interactive instruments

Software für Wissenschaft und Technik

Property Stereotype for Metadata

Johannes Echterhoff (interactive instruments GmbH)

Version:	1.0
Status:	Final
Date:	Nov 29, 2019

History

Version	Status	Date	Author(s)	Description
1.0	Final	Nov, 29, 2019	Johannes Echter-hoff	Finalize documentation

Contents

Figures

Tables

Listings

1 Terms and Definitions

1.1 Abbreviated Terms

DDL	Data Definition Language
FES	Filter Encoding Specification
GEOINT	Geospatial Intelligence
GFM	General Feature Model
GML	Geography Markup Language
GML-SF	GML Simple Features
IETF	Internet Engineering Task Force
INSPIRE	Infrastructure for Spatial Information in Europe
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
NAS	NSG Application Schema
NEO	NSG Enterprise Ontology
NSG	U.S. National System for Geospatial Intelligence
O&M	Observations and Measurements
OCL	Object Constraint Language
OGC	Open Geospatial Consortium
OWL	Web Ontology Language
RDF	Resource Description Framework
RFC	Request for Comments
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
SVN	Subversion
UGAS	UML to GML Application Schema
UML	Unified Modeling Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XPath	XML Path Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformation

1.2 Definitions

1.2.1 Application schema

Conceptual schema (1.2.5) for data required by one or more applications.

[SOURCE: ISO 19101-1:2014, 4.1.2]

1.2.2 Class

Unified Modeling Language (UML) - Description of a set of objects that share the same specifications of features (attributes and operations), constraints, and semantics.

[SOURCE: UML 2.4.1, modified – slightly reworded and content in parentheses added as clarification]

1.2.3 Conceptual formalism

Set of modeling concepts used to describe a *conceptual model* (1.2.4)

[SOURCE: ISO 19101-1:2014, 4.1.4]

Example: UML meta model, EXPRESS meta model.

Note: One conceptual formalism can be expressed in several *conceptual schema languages* (1.2.6).

1.2.4 Conceptual model

Model (1.2.7) that defines concepts of a *universe of discourse* (1.2.10)

[SOURCE: ISO 19101-1:2014, 4.1.5, modified – Notes have been added.]

Note: It is often the case that the concepts defined by a *conceptual model* are formed into groups, which are meaningful to one or more application domains. These groups, being subsets of the whole *model*, are themselves *conceptual models*. A group typically consists of a set of concepts, but may also consist of only a single concept.

Note: Conceptual models are formally described by c*onceptual schemas* (1.2.5). A *conceptual model*, more specifically its subsets, can thus be described by multiple *conceptual schemas*. With UML being used as *conceptual schema language* (1.2.6), a UML model as a whole formally represents a conceptual model. The UML model may also be comprised of multiple UML packages, each of which may formally represent a *conceptual schema*.

1.2.5 Conceptual schema

Formal description of a *conceptual model* (1.2.4)

[SOURCE: ISO 19101-1:2014, 4.1.6, modified – Note has been added.]

Note: Conceptual schemas for geographic information are typically modeled using UML as the *conceptual schema language* (1.2.6). UML defines the concept of *stereotype* (1.2.9), which is essential to this report. ISO 19103 and ISO 19109, for example, define stereotypes <<CodeList>> and <<FeatureType>> for classes, which give these classes specific meaning.

1.2.6 Conceptual schema language

Formal language based on a *conceptual formalism* (1.2.3) for the purpose of representing a *conceptual schema* (1.2.5)

EXAMPLES UML, EXPRESS, IDEF1X

NOTE A conceptual schema language may be lexical or graphical. Several conceptual schema languages can be based on the same conceptual formalism.

[SOURCE: ISO 19101-1:2014, 4.1.7]

1.2.7 Model

Abstraction of some aspects of reality

[SOURCE: ISO 19109:2015, 4.15, modified – Note has been added.]

Note: The term *model* is often used as a shorthand-notation for the term *conceptual model* (1.2.4).

1.2.8 Schema

Formal description of a *model* (1.2.7)

[SOURCE: ISO 19101-1:2014, 4.1.34, modified – Note has been added.]

Note: The term *schema* is often used as a shorthand-notation for the term *conceptual schema* (1.2.5).

1.2.9 Stereotype

<UML> extension of an existing metaclass that enables the use of platform or domain specific terminology or notation in place of, or in addition to, the ones used for the extended metaclass

[SOURCE: UML 2.4.1]

1.2.10 Universe of discourse

View of the real or hypothetical world that includes everything of interest

[SOURCE: ISO 19101-1:2014, 4.1.38]

2 Property Stereotype for Metadata in Application Schemas

NOTE: This document was created during the OGC UML-to-GML Application Schema (UGAS) Pilot (UGAS-2019), an initiative within the OGC Innovation Program.

2.1 Current NAS Modeling Approach

The NSG Application Schema (NAS) is based upon the General Feature Model (GFM), which is defined by ISO 19109:2015. As such, the NAS defines a number of feature types, which represent objects that are important to geospatial intelligence (GEOINT) applications. These feature types typically have a number of properties, *i.e*. attributes (whose values do not have identity) and association roles (whose values do have identity and can therefore be shared).

The NAS supports metadata for attribute values, which can be used to (for example):

- Interpret a value, *e.g*., numeric unit-of-measure, numeric precision.
- Relate a value to other values, *e.g*., reference datum (numeric zero point).
- Specify the quality of a value, *e.g*., accuracy.
- Specify the provenance of a value, *e.g*., source or process by which, and/or conditions under which, the value was determined, estimated, or predicted.
- Characterize the absence of an expected value, *e.g*., inapplicable in the context in which the data collection took place.

Note: Metadata for values of association roles is currently not supported by the NAS.

The NAS uses specific Meta- and Reason-classes to incorporate this metadata. Figure 1 provides an example.

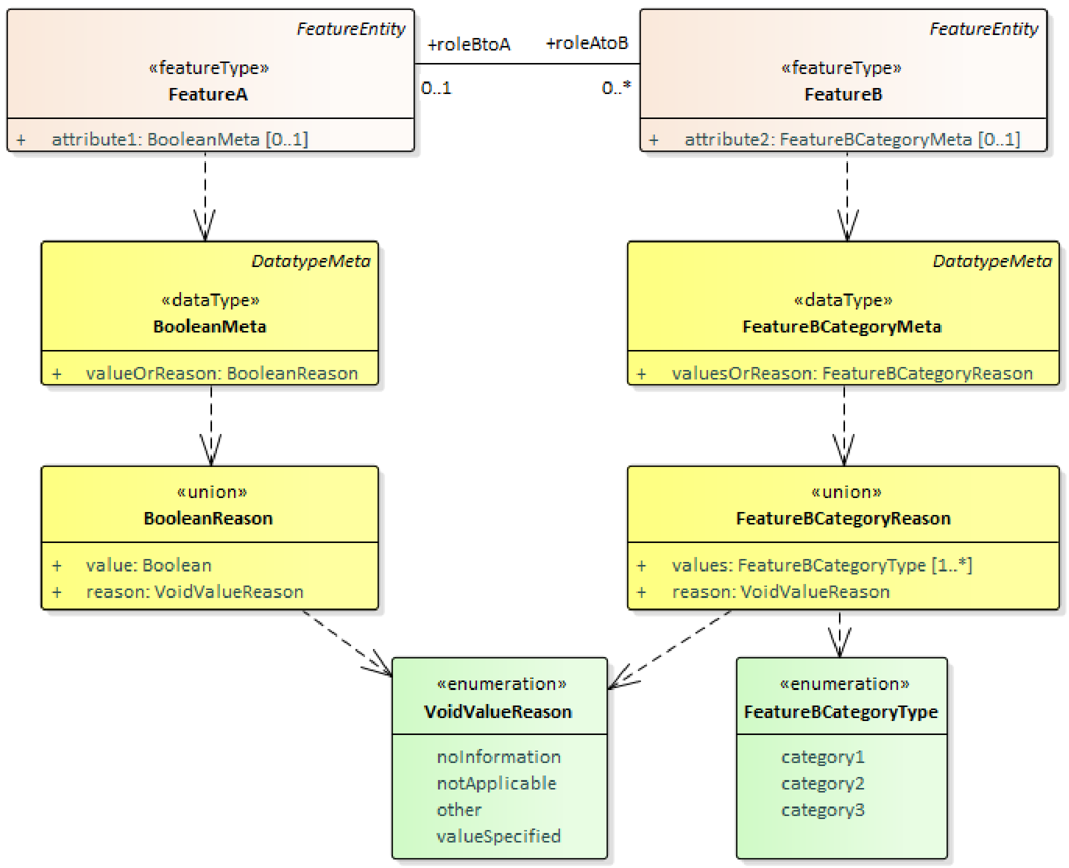


Figure 1 – Current approach for modeling property metadata in the NAS, using specific Meta- and Reason-classes

Classes *FeatureA* and *FeatureB*, together with *FeatureBCategoryType* and *Boolean* (not shown in Figure 1), represent "normal" data, while the other classes represent a way to explicitly model metadata about the values of the attributes.

The metadata is only hinted at in Figure 1 by the presence of *DatatypeMeta*, a supertype of the two classes *BooleanMeta* and *FeatureBCategoryMeta*. Figure 2 shows the content model of the abstract type *DatatypeMeta*. Note that, for brevity, only a few metadata characteristics defined by the NAS are shown.

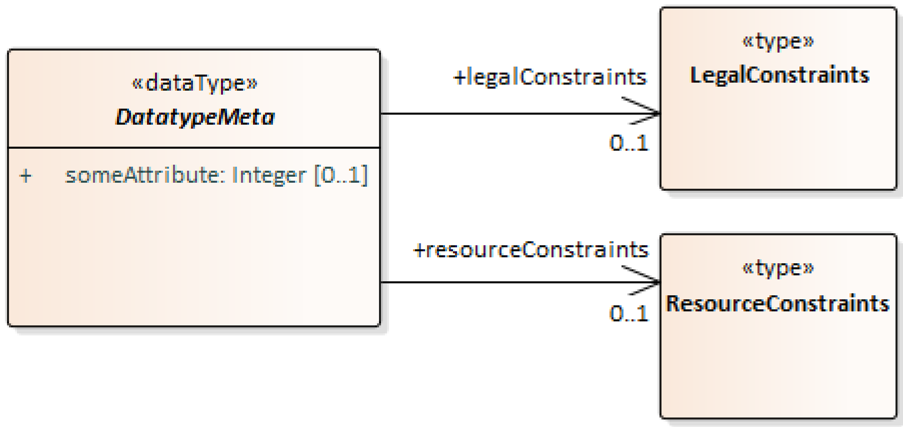


Figure 2 - The datatype which carries the metadata characteristics in the current NAS modeling approach

This approach to modeling metadata for attribute values introduces a high degree of complexity, both in the actual model and in platform-specific encodings (*e.g*. Extensible Markup Language (XML) and Resource Description Framework (RDF) / Web Ontology Language (OWL)). Issues that have been raised for this approach are:

- The actual cardinality of a feature property is not directly visible for that property. For example, the cardinality of *FeatureB.attribute2* is shown as 0..1, but – when combined with the cardinality of *FeatureBCategoryReason.values* – it is 0..*.
- The Meta- and Reason-classes introduce a level of indirection which is unfamiliar to the typical user (modeling expert, software developer, *etc*.) and can result in reduced comprehension due to the increase in model complexity.
  - For example, instead of saying "*FeatureB* has *attribute2* with values *category1* and *category2*", one needs to say "*FeatureB* has *attribute2* with a *FeatureBCategoryMeta* whose *valuesOrReason* is a *FeatureBCategoryReason* whose *values* property has the values *category1* and *category2*".
  - An Object Constraint Language (OCL) constraint intended to restrict the value of FeatureA.attribute1 to *true* must take the Meta- and Reason-classes into account within the OCL expression. Instead of the simple invariant (in the context of class *FeatureA*) "attribute1 = true", the expression would have to be "attribute1.valueOrReason.value = true". With a more elaborate expression, for example checking properties based upon the values of other properties, the level of complexity caused by the Meta- and Reason-classes increases exponentially.
- Humans and – sometimes – software tools struggle with the size of the application schema, which is considerably increased by the presence of the Meta- and Reason-classes.

To summarize: The current approach to explicitly modeling attribute metadata in the NAS is difficult to understand and maintain. A primary goal of the UGAS-2019 Pilot therefore is to make the application schema of the NAS more user-friendly and the actual (NAS-conformant) data more accessible.

2.2 Platform Independent Modeling with Property Stereotype for Metadata

In order to solve the complexity issues caused by the current modeling approach of the NAS, a new property stereotype is introduced: <<propertyMetadata>>[1]. This stereotype, when assigned to a property, indicates that metadata is associated with values of the property.

Note: Instead of using multiple different metadata types, applications typically only use a single type of metadata for describing an object or a (set of) property value(s). Accordingly, the analysis is based on the constraint that at most one property stereotype for metadata is defined per UML property.

When applying the property stereotype to the example from Figure 1, the conceptual model can be simplified significantly – see Figure 3.

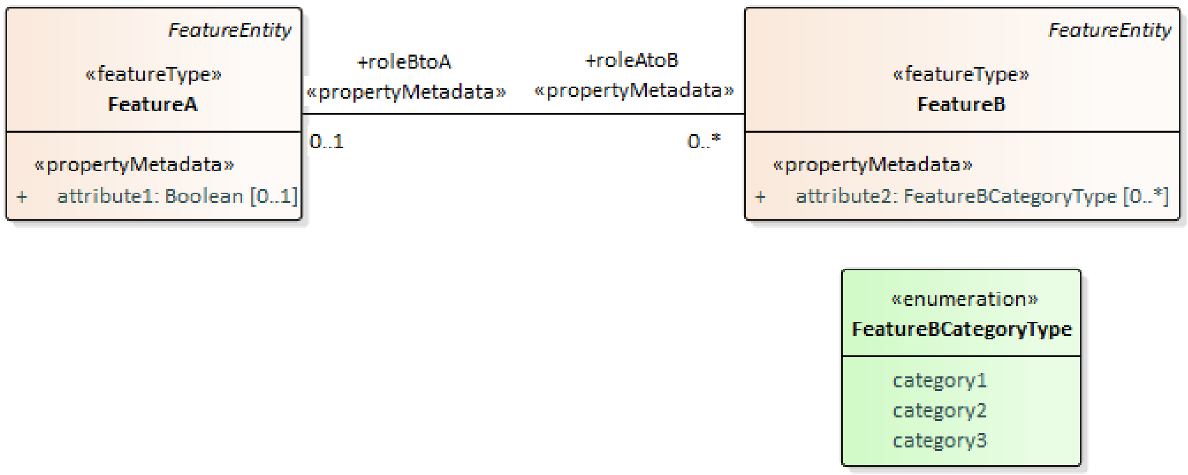


Figure 3 – New approach for modeling property metadata in the NAS, using property stereotype <<propertyMetadata>>

Note: As mentioned in section 2.1, the NAS currently does not support metadata for values of association roles. Consequently, the association in Figure 1 looks "normal", *i.e.* there are no specific stereotypes on that association. In Figure 3, however, the roles of the association do have a stereotype: <<propertyMetadata>>. This may look like added complexity. However, it is just an example that shows that the new stereotype can be applied on properties in general, i.e. both attributes <u>and</u> association roles. That is a considerable improvement, not only regarding simplicity

[1] A community might prefer a different name for the new stereotype, for example <<propMeta>>. This is possible, especially if tools they use to process the conceptual model can determine that a stereotype with another name identifies the same concept as stereotype <<propertyMetadata>>. ShapeChange, for example, allows users to configure stereotype aliases, with which <<propMeta>> can be mapped to the well-known stereotype <<propertyMetadata>>.

(of the model) but also regarding functionality (since metadata can also be associated with the values of association roles).

Different properties may be associated with different types of metadata, and applications may need to identify which type of metadata is associated with a specific property. To support this, tagged value *metadataType* has been defined. It can be used in combination with stereotype <<propertyMetadata>>. The tagged value identifies the applicable metadata type, as follows:

- If the type is defined by the schema that contains the property, then the tagged value simply provides the name of the type.
- Otherwise, the tagged value shall identify the type by its full package-qualified name, starting with the application schema package. For example: "Some Application Schema::Some Subpackage::Another Subpackage::MetadataType".
    - NOTE: In order for the qualified name to lead to the intended type, it is important that the schema structure and the type name does not change – also that the type actually is contained in the model. In rare cases, model transformations could result in such changes. Another situation where such a change can occur is when a schema that uses <<propertyMetadata>> - with metadataType referring to a type from an external schema - is shared with another user[2], and the other user has a slightly different version of the external schema in his UML model, for example with altered package structure. If ShapeChange cannot find the type based upon its qualified name, but requires that type to perform a certain task (*e.g*. applying a model transformation as described in section 2.3.1.3), ShapeChange will log an error.

Figure 4 provides an example of how NAS metadata types could be modeled. Note the two types AttMeta and RoleMeta, which represent examples of different metadata types, one for attributes and one for association roles.

[2] For example via XML Metadata Interchange (XMI) exports that are hosted in a Subversion (SVN) repository to which both users have access.

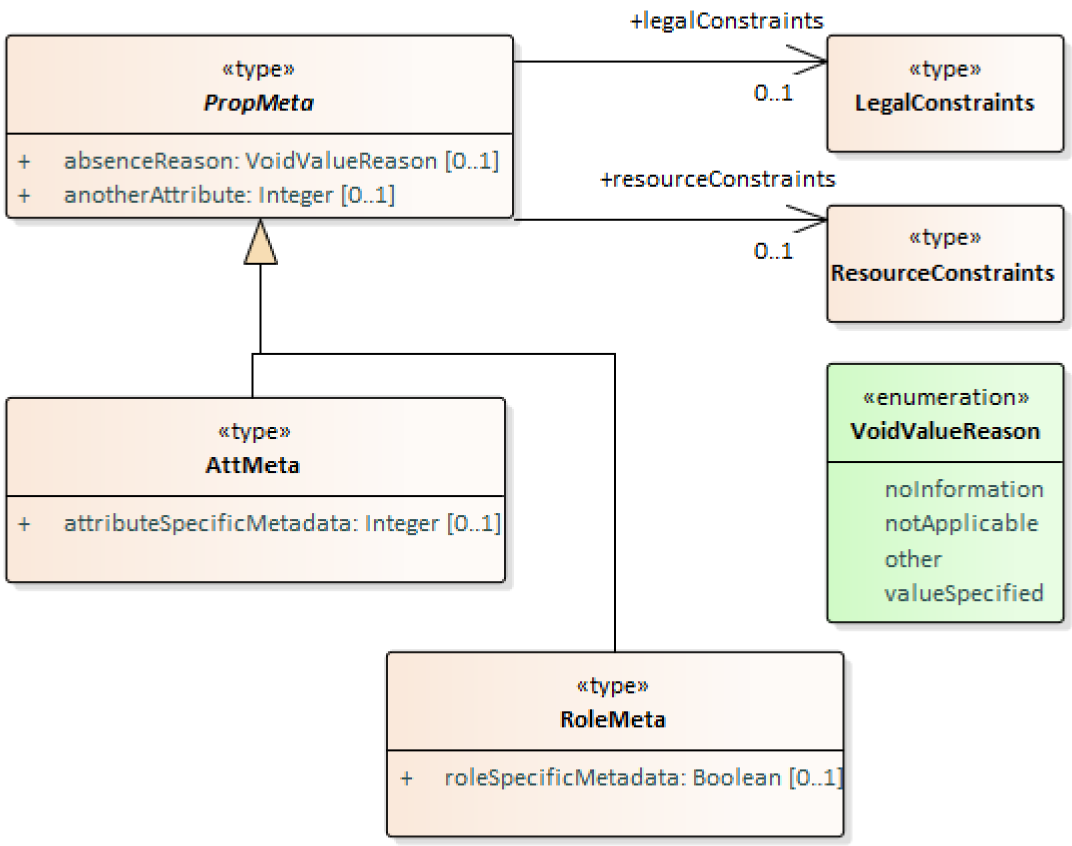


Figure 4 – Example of instantiable metadata types, with identity

Note: If metadata shall be referenceable, the metadata type should have identity, and should consequently be modeled accordingly (*e.g.* as shown in Figure 4). Metadata with identity can be shared and re-used. In case of the NAS, security markings are well suited for sharing and re-use. The way that metadata is identified and the way in which metadata can be referenced depends on the approach for governing metadata, and is the responsibility of each community.

Figure 5 illustrates how tagged value *metadataType* would be set for the <<propertyMetadata>> properties from the example in Figure 3, in order to reference the two metadata types from the example in Figure 4.

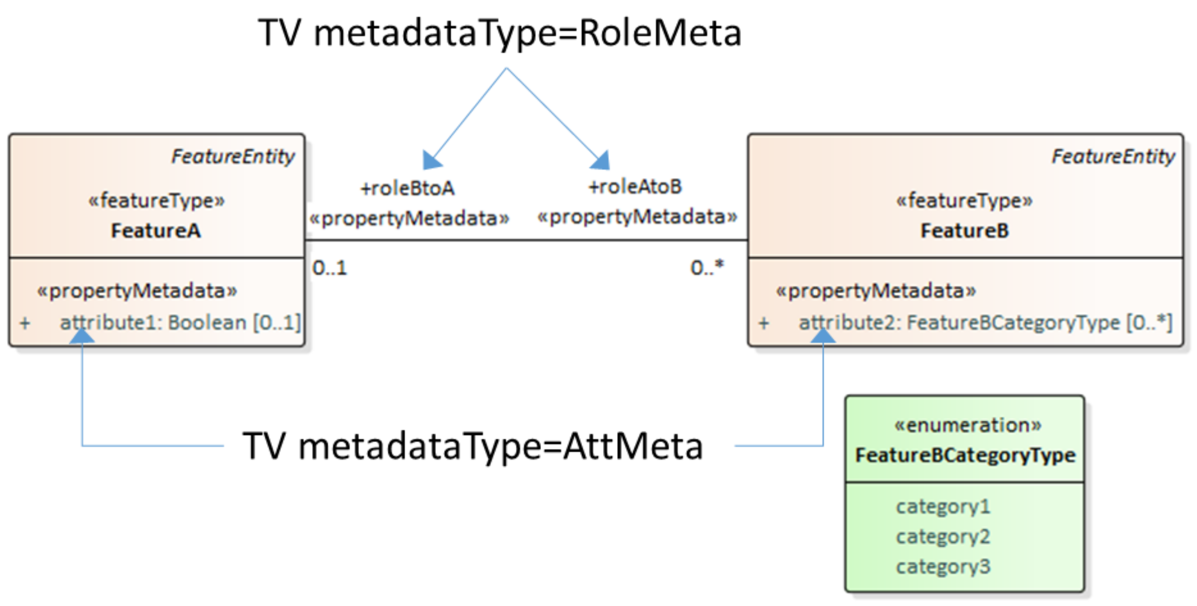


Figure 5 – Example of properties with stereotype <<propertyMetadata>> referencing different metadata types via tagged value "metadataType"

In platform-specific encodings, the data can include the metadata, it can reference the metadata, or it can be referenced by the metadata. Whether metadata is referenced by or references the actual data depends on both the data and the metadata formats and/or the rules regarding how both are maintained, stored and published. It may even be possible to have bi-directional links between data and metadata. The topic of navigability to/from metadata will be discussed in more detail in section 2.3, when approaches for representing the <<propertyMetadata>> stereotype in implementation schemas are presented.

The approach of using a property stereotype for modeling property metadata has the following advantages:

- The simplified model looks like any other application schema. It should be much easier to understand and maintain. Only the property stereotype needs to be added where applicable.
- Without the Meta- and Reason-classes:
  - The cardinality of a property is directly visible.
  - OCL expressions can be formulated in a natural way.
  - The size of the model has decreased significantly.
  - The model clearly separates the domain aspects from the metadata.

There is precedence for the use of property stereotypes with specific meaning in application schemas:

- The UML profile defined by ISO 19109[3] includes the property stereotype *<<estimated>>*, which indicates that the property value is assigned using an observation procedure.
- The UML profile defined by the Infrastructure for Spatial Information in Europe (INSPIRE) Generic Conceptual Model[4] contains *<<voidable>>* as stereotype for properties. Such a property may have the value of *void* (often also called *null*) to indicate that the characteristic represented by the property is not present in the spatial data set, but may be present or applicable in the real world. In INSPIRE, it is possible to qualify a value of *void* in the data with a reason using the INSPIRE *VoidReasonValue* type.

Note: The *PropMeta* type in Figure 4 defines an *absenceReason*, which, like the *reason* property from the Reason-classes in the current NAS, can be used to qualify the reason why no value or a void/null value is given for a property. Alternatively, an additional property stereotype like <<voidable>> (or a stereotype with same meaning) can be used to indicate that in actual data, void/null values can be qualified with a reason – instead of explicitly encoding such a reason in the metadata type(s). The set of applicable absence reason codes may depend on the value type of a property[5]. It would be the same for all properties with that value type. If different sets of reason codes need to be defined by the model[6] then the model should define these sets as enumerations. In order to indicate which absence reasons apply for a given property, set tagged value *voidReasonType* for that property[7]. The value of that tag identifies the enumeration - using the same approach as defined for tag *metadataType*.

For the NAS, it is recommended to not model absence reasons within property metadata types. Instead, the fact that a property value can be void and should be qualified with an absence reason should be modeled using stereotype <<voidable>> (in conjunction with tagged value *voidReasonType*, to indicate the applicable set of absence reasons). If metadata for that property can be defined as well, the <<propertyMetadata>> stereotype should be added to the property. Consequently, a UML property in the NAS can have multiple stereotypes, for example <<propertyMetadata, voidable>>. Figure 6 illustrates the use of multiple stereotypes per property, based upon the example model from Figure 3.

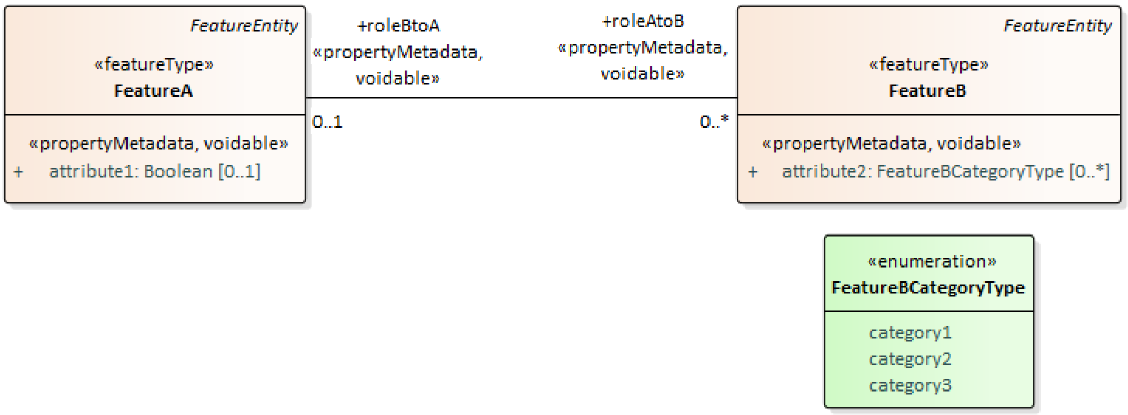
[3] See ISO 19109:2015, section 8.2.2, table 17.

[4] See INSPIRE Generic Conceptual Model, version 3.4, section 9.4.6 requirement 18, and section 9.6.3, table 2.

[5] For the NAS, that is the case: the sets of absence reasons for numeric and other types are different.

[6] An alternative could be to apply an external validation mechanism on encoded data.

[7] A UML profile that contains stereotype <<voidable>> could add tagged value *voidReasonType* to that stereotype, with allowed values restricted to a certain set of enumeration names, and a default value.



FeatureEntity
«featureType»
FeatureA
«propertyMetadata, voidable»
+ attribute1: Boolean [0..1]
+roleBtoA
«propertyMetadata, voidable»
0..1
+roleAtoB
«propertyMetadata, voidable»
0..*
FeatureEntity
«featureType»
FeatureB
«propertyMetadata, voidable»
+ attribute2: FeatureBCategoryType [0..*]
«enumeration»
FeatureBCategoryType
category1
category2
category3

Figure 6 - New approach for modeling in the NAS, using property stereotypes <<propertyMetadata>> and <<voidable>>

Note: Property metadata can be defined for individual property values, and/or for the whole set of values of a property. In principle, the new approach to use the <<propertyMetadata>> stereotype for properties that can have metadata supports both. However, the platform-specific metadata format may only support one way (metadata for individual property values or for all values of a property). Ultimately, the application community has to decide which kind of property metadata it needs, and choose the platform-specific metadata format accordingly.

2.3 Platform-Specific Encoding for Property Stereotype <<propertyMetadata>>

2.3.1 XML Encoding

Four alternative approaches for encoding the <<propertyMetadata>> stereotype in XML have been identified:

- data referencing metadata
- metadata referencing data
- metadata as additional data
- data with metadata range

The following sections document each approach. Section 2.3.1.5 provides a summary, including a table for comparing the approaches.

Note that all options assume that the XML encodings of the data use the object-property-value pattern as used by ISO 19136 Geography Markup Language (GML) and the ISO/TS 19115-3 / ISO/TS 19139 encoding rule.

2.3.1.1 Data Referencing Metadata

This encoding approach adds another XML attribute to property elements, with which metadata objects can be referenced. In the example shown in Listing 1, the name of that XML attribute is *metadata*[8].

Note: Using an XML attribute to convey additional information for a property value, in this case to reference a metadata object, is an already established approach in GML: the *nilReason* XML attribute can also be used to provide additional information for a property value.

Listing 1 – XML example for Data Referencing Metadata approach

```
<xml xmlns:gml="http://www.opengis.net/gml/3.2"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink">
 <FeatureA gml:id="FA">
 <attribute1 metadata="#M1" xsi:nil="true" nilReason="…"/>
 <roleAtoB metadata="#M2" xlink:href="#FB"/>
 </FeatureA>
 <FeatureB gml:id="FB">
 <attribute2 metadata="#M3">category1</attribute2>
 <attribute2 metadata="#M3">category2</attribute2>
 <roleBtoA metadata="#M4" xsi:nil="true" nilReason="…"/>
 </FeatureB>
 <AttMeta gml:id="M1">...</AttMeta>
 <RoleMeta gml:id="M2">...</RoleMeta>
 <AttMeta gml:id="M3">...</AttMeta>
 <RoleMeta gml:id="M4">...</RoleMeta>
</xml>
```

Each XML element that represents a property can use this XML attribute to reference a metadata object. In the example, the property *attribute2* has multiple values, but each property element references the same metadata object (#M3). The property elements could just as well reference different metadata objects. The example was constructed with the current property metadata scope of the NAS in mind (where a single metadata instance is often provided for the whole set of values of a property).

Note: The example shows the *nilReason* XML attribute. The content of *PropMeta* objects has been omitted, even though the conceptual model shown in Figure 4 defines the *absenceReason* attribute for *PropMeta*. Whether or not *nilReason* should play a role in the XML encoding depends on the particular schema:

[8] The type of the *metadata* XML attribute would be xs:anyURI. Absolute and relative URIs to reference metadata objects are allowed. This is similar to the XML attribute *xlink:href*.

- If 1) *nilReason* is already used in the current schema encoding, or 2) the metadata format does not contain an XML element corresponding to the *absenceReason* UML attribute, then use the *nilReason* XML attribute.
- However, if the metadata format does contain a suitable XML element, then use it and ignore the *nilReason* XML attribute.

This encoding approach is compatible with GML, and consistent with existing encoding rules. It avoids cluttering data with metadata, even though access to the metadata of a property is still straightforward. XML encoded data would simply contain additional *metadata* XML attributes for all XML elements that encode property values for which metadata is available. A *metadata* XML attribute contains the information to get this metadata. It is therefore comparable to the GML *byReference* encoding, where an xlink:href XML attribute contains the information to get the referenced object.

2.3.1.2 Metadata Referencing Data

In this encoding approach, a metadata object has some means to identify the object and the property for which metadata is provided. The actual data has no link to the metadata. It is the other way round (metadata references data).

The conceptual model of the metadata types from the example in Figure 4 do not have means to reference the relevant object and property. Thus, a model transformation is used to add properties – with which such references can be made – to relevant property metadata types. Figure 7 shows the result of this transformation, when applied to the metadata types from Figure 4. The two properties “object” and “property” have been added to *PropMeta*.

Note: If the conceptual model of metadata types already had the means to identify the object and property for which metadata is provided, then of course such a model transformation would not be necessary.

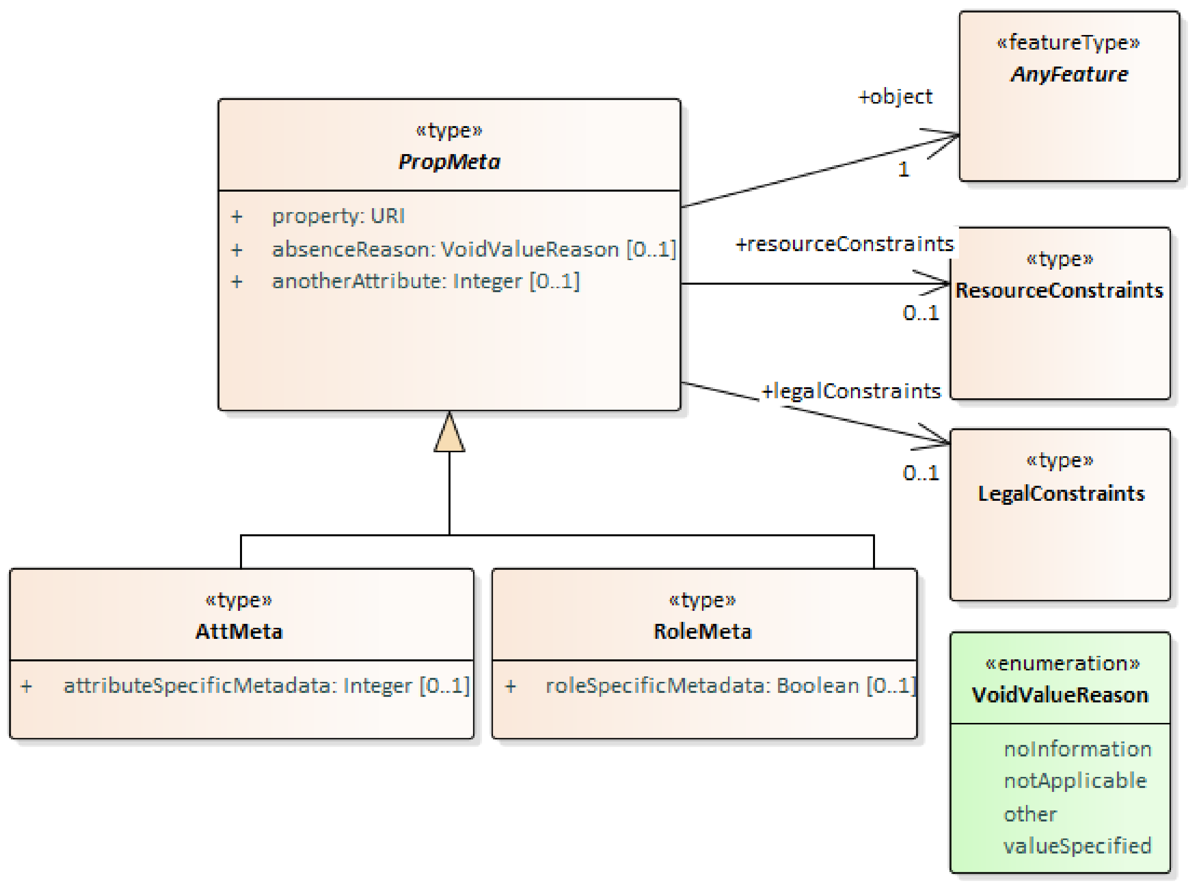


Figure 7 – Example of transformed metadata types for the Metadata References Data approach

There are conceptual issues with this approach:

- The value type of property *object* is *AnyFeature*. The type is specified in ISO 19109:2015 as "an abstract class that is the generalization of all feature types". ISO 19109:2015 requires that any feature type in an application schema has to be a subtype of AnyFeature[9]. This requirement is implemented in an application schema that is compliant to ISO 19109:2015. However, application schemas that are only compliant to an older version of ISO 19109, and not to ISO 19109:2015, do not use of AnyFeature. That would result in a mismatch between the transformed metadata type model – which uses AnyFeature to reference the object to which the metadata applies – and the application schema – whose feature types do not inherit from AnyFeature.

[9] Requirement /req/uml/feature states: "An instance of FeatureType shall be implemented as a CLASS. The CLASS shall carry the STEREOTYPE «FeatureType». The CLASS shall have a GENERALIZATION ASSOCIATION with AnyFeature."

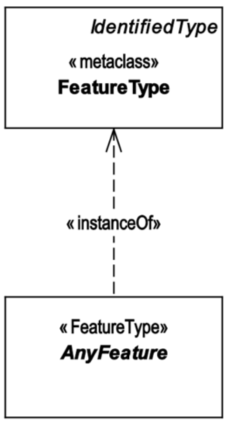


Figure 8 – AnyFeature in ISO 19109:2015

- The value type of property *property* is *URI*, but conceptually it would be the metaclass PropertyType (from ISO 19109:2015). That is, valid values of *property* are instances of *PropertyType*, *i.e.*, elements in the model / application schema. In an application schema specified in UML this would be an attribute or association role of a feature. As this would use a metaclass in the application schema or a model element in instance data, the abstraction levels are no longer clearly separated. There is no generally supported mechanism to represent references to such model elements in instance data, for example in an XML instance document. This issue is reflected in the fact that a general URI is used as the value type. Data publishers will have to specify how URIs identify the property.

Note that ISO 19156 / O&M basically has the same issues (properties *featureOfInterest* and *observedProperty*).

With the conceptual model for data shown in Figure 3, and the conceptual model for the metadata types illustrated in Figure 7, XML data where metadata references data can be created. Listing 2 provides an example.

Listing 2 – XML example for Metadata Referencing Data approach

```
<xml xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink">
 <FeatureA gml:id="FA">
 <attribute1 xsi:nil="true" nilReason="…"/>
 <roleAtoB xlink:href="#FB"/>
 </FeatureA>
 <FeatureB gml:id="FB">
 <attribute2>category1</attribute2>
 <attribute2>category2</attribute2>
 <roleBtoA xsi:nil="true" nilReason="…"/>
 </FeatureB>
 <AttMeta gml:id="M1">
 <object xlink:href="#FA"/>
 <property>http://example.org/schema/FeatureA/attribute1</property>
 ...
 </AttMeta>
 <RoleMeta gml:id="M2">
 <object xlink:href="#FA"/>
 <property>http://example.org/schema/FeatureA/roleAtoB</property>
 ...
 </RoleMeta>
 <AttMeta gml:id="M3">
 <object xlink:href="#FB"/>
 <property>http://example.org/schema/FeatureB/attribute2</property>
 ...
 </AttMeta>
 <RoleMeta gml:id="M4">
 <object xlink:href="#FB"/>
 <property>http://example.org/schema/FeatureB/roleBtoA</property>
 ...
 </RoleMeta>
</xml>
```

The example uses an encoding that is somewhat similar to the encoding of O&M observations. While object references via xlink:href are common enough, the identification of a property is not. The example shows how metadata for all values of a property can be defined, though in a rather application specific way[10]. If metadata for individual property values was needed, the referencing mechanism within the metadata format would have to become even more elaborate.

Notes:

- The discussion regarding the encoding of reasons for the absence of a property value is the same as for the Data Referencing Metadata approach.
- The example encodes metadata objects as ISO 19136 GML objects (with gml:id). XML Schema definitions for metadata object types can also be derived using the ISO 19139 encoding rule or any other encoding.
- In the example, property references are encoded as *property* elements with URI content. Alternatively, XLink XML attributes could have been used to reference property type definitions.
- The transformation for adding the means to reference object and property would have to be defined in detail, so that it can be implemented (*e.g.* in ShapeChange). For example, one aspect would be to which classifiers the new properties should be added. That would be the classifiers identified by the *metadataType* tagged value of properties with stereotype <<propertyMetadata>>. If such a classifier is a subtype of another classifier that is referenced that way, the new properties would only be added to the classifier that represents the root of such an inheritance tree.

[10] In the example, a URI is used. This can resolve to a property definition of any kind, for example an ontological definition of the property. The application will have to understand the format of the referenced resource, which leads to a community-specific way of identifying properties.

2.3.1.3 Metadata as Additional Data

This approach depends upon a transformation of the property stereotype <<propertyMetadata>> into additional metadata properties. For each property with that stereotype, a new property is created, with the metadata type that applies for the property (identified by tagged value *metadataType*) as its value type. Figure 9 illustrates the result of this transformation, based upon the conceptual model shown in Figure 3.

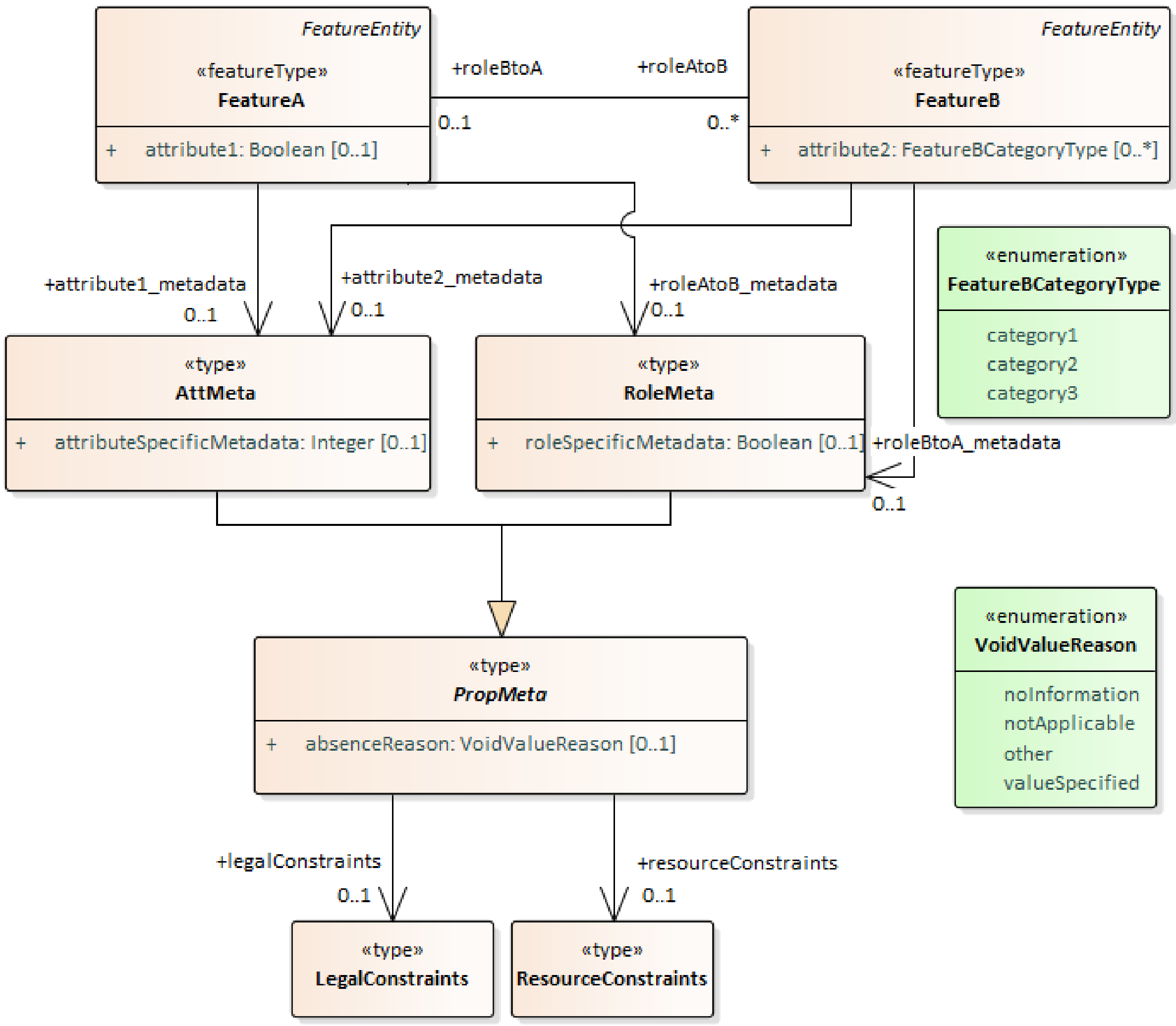


Figure 9 – Property stereotype for metadata transformed to metadata properties

Since the metadata type is modeled as a type with identity, the transformation created directed associations to that type, one for each property with the property stereotype <<propertyMetadata>>. The names of such properties are used for constructing the names of the new navigable association roles. In the example, the suffix "_metadata" is added to these names. Any other, non-empty suffix would work as well[11]. Once the associations to the metadata type have been created, the property stereotype <<propertyMetadata>> is removed.

For the XML encoding, the transformed model can then have normal encoding rules (*e.g.*, ISO 19136 or 19139) applied. Other than in the first approach (see section 2.3.1.1), the metadata values are now encoded in additional XML elements.

Listing 3 – XML example for Metadata as Additional Data approach

```
<xml xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink">
 <FeatureA gml:id="FA">
 <attribute1 xsi:nil="true" nilReason="…"/>
 <attribute1_metadata xlink:href="#M1"/>
 <roleAtoB xlink:href="#FB"/>
 <roleAtoB_metadata xlink:href="#M2"/>
 </FeatureA>
 <FeatureB gml:id="FB">
 <attribute2>category1</attribute2>
 <attribute2>category2</attribute2>
 <attribute2_metadata xlink:href="#M3"/>
 <roleBtoA xsi:nil="true" nilReason="…"/>
 <roleBtoA_metadata xlink:href="#M4"/>
 </FeatureB>
 <AttMeta gml:id="M1">...</AttMeta>
 <RoleMeta gml:id="M2">...</RoleMeta>
 <AttMeta gml:id="M3">...</AttMeta>
 <RoleMeta gml:id="M4">...</RoleMeta>
</xml>
```

Notes:

- The example in Listing 3 only shows metadata by reference. Metadata could also be encoded inline. The transformation could be extended to set the *inlineOrByReference* tagged value on newly created metadata properties (to *inline*, *byReference*, or *inlineOrByReference*). The target encoding rules (*e.g.* for XML Schema, JavaScript Object Notation (JSON) Schema, and Structured Query Language (SQL) Data Definition Language (DDL)) ultimately define which encodings of metadata property values are available.
- In order to encode the new metadata properties in sequence with their corresponding properties (*e.g. attribute1_metadata* directly after *attribute1*), as shown in the example, the transformation will need to set the *sequenceNumber* tagged values appropriately. The transformation could also support different behaviors, for example to encode the new metadata properties after the whole set of actual properties.

[11] The suffix could be a configuration parameter of the model transformation.

One drawback of this approach is that it only supports a single metadata object per property; providing metadata for individual property values would not be supported. In order to also support metadata for individual property values, a more complex transformation would be needed. That transformation could:

- Create association classes, which would provide information with which individual values can be identified. However, it is not clear what kind of information this would be. A new mechanism would likely need to be invented, creating an additional burden for implementers.
- Create new metadata (data) types, with two properties: a *value* (with the value type of the property that had a property stereotype for metadata) and *metadata* (whose value type would be the metadata type that belongs to the property stereotype for metadata). This would allow provision of metadata per value, with one level of indirection. It would mean, however, that OCL expressions will likely need to be transformed, which would be a very complex task.

Note that the discussion regarding the encoding of reasons for the absence of a property value is the same as for the Data Referencing Metadata approach (see section 2.3.1.1).

2.3.1.4 Data with Metadata Range

In this approach, new metadata types are created for properties with stereotype <<propertyMetadata>>. Listing 4 provides an example, where new classes *BooleanMetadata*, *FeatureAMetadata*, *FeatureBMetadata* and *FeatureBCategoryTypeMetadata* have been created.

Listing 4 – XML example for Data with Metadata Range approach

```
<xml xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink">
 <FeatureA gml:id="FA">
 <attribute1>
  <BooleanMetadata>
  <resourceConstraints>...</resourceConstraints>
  <absenceReason>...</absenceReason>
  <value xsi:nil="true" nilReason="…"/>
  </BooleanMetadata>
 </attribute1>
 <roleAtoB>
  <FeatureBMetadata>
  <resourceConstraints>...</resourceConstraints>
  <value xlink:href="#FB"/>
  </FeatureBMetadata>
 </roleAtoB>
 </FeatureA>
 <FeatureB gml:id="FB">
 <attribute2>
  <FeatureBCategoryTypeMetadata>
  <resourceConstraints>...</resourceConstraints>
  <value>category1</value>
  </FeatureBCategoryTypeMetadata>
 </attribute2>
 <attribute2>
  <FeatureBCategoryTypeMetadata>
  <resourceConstraints>...</resourceConstraints>
  <value>category2</value>
  </FeatureBCategoryTypeMetadata>
 </attribute2>
 <roleBtoA>
  <FeatureAMetadata>
  <resourceConstraints>...</resourceConstraints>
  <absenceReason>...</absenceReason>
  <value xsi:nil="true" nilReason="…"/>
  </FeatureAMetadata>
 </roleBtoA>
 </FeatureB>
</xml>
```

This is somewhat similar to the Meta- and Reason-classes in the current NAS. However, Reason-classes are no longer needed since value and absence reason are both properties of the new metadata types.

This approach supports metadata for individual property values. With slight adjustments, it could also be used to support metadata for sets of property values.

The approach could be realized with a model transformation. However, overall, this approach leads to a rather complex encoding. OCL constraints would need to be transformed as well, which is a very complex task. In addition, sharing and re-use of metadata objects is restricted, since the metadata is now coupled with the property value.

Note: Sharing and re-use of metadata objects is not fully eliminated, because classes referenced by metadata types, such as *ResourceConstraints* and *SecurityConstraints* in the example from Figure 4, have not been merged. Instances of these classes can still be shared and re-used. Only the metadata types themselves, *AttMeta* and *RoleMeta* in the example from Figure 4, have been merged and thus sharing and re-use of these metadata objects is no longer possible.

Note that the discussion regarding the encoding of reasons for the absence of a property value is the same as for the Data Referencing Metadata approach (see section 2.3.1.1).

2.3.1.5 Summary

The following table compares the XML encoding approaches. Note that the investigation is based on a conceptual model using the new property stereotype for metadata.

Table 1 – Comparison of XML encoding approaches for property stereotype <<propertyMetadata>>

	Data Referencing Metadata	Metadata Referencing Data	Metadata as Additional Data	Data with Metadata Range
Does the approach support metadata for individual values?	Yes (because each property value element can have a metadata reference)	No (at least not without increasing the level of complexity)	No (at least not without increasing the level of complexity)	Yes (because the value of each property element is a complex element that contains the actual value and metadata for that value)
Is access to metadata for a given property straightforward?	Yes (just follow the reference provided by the *metadata* XML attribute)	No (since a non-standardized mechanism is needed to look-up the metadata object that applies to a property)	Yes (just follow the reference provided by the metadata property element, or directly access the metadata if encoded inline)	Yes (since the metadata is encoded together with the value)
Can the necessary referencing (of data or metadata) be realized with standard referencing mechanisms?	Yes (normal object referencing mechanisms can be used)	No (a special mechanism is needed to identify the feature property)	Yes (normal object referencing mechanisms can be used; metadata could also be exclusively encoded inline, avoiding the need for a referencing mechanism)	Not applicable (since metadata is directly encoded with the data)
Is it easy to ignore the metadata (if all you are interested is the data)?	Yes (since without the new *metadata* XML attribute, everything else, i.e. the actual data, is encoded as usual; so the new XML attribute could simply be ignored)	Yes (because the actual data is encoded as usual)	Yes (if 1. new metadata properties can be distinguished from the properties with actual data and 2. metadata is not encoded inline)	No (because the metadata has been merged with the actual data)
Does the approach require	No	No	No	Yes

	Data Referencing Metadata	Metadata Referencing Data	Metadata as Additional Data	Data with Metadata Range
transformations of OCL expressions? (Presumes that these expressions do not span from data to metadata)				(because the merging introduces a level of indirection)
Is filtering of data with certain metadata characteristics with Extensible Stylesheet Language (XSL) Transformation (XSLT) efficient?	No (because metadata objects need to be dereferenced)	No (because the metadata that applies to a certain value always needs to be identified through a search within the whole set of metadata objects)	Inline: Yes (since the metadata is encoded with the data) By-reference: No (because metadata objects need to be dereferenced)	Yes (since the metadata is encoded with the data)
Is filtering of data with certain metadata characteristics with a Web Feature Service 2.0 efficient?	No (there is no XML Path Language (XPath) expression supported by Filter Encoding Specification (FES) 2.0 to identify the metadata elements of a property)	No (it is not possible at all as there is no way to get from the feature property to the metadata)	Inline: Yes (as the metadata elements can be identified using XPath expressions) By-reference: No (there is no XPath expression supported by FES 2.0 to identify the metadata elements of a property)	Yes (as the metadata elements can be identified using XPath expressions)
ShapeChange: **Does this approach require a model transformation?**	No	If the conceptual model is changed to include the "object" and "property" properties (Figure 5):	Yes	Yes

	Data Referencing Metadata	Metadata Referencing Data	Metadata as Additional Data	Data with Metadata Range
		No Otherwise (the conceptual model is as shown in Figure 4): Yes		
ShapeChange: **Is an XML Schema target customization required?**	Yes (to add the *metadata* XML attribute)	No	No	No
ShapeChange: If absenceReason is mapped to xsi:nil and nilReason, is an XML Schema target customization required?	Yes (the changes would be similar for all options; in addition https://shapechange.net/targets/xsd/extensions/#rule-xsd-cls-union-direct can be deprecated or deleted)			

The comparison of the four approaches shows that - for the purpose of GML based exchange of NAS data – the first approach is most promising. It is therefore recommended to realize the property stereotype <<propertyMetadata>> in GML application schemas using the Data Referencing Metadata approach (see section 2.3.1.1).

Note: This recommendation does not apply to GML application schemas with limited encoding options, where the addition of the metadata XML attribute is not allowed or supported, such as the GML Simple Features (GML-SF) encoding. Section 2.3.2 explains how the property stereotype <<propertyMetadata>> can be implemented in such encodings.

2.3.2 JSON, SQL, GML-SF Encodings

The recommended approach for representing the property stereotype <<propertyMetadata>> in the full GML encoding (see section 2.3.1.1) is based on the ability to provide additional information for an XML element – in this case the reference to a metadata object – in the form of an XML attribute. Encodings like JSON, SQL, and GML Simple Features (GML-SF), do not have this ability, or an equivalent mechanism. Therefore, these encodings require a different approach for realizing the property stereotype.

The *Metadata as Additional Data* approach (see section 2.3.1.3) is recommended for the JSON, SQL, and GML-SF encodings. A common model transformation is applied, to create an additional property corresponding to each property that has that stereotype in the original model. The type of such a new property is the metadata type that is identified by tagged value *metadataType*.

The encoding of the transformed model would be supported for JSON, SQL, and GML-SF encodings, since the stereotype <<propertyMetadata>> has been turned into regular model elements (additional properties), which are supported in these encodings[12].

Listing 5 shows how this could look like for JSON. It represents the content of the example from Listing 3, with the exception of nilReason attributes[13] and the content of the metadata objects (which is only hinted at in Listing 3, anyway). Note that object references are encoded in this example as JSON Pointers[14], since all objects from the example are contained in one JSON document. If all objects were available online in separate documents, full URLs should be used.

[12] Creating the encoding may require subsequent model transformations. Chapter 7 of the *OGC Testbed-13: NAS Profiling Engineering Report* (OGC document 17-020r1, available online at http://docs.opengeospatial.org/per/17-020r1.html), for example, documents the transformations that are necessary to generate a GML-SF0 encoding from a NAS schema profile.

[13] The OGC Testbed-14: Application Schemas and JSON Technologies Engineering Report (OGC document 18-091r2, available online at http://docs.opengeospatial.org/per/18-091r2.html), chapter 5.3.3, presents ideas for JSON Schema conversion rules, one of which is the creation of *nilReason* properties. Note, however, that the Engineering Report recommends to develop new JSON Schema conversion rules, and to revise the ShapeChange JSON Schema target to support these rules. This work will need to be done in order to derive JSON Schemas from the NAS.

[14] See IETF RFC 6901, *JavaScript Object Notation (JSON) Pointer*, available online at https://tools.ietf.org/html/rfc6901.

Listing 5 – JSON example for Metadata as Additional Data approach

```
[
 {
  "id": "FA",
  "attribute1": null,
  "attribute1_metadata": "/2",
  "roleAtoB": "/1",
  "roleAtoB_metadata": "/3"
 },
 {
  "id": "FB",
  "attribute2": [
   "category1",
   "category2"
  ],
  "attribute2_metadata": "/4",
  "roleBtoA": null,
  "roleBtoA_metadata": "/5"
 },
 {
  "id": "M1"
 },
 {
  "id": "M2"
 },
 {
  "id": "M3"
 },
 {
  "id": "M4"
 }
]
```

A benefit of this encoding approach is that metadata objects are encoded as individual objects, and can be stored and referenced individually, as in the recommended approach for the XML encoding. The implementation for managing the metadata can thus be similar for multiple encodings. A metadata service, for example, could expose metadata objects in both XML and JSON encoding.

Note: As documented in section 2.3.1.3, the transformation currently only supports metadata for all values of a property, not for individual values. If metadata for individual values was necessary, then a more complex transformation would be needed (section 2.3.1.3 presents two options).

2.3.3 Linked Data Encoding

Creating a linked data encoding from an application schema that uses property stereotype <<propertyMetadata>> is out-of-scope for this project. Nevertheless, a short analysis was performed to identify how such an encoding could be realized, and to get a better understanding of the state-of-the-art in this regard.

Linked data is represented through graph models, the most prominent of which are RDF and Property Graphs. RDF is standardized by W3C. Property Graphs have not yet been standardized. Graph models store statements in the form of subject-predicate-object triples. A predicate thereby represents a UML property. In RDF, the same subject may be connected to different objects via the same predicate. This resembles a UML property with a set of values. Property Graphs additionally support multiple triples with the same subject, predicate, and object. The question is if and how annotations of these triples are supported by these two graph models.

[1], a position statement from a W3C workshop held early 2019, provides valuable insights into this topic, also providing references to useful further reading.

According to these sources, the following approaches for property metadata in linked data exist:

- Property Graph (Edges)
- RDF Reification
- Singleton Properties
- RDF* and SPARQL* (extensions of RDF and SPARQL Protocol and RDF Query Language (SPARQL))

Shortcomings of the first three approaches are briefly discussed in [2]. However, the paper proposes an extension of RDF and SPARQL – aptly named RDF* and SPARQL* - which is quite promising:

- It proposes to make statements about RDF triples a first-class citizen of RDF as well as the RDF query language SPARQL. As a result, the verbosity that would be caused by using RDF Reification can be avoided.
- Conversions from/to RDF data using RDF Reification as well as Property Graphs exist. A user could thus start capturing linked data as RDF, using RDF Reification to encode property metadata, and convert the RDF data to RDF* data or a Property Graph later on – when necessary or desirable (*e.g.* for performant execution of specific graph queries). Note that when using RDF Reification, queries and OWL based reasoning on the “normal” data, *i.e.* not the metadata statements, can be performed as usual. SPARQL queries that involve metadata are possible, although they would be verbose (with pure RDF, RDF Reification, and SPARQL - not so with RDF* and SPARQL*). The extent to which OWL reasoning on metadata statements is possible and useful would require further analysis[15].
- The extension can be implemented as a wrapper on top of existing triple stores. At the same time, it allows for native implementations, with potentially better performance.

Note that a lossless conversion between an RDF* graph and a Property Graph is possible, but if the RDF* graph contained metadata statements on triples with literal value, the resulting Property Graph would be atypical. That is because typically, the properties of nodes and edges in a

[15] The current NSG Enterprise Ontology (NEO) is based on OWL to support reasoning. However, useful information may also be deduced using other or additional approaches, for example using query languages like SPARQL and Cypher (a query language for Property Graphs).

Property Graph only have literal values, and while edges in a Property Graph have identity themselves and can carry properties, that is not the case for the properties of a node. A metadata statement on an RDF triple with literal value would therefore require that the Property Graph represents the literal value with a node. This is described in more detail in [3].

To summarize: The linked data community has developed multiple approaches for representing property metadata, and is still actively working to improve the functional capabilities for handling such data. RDF Reification presents a standardized approach for representing property metadata, which works for any kind of RDF triple (unlike Property Graphs, which typically support and use metadata only for properties whose value is an identifiable object). RDF* offers a way to convert RDF data which uses RDF Reification to a less verbose format that is easier to query (using SPARQL*), and to also convert this data to Property Graphs (which can improve performance when performing certain graph analysis tasks). The property stereotype <<propertyMetadata>> can therefore be represented in a linked data encoding with existing approaches. RDF Reification, for example, would be a native and standardized representation of property metadata. Listing 6 and Listing 7 give an example of encoding the property metadata from the example in Listing 1 using RDF Reification. The underlying RDF data in Listing 6 and Listing 7 is the same; just the encoding is different (in the first case Turtle; in the second case RDF/XML).

Listing 6 – Example of property metadata encoded using RDF Reification – Turtle syntax

```
@prefix rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:      <http://www.w3.org/2001/XMLSchema#> .
@prefix metainst: <http://example.org/metadata/instances/> .
@prefix meta:     <http://example.org/metadata/ontology/> .
@prefix exinst:   <http://example.org/data/instances/> .
@prefix ex:       <http://example.org/data/ontology/> .

exinst:FA
  a ex:FeatureA ;
  ex:roleAtoB exinst:FB .

exinst:FB
  a ex:FeatureB ;
  ex:attribute2 "category1", "category2" .

_:b0
  a rdf:Statement ;
  rdf:subject exinst:FA ;
  rdf:predicate ex:roleAtoB ;
  rdf:object exinst:FB ;
  meta:metadata metainst:M2 .

_:b1
  a rdf:Statement ;
  rdf:subject exinst:FB ;
  rdf:predicate ex:attribute2 ;
  rdf:object "category1" ;
  meta:metadata metainst:M3 .

_:b2
  a rdf:Statement ;
  rdf:subject exinst:FB ;
  rdf:predicate ex:attribute2 ;
  rdf:object "category2" ;
  meta:metadata metainst:M3 .
```

Listing 7 – Example of property metadata encoded using RDF Reification – RDF/XML syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:ex="http://example.org/data/ontology/" xmlns:meta="http://exam-
ple.org/metadata/ontology/">
 <rdf:Description rdf:about="http://example.org/data/instances/FA">
  <rdf:type rdf:resource="http://example.org/data/ontology/FeatureA"/>
  <ex:roleAtoB rdf:resource="http://example.org/data/instances/FB"/>
 </rdf:Description>
 <ex:FeatureB rdf:about="http://example.org/data/instances/FB">
  <ex:attribute2>category1</ex:attribute2>
  <ex:attribute2>category2</ex:attribute2>
 </ex:FeatureB>
 <rdf:Statement rdf:nodeID="b0">
  <rdf:subject rdf:resource="http://example.org/data/instances/FA"/>
  <rdf:predicate rdf:resource="http://example.org/data/ontology/roleAtoB"/>
  <rdf:object rdf:resource="http://example.org/data/instances/FB"/>
  <meta:metadata rdf:resource="http://example.org/metadata/instances/M2"/>
 </rdf:Statement>
 <rdf:Statement rdf:nodeID="b1">
  <rdf:subject rdf:resource="http://example.org/data/instances/FB"/>
  <rdf:predicate rdf:resource="http://example.org/data/ontology/attrib-
ute2"/>
  <rdf:object>category1</rdf:object>
  <meta:metadata rdf:resource="http://example.org/metadata/instances/M3"/>
 </rdf:Statement>
 <rdf:Statement rdf:nodeID="b2">
  <rdf:subject rdf:resource="http://example.org/data/instances/FB"/>
  <rdf:predicate rdf:resource="http://example.org/data/ontology/attrib-
ute2"/>
  <rdf:object>category2</rdf:object>
  <meta:metadata rdf:resource="http://example.org/metadata/instances/M3"/>
 </rdf:Statement>
</rdf:RDF>
```

Even though RDF Reification has some shortcomings, these can be solved through conversions to other formats such as RDF* and Property Graphs.

The definition of conversion rules for property stereotype <<propertyMetadata>> in a linked data encoding is future work. However, here are some ideas:

- Ignore the stereotype. In actual RDF data, RDF Reification would be used to link to the metadata (object) for a given property value. That link could be represented by an RDF property, which should be standardized in order to enhance interoperability.

    - Note that this would allow an implementation for managing metadata, mentioned in section 2.3.2, to not only provide metadata objects encoded in XML and JSON, but also in a linked data format (RDF).
- Instead of ignoring the stereotypes, add sub-properties of the metadata property proposed in the previous bullet item to the ontology – one for each type identified by tagged value *metadataType* on properties with stereotype <<propertyMetadata>> from the application schema that is converted. The range of such a sub-property would be the metadata class that has been derived from the appropriate metadata type (*e.g.* AttMeta and RoleMeta as illustrated in Figure 4).
- In an ontology, a metadata class could be defined as a subclass of RDF Statement. Then the actual metadata information could be defined directly as such a statement, which would prevent the need for an intermediate metadata property (such as meta:metadata used in the example from Listing 7).

Note: RDF Reification supports metadata for individual property values, since the object of an RDF statement is a single value. However, it is unclear if metadata for whole sets of property values is supported, and whether this would be different for RDF*. Further analysis is required. Since edges in Property Graphs have their own identity, metadata is necessarily also given per object reference, and thus per object value. Note, however, that Property Graphs typically do not support metadata for properties with a literal value.

Note: Interestingly enough, an RDF Statement would solve the issue with the second approach for encoding the property stereotype for metadata (see section 2.3.1.2, Metadata Referencing Data) – at least regarding the encoding. The reason is that such a statement has well-defined properties to represent a subject, a predicate, and an object. However, it really only provides a well-defined solution for the (RDF) encoding, and does not solve the issues with conceptual modeling (as documented in section 2.3.1.2).

3 Annex A - Multiple Property Stereotypes for Metadata

This Annex documents a slightly different approach for solving the complexity issues caused by the current modeling approach of the NAS. The differences from the recommended approach – of having a single stereotype for metadata – are explained, as well as the reasons why the different approach is inferior to the recommended one.

This alternative approach is based on using multiple property stereotypes for metadata instead of a single stereotype. Figure 10 shows the differences from the recommended approach.

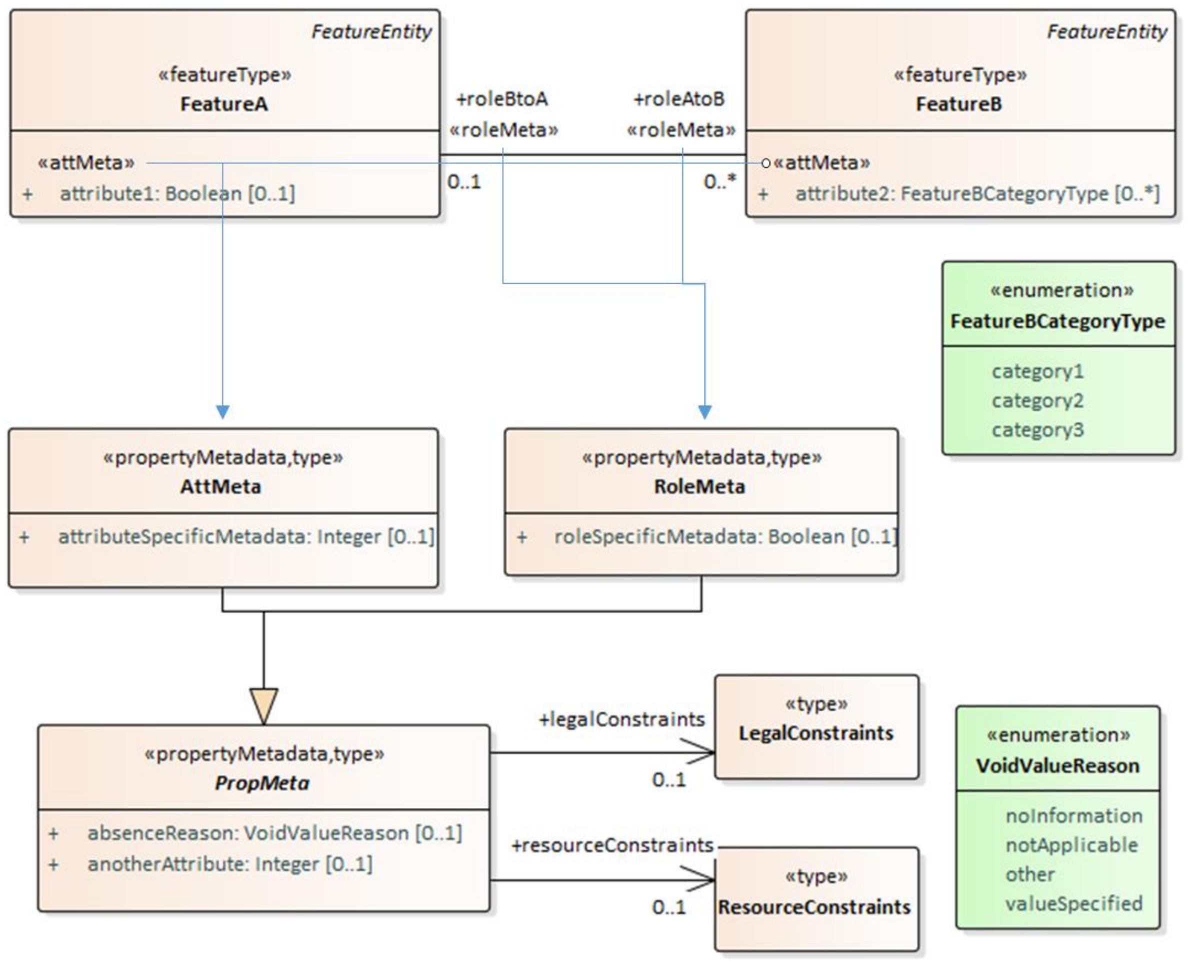


Figure 10 - Example for an alternative approach for modeling property metadata in the NAS, using multiple property stereotypes for metadata

The figure is a combination of the UML shown in Figure 3 and Figure 4, with the following differences: The properties use the stereotypes <<attMeta>> and <<roleMeta>> and the metadata types have an additional stereotype (<<propertyMetadata>>).

The idea behind having multiple property stereotypes for metadata was that each of them would visually indicate to the user which type of metadata is associated with a given property. The names of these stereotypes would be equal to (ignoring case) the names of the applicable metadata type. In the example, <<attMeta>> would thus identify type AttMeta as the metadata type. It would be possible to use different stereotypes for different attributes and different association roles. This capability was deemed to be useful if different metadata types apply for these properties.

Eventually, however, the decision was made to use a single property stereotype, <<propertyMetadata>>, instead of using multiple property stereotypes for metadata. The reasons for that decision are described in the following paragraphs.

Multiple property stereotypes for metadata would all have represented the same concept: that metadata is associated with the property. A benefit would have been that the stereotype name directly indicated of which type the metadata for a property is. However, the approach required that metadata types had a specific stereotype (<<propertyMetadata>>, as shown in Figure 10), in order to know which of potentially multiple stereotypes defined for a property actually are property stereotypes for metadata. That stereotype would have enabled tools like ShapeChange to determine that a property stereotype – for example <<attMeta>> – really is a property stereotype for metadata, and to differentiate this from situations in which a property stereotype coincidentally has the same name as one of the types contained in the UML model.

A single property stereotype suffices to indicate that metadata is associated with the property. A single, well-known property stereotype directly provides the relevant meaning to tools like ShapeChange, which therefore no longer need to search for metadata types with a specific stereotype. In fact, the approach with a single property stereotype for metadata works without the need for a specific stereotype on metadata types. In the recommended approach, the metadata type for a property is defined by a tagged value (metadataType).

The recommended approach is consistent with the recommended approach for the use of <<voidable>> (described towards the end of section 2.2), where the set of void reasons that are applicable for a given property is represented by an enumeration, which is identified by a tagged value (voidReasonType) and does not require any additional specific stereotype.

To summarize, the approach of only using a single property stereotype is superior to that of using multiple property stereotypes for metadata, because it:

- improves clarity (a single, well-known stereotype identifies the concept of a property being associated with metadata),
- ensures consistency (because it is in line with how the <<voidable>> stereotype and the link to the void reason type is modeled), and
- avoids modeling overhead (because it avoids having multiple stereotypes for properties with different types of metadata and it does not require the presence of a specific stereotype on these metadata types).

4 Bibliography

[1] O. Hartig, "Position Statement: The RDF* and SPARQL* Approach to Annotate Statements in RDF and to Reconcile RDF and Property Graphs," 2019.

[2] O. Hartig, "RDF* and SPARQL*: An Alternative Approach to Annotate Statements in RDF," in *International Semantic Web Conference*, 2017.

[3] O. Hartig, "Reconciliation of RDF* and Property Graphs," 2014.