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type
M14860	M	298.1	308.6	1551	42.8	0	0	No Failure
L47181	L	298.2	308.7	1408	46.3	3	0	No Failure
L47182	L	298.1	308.5	1498	49.4	5	0	No Failure

For prediction purposes, type and Product ID are omitted and are as such not included in the RDF. The model outputs the entries of the target column which is then converted to ’No-failure’ or ’Failure’. For brevity, conversion of target to failure type is omitted.

Data Mapping Description

```
INSERT DATA {
  ns-1:air-temperature-value a iof-core:MeasuredValueExpression ;
    bfo:continuantPartOf ns-1:data-vector-1 .
  ns-1:process-temperature-value a iof-core:MeasuredValueExpression ;
    bfo:continuantPartOf ns-1:data-vector-1 .
  ns-1:tool-wear-value a iof-core:MeasuredValueExpression ;
    bfo:continuantPartOf ns-1:data-vector-1 .
  ns-1:torque-value a iof-core:MeasuredValueExpression ;
    bfo:continuantPartOf ns-1:data-vector-1 .
  ns-1:rotational-speed-value a iof-core:MeasuredValueExpression ;
    bfo:continuantPartOf ns-1:data-vector-1 .
  ns-1:data-vector-1 a ns-1:DataSet .
  ns-1:DataSet rdfs:subClassOf iof-core:InformationContentEntity .
  ns-1:machine-learning-execution a iof-core:ComputingProcess ;
    iof-core:hasInput ns-1:data-vector-1 ;
    iof-core:hasSpecifiedOutput ns-1:classification-result ;
    bfo:hasParticipant ns-1:computer .
  ns-1:classification-result a iof-core:ValueExpression .
  ns-1:computer a iof-core:Agent .
  ns-1:multilayerperceptron a iof-core:EncodedAlgorithm ;
    bfo:isConcretizedBy ns-1:machine-learning-execution ;
    bfo:genericallyDependsOn ns-1:computer .
}
```

Data Validation

```
# Shape for MeasuredValueExpression
ns-1:MeasuredValueExpressionShape a sh:NodeShape ;
    sh:targetClass iof-core:MeasuredValueExpression ;
    sh:property [
        sh:path bfo:continuantPartOf ;
        sh:class ns-1:DataSet ;
        sh:nodeKind sh:IRI ;
        sh:message "Each MeasuredValueExpression must be part of a DataSet." ;
    ] .

# Shape for ComputingProcess
ns-1:ComputingProcessShape a sh:NodeShape ;
    sh:targetClass iof-core:ComputingProcess ;
    sh:property [
        sh:path iof-core:hasInput ;
        sh:class ns-1:DataSet ;
        sh:nodeKind sh:IRI ;
        sh:message "ComputingProcess must have an input of type DataSet." ;
    ] ;
    sh:property [
        sh:path iof-core:hasSpecifiedOutput ;
        sh:class iof-core:ValueExpression ;
        sh:nodeKind sh:IRI ;
        sh:message "ComputingProcess must have an output of type ValueExpression." ;
    ] ;
    sh:property [
        sh:path bfo:hasParticipant ;
        sh:class iof-core:Agent ;
```

```
        sh:nodeKind sh:IRI ;
        sh:message "ComputingProcess must have a participant of type Agent." ;
    ] .

# Shape for EncodedAlgorithm
ns-1:EncodedAlgorithmShape a sh:NodeShape ;
    sh:targetClass iof-core:EncodedAlgorithm ;
    sh:property [
        sh:path bfo:isConcretizedBy ;
        sh:class iof-core:ComputingProcess ;
        sh:nodeKind sh:IRI ;
        sh:message "EncodedAlgorithm must be concretized by a ComputingProcess." ;
    ] ;
    sh:property [
        sh:path bfo:genericallyDependsOn ;
        sh:class iof-core:Agent ;
        sh:nodeKind sh:IRI ;
        sh:message "EncodedAlgorithm must generically depend on an Agent." ;
    ] .
```

9 Information

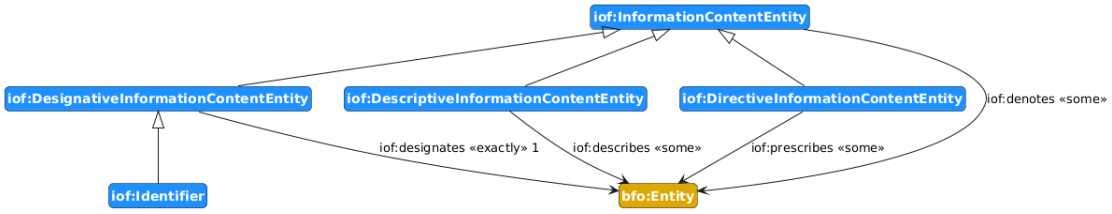
9.1. Information about some entity(s)

Created by: Arkopaul Sarkar
Modified by:

Scenario Objective

This scenario illustrates how various types of information about an entity (e.g., object, process, properties) can be captured. This scenario introduces the fundamental classes and properties to model information. Although IOF contains many types of informational entities, this scenario only introduces the generic types of information based on how they refer to the entities they are about.

General Pattern Description



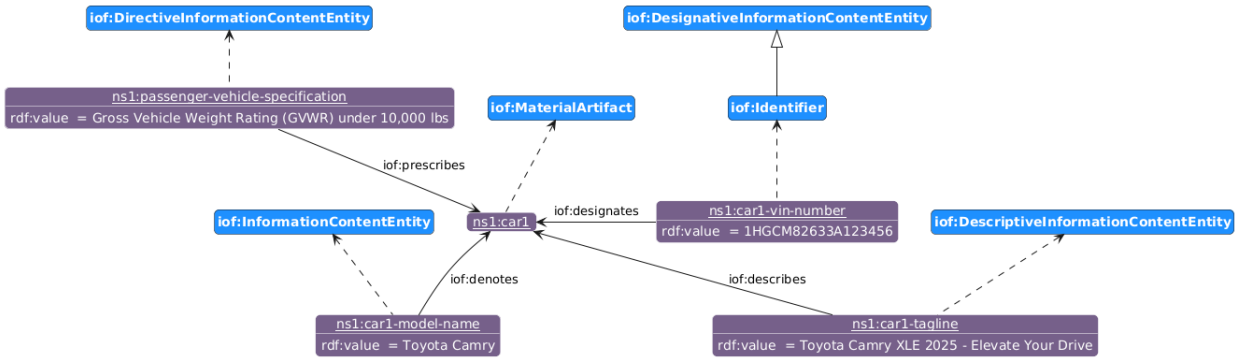
The primitive and generic relationship `iof:isAbout` establishes a connection between an Information Content Entity (ICE) and a referenced entity, encapsulating the fundamental notion of "aboutness"—that is, the ICE instance conveys information about the entity.

Three distinct modes of referring to an entity are captured through three specialized sub-properties of `iof:isAbout`: `iof:denotes` serves to distinguish one or more instances of an entity. `iof:designates`, a functional specialization of `iof:denotes`, uniquely identifies a single specific instance, ensuring exclusive reference. `iof:describes` conveys attributes or characteristics of an entity without necessarily providing unique identification. It emphasizes detailing the entity rather than establishing a direct referential link. `iof:prescribes` refers to entities that may not yet exist but are expected to adhere to prescribed rules, guidelines, or specifications once instantiated. It thus encompasses potential future entities, such as artefacts constructed according to a design specification or processes executed in alignment with a plan specification. Accordingly, the IOF Core ontology defines three corresponding classes of ICE—Designative ICE, Descriptive ICE, and Prescriptive ICE — each qualified based on these three sub-properties of `iof:isAbout`.

Use Case: Information about a car

In this use case, information about a car are asserted using different types of "about-ness" relations. The types of the ICE instances may be inferred based on the asserted sub-property of `iof:isAbout`. In this example, the VIN of the car acts as a unique identifier of the vehicle and the model of the car denotes the structural style of the car. A branding slogan euphemistically describes the car's ability. Lastly, a directive from the Department of Transportation prescribes the car's weight rating, i.e., the car follows the specifications for a passenger car as prescribed by 'Gross Vehicle Weight Rating (GVWR)' from the Department of Transportation.

Use Case Pattern Description



The designative information content entity (iof:designates) uniquely identifies a specific instance of the car, typically using a Vehicle Identification Number (VIN) or a similar unique identifier, ensuring that a particular vehicle unit is individually referenced within a system. The denotative information content entity (iof:denotes) assigns a model name (Toyota Camry) to the vehicle, referring to a general category rather than an individual instance, which helps differentiate between different models within a brand. The descriptive information content entity (iof:describes) provides branding and marketing information, such as the tagline ”Toyota Camry XLE 2025 – Elevate Your Drive”, which conveys the product’s perception but does not serve as a unique identifier.

Meanwhile, the directive information content entity (iof:prescribes) applies to regulatory constraints, such as ”Gross Vehicle Weight Rating (GVWR) under 10,000 lbs”, ensuring compliance with passenger vehicle classification standards before the car is operational. The ontology’s structured approach enables semantic differentiation between designation (unique instance reference), denotation (model reference), description (attributes and perception), and prescription (rules and requirements). This classification ensures that different stakeholders—manufacturers, regulators, marketers, and data systems—can process and interpret vehicle-related information accurately for production, sales, legal compliance, and operational tracking.

Use-Case Example Data

```
PREFIX iof: <http://example.org/iof#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ns1: <http://example.org/ns1#>

INSERT DATA {
  ns1:car1 a iof:MaterialArtifact .
  ns1:car1-vin-number a iof:Identifier ;
      iof:designates ns1:car1 ;
      rdf:value "1HGCM82633A123456" . # Example VIN
  ns1:car1-model-name a iof:InformationContentEntity ;
      iof:denotes ns1:car1 ;
      rdf:value "Toyota Camry" .
  ns1:car1-tagline a iof:DescriptiveInformationContentEntity ;
      iof:describes ns1:car1 ;
      rdf:value "Toyota Camry XLE 2025 - Elevate Your Drive" .
  ns1:passenger-vehicle-specification a iof:DirectiveInformationContentEntity ;
      iof:prescribes ns1:car1 ;
      rdf:value "Gross Vehicle Weight Rating (GVWR) under 10,000 lbs" .
}
```

Data Mapping

Data Validation

URI of a website; social security number of a person (living in the United States), a global location number assigned to the Amazon regional distribution center at 12300 Bermuda Rd in Henderson, NV; the lot identifier assigned to a batch of rivets just received from China by the Airbus final assembly plant in Toulouse, FR; the VIN number assigned to the Tesla in my garage; a credit card number, the value of a field in a company's internal IT systems system used to uniquely identify a particular product and product revision.

Batch RV123456 - Rivets from China Lot number: RV123456 Supplier: China Aerospace Rivets Ltd.

9.2. descriptions

We have to address:

simple description. e.g., name, „,

Value expressions. subtypes of value expressions

1cm is the value expression of the diameter of a screw head that is specified in its design; 37C is the value expression of the temperature of a bioreactor measured during the production process; "low risk" is the value expression of a process parameter based on the risk analysis classification scheme; 3 g/l is the value expression of titer of an antibody generated by a process simulation - need to be genric...

9.3. Specifications

10 Measurement

10.1. Simple vs Aggregate Measurement

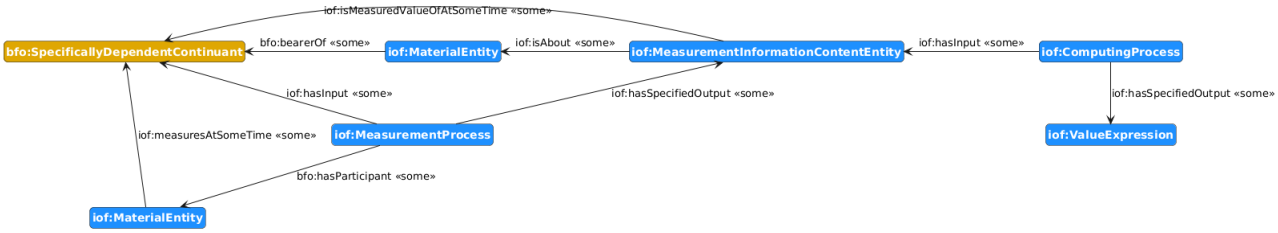
Scenario Objective

This scenario demonstrates how to represent measurements within the context of the **BFO/IOF ontology**. It then also represents how to capture processing simple measurements to get an aggregate result (such as doing an average).

Key Points

- This pattern highlights the connection between a measurement and the attribute that is being measured
- The pattern demonstrates how measurements are processed and introduces a distinction between measured data and process data and its association with an attribute
- For the examples in hand QUDT is used for representing units according to the guide x. It should be noted that using QUDT is not normative for the IOF

General Pattern Description



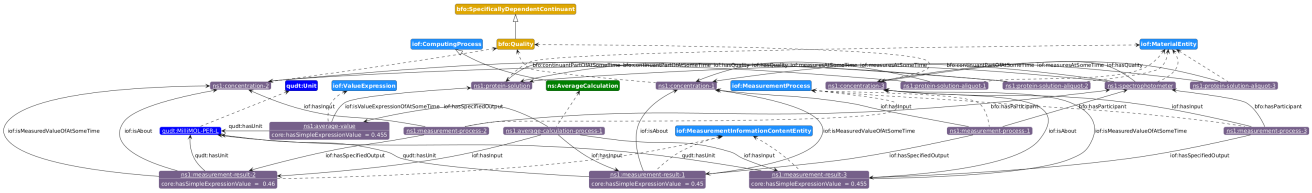
10.1.1.

Use case: Measuring protein concentration with a Bicinchoninic Acid (BCA) Assay

Within a laboratory environment protein concentration during batch purification is typically measured by using the BCA assay. The assay is based on a colorimetric detection method, where the analyte reacts with a specific reagent to produce a quantifiable color change. The absorbance of the reaction product is measured at a defined wavelength using a spectrophotometer, with the signal intensity correlating to the analyte concentration according to a pre-established standard curve. The details of converting absorbance into concentration are excluded in the pattern given here. To capture this conversion ontologically the reader should take a look at [placeholder UC]

Each concentration measurement is performed in triplicate to ensure accuracy, reproducibility, and compliance with quality control standards. Three independent aliquots (samples) are analyzed under identical experimental conditions, with results recorded separately. The final reported concentration value is then derived from the mean of these replicates. It should be noted that in addition to the mean the standard deviation is reported. However, this is outside of the scope of the pattern. For representing standard deviation the reader can look at [placeholder]

Use-Case Pattern Description



Actors

The primary actors involved in this use case include:

- **Spectrophotometer (ns1:spectrophotometer):** A material entity that participates in the measurement process by performing the actual measurement of the protein solution.
- **Measurement Process (ns1:measurement-process-1):** The process responsible for recording the protein concentration from the sample. (Note: Additional measurement processes, such as `ns1:measurement-process-2` and `ns1:measurement-process-3`, follow the same pattern.)
- **Protein Solution Aliquot (ns1:protein-solution-aliquot-1):** The sample containing the protein solution that is subject to measurement.
- **Concentration Quality (ns1:concentration-1):** The quality attribute of the protein solution representing the measured concentration.
- **Measurement Result (ns1:measurement-result-1):** The output produced by the measurement process, capturing the numerical concentration value along with its unit.
- **Computing Process (Average Calculation) (ns1:average-calculation-process-1):** The process that aggregates multiple measurement results to compute an average concentration value.
- **Average Value (ns1:average-value):** The computed average concentration value, representing a more reliable measurement of the protein concentration.
- **Protein Solution (ns1:protein-solution):** The overall material entity to which the measurement and computed average are associated.

Process Steps

The process unfolds in the following steps:

1. **Measurement Execution:**
   - The spectrophotometer (`ns1:spectrophotometer`) measures the protein concentration of the aliquot (`ns1:protein-solution-aliquot-1`).
   - The measurement process (`ns1:measurement-process-1`) records a result (`ns1:measurement-result-1`) that includes a concentration value (e.g., ”0.45”) and its unit (`qudt:MilliMOL-PER-L`).
2. **Data Association:**
   - The measurement result (`ns1:measurement-result-1`) is linked to the concentration quality (`ns1:concentration-1`) of the protein solution.
   - The result is annotated with the appropriate unit of measure, ensuring data consistency.
3. **Average Calculation:**
   - A computing process (`ns1:average-calculation-process-1`) aggregates multiple measurement results.
   - An average value (`ns1:average-value`) is computed, offering a reliable estimate of the protein concentration.
   - The computed average is associated with the overall protein solution (`ns1:protein-solution`).

By following this pattern, the protein solution retains references to both individual measurements and the computed average, ensuring traceability.

Use-Case Example Data

Three rows of a CSV table are given bellow where each row constitutes an id of the material from which the aliquots are taken from. The concentration of each aliquot and finally the aggregate measurement.

Concentrations are reported in g/l.

Table 10.1: Triplicate Measurement Data after a purification step (Concentrations in mmol/l)

Material ID	Aliquot 1 (mmol/l)	Aliquot 2 (mmol/l)	Aliquot 3 (mmol/l)	Aggregate (mmol/l)
ID001	0.45	0.46	0.455	0.455
ID002	0.50	0.52	0.51	0.51
ID003	0.42	0.43	0.425	0.425

Data Mapping Description

```
INSERT DATA {
# Classes
   ns:AverageCalculation rdfs:subClassOf iof:ComputingProcess .
  # Individuals
  ns1:measurement-process-1 a iof:MeasurementProcess .
  ns1:measurement-result-1 a iof:MeasurementInformationContentEntity .
  ns1:protein-solution-aliquot-1 a iof:MaterialEntity .
  ns1:concentration-1 a bfo:Quality .
  ns1:spectrophotometer a iof:MaterialEntity .
  ns1:average-calculation-process-1 a ns:AverageCalculation .
  ns1:average-value a iof:ValueExpression .
  ns1:protein-solution a iof:MaterialEntity .
  qudt:MilliMOL-PER-L a qudt:Unit .

  # Relationships
  ns1:measurement-process-1 iof:hasSpecifiedOutput ns1:measurement-result-1 .
  ns1:measurement-process-1 iof:hasInput ns1:concentration-1 .
  ns1:measurement-process-1 bfo:hasParticipant ns1:spectrophotometer .

  ns1:spectrophotometer iof:measuresAtSomeTime ns1:concentration-1 .
  ns1:protein-solution-aliquot-1 iof:hasQuality ns1:concentration-1 .
  ns1:measurement-result-1 iof:isMeasuredValueOfAtSomeTime ns1:concentration-1 .
  ns1:measurement-result-1 iof:isAbout ns1:concentration-1 .

  ns1:measurement-result-1 qudt:hasUnit qudt:MilliMOL-PER-L .
  ns1:measurement-result-1 core:hasSimpleExpressionValue "0.45" .

  ns1:average-calculation-process-1 iof:hasInput ns1:measurement-result-1 .
  ns1:average-calculation-process-1 iof:hasSpecifiedOutput ns1:average-value .
  ns1:average-value iof:isValueExpressionOfAtSomeTime ns1:protein-solution .
  ns1:average-value core:hasSimpleExpressionValue "0.455" .
  ns1:average-value qudt:hasUnit qudt:MilliMOL-PER-L .

  ns1:protein-solution-aliquot-1 bfo:continuantPartOfAtSomeTime ns1:protein-solution .

  # Additional measurement instances (measurement 2 & 3 follow the same pattern)
  ns1:measurement-process-2 a iof:MeasurementProcess .
  ns1:measurement-result-2 a iof:MeasurementInformationContentEntity .
  ns1:protein-solution-aliquot-2 a iof:MaterialEntity .
  ns1:concentration-2 a bfo:Quality .

  ns1:measurement-process-2 iof:hasSpecifiedOutput ns1:measurement-result-2 .
  ns1:measurement-process-2 iof:hasInput ns1:concentration-2 .
  ns1:measurement-process-2 bfo:hasParticipant ns1:spectrophotometer .
```

```
  ns1:spectrophotometer iof:measuresAtSomeTime ns1:concentration-2 .
  ns1:protein-solution-aliquot-2 iof:hasQuality ns1:concentration-2 .
  ns1:measurement-result-2 iof:isMeasuredValueOfAtSomeTime ns1:concentration-2 .
  ns1:measurement-result-2 iof:isAbout ns1:concentration-2 .

  ns1:measurement-result-2 qudt:hasUnit qudt:MilliMOL-PER-L .
  ns1:measurement-result-2 core:hasSimpleExpressionValue "0.46" .

  ns1:protein-solution-aliquot-2 bfo:continuantPartOfAtSomeTime ns1:protein-solution .

  ns1:measurement-process-3 a iof:MeasurementProcess .
  ns1:measurement-result-3 a iof:MeasurementInformationContentEntity .
  ns1:protein-solution-aliquot-3 a iof:MaterialEntity .
  ns1:concentration-3 a bfo:Quality .

  ns1:measurement-process-3 iof:hasSpecifiedOutput ns1:measurement-result-3 .
  ns1:measurement-process-3 iof:hasInput ns1:concentration-3 .
  ns1:measurement-process-3 bfo:hasParticipant ns1:spectrophotometer .

  ns1:spectrophotometer iof:measuresAtSomeTime ns1:concentration-3 .
  ns1:protein-solution-aliquot-3 iof:hasQuality ns1:concentration-3 .
  ns1:measurement-result-3 iof:isMeasuredValueOfAtSomeTime ns1:concentration-3 .
  ns1:measurement-result-3 iof:isAbout ns1:concentration-3 .

  ns1:measurement-result-3 qudt:hasUnit qudt:MilliMOL-PER-L .
  ns1:measurement-result-3 core:hasSimpleExpressionValue "0.455" .

  ns1:protein-solution-aliquot-3 bfo:continuantPartOfAtSomeTime ns1:protein-solution .
}
```

Data Validation

```
# Shape for MeasurementProcess
ns:MeasurementProcessShape a sh:NodeShape;
    sh:targetClass iof:MeasurementProcess;
    sh:property [
        sh:path iof:hasSpecifiedOutput;
        sh:class iof:MeasurementInformationContentEntity;
        sh:minCount 1;
    ];
    sh:property [
        sh:path iof:hasInput;
        sh:class bfo:Quality;
        sh:minCount 1;
    ];
    sh:property [
        sh:path bfo:hasParticipant;
        sh:class iof:MaterialEntity;
        sh:minCount 1;
    ].

# Shape for MeasurementInformationContentEntity
ns:MeasurementResultShape a sh:NodeShape;
    sh:targetClass iof:MeasurementInformationContentEntity;
```

```
    sh:property [
        sh:path iof:isMeasuredValueOfAtSomeTime;
        sh:class bfo:Quality;
        sh:minCount 1;
        sh:maxCount 1;
    ];
    sh:property [
        sh:path iof:isAbout;
        sh:class bfo:Quality;
        sh:minCount 1;
    ];
    sh:property [
        sh:path qudt:hasUnit;
        sh:class qudt:Unit;
        sh:minCount 1;
    ];
    sh:property [
        sh:path core:hasSimpleExpressionValue;
        sh:datatype xsd:string;
        sh:minCount 1;
    ].

# Shape for Average Calculation Process
ns:AverageCalculationShape a sh:NodeShape;
    sh:targetClass iof:ComputingProcess;
    sh:property [
        sh:path iof:hasInput;
        sh:class iof:MeasurementInformationContentEntity;
        sh:minCount 1;
    ];
    sh:property [
        sh:path iof:hasSpecifiedOutput;
        sh:class iof:ValueExpression;
        sh:minCount 1;
        sh:maxCount 1;
    ].

# Shape for ValueExpression
ns:ValueExpressionShape a sh:NodeShape;
    sh:targetClass iof:ValueExpression;
    sh:property [
        sh:path iof:isValueExpressionOfAtSomeTime;
        sh:class iof:MaterialEntity;
        sh:minCount 1;
    ];
    sh:property [
        sh:path core:hasSimpleExpressionValue;
        sh:datatype xsd:string;
        sh:minCount 1;
    ];
    sh:property [
        sh:path qudt:hasUnit;
        sh:class qudt:Unit;
        sh:minCount 1;
    ].
```

10.2. Unitless measurements

count, percentage. ratio

10.3. Multiple measurements of the same object at the same time

different measurement processes

Scenario Objective

This scenario demonstrates how to represent measurements conducted on different attributes of the same material entity within the context of the **BFO/IOF ontology**.

Key Points

- This pattern demonstrates how to capture and represent various attributes (e.g., physical, chemical, mechanical) measured on the same material entity.
- The pattern demonstrates the temporal coincidence of different measurements.
- The pattern highlights how measurements of various attributes can be traced back to the same material entity by using the IOF Core

10.3.1.

Use case: Measurement of temperature and pH during fermentation

This use case describes a scenario in which temperature and pH measurements are captured during a fermentation process. The temperature sensor records thermal conditions critical for maintaining optimal enzymatic reactions and microbial activity, as deviations can lead to inefficient fermentation or undesirable by-products. Concurrently, the pH sensor monitors the acidity or alkalinity of the fermenting mixture, providing essential insights into the chemical environment that directly affects enzyme functionality, product quality or overall process efficiency. This use case will focus on capturing a particular (discrete) measurement of temperature and pH. In practice, both attributes are monitored and captured continuously. For the details on how continuous measurements are captured the reader should look at the time-series measurement usecase.

10.4. Measurements of the same attribute by using different instruments

Scenario Objective

This scenario demonstrates how to represent measurements conducted on same attribute but with different measurement instruments within the context of the **BFO/IOF ontology**.

10.4.1.

Use case: Measurement of glucose by Raman spectroscopy and a glucose sensor during fermentation

In this biomanufacturing use case, glucose concentration—a key parameter for controlling cell culture metabolism—is measured using both Raman spectroscopy and a traditional electrochemical glucose sensor to ensure accuracy and process robustness. The Raman spectrometer provides real-time, non-invasive monitoring of glucose levels by analyzing molecular vibrational signatures, offering a continuous and label-free measurement method. In parallel, an electrochemical glucose sensor, based on enzymatic oxidation, provides direct, quantitative readings through periodic offline sampling. The Raman data enables trend analysis and predictive modeling, while the electrochemical sensor serves as a validation tool to confirm Raman-derived values and detect potential spectral interferences. This hybrid measurement approach enhances confidence in glucose monitoring, ensuring precise feed control strategies that optimize cell growth, productivity, and overall biomanufacturing efficiency.

10.5. Time-series measurement

(process characteristics)

10.6. uncertainty, range of values

if time permits

A Instructions to the authors

This document is a compendium of IOF Core patterns. Patterns are small, but repetitive fragments of a knowledge graph. The purpose of this work is to develop patterns that can be used by data modellers or developers in mapping legacy data using the IOF Core vocabulary. This pattern shows how to properly use the IOF Core in real-life data modelling practice.

We divide this document into a set of aspects, which are common topics for industrial data (maybe for other kinds of data too), e.g., person and agent, measurements, and time.

Many different scenarios can be addressed for each aspect, e.g., for Person and agent-related information, one scenario may address how to store a natural person's information and another scenario may address how the same person can be employed or assigned to different job positions or accounts. Within the aspect 'Time', scenarios can be how to express duration or clock time, how to link time to space, etc.

Each scenario will be developed by some author. However, the scenarios will be discussed in the group before they start developing them. Each scenario will be peer-reviewed, i.e., each author will review other authors' work.

Each scenario must include a description of the pattern in general and at least one use case demonstrating how the pattern can be practically used. There can be more than one use case but each use case must include an object diagram, example data set, and SPARQL INSERT query showing the data mapping. It is also highly recommended to provide a data validation mechanism using SPARQL queries and SHACL validator. A template is available in the appendix B which also contains instructions for each section.

Project setup

Once a scenario is allocated to you as an author, please follow the following steps to set up your work environment. Please make sure you have access to the Overleaf project called **IOF Core Pattern v2** and the GitHub repository of the same name.

1. In overleaf project, create a new folder for your scenario and make two more folders named 'image' and 'data' under it.
2. Create a new file called `<scenario-name>.tex`.
3. Copy the 'Scenario Template' from appendix B the new file. Edit the scenario name.
4. Sync the project with Github: Go to 'Github' in the 'Menu' and then press 'Push Overleaf changes to Github' button.
5. Clone the Git repository or pull in your local repository. You will find the image and data folder under your scenario folder. You can use these folders to develop diagrams, queries and data files or any other work that cannot be performed in Overleaf.

   Your Project setup is complete!

Diagramming

Each author is required to develop at least two diagrams, one **class-relation diagram** for 'General Pattern' and one 'object diagram' for every 'Use Case', for each scenario.

To harmonize the style of the diagram across all scenarios, the authors need to select one of the following two methods of drawing diagrams:

PlantUML using IOF visual notation library

PlantUML diagrams can be drafted using online editors, such as, PlantUML official server, PlantText, PlantUML Editor, and Pladitor. You can also set up PlantUML rendering in Visual Studio Code (VS Code) for live preview using PlantUML plugin. For setting up your own PlantUML server and use it in VS Code, please see Paregov's short instruction.

A standard library for PlantUML is developed (iofoundry/ontopuml) for ontology notations and IOF styling. The standard library provides a set of procedures which can be conveniently used to draw Class-relation and Object diagrams. Authors only need to import the library URL in their diagram as follows to be able to use the library in any PlantUML editor.

```
@startuml
!include https://raw.githubusercontent.com/iofoundry/ontopuml/refs/heads/main/iof.iuml
'...
' Write your code here!
'...
@enduml
```

Although the standard library provides full support to notations for every construct in OWL 2.0, only a few of them are required for drawing a Class-relation or Object diagram. A quick introduction to both of these diagrams and the minimum syntax for getting you started can be found here.

Draw.io using IOF visual notation library

A library with all basic patterns is available to be used with draw.io. Please follow the steps below to set up draw.io for diagramming.

1. Draw.io may be used as an online app or installed on desktop.
   (a) To use it online, simply go to Flowchart Maker & Online Diagram Software
   (b) Download and install draw.io from draw.io
2. Download "IOF Visual Notation" library from here.
3. In draw.io, go to File > New and select "Blank Diagram". The online app may ask you to choose a location to save the file.
4. Go to File > Open Library… (for the online app, choose Device
   (a) In the explorer select the IOF-visual-notation.drawio file.
   (b) The library with all basic patterns will be loaded on the left sidebar.
5. Start diagramming!

Naming and style convention

1. **Namespace:** Every label of ontological terms that are used in a diagram or text must accompany a namespace. Only a namespace prefix should be used, e.g., bfo:Occurrent. The prefixes should be common across all diagrams and text and should be declared in the front matter.
   (a) individuals may use namespace `ns1`, `ns2` etc. Multiple namespaces may be used to denote data coming from different sources. However, using a namespace for individuals is not obligatory.

Table A.1: Colors and fonts

Block type	Prefix	Background Color	Font Color	Border Color	Font style
Class	bfo	Golden Yellow #DFA702	White	Black	Bold
	iof	Dodger Blue or Vivid Blue #1e90ff	White	Black	Bold
	ns	Green #008000	White	Black	Bold
	others	Sunflower Yellow #DFA702	White	Black	Bold
	Without prefix	Blue #0000ff	White	Black	Bold
Individual	ns1	Mauve #76608A	White	White	Underlined

(b) Any new class that is not part of BFO and IOF must use namespace `ns`.

2. **Colors and font style**: The colours and font styles for diagrams are mentioned in table A.1.
3. See the table A.2 for choosing a proper case style of the label according to the element type.

Common name for different roles

Different intended audiences of this document need to be addressed in a unified manner. For example, the sentence: "...the developer/data modeller/architect may use the following SPARQL..." addresses some purported user of some SPARQL query. In the following table, a set of unified names is given for various roles that may be used in this document. This list will grow with time.

- **Ontologiest**:
- **Data modeller**:
- **Developer**:

Notes:

1. While referring to non-existing sections or chapters, mention a label prefixed by `?`. e.g.,

```
...which is detailed in \cref{?chapter-space-location}.
```

   Although this will cause the reference not to print correctly, these markings will help us correct them later.

2. All overflowing inline verbatim should be split with a next line, e.g.,

```
...while projecting on a single \texttt{bfo:SpatialReg\\ion}.
```

3. Use cases, particularly those involving data management, should not necessitate the axiomatization of a class. Instead, the system should facilitate the creation of new subclasses dynamically, as required, to accommodate evolving requirements and domain-specific needs.
4. It was decided on 12/23/2024 that scenarios will only show `INSERT DATA` SPARQL query, including at least the triples necessary to create the first row or multiple rows with some pattern.
5. The same INSERT DATA query, or preferably a fully mapped query, should be saved in a `*.q` type file under the data folder.
6. A workflow needs to be separately written to create SPARQL and SHACL uniformly.

Other tools

1. For generating LaTeX table, use https://www.tablesgenerator.com/. Please follow the format of the table as in table A.1.

Table A.2: Case styles

Type	Case style
Class (`owl:Class`)	The class name should be written in PascalCase (The first letter of every word is in uppercase and the spaces are removed). It is better to use the label (`rdf:Label`) for better comprehension. If the original `rdf:Label` has spaces, they should be removed to convert to PascalCase. Example: `bfo:GenericallyDependentContinuant`
Individual (`owl:Individual`)	The individual name should be in kebab-case (all letters are in lowercase, and spaces are replaced by a single hyphen). Avoid using abbreviated or shortened strings as names, e.g., `bn1`. Do not use the same name as the class it is a member of. Use the same name suffixed by indices for multiple individuals of the same class. Examples: `my-bicycle`, `house-of-joe`, `engine-1`, `engine-2`.
Object Property (`owl:ObjectProperty`)	The object property name should be written in camelCase (The first letters of every word except the first word are in uppercase, and spaces are removed). It is better to use the label (`rdf:Label`) of the property for better comprehension. If the original `rdf:Label` has spaces, they should be removed to convert to camelCase. Example: `bfo:hasContinuantPartAtSomeTime`. Additional symbols may be added before the name to represent qualifiers and cardinalities. Please see details in the next section.
Data Property (`rdf:DatatypeProperty`)	The data property name should be written in camelCase (The first letters of every word except the first word are in uppercase, and spaces are removed). It is better to use the label (`rdf:Label`) of the property for better comprehension. If the original `rdf:Label` has spaces, they should be removed to convert to camelCase. Example: `iof:hasSimpleExpressionValue`.

B Scenerio Template

This template should be used to create a scenario. Please copy to a dedicated folder under folder 'scenario'. All images should be referred from 'image' folder under the dedicated folder. For example, please see:
section 7.1

<Scenario title>

Short Phrase (e.g., Change of object's location over time)

Created by: Arkopaul Sarkar
Modified by: Arkopaul Sarkar

Scenario Objective

Describe the general purpose of representing the pattern (e.g., demonstrating how you can depict measurements with IOF core).

General Pattern Description

Provide a figure containing the pattern independent of the use case (e.g., just measurements in general as opposed to measuring a particular thing such as pH or length).
This diagram is generally a simple class-relation diagram. Please see examples of class-relation diagram here.

Describe the general pattern, including any background why such types and relations are used. Also mention any alternative or shortcuts. Please provide a background of the types and relations in the context of IOF (please prefer logical arguments to didactic arguments).

Use Case: <Use-Case title>

Describe a concrete use case for the pattern. (Multiple use cases can be described for one pattern. If multiple, please use distinct use case titles and repeat all subsubsections for each use case.)

Use-Case Pattern Description

Provide a diagram which matches the use-case pattern description.
This diagram is generally an object diagram. Please see examples of object diagram here.

Describe how the general pattern is applied to the use case. It is important to highlight within the description how the use-case-specific concepts align with the general pattern (e.g., SubClassOf Class for general pattern).

Use-Case Example Data

Provide a description of the data set used to demonstrate the use case pattern (e.g., format as JSON, CSV, XML, column/attribute/node name and their description, relation between datasets if multiple are used). Use not less than 3 and not more than 7 records/transactions.

Data Mapping

Describe how the data was mapped to RDF. Provide an INSERT DATA/ INSERT SPARQL for mapping. Use INSERT only when the use case example data needs further manipulation.
For INSERT DATA SPARQL, use only 2/3 records/transactions with named class and individuals.
For INSERT SPARQL, declare column names by '{ }' to the variable. For INSERT query, For both, do not use blank nodes.

Data Validation

Data validation can be performed in two ways: accessing interesting facts using SPARQL or validating whether the entire data conforms to the ontology using SHACL. It is preferable to provide both.
Provide the SPARQL query in the code block along with the result of the query.

C About the LaTex Template

This template aims to simplify and improve the (Xe)LaTeX report/thesis template by Delft University of Technology with the following three main design principles:

- **Simplicity First:** A class file that has been reduced by nearly 70% to simplify customization;
- **Effortless:** A careful selection of common packages to get started immediately;
- **Complete:** Ready-to-go when it comes to the document and file structure.

This template works with pdfLaTeX, XeLaTeX and LuaLaTeX. In order to adhere to the TU Delft house style, either XeLaTeX or LuaLaTeX is required, as it supports TrueType and OpenType fonts. BibLaTeX is used for the bibliography with as backend biber. Please visit https://dzwaneveld.github.io/report/ for the full documentation.

Documentation (Abridged)

As a report/thesis is generally a substantial document, the chapters and appendices have been separated into different files and folders for convenience. The folders are based on the three parts in the document: the frontmatter, mainmatter and appendix. All files are inserted in the main file, `report.tex`, using the `\input{filename}` command. The document class, which can be found in `tudelft-report.cls`, is based on the book class.

The template will automatically generate a cover when the `\makecover` command is used. The title, subtitle and author will also be present on the title page. To give greater flexibility over the title page, the layout is specified in `title-report.tex`. A title page for theses is also available: `title-thesis.tex`. Change the corresponding `\input{...}` command in the main file to switch.

The bibliography has been set up in `report.tex` to allow for easy customization. It is included in the table of contents and renamed to 'References' using the `heading=bibintoc` and `title=References` options of the `\printbibliography` command respectively. If you would like to use a different .bib file, change the command `\addbibresourcereport.bib` accordingly.

→ *Visit https://dzwaneveld.github.io/report/ for the full documentation.*

License