Faceted Data Class Design Pattern

Table of Contents

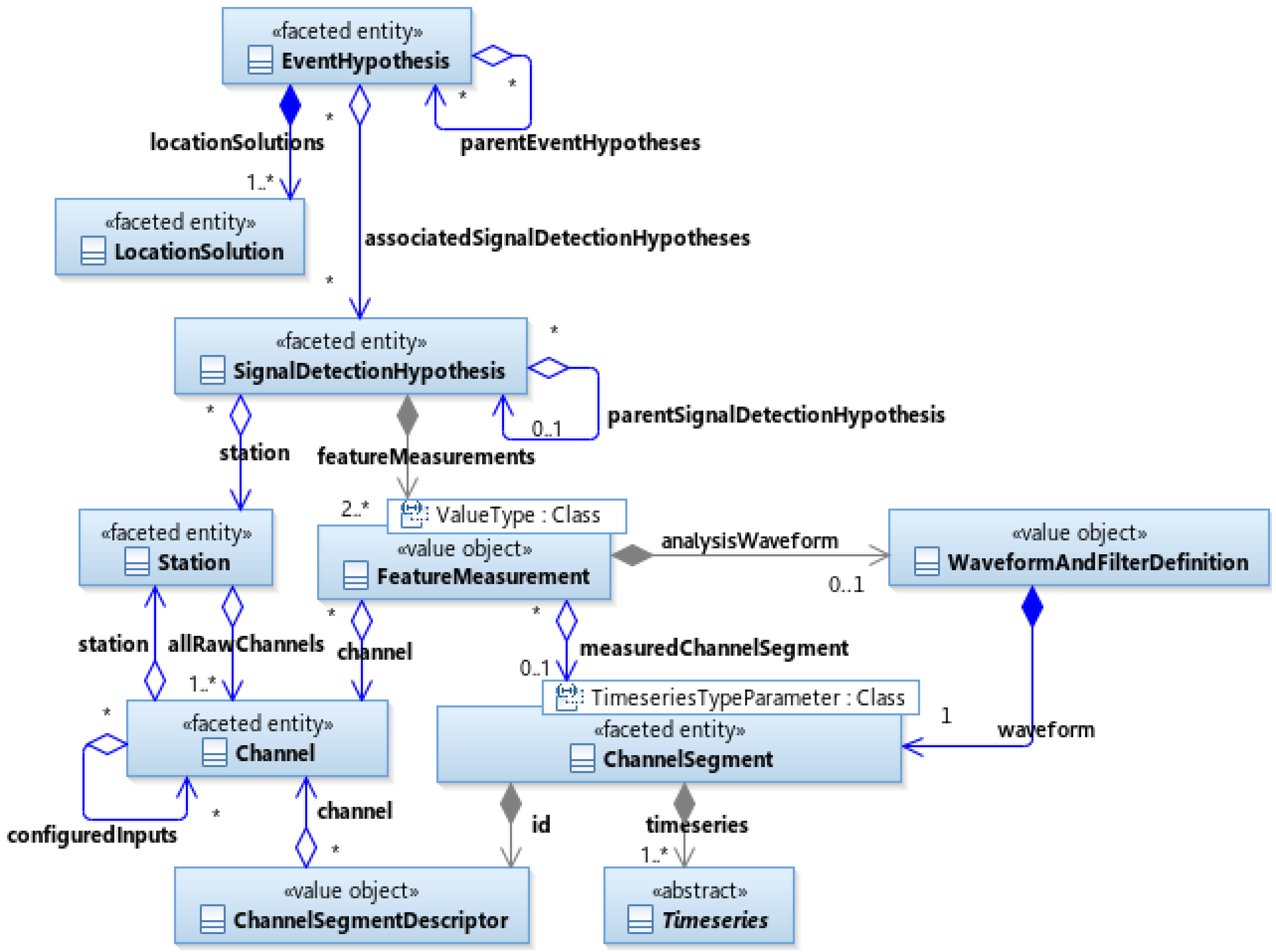
List of Figures

List of Tables

Motivation

The GMS COI data model is highly interconnected, with data objects often including a large transitive closure of associated objects, raw data, and provenance information. The figure below shows an example: a fully populated **EventHypothesis** object may contain a large collection of associated data. Among other things, an **EventHypothesis** includes an associated **SignalDetectionHypothesis** collection, each of which contains a **FeatureMeasurement** collection, each of which is associated with a measured **ChannelSegment**. A typical measured **ChannelSegment** is derived from an input waveform **ChannelSegment** collection produced by **Channel** objects organized into groups for a given **Station**.

Figure 1: Faceting in the closure of a COI **EventHypothesis**


«faceted entity» EventHypothesis
locationSolutions
1..*
*
parentEventHypotheses
«faceted entity» LocationSolution
associatedSignalDetectionHypotheses
«faceted entity» SignalDetectionHypothesis
parentSignalDetectionHypothesis
0..1
station
featureMeasurements
«faceted entity» Station
2..*
ValueType : Class
«value object» FeatureMeasurement
analysisWaveform
0..1
«value object» WaveformAndFilterDefinition
station
allRawChannels
1..*
channel
measuredChannelSegment
0..1
TimeseriesTypeParameter : Class
1
waveform
«faceted entity» Channel
«faceted entity» ChannelSegment
configuredInputs
channel
id
timeseries
«value object» ChannelSegmentDescriptor
«abstract» Timeseries

Faceting allows clients to select a subset of the closure of associated information for a given data product, thus avoiding the performance costs and processing complexity of retrieving unneeded data. Continuing with the **EventHypothesis** example, an application may need access to **EventHypothesis** content without the accompanying **SignalDetectionHypothesis** collection; faceting provides the client the ability to indicate within an access request that the **EventHypothesis** response should exclude the **SignalDetection** contents. FacetingDefinition objects define the faceting rules. In the GMS pattern, faceting applies to the point of association between objects. Specifically, it provides a means of substituting object identifiers in place of the objects themselves. In the figure, each blue arrow indicates an **EventHypothesis** class attribute which includes another object via aggregation or composition. Through faceting, each of these included objects may be substituted with its identifier.

Faceted Data Classes

The GMS COI data model indicates faceted classes using the stereotypes <<*faceted entity*>> and <<*faceted value object*>>. These stereotypes indicate the class has two facets:

1. Object identifier facet
2. Data contents facet

Instances of the faceted class always populate the identifier facet but optionally populate the data contents facet. An object populated with both the identifier and data contents facets may be referred to as "fully populated".

Note

An instance of a faceted class may include within its data contents an instance of another faceted class. Within the populated data contents of an instance of the first faceted class, the instance of the second class may be populated with only its identifier facet or with both its identifier and data contents facets. In both cases, the first object is considered "fully populated" since its identifier and data contents facets are both populated. Returning to the **EventHypothesis** example, a "fully populated" **EventHypothesis** object may include a collection of **SignalDetectionHypothesis** objects populated as identifier-only or fully populated instances.

Serialized representations of faceted objects include only the populated facets. The serialized contents match the structure defined in the COI data model and do not reveal any implementation details of the faceting pattern.

Implementation Pattern

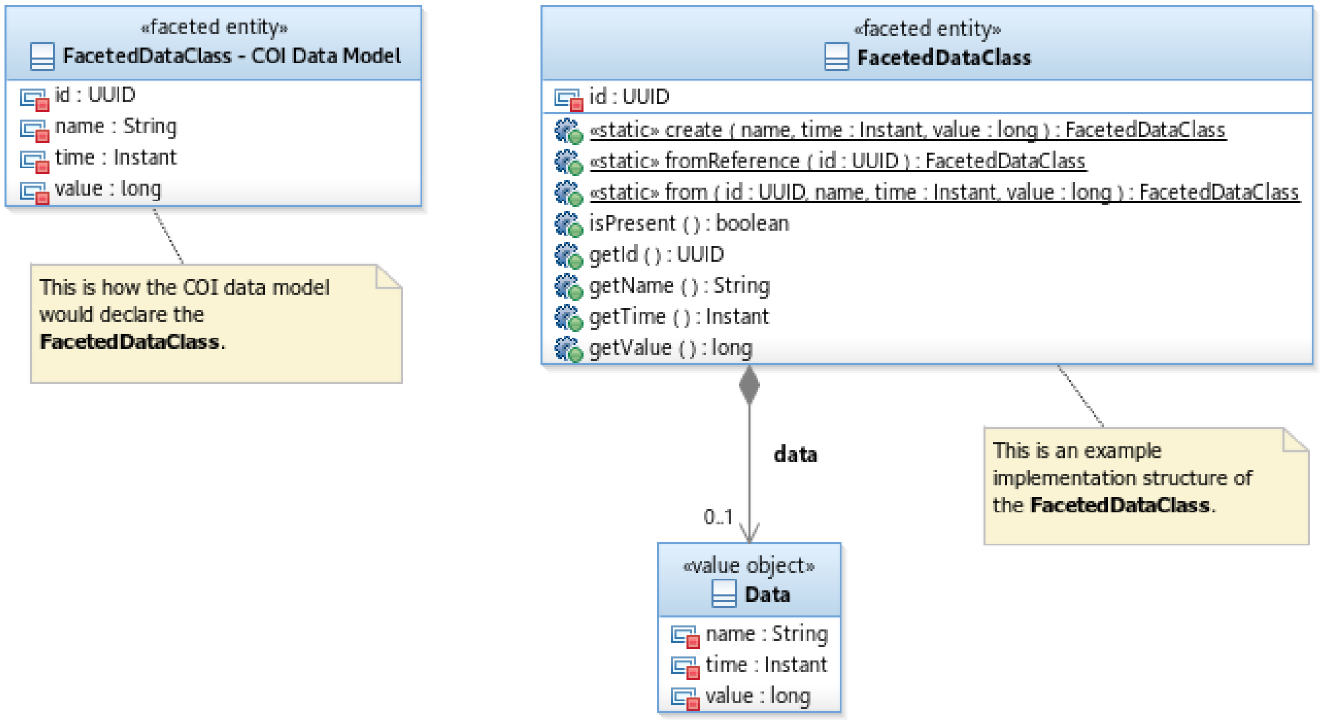
This section describes an implementation pattern for a faceted data class. The faceted data class implementation includes an attribute for each facet, typically:

1. An identifier attribute.
2. A **Data** class attribute which includes all of the data contents.

As shown in the figure below, the COI data model definition for a faceted class typically will not include the **Data** class. Instead, it will declare the faceted class to include separate attributes for each element of its data contents. The **Data** class is an implementation detail encapsulated within the faceted class implementation. This is a design decision chosen to prevent the faceted class implementation from having to include separate optional attributes for each element of their data contents. For example, if the data contents include three attributes (e.g. *name*, *time*, and *value* in the figure below), then none of the three attributes will be populated in an identifier-only object while all of them will be populated in a fully populated object. Since there is an instantiation where the attributes are not populated, they must be optional with a constraint that either they are all populated or none of them are populated. Including the three attributes in the **Data** class allows the faceted class to include an optional **Data** attribute, while the **Data** class' individual attributes are not optional. This allows the three data attributes to be easily populated or unpopulated as a group. Because it is an implementation detail, the **Data** class and its structure are not revealed in serialized representations of the faceted data class.

The figure below shows the example **FacetedDataClass** with this structure.

Figure 2: **FacetedDataClass** static structure


«faceted entity»
FacetedDataClass - COI Data Model
id : UUID
name : String
time : Instant
value : long
This is how the COI data model would declare the FacetedDataClass.
«faceted entity»
FacetedDataClass
id : UUID
«static» create (name, time : Instant, value : long) : FacetedDataClass
«static» fromReference (id : UUID) : FacetedDataClass
«static» from (id : UUID, name, time : Instant, value : long) : FacetedDataClass
isPresent ( ) : boolean
getId ( ) : UUID
getName ( ) : String
getTime ( ) : Instant
getValue ( ) : long
data
0..1
This is an example implementation structure of the FacetedDataClass.
«value object»
Data
name : String
time : Instant
value : long

FacetedDataClass is an example class implemented with the faceting pattern. It includes an identifier facet (the *id* attribute) and a data contents facet (the combination of the *name, time,* and *value* attributes). In the COI data model, the **FacetedDataClass** class would have four attributes (*id*, *name*, *time*, and *v alue*). Following the GMS faceting implementation pattern, the implemented class has two attributes: the required *id* and the optionally populated **Data** instance.

The existence of the **Data** class within the **FacetedDataClass** implementation is hidden from serialized representations of **FacetedDataClass** objects. In particular, **FacetedDataClass** objects can be serialized in two ways:

1. An identifier-only instance is serialized to JSON as:

```
{"id":"cd344524-9b81-4c6c-9ab4-721436a05d96"}
```

2. A fully populated instance is serialized to JSON as:

```
{"id":"cd344524-9b81-4c6c-9ab4-721436a05d96", "name":"Faceted Class - fully populated", "time":"2020-04-
01T20:36:14.822267Z", "value":1234}
```

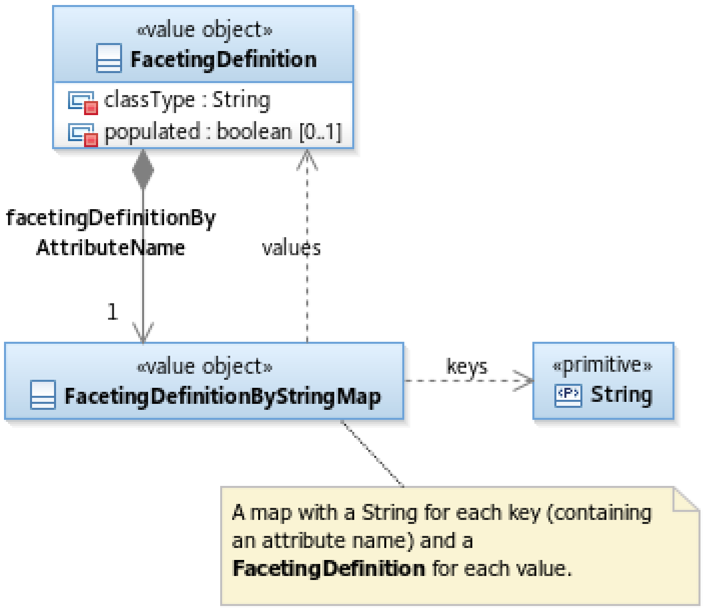
FacetedDataClass deserializers must correctly construct instances from both serializations.

Data Access Queries for Faceted Data Classes

Data access operations which construct objects containing attributes that are instances of faceted data classes must know how to populate the faceted objects. In some cases, the operation will declare the specific population it uses. Returning to the **EventHypothesis** example, an operation may declare it returns **EventHypothesis** objects with aggregated instances of the faceted **SignalDetectionHypothesis** class populated with only their identifier facets. In other cases, data access operations will be required to create the faceted objects with any of their possible populations. When this occurs, the data access operation accepts a **FacetingDefinition** parameter describing how it must populate each attribute that is an instance of a faceted data class. In the **EventH ypothesis** example, the **FacetingDefinition** object defines how the elements in the **EventHypothesis** object's associated **SignalDetectionHypothesis** ca lculation should be populated.

Faceting Definition

Figure 3: **FacetingDefinition** static structure


«value object»
FacetingDefinition
classType : String
populated : boolean [0..1]
facetingDefinitionBy AttributeName
values
1
«value object»
FacetingDefinitionByStringMap
keys
«primitive»
String
A map with a String for each key (containing an attribute name) and a FacetingDefinition for each value.

FacetingDefinition objects define how to populate a data object which has attributes that are instances of faceted classes. A **FacetingDefinition** object aggregates other **FacetingDefinition** objects in its *facetingDefinitionByAttributeName* map, allowing it to declare how to populate arbitrarily deep nestings of faceted attributes within faceted attributes.

FacetingDefinition has the following attributes:

Table 1: **FacetingDefinition** attribute descriptions

Attribute	Data Type	Units	Range	Populated	Description
classType	String	N/A	N/A	Always	A string corresponding to the name of a class with attributes that are instances of faceted classes. Must uniquely identify a single class. Generally corresponds to the name of a class defined in the COI data model.
facetingDefin itionByAttrib uteName	Map with a String for each key and a **Faceting Definition** object for each value	N/A	N/A	Always	A map with each entry containing: 1. A String key containing the name of an attribute within the *classType* class. 2. A **FacetingDefinition** value defining how the attribute should be populated. If the map does not contain an entry for an attribute within the *classType* class which uses a faceted data class for its type, then the attribute will be populated according to its default defined in the COI data model for the *classType* class or an alternate default defined by a specific COI data access operation. If the *classType* class includes an attribute with a type of a non-faceted class which contains faceted attributes, then the map may contain an entry for the attribute of the non-faceted class. The associated **FacetingDefinition** leaves the attribute *populated* unpopulated and each of its *facetingDefinitionByAttributeName* map entries describes how to populate one of the non-faceted class' attributes with a type of a faceted class. This pattern may repeat more than once (e.g. the non-faceted **FeatureMeasurement** attribute *analysisWaveform* has type **WaveformAndFilterDefinition**; **WaveformAndF ilterDefinition** is not faceted but includes an attribute with type **ChannelSegment**, which is faceted. A **FacetingDefinition** object for the **SignalDetectionHypothesis** class may include a *facetingDefinitionByAttributeName* map with nested entries describing how the **ChannelSegment** objects within the **FeatureMeasurement** objects' **WaveformAndFilterDefin ition** attributes should be populated.)

populated	boolean	N/A	N/A	Optional	Indicates how the attributes of the *classType* objects populated based on this **FacetingDefinition** should be populated. The boolean values have the following meanings: **1.** FALSE: the *classType* object should be instantiated as an identifier-only instance (i.e. only its identifier facet should be populated). The *facetingDefinitionByAttributeName* map is not used and should be empty. **2.** TRUE: the *classType* object should be instantiated as a fully populated instance (i.e. both its identifier and data contents facets should be populated). The *facetingDefinitionByAttributeName* entries define how to populate the object's data contents attributes which are instances of faceted classes (see the *facetingDefinitionByAttributeName* description for details). Unpopulated when this **FacetingDefinition** object describes how to populate the faceted attributes of a non-faceted class. Always populated when this **FacetingDefinition** object describes how to populate a faceted class. Note This attribute is optional because an instance of a non-faceted class must be populated. Alternatively, the attribute could have been defined as always populated with a constraint that its value must be TRUE for a **FacetingDefinition** object describing how to populate the faceted attributes of a non-faceted class. However, that approach gives the illusion the **FacetingDefinition** could have defined the object to be unpopulated. Since that is not possible, making the attribute optionally populated better reveals intent.

Faceting Definition Example - Event Hypothesis

Considering only the contents shown in the above *Figure - Faceting in the closure of a COI* ***EventHypothesis***, the **EventHypothesis** and associated classes have the following structure:

Table 2: **FacetingDefinition** example - attributes in the **EventHypothesis** closure

Attribute	Data Type	Data Type Is Faceted
	EventHypothesis	
associatedSignalDetectionHypotheses	**SignalDetectionHypothesis**[*]	Yes (individual elements)
locationSolutions	**LocationSolution**[1..*]	Yes (individual elements)
parentEventHypotheses	**EventHypothesis**[*]	Yes (individual elements)
	SignalDetectionHypothesis	
featureMeasurements	**FeatureMeasurement**[2..*]	No
parentSignalDetectionHypothesis	**SignalDetectionHypothesis** (optional)	Yes
station	**Station**	Yes
	FeatureMeasurement	
analysisWaveform	**WaveformAndFilterDefinition**	No
channel	**Channel**	Yes
measuredChannelSegment	**ChannelSegment** (optional)	Yes
	ChannelSegment	
id	**ChannelSegmentDescriptor**	No
timeseries	**Timeseries**[1..*]	No
	ChannelSegmentDescriptor	
channel	**Channel**	Yes
	WaveformAndFilterDefinition	
waveform	**ChannelSegment**	Yes
	Channel	
configuredInputs	**Channel**[*]	Yes (individual elements)
station	**Station**	Yes
	Station	
allRawChannels	**Channel**[1..*]	Yes (individual elements)

A variety of **FacetingDefinition** objects are possible, with each defining a different way a data access operation must populate the data contents of the **EventHypothesis** objects it creates. The two simplest **FacetingDefinition** objects define an **EventHypothesis** with only its identifier facet populated:

```
{"classType":"EventHypothesis", "populated":false, "facetingDefinitionByAttributeName":{}}
```

and an **EventHypothesis** with both its identifier and data contents facets populated, but with attributes of faceted classes within the data contents populated according to their defined defaults:

```
{"classType":"EventHypothesis", "populated":true, "facetingDefinitionByAttributeName":{}}
```

Within its data contents, the **EventHypothesis** class includes several attributes with types that are faceted data classes. A **FacetingDefinition** may define how each of these attributes should be populated. When an attribute is a collection, the **FacetingDefinition** defines a single population used for every element. For example, this **FacetingDefinition** defines a fully populated **EventHypothesis** object, with the elements in its *associatedSignalDetectionHypotheses* and *locationSolutions* collections populated according to their defined defaults, and the elements in its *parentEventHypotheses* collection populated as identifier-only instances:

```
{"classType":"EventHypothesis", "populated":true, "facetingDefinitionByAttributeName":{
        "associatedSignalDetectionHypotheses" : {"classType":"SignalDetectionHypothesis", "populated":true, 
"facetingDefinitionByAttributeName":{}},
         "locationSolutions" : {"classType":"LocationSolution", "populated":true, 
"facetingDefinitionByAttributeName":{}},
          "parentEventHypotheses" : {"classType":"EventHypothesis", "populated":false, 
"facetingDefinitionByAttributeName":{}}
}}
```

The concept repeats for each attribute which is an instance of a class containing attributes that are instances of faceted classes. For example, a **FacetingDefinition** may define an **EventHypothesis** with populated **SignalDetectionHypothesis** objects in its *associatedSignalDetectionHypotheses* collection; populated **ChannelSegment** objects for each *measuredChannelSegment* in each **SignalDetectionHypothesis** object's **FeatureMeasurement** collection; populated **Channel** objects in each measured **ChannelSegment** object's identifier (**ChannelSegmentDescriptor** objects); populated **ChannelSegment** objects in each **FeatureMeasurement** object's *analysisWaveform*; and other attributes populated according to their defined defaults:

```
{"classType":"EventHypothesis", "populated":true, "facetingDefinitionByAttributeName":{
        "associatedSignalDetectionHypotheses" : {"classType":"SignalDetectionHypothesis", "populated":true, 
"facetingDefinitionByAttributeName":{
                "featureMeasurements" : {"classType":"FeatureMeasurement", "facetingDefinitionByAttributeName":{
                         "measuredChannelSegment" : {"classType":"ChannelSegment", "populated":true, 
"facetingDefinitionByAttributeName":{
                                  "id" : {"classType":"ChannelSegmentDescriptor", 
"facetingDefinitionByAttributeName":{
                                        "channel" : {"classType":"Channel", "populated":true, 
"facetingDefinitionByAttributeName":{}}
                                }}
                           }},
                          "analysisWaveform" : {"classType":"WaveformAndFilterDefinition", 
"facetingDefinitionByAttributeName":{
                                  "waveform" : {"classType":"ChannelSegment", "populated":true, 
"facetingDefinitionByAttributeName":{}}
                        }}
                }}
        }}
}}
```

In some cases, a class is not a faceted class but it includes attributes with types that are faceted classes. When this occurs, the *facetingDefinitionByAttributeName* contains an entry for each non-faceted attribute. The corresponding **FacetingDefinition** must not populate attribute *populated*, and its *facetingDefinitionByAttributeName* defines the population approaches for the class' attributes with types of faceted classes. For example, the **SignalDetectionHypothesis** class includes a **FeatureMeasurement** collection. **FeatureMeasurement** is not faceted, but it includes a *measuredChannelSegment* attribute of type **ChannelSegment**, and the **ChannelSegment** class is faceted. The **SignalDetectionHypothesis** object's *facetingDefinitionByAttributeName* map contains the entry with key "featureMeasurements" mapped to a **FacetingDefinition** defining how faceted attributes of the **FeatureMeasurement** objects should be populated. This **FacetingDefinition** leaves the attribute *populated* unpopulated. That **FacetingDefinition** includes entries describing how the *measuredChannelSegment* should be populated. This concept repeats for the **ChannelSegment** attribute *id* with type of the non-faceted **ChannelSegmentDescriptor** class, which contains the attribute *channel* with type of the faceted class **Channel**. This concept also repeats for the **FeatureMeasurement** attribute *analysisWaveform* with type of the non-faceted **WaveformAndFilterDefinition** class, which contains the attribute *waveform* with type of the faceted class **ChannelSegment**.

Implementation Concept

The **FacetingDefinition** class' nested structure provides clients the ability to use a single data access operation to request an object with data contents populated in a variety of ways. This flexibility complicates the data access operation implementations compared to data access operations which do not populate faceted objects. This section describes a potential implementation pattern.

A **FacetingDefinition** includes two pieces of information:

1. The *populated* flag indicating whether an instance of a faceted class should be populated.
2. The *facetingDefinitionByAttributeName* map containing nested **FacetingDefinition** objects describing how attributes with types of faceted classes should be populated.

Handling each of these individually is straightforward. First, consider the options for the *populated* flag:

Table 3: Processing each possibility of the **FacetingDefinition** attribute *populated*

	FacetingDefinition attribute *populated* value	
Object's current population	**FALSE**	**TRUE**
Identifier only	Object already populated as requested.	Use a "find by identifier" query to load the object.
Fully populated	Replace populated object with identifier only object.	Object already populated as requested.

Each case is trivial to handle. The most complicated case only requires using a standard query to find an object by its identifier.

Second, consider the nested **FacetingDefinition** objects in the *facetingDefinitionByAttributeName* map. Each entry is a **FacetingDefinition** which can be processed using the logic described in the table above. The only additional behavior is to match the attribute names in the map's keys with specific attributes within a class so the correct object population behavior can be accessed for each attribute. One way to provide this logic is through a utility operation for each COI class which is responsible for populating the attributes within that class whose types are a COI class or include a COI class (e.g. as part of a collection). The section below describes example behavior for one of these utility operations. The behavior is straightforward and is mostly boilerplate.

Faceting Utility Example - Event Hypothesis

Returning to the **EventHypothesis** example, a utility operation to populate an **EventHypothesis** object could have the following behavior:

Note

This example operation behavior only includes the **EventHypothesis** attributes included in the running example discussed in this architecture description. The **EventHypothesis** COI data model includes additional attributes and an actual *populateFacets(eventHypothesis : EventHypothesis, facetingDefinition : FacetingDefinition) : EventHypothesis* operation implementation would include behavior to populate those attributes.

populateFacets(eventHypothesis : EventHypothesis, facetingDefinition : FacetingDefinition) : EventHypothesis - this operation populates the attributes of the provided **EventHypothesis** object based on the provided **FacetingDefinition**. This operation performs the following:

1. Verifies the provided **FacetingDefinition** defines how an **EventHypothesis** object should be populated: checks the **FacetingDefinition** object's *classType* attribute is assigned to the value "EventHypothesis". If the attribute contains any other value, this operation fails with an error response.
2. Handles each possible combination of the provided **FacetingDefinition** object's *populated* attribute's value and the provided **EventHypothesis** object's current population:
    a. Case 1: *populated* is FALSE and the provided **EventHypothesis** is an identifier-only instance:
        i. Returns the provided **EventHypothesis** object.
    b. Case 2: *populated* is FALSE and the provided **EventHypothesis** is a fully populated instance:
        i. Creates and returns a new identifier-only **EventHypothesis** object containing the provided **EventHypothesis** object's identifier.
    c. Case 3: *populated* is TRUE and the provided **EventHypothesis** is an identifier-only instance:
        i. Uses a *findEventHypothesisById(UUID) : EventHypothesis* query operation, providing the provided **EventHypothesis** object's identifier as a parameter, to load a fully populated instance of the **EventHypothesis**.
        ii. Processes the provided **FacetingDefinition** object's *facetingDefinitionByAttributeName* map as described below.
    d. Case 3: *populated* is TRUE and the provided **EventHypothesis** is a fully populated instance:
        i. Processes the provided **FacetingDefinition** object's *facetingDefinitionByAttributeName* map as described below.
3. Populates the **EventHypothesis** object's faceted attributes according to the provided **FacetingDefinition** object's *facetingDefinitionByAttributeName* map:

    a. Populates the **SignalDetection** objects within the **EventHypothesis** object's *associatedSignalDetectionHypotheses* collection:
        i. Extracts from the *facetingDefinitionByAttributeName* map the **FacetingDefinition** associated to the key "associatedSignalDetectionHypotheses".
        ii. Creates an empty **SignalDetectionHypothesis** collection.
        iii. For each **SignalDetectionHypothesis** in the *associatedSignalDetectionHypotheses* collection:
            1. Calls the operation *populateFacets(SignalDetectionHypothesis, FacetingDefinition) : SignalDetectionHypothesis,* providing the **SignalDetection** object and the **FacetingDefinition** extracted from the *facetingDefinitionByAttributeName* map as parameters.
                a. The operation responds with a **SignalDetectionHypothesis** object.
            2. Adds the **SignalDetectionHypothesis** object to the **SignalDetectionHypothesis** collection.
    b. Populates the **LocationSolution** objects within the **EventHypothesis** object's *locationSolutions* collection:
        i. Implements behavior equivalent to that described above for the *associatedSignalDetectionHypotheses* collection, using the operation *populateFacets(LocationSolution, FacetingDefinition) : LocationSolution* to populate the **LocationSolution** objects.
    c. Populates the **EventHypothesis** objects within the **EventHypothesis** object's *parentEventHypotheses* collection:
        i. Implements behavior equivalent to that described above for the *associatedSignalDetectionHypotheses* collection, using the operation *populateFacets(EventHypothesis, FacetingDefinition) : EventHypothesis* to populate the **EventHypothesis** objects (i.e. a recursive call to this operation).
    d. Updates the **EventHypothesis** object with the new *associatedSignalDetectionHypotheses, locationSolutions,* and *parentEventHypothese* collections.
4. Returns the updated **EventHypothesis** object.

Notes

1. None

Change History

1. PI27
    a. 02/2024: Refactoring to support Data Fabric descriptions
        i. Clarified description of the faceting concept.
        ii. Removed description of the "Faceting Utility" concept.
        iii. Updated **FacetingDefinition** to make attribute *populated* optional. This replaces the "dot notation" implementation concept for referencing attributes within attributes within **FacetingDefinition** attribute *facetingDefinitionByAttributeName*.
2. PI14 - initial description

Open Issues

1. None