OGC Model Driven Standards (D143)

2021-11-12

Table of Contents

Introduction

Refocussed Deliverable

Work for this Testbed 17 Deliverable 143 was originally anticipated to deliver a:

> *software tool capable of exercising UML model to platform specific model transformation for the work items identified in the Transformation Exercises table*

However, as per SURROUND Australia Pty Ltd's (SURROUND's) response to the Testbed, outlined in its Statement of Work[1], an approach to the generation of Standards' documents "directed by the World Wide Web Consortium (W3C)[2]'s *Profiles*

Vocabulary" (PROF) [PROF] and one that is "provenance and profile dependencies" aware was undertaken, rather than a tool build directly. This was for two reasons:

1. The OGC has an embedded tool-based approach (see Testbed 17, D135[3])
2. A quick assessment by this author indicated that to model-drive a whole Standard, and not just the _Domain Model_[4], lots more Standard element modelling was required

The focus on PROF was undertaken because it privides a formal, Semantic Web (thus model-driven) vocabulary to "relate profiles (standards that profile other standards) to one another and to link profile parts to one another". This then provides "several options for describing PIM/PSM relations".

The form of deliverables for this Deliverable is now a Report - this document - and a series of technical resources - ontologies, code examples and images - that are listed in the next section.

Deliverable Resources

- Report - this document
- Additions to the MDS Engineering Report[5]
- oMDS tool code - uploaded to https://gitlab.ogc.org/ogc/T17-D143-MDA-Tool-1 (https://gitlab.ogc.org/ogc/T17-D143-MDA-Tool-1)

Standard relations modelling & scenario selection

As per the SURROUND SoW, two potential options were presented fro relating Platform-Independent standard elements and Platform-Specidif ones (PIM/PSM):

1. Declaring the PIM and any PSMs to be parts of a single, conceptual, Standard
2. Defining the PIM and each PSM as a Standard and relating them with PROF properties

PROF provides a small vocabulary of roles that elements of a Standard might play[6] that could be used in scenario 1 to indicated PIM/PSM relations. Before this work, no assessment of whether these roles we related to PIM/PSM Standard concerns had been conducted. PROF provides only one Standard-to-Standard relationship, `isProfileOf`, that can be used to related multiple standards, as per scenario 2. There has been some talk of creating other relationships or specailised forms of `isProfileOf` but this has not happened yet.

For this Report, scenario 1 only is considered further for this reason: PIMs, PSMs and other Standads' elements seem to be easily considered parts of a conceptual standard and more modelling skill, in PROF, is available to describe and relate them than to relate multiples Standards together. Also, things modelled as per scenario 1 may be able to be re-implemented according to scenario 2 at a later date if, for innstance, someone wishes to take a part of a standard out of that standard and give it it’s own life. So if scenario 1 can be made to work, scenario 2 could be implemented later.

Exemplar selection

The SoW indicated that the development of the OGC’s GeoSPARQL standard' 1.1 version[7] would be use to exemplify the use of PROF. Within this Testbed activity, the Australian/New Zealand 3D Cadastre Standard[8] has also been used to demonstrate things.

These exemplars will be referred to throughout as *GeoSPARQL 1.1* and *3D Cad*.

Additional work

An Annex regarding the relationships between RDF graphs and Property Graphs is included in this document as this was generated in response to questions on that topic within the Testbed.

Standard modelling

As stated in the Section Refocussed Deliverable, a "software tool" was not produced *per se*, however a metholdology was. That methdology starts with modelling the elements of a Standard.

PROF Standards' elements modelling

PROF provides model (ontology) elements to describe and relate parts of a Standard. This mostly involves defining basic properties of the elements for technical access (format etc.) but also the Role or Roles that the various parts play with PROF providing a roles vocabulary[9] that contains:

- Constraints
- Example
- Guidance
- Mapping
- Schema
- Specification
- Validation
- Vocabulary

Definitions for these Roles are given by PROF and PROF encourages uses of it to extend this vocabulary for their own purposes. SURROUND has extended this vocabulary with a couple of additional roles for this work and other projects. That extended vocabulary can be seen at:

- https://data.surroundaustralia.com/def/prof-roles (https://data.surroundaustralia.com/def/prof-roles)

So far, the only additional Roles are:

- Repository (https://data.surroundaustralia.com/def/prof-roles/repository)

Since GeoSPARRQL 1.1 has had its elements modelled according to PROF and had them assigned these roles, some of its elements and their roles are given as an example of this modelling in Figure 2. using PROF's diagramming notation, a key of which is given in Figure 1.

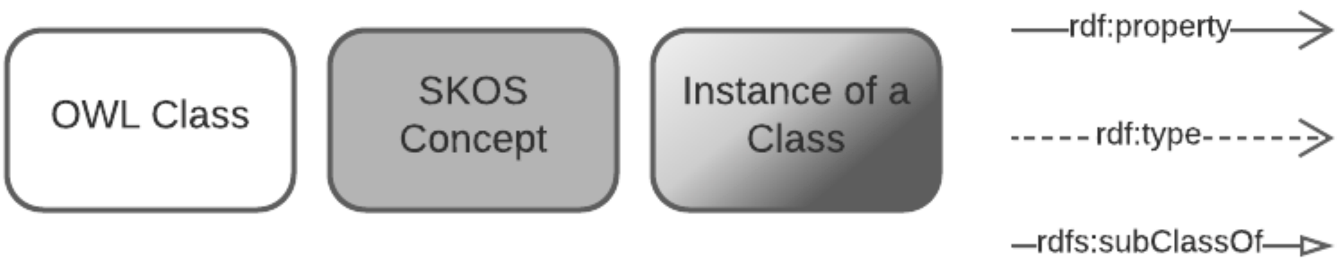


Figure 1. Figure Element Key

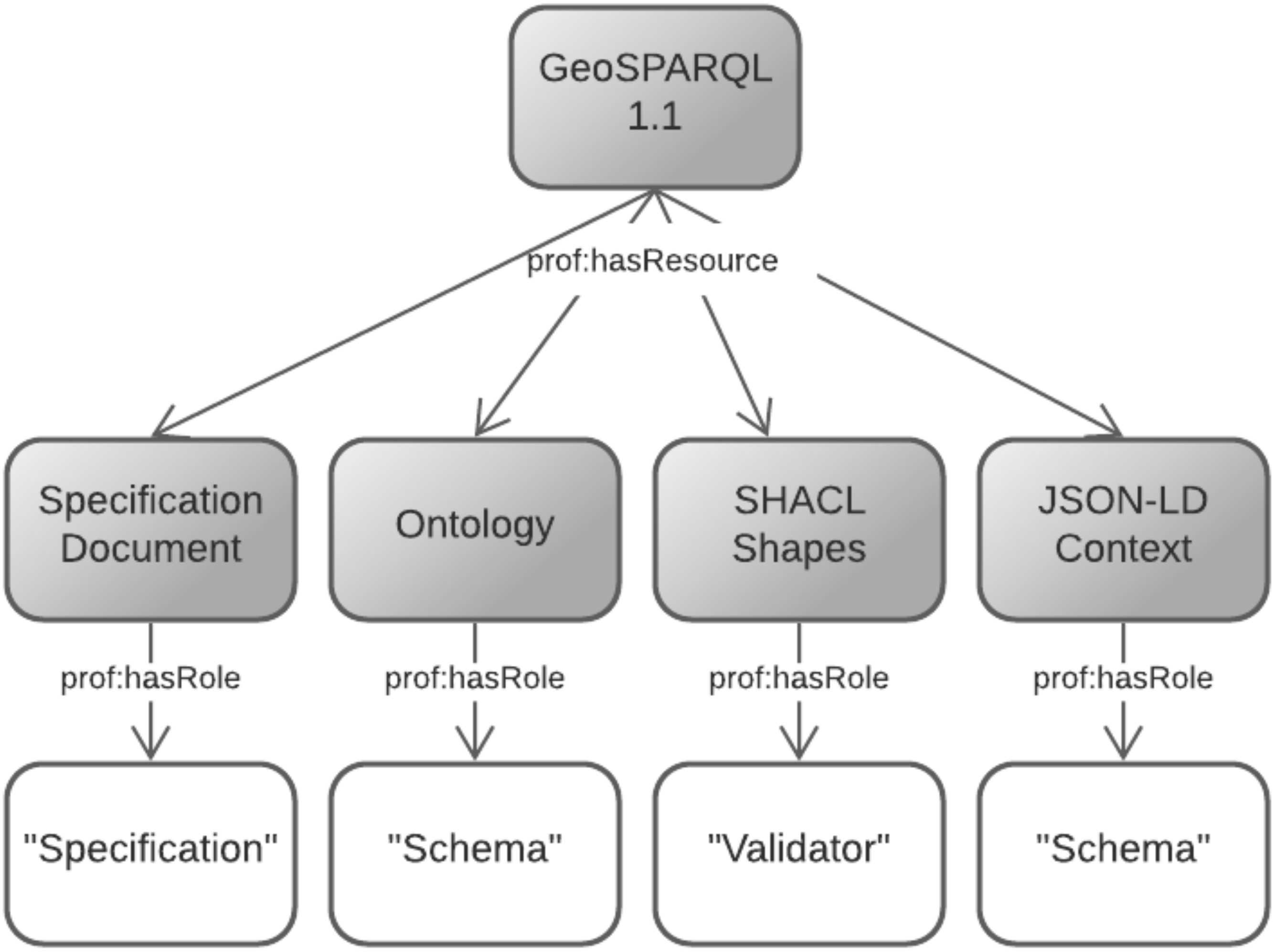


Figure 2. GeoSPARQL 1.1 Profile elements, modelled according to PROF, using figure elements from Figure 1

Note that, according to PROF, the thing that most people see as "The Standard", is termed a *Specification* (document) and that there are other, potentially equally important, elements of the Standard, such as *Schema* - technical data model descriptions - *Validators* - technical assets to validata data claiming conformance to the Standard and so on.

There is no direct handling of PIM or PSM elements in the PROF Roles vocabulary however:

- such terminology could be introduced in a Roles vocabulary extension, such as SURROUND’s linked to above
- we may infer that a "Specification" is somewhat abstract (PIM) and that "Schema" is not (PSM)

With this modelling in mind, the next section considers PIM/PSM modelling.

PROF-based PIM/PSM modelling

As stated above, PROF’s Roles for Resources don’t explicitly model PIM & PSM concerns within Standards and neither do other PROF elements (classes or properties), however, at least two forms of PIM/PSM splits were foreseen at the start of the Testbed 17 MDS project, as indicated in the Introduction's Refocussed Deliverable above:

1. Declaring the PIM and any PSMs to be parts of a single, conceptual, Standard
2. Defining the PIM and each PSM as a Standard and relating them with PROF properties

Approach 1: PIM & PSM parts in a single Standard

For Approach 1, the PIM and various PSMs of an imagined *Spatial Standard* could be declared using the following RDF conforming to PROF:

TURTLE

```
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX prof: <http://www.w3.org/ns/dx/prof/>
PREFIX role: <http://www.w3.org/ns/dx/prof/role/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX : <http://example.com/def/spatialstandard/>

<http://example.com/def/spatialstandard>
    a prof:Profile , dcterms:Standard ;
    dcterms:title "Spatial Standard"@en ;
    dcterms:description "Profile declaration of the Spatial Standard"@en ;
    prof:hasResource
        :pim-spec ,
        :td-rdf ,
        :psm-json-schema ,
        :psm-json-ld ,
        :psm-xml ,
        :psm-xml-validator ;
.

:pim-spec
    a prof:ResourceDescriptor ;
    dcterms:title "Spatial Standard Specification"@en ;
    dcterms:description "This Standard's normative, Platform Independent Model"@en ;
    dcterms:conformsTo <https://www.ogc.org/standards/modularspec> ;
    dcterms:format "application/pdf"^^dcterms:IMT ;
    prof:hasRole role:specification ;
    prof:hasArtifact "http://somewhere.com/resources/3"^^xsd:anyURI ;
.

:td-rdf
    a prof:ResourceDescriptor ;
    dcterms:title "Terms & Definitions Vocabulary"@en ;
    dcterms:description "This Standard's PIM's Terms & Definitions presented as a SKOS Vocbaulary, in RDF
(machine-redable)"@en ;
    dcterms:conformsTo <https://w3id.org/profile/vocpub> ;
    dcterms:format "text/turtle"^^dcterms:IMT ;
    prof:hasRole role:vocabulary ;
    prof:hasArtifact "http://somewhere.com/resources/12"^^xsd:anyURI ;
.

:psm-json-schema
    a prof:ResourceDescriptor ;
    dcterms:title "Spatial Standard JSON Schema"@en ;
    dcterms:conformsTo <https://datatracker.ietf.org/doc/draft-bhutton-json-schema/00/> ;
    dcterms:format "application/schema+json"^^dcterms:IMT ;
    prof:hasRole role:schema ;
    prof:hasArtifact "http://somewhere.com/resources/101"^^xsd:anyURI ;
.

:psm-json-ld
    a prof:ResourceDescriptor ;
    dcterms:title "Spatial Standard JSON-LD Schema"@en ;
    dcterms:conformsTo <https://www.w3.org/TR/json-ld11/> ;
    dcterms:format "application/ld+json"^^dcterms:IMT ;
    prof:hasRole role:schema ;
    prof:hasArtifact "http://somewhere.com/resources/36"^^xsd:anyURI ;
.

:psm-xml
    a prof:ResourceDescriptor ;
    dcterms:title "Spatial Standard XML Schema"@en ;
    dcterms:format "application/xm"^^dcterms:IMT ;
    prof:hasRole role:schema ;
```

```
      prof:hasArtifact "http://somewhere.com/resources/27"^^xsd:anyURI ;
  .

  :psm-xml-validator
      a prof:ResourceDescriptor ;
      dcterms:title "XML Validator"@en ;
      dcterms:format "application/xm"^^dcterms:IMT ;
      prof:hasRole role:validator ;
      prof:hasArtifact "http://somewhere.com/resources/55"^^xsd:anyURI ;
  .
```

Figure Figure 3 below is an informal diagram of the elements of the RDF code above.

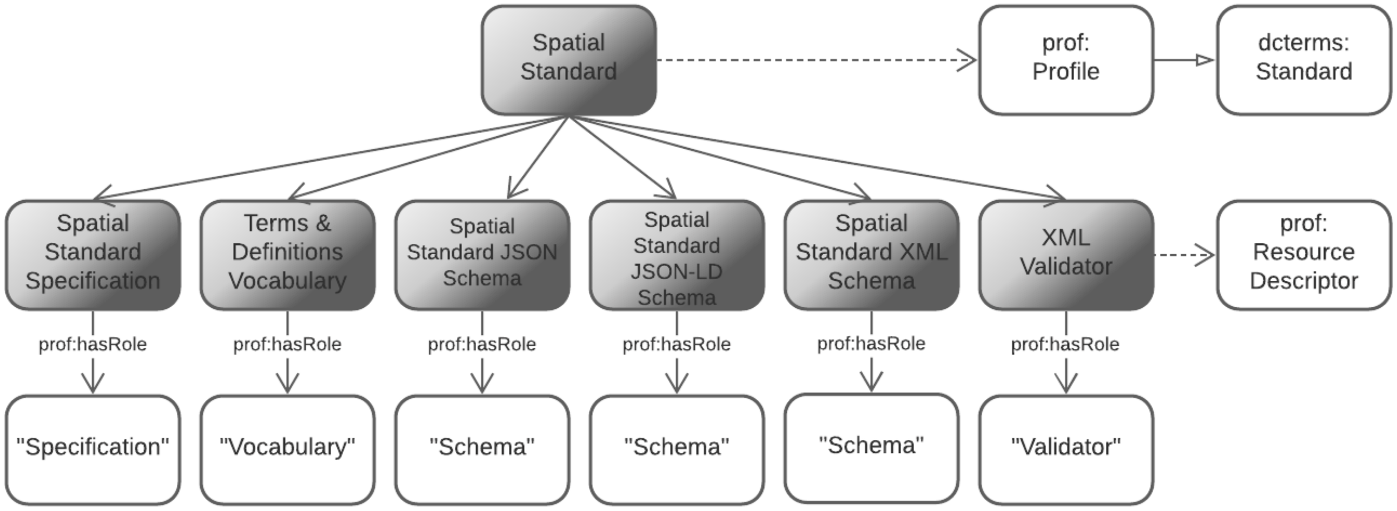

Spatial Standard
prof: Profile
dcterms: Standard
Spatial Standard Specification
Terms & Definitions Vocabulary
Spatial Standard JSON Schema
Spatial Standard JSON-LD Schema
Spatial Standard XML Schema
XML Validator
prof: Resource Descriptor
prof:hasRole
"Specification"
"Vocabulary"
"Schema"
"Schema"
"Schema"
"Validator"

Figure 3. Spatial Standard example, modelled according to PROF as parts (`Resource` *instances) within a single* `Standard`*, using figure elements from Figure 1*

In the dummy "Spatial Standard" code above, the Standard is identified as a conceptual thing of type (of OWL Class) `dcterms:Standard` and not as any particular resource, such as a Specification. A series of resources are linked to the conceptual Standard and described using PROF's `ResourceDescriptor` class which is made for this purpose. Using the PROF Roles listed in PROF Standards' elements modelling, given is a `:pim-spec` object, with role `role:specification`, a `:psm-xml-validator` object with role `role:validator` etc. Here we may infer that the Specification resource is indeed a Platform Independent Model and we may expect it to be a document (it's format is given as PDF) with at least a *Domain Model* and likely a number of other, common, elements, such as a *Terms & Definitions* section too. See Specification elements for a description of Specification element modelling.

The *Terms & Definitions* content is also available in machine-readable form (RDF, according to the _VocPub_[10] profile of _SKOS_[11]) and this is hard to classify as either a PIM or a PSM thing.

A JSON Schema (`:psm-json-schema`), a JSON-LD (`:psm-json-ld `) and an XML (` :psm-xml) Schema are all clearly PSMs. The XML Schema schema is also accompanied by an XML validator ( `:psm-xml-validator`).

With this way of modelling PIM & PSM object using PROF roles, it is relatively easy to identify a PIM-like resource - the Specification - and PSM-like resources - various Schema - but there are other elements too - the Vocabulary and the Validator - that PROF enables the identification and description of, and which are common to find in real-world Standards - but which do not easily lend themeles to PIM or PSM classification.

There are several options for formally identifying Standard elements described in this way using PROF as PIM or PSM things:

1. Create PIM & PSM PROF Roles
   - resources described using PROF may have more than one Role
   - the `:psm-json-schema` described above might have both the roles of `role:schema` and `ex:psm`
2. Create a PROF extension for PIM/PSM descriptions
   - PROF could be extended with new properties to describe PIM/PSM attributes
   - `:psm-json-schema` above could then have a PROF Role of `role:schema` and an orthogonal property indicating PIM/PSM classification, perhaps `ex:mdsRole ex:psm`

Approach 2: PIM and each PSM separate Standards

Information similar to that given in the section above could be declared for a PIM and a PSMs that are described as separate standards but related to one another. This may look as follows:

TURTLE

```
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX prof: <http://www.w3.org/ns/dx/prof/>
PREFIX role: <http://www.w3.org/ns/dx/prof/role/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX : <http://example.com/def/spatialstandard/>

<http://example.com/def/spatialstandard>
    a prof:Profile , dcterms:Standard ;
    dcterms:title "Spatial Standard"@en ;
    dcterms:description "Profile declaration of the Spatial Standard"@en ;
    prof:hasResource [
        a prof:ResourceDescriptor ;
        dcterms:title "Spatial Standard Specification"@en ;
        dcterms:description "This Standard's normative, Platform Independent Model"@en ;
        dcterms:conformsTo <https://www.ogc.org/standards/modularspec> ;
        dcterms:format "application/pdf"^^dcterms:IMT ;
        prof:hasRole role:specification ;
        prof:hasArtifact "http://somewhere.com/resources/3"^^xsd:anyURI ;
    ] ,
    [
        a prof:ResourceDescriptor ;
        dcterms:title "Terms & Definitions Vocabulary"@en ;
        dcterms:description "This Standard's PIM's Terms & Definitions presented as a SKOS Vocbaulary, in RDF
(machine-redable)"@en ;
        dcterms:conformsTo <https://w3id.org/profile/vocpub> ;
        dcterms:format "text/turtle"^^dcterms:IMT ;
        prof:hasRole role:vocabulary ;
        prof:hasArtifact "http://somewhere.com/resources/12"^^xsd:anyURI ;
    ]
.

<http://example.com/def/spatialstandard-json>
    a prof:Profile , dcterms:Standard ;
    dcterms:title "Spatial Standard in JSON"@en ;
    dcterms:description "Profile declaration of the JSON PSM of the Spatial Standard"@en ;
    prof:hasResource [
        a prof:ResourceDescriptor ;
        dcterms:title "Spatial Standard JSON Schema"@en ;
        dcterms:conformsTo <https://datatracker.ietf.org/doc/draft-bhutton-json-schema/00/> ;
        dcterms:format "application/schema+json"^^dcterms:IMT ;
        prof:hasRole role:schema ;
        prof:hasArtifact "http://somewhere.com/resources/101"^^xsd:anyURI ;
    ] ;
    prof:isProfileOf <http://example.com/def/spatialstandard> ;
.

<http://example.com/def/spatialstandard-json-ld>
    a prof:Profile , dcterms:Standard ;
    dcterms:title "Spatial Standard in JSON"@en ;
    dcterms:description "Profile declaration of the JSON PSM of the Spatial Standard"@en ;
    prof:hasResource [
        a prof:ResourceDescriptor ;
        dcterms:title "Spatial Standard JSON-LD Schema"@en ;
        dcterms:conformsTo <https://www.w3.org/TR/json-ld11/> ;
        dcterms:format "application/ld+json"^^dcterms:IMT ;
        prof:hasRole role:schema ;
        prof:hasArtifact "http://somewhere.com/resources/36"^^xsd:anyURI ;
    ] ;
    prof:isProfileOf <http://example.com/def/spatialstandard-json> ;
.


<http://example.com/def/spatialstandard-xml>
```

```
        a prof:Profile , dcterms:Standard ;
        dcterms:title "Spatial Standard in XML"@en ;
        dcterms:description "Profile declaration of the XML PSM of the Spatial Standard"@en ;
        prof:hasResource [
            a prof:ResourceDescriptor ;
            dcterms:title "Spatial Standard XML Schema"@en ;
            dcterms:format "application/xm"^^dcterms:IMT ;
            prof:hasRole role:schema ;
            prof:hasArtifact "http://somewhere.com/resources/27"^^xsd:anyURI ;
        ] ,
        [
            a prof:ResourceDescriptor ;
            dcterms:title "XML Validator"@en ;
            dcterms:format "application/xm"^^dcterms:IMT ;
            prof:hasRole role:validator ;
            prof:hasArtifact "http://somewhere.com/resources/55"^^xsd:anyURI ;
        ] ;
        prof:isProfileOf <http://example.com/def/spatialstandard> ;
    .
```

Figure Figure 4 below is an informal diagram of the elements of the RDF code above.

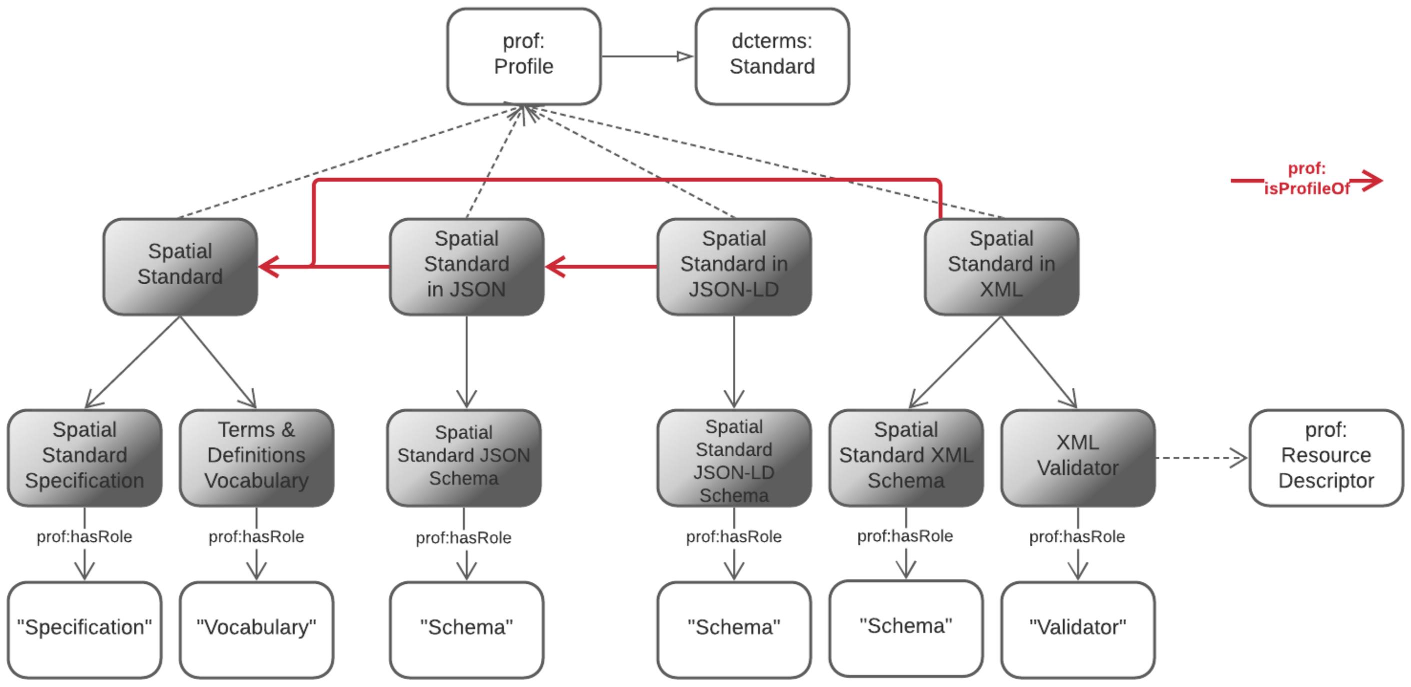

prof: Profile
dcterms: Standard
prof: isProfileOf
Spatial Standard
Spatial Standard in JSON
Spatial Standard in JSON-LD
Spatial Standard in XML
Spatial Standard Specification
Terms & Definitions Vocabulary
Spatial Standard JSON Schema
Spatial Standard JSON-LD Schema
Spatial Standard XML Schema
XML Validator
prof: Resource Descriptor
prof:hasRole
"Specification"
"Vocabulary"
"Schema"
"Schema"
"Schema"
"Validator"

Figure 4. Spatial Standard example, modelled according to PROF as multiple `Standard` *instances with profile relations between them, using figure elements from Figure 1*

Above, four Stadards are declared - `http://example.com/def/spatialstandard` , `http://example.com/def/spatialstandard-json` , `http://example.com/def/spatialstandard-json-ld` & `http://example.com/def/spatialstandard-xml` - with the second and fourth being shown to be profiles of the first (the JSON & XML PSMs of the PIM) and the third to be a profile of the second (the JSON-LD PSM of the JSON PSM). Currently there is only one semantic way of relating muliple Profiles/Standards to one another in PROF and that is by using `prof:isProfileOf` , as is done in these cases. Note that we can learn, with this way of working, that there is a relationship between the JSON & JSON-LD PSM forms.

This method of characterisation follows the ISO TC 211 pattern of having a so-called Dash One ("-1") conceptual (PIM) stnadard with Dash two, Three etc ("-2", "-3" etc.) for various encodings.

For Standards modelled in this way, PIM/PSM identification could be achieved by:app-name:

1. Using method 2. from the previous section
   - "Create a PROF extension for PIM/PSM descriptions"
   - applied to Standards, not resources described within them
2. Specialization of the `prof:isProfileOf` property
   - perhaps a property such as `ex:isPlatformSpecificProfileOf` could be used to link a PSM to a PIM

The method 1. above - "Create PIM & PSM PROF Roles" - could not be used in this modelling as the Roles apply to parts within a Standard, not across Standard instances. There is no sensible way to defined Roles in an Open World modelling of multiple Standard instances since there is no defined envelope in which all the Standard instnaces exist, unlike resources within a Standard thus there is no way to indicate what the role of a Standard is with respect to.

Approach review

Both the approaches above have merit. The first approach - Parts - is easier to implement but perhaps more constrained: all the PIM & PSM elements must conceptually be within a single Standard.

It may be that Approach 2 is required for real world implementation, given the fitful nature of standards development (not all the parts, PIMs & PSMs etc. are developed in a coordinated manner).

The complete examples of modelled Standards in this report in GeoSPARQL Standard Example and [_anz_3d_cadastre_standard_example] use Approaches 1 and 2 respectively. This is due to requirements set outside of this Testbed activity which required GeoSPARQL to be one Stnadard and ANZ 3D Cad to deliver distinct *Conceptual* and *Implementation* Stnadards which are essentially equivalent to PIMs and PSMs.

Specification elements

To sensibly model Standards according to PROF, we need to be able to model the parts of a Standard’s Specification.

The Specification document of a Standard is often considered to be THE Standard. PROF modelling deliberately separates the conceptual Stnadard from the concrete Specification.

A conjecture is posed which is that:

> *Different parts of a Standard’s Specification are best modelled with different model forms*

For example, the conceptual area of the Standard’s concern, which is termed here a *Domain Model*, is likely best modelled with a form of ontology, perhaps an OWL model or a UML Class model, however the *Terms & Definitions* section often found in Specifications is likely best modelled with a taxonomy model, perhaps [SKOS].

The following Figure indicates the parts in a number of Specifications and proposed forms of model to model their content.

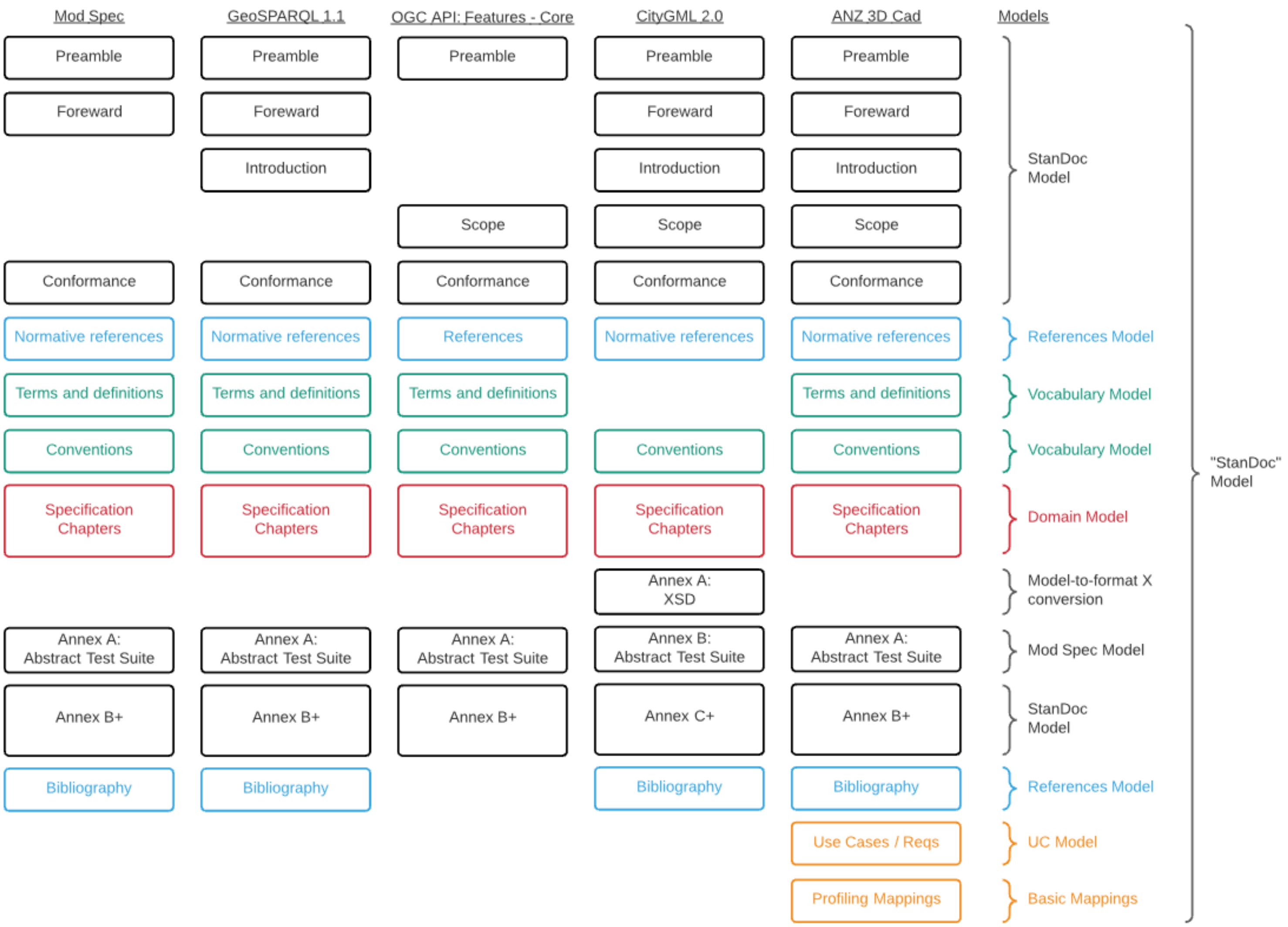


Figure 5. Part the Specifications documents of several Standards

Specification part models

Document Structure

A so-called *StanDoc* ("Standards Document") model is indicated in the figure above to model the overall structure of Specifications and several parts within them. SURROUND has used the Semantic Publishing and Referencing Ontologies (SPAR Ontologies)[12] to model structured documents such as Standard thus SPAR Ontologies are proposed as a candidate StanDoc model.

SPAR is a set of related but orthogonal models originally designed for academic and technical publication structural and semantic modelling. Simple SPAR modelling describes elements familiar to all structured document producers: *Section*, *Table of Content*, *Title* etc.

Proposing the use of a semantic model for document structure is a more formal way of defining Standard content than the use of a set of templates or descriptions of document elements, as the OGC currently uses. Ontology *Shapes* may be defined in a profile of SPAR tuned to the OGC's Standard's needs. This profile, which has not been created in this Testbed, could be made to meet current OGC requirements and even map to non-Semantic document systems, such as ASCIIDOC templating.

A major benefit to using a semantic document structural model is that with such a model implemented, Specifications' structural elements become data similarly to their Domain Model and other parts. Use then of ontologies for document structure is the ultimate Model-Driven Standard step. Below is shown an extract of the ANZ 3D Cad Standard's

Specification’s SPAR modelling. The content follows the commonly seen OGC Specification structure.

TURTLE

```
<https://example.com/anz-3d-cad>
  a frbr:Work , owl:Ontology ;
  co:firstItem _:b626767 ;
  co:item [
      a co:ListItem ;
      co:index 1 ;
      co:itemContent _:b626767 ;
    ] ;
  ...
  co:item [
      a co:ListItem ;
      co:index 9 ;
      co:itemContent _:b865408 ;
    ] ;
  po:contains :preamble ;
  po:contains :scope ;
  po:contains :related-domains ;
  ...
  po:contains :annex-b ;
  po:contains :annex-c ;
  dcterms:published "2021-11-23"^^xsd:date ;
  dcterms:title "ICSM Conceptual Model for 3D Cadastral Survey Dataset Submissions"@en ;
  frbr:embodiment [
      a prof:ResourceDescriptor ;
      dcterms:format "text/html" ;
      rdfs:comment "OGC-style Specification online"@en ;
      rdfs:label "PDF of the ICSM Conceptual Model for 3D Cadastral Survey Dataset Submission"@en ;
      prof:hasArtifact "http://cad-spec.s3-website-ap-southeast-2.amazonaws.com/"^^xsd:anyURI ;
    ] ;
.
```

Terms & Definitions Model

Specifications often contain *Terms & Definitions* and sometimes *Conventions* sections. Both of these types of sections are easily modelled using a semantic vocabulary model such as [SKOS]. Vocabularies of terms & definitiosn are both critical for Specifications, easy to model in something like SKOS but hard to model well in *Domain Model* systems such as UML Class models.

Double modelling can take place for these sections where defined terms and conventions are both modelled as document structural elements using SPAR Ontologies and also as vocabulary concepts using SKOS which allow for vocab content use in isloation - outside a document. Having two forms of modelling, sibe-by-side, is entirely possible in SW modelling and specifications such as *Content Negotiation by Profile* [CONNEGP] indicate how to indicate when one or other form of content’s modelling should be used.

An intention of the OGC Naming Authority is to present a set of definitions, via their Definitions Server[13], that both includes all the defined *Terms & Definitions* and *Conventions* from OGC Specifications and makes them available for reuse. So, if a Specification needed to contain a definition for the term *Boundary*, it would contain the ASCIIDOC content:

ASCCIDOC

```
==== Boundary

Set that represents the limit of an entity. Note 1 to entry: Boundary is most commonly used in the context of
geometry, where the set is a collection of points or a collection of objects that represent those points. In other
arenas, the term is used metaphorically to describe the transition between an entity and the rest of its domain of
discourse

[ ISO 19107:2003, 4.4]
```

It could, instead, use the SKOS Concept from the OGC Definitons Server at https://www.opengis.net/def/CaLAThe/4.0/Boundary (https://www.opengis.net/def/CaLAThe/4.0/Boundary):

```
TURTLE
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .

<https://www.opengis.net/def/CaLAThe/4.0/Boundary>
    a skos:Concept ;
    skos:broader <https://www.opengis.net/def/CaLAThe/4.0/LandDivision> ;
    skos:definition "Set that represents the limit of an entity. Note 1 to entry: Boundary
        is most commonly used in the context of geometry, where the set is a collection of
        points or a collection of objects that represent those points. In other arenas, the
        term is used metaphorically to describe the transition between an entity and the
        rest of its domain of discourse."@en ;
    dcterms:source "https://www.iso.org/standard/26012.html"^^xsd:anyURI ;
    skos:narrower <https://www.opengis.net/def/CaLAThe/4.0/BoundaryMark> ;
    skos:prefLabel "Boundary"@en ;
.
```

That SKOS RDF code could easily be converted to the equivalent ASCIIDOC code for display in a document form of the Specification but the use of a Concept in the NA’s Definitions Server makes the defined term far more machine-readable and more accessible and tracable generally.

OGC Specification' *Terms & Definitions* and *Conventions* elements usually support *Domain Model* elements which can be said to depend upon their definitions. Where the *T&D* & *Convention* elements, in turn, depend on Naming Authority or other defined terms, we may have relations as per Figure 7 below.

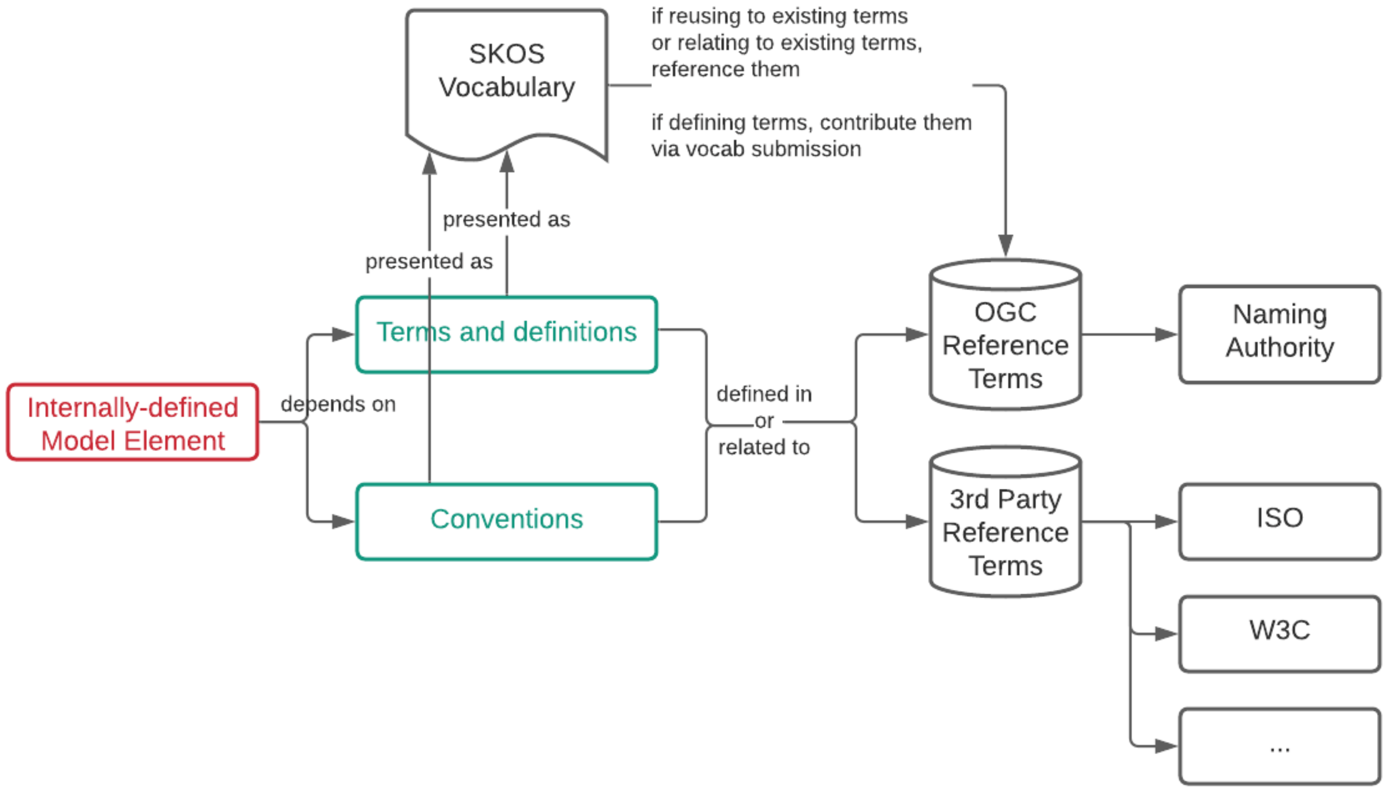

SKOS Vocabulary
if reusing to existing terms or relating to existing terms, reference them
if defining terms, contribute them via vocab submission
presented as
presented as
Terms and definitions
Internally-defined Model Element
depends on
Conventions
defined in or related to
OGC Reference Terms
Naming Authority
3rd Party Reference Terms
ISO
W3C
...

Figure 6. Conceptual relations between Domain Model elements ("Internally-Defined Model Elements") and OGC and external reference terms

This element of Model Driven Standards moves from driving a particular Standard's section - *Terms & Definitions* within a *Specification* from being driven by a local model, i.e. for this Standard only, to the section being driven by a combination of new and previously definied elements from a larger, Standards' baseline set of models.

References Model

Most OGC (and ISO and W3C) Specifications contain normative and non-normative reference sections, as shown in Figure 5. The SPAR Ontologies contain a detailed handing of reference modelling. The same SPAR reference handing can also be used for Specification's Bibliographies.

Figure 6 is a conceptual diagram of how different parts of a Specification reference and define *Terms & Definitions*, *Normative References* and *Bibliography* elements.

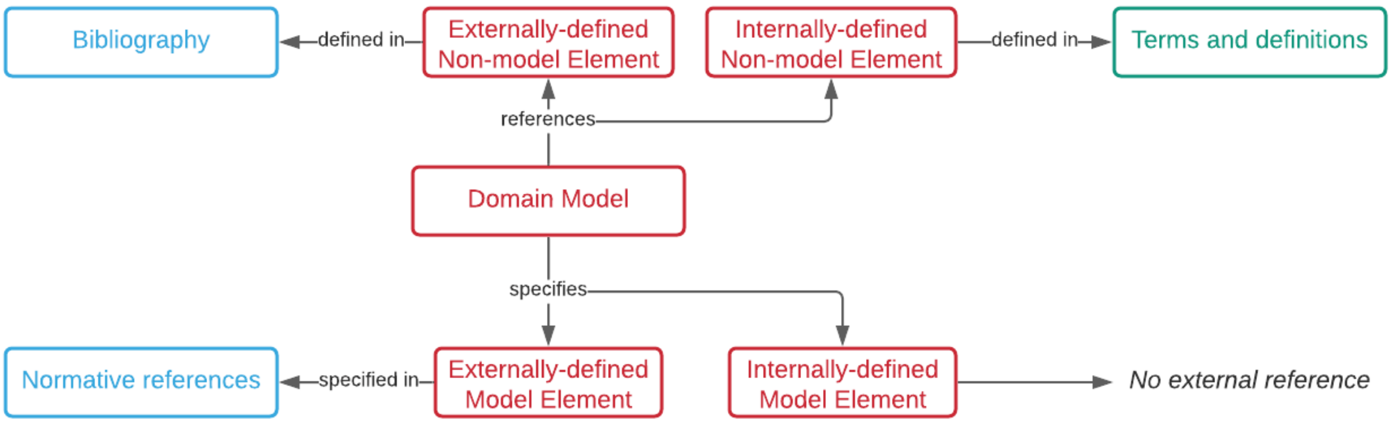


Figure 7. Conceptual modelling of how a Specification's Domain Model elements relate to Terms & Definitions, Normative References and Bibliography section elements.

Modelling such as this, along with its formal realisation in SPAR Ontologies, would machine-readable linking between OGC Specifications elements that have previously not been linked in this way: currently OGC Specifications' do not link *Domain Model* elements to other definitional Specification elements. Use of reference modelling in this way will expand the envelope of the model driven elements of a Standard.

Test Suite (Requirements) Model

Many OGC Specifications contain an Abstract Test Suite and there are many OGC Specifications that contain strictly structured Abstract Test Suide elements, for example the OGC API *Features* core specification [FEATURES]. *Features* defines "Conformance Classes" and "Abstract Tests", all identified with URIs with relationships - the Test Suite / Conformance model - taken from the OGC's "ModSpec" (The Specification Model — A Standard for Modular specifications) [08-131r3]. The model is presented in UML and could easily be presented in OWL. This has not been done in this Testbed however the GeoSPARQL 1.1 release contains not only "Conformance Classes" and "Abstract Tests" presented in a similar way to *Features* but also contains a SKOS vocabulary of the items too[14], so it is close to this modelling. It would require little effort to build the sorts of Requirements / Tests listing seen in OGC specifications from this vocabulary or from a more semantically strong OWL version of this content.

One example of a Requirement and its related Conformance Test from GeoSPARQL 1.1's vocabulary of them is given in the listing below:

TURTLE

```
reqs10tve:rcc8-spatial-relations
    a spec:Requirement ;
    skos:prefLabel "GeoSPARQL 1.0 Requirement: RCC8 Spatial Relations"@en ;
    skos:definition "Implementations shall allow the properties geo:rcc8eq, geo:rcc8dc, geo:rcc8ec, geo:rcc8po,
geo:rcc8tppi, geo:rcc8tpp, geo:rcc8ntpp, geo:rcc8ntppi to be used in SPARQL graph patterns" ;
.

conf10tve:rcc8-spatial-relations
    a spec:ConformanceTest ;
    skos:prefLabel "GeoSPARQL 1.0 Conformance Test: RCC8 Spatial Relations"@en ;
    skos:definition "Verify that queries involving these properties return the correct result for a test dataset"
;
    spec:testPurpose "check conformance with this requirement" ;
    spec:testType spec:Capabilities ;
    rdfs:seeAlso reqs10tve:rcc8-spatial-relations ;
.
```

While above one Requirement is indicated by one Conformance test, other Specifications could contain non-1:1 relations.

GeoSPARQL 1.1 also groups Requirements so that *Conformance Classes* are created as collections of Confromance Tests. The tests for the GeoSPARQL 1.1 "core" *Confronace Class* are listed below:

```turtle
confs11:core
    a spec:ConformanceClass, skos:Collection ;
    skos:member conf11core:feature-collection-class ,
                conf11core:spatial-object-class ,
                conf11core:sparql-protocol ,
                conf11core:spatial-object-collection-class ,
                conf11core:spatial-object-properties ;
    skos:prefLabel "GeoSPARQL 1.1 Conformance Class: Core"@en ;
.
```

Most of the Conformance Test / Class and Requirements modelling is just strightforward [SKOS]-basd vocbaulary modelling of concepts with a few special properties to indicate tests purpose (`spec:testPurpos`) and similar.

Domain Model

UML to OWL

Domain models within the OGC often reference a "baseline" of ISO models, such as ISO 19107 *Geographic information — Spatial schema* [ISO_19107_2019] and GeoSPARQL 1.1 and the ANZ 3D Cad models are no different: GeoSPARQL's *Standard Structure* introductory text describes its use of these models[15].

The ISO's TC 211 presents not only the consolidated UML versions of these models as a "harmonised model" but also a consolidated OWL version of them too via TC 211's *Group on Ontology Maintenance* (GOM)'s public code repository [16]. While this consolidated OWL information is subject to many questions regarding its exhibition of _Ontology Design Patterns_[17] or, generally, "Best Bractices", a UML to OWL coversion for multiple (ISO) standard's Domain Models is demonstrated.

There are many UML-to-OWL academic papers and studies, for example:

- Brockmans, Sara, Raphael Volz, Andreas Eberhart, and Peter Löffler. "Visual Modeling of OWL DL Ontologies Using UML." In The Semantic Web – ISWC 2004, edited by Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, 3298:198–213. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. https://doi.org/10.1007/978-3-540-30475-3_15 (https://doi.org/10.1007/978-3-540-30475-3_15).
- Jetlund, Onstein, and Huang. "Adapted Rules for UML Modelling of Geospatial Information for Model-Driven Implementation as OWL Ontologies." ISPRS International Journal of Geo-Information 8, no. 9 (August 22, 2019): 365. https://doi.org/10.3390/ijgi8090365 (https://doi.org/10.3390/ijgi8090365).

The first publication was made early in the days of OWL modelling but remains relevant. More recent, simple UML-to-OWL mappings/models exist[18], but the fundamentals remain unchanged. These sorts of mappings are the types used to generate GOM Semantic Web data.

The second publication uses essentially current ISO TC 211 Standards' UML (circa 2019) and concludes that "conversion challenges [from UML to OWL] are addressed by adding more semantics to UML models: global properties and reuse of external concepts". This author agrees with the particular solution of that paper and feels that a next-generation of automatically-produced OWL from the current TC 211 UML could be made that exhibits many more Ontology Design Patterns.

OWL to UML

The GeoSPARQL (1.0 and 1.1) and ANZ 3D Cad standards have Semantic Web-first models, that is they have models presented initially as OWL ontologies and then, informally, UML diagram versions of them, or parts of them. GeoSPARQL 1.1's main Domain Model diagram, from its Specification document, is presented below in Figure 7: it is not a UML diagram but an informal OWL diagram (See [_informal_owl_diagrams] for information on this type of diagramming).

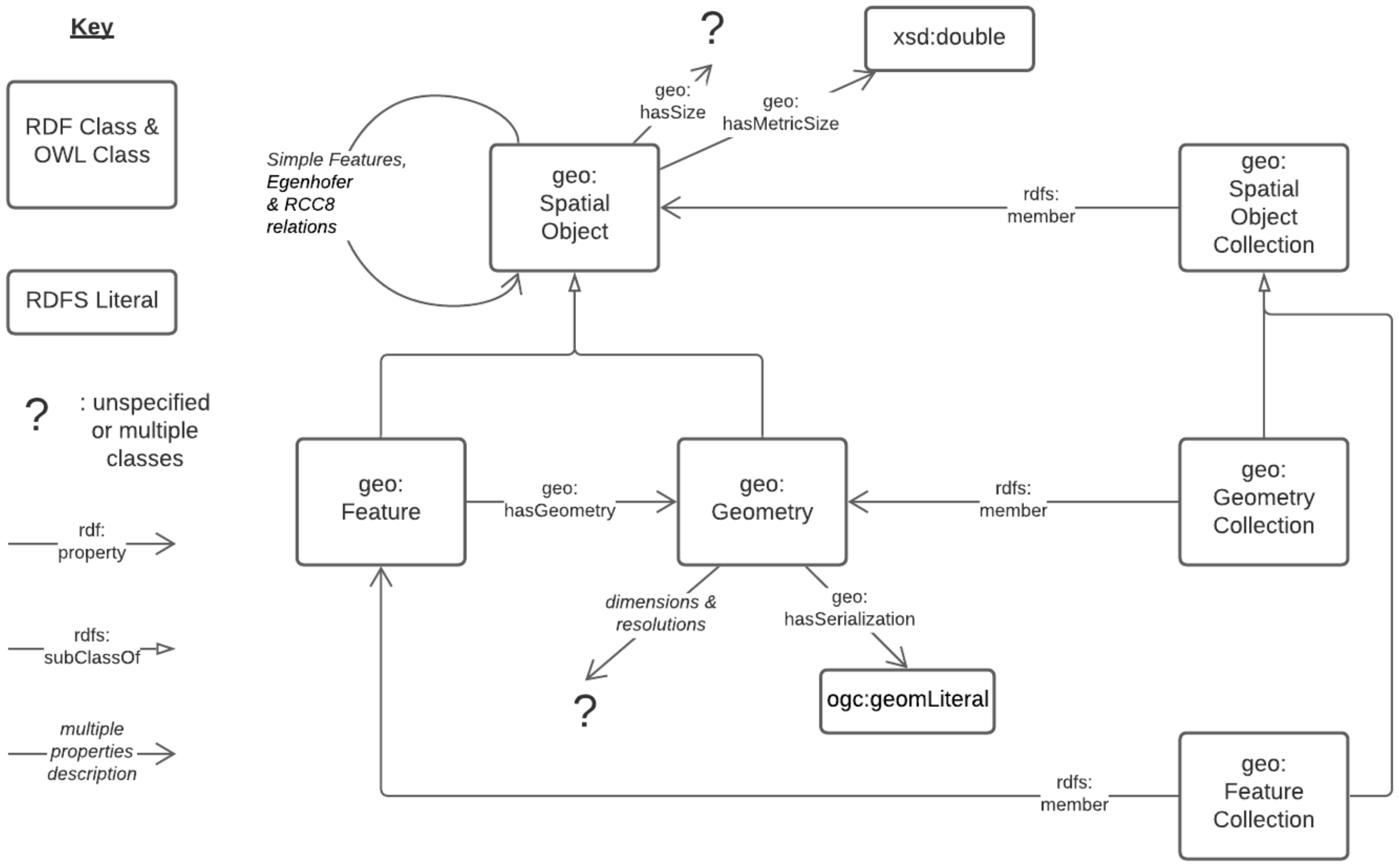


Figure 8. An informal model diagram overview of GeoSPARQL 1.1's Domain Model. After https://opengeospatial.github.io/ogc-geosparql/geosparql11/spec.html#_core (https://opengeospatial.github.io/ogc-geosparql/geosparql11/spec.html#_core)

A simple UML diagram of a model presented at the level of detail of the figure above is easy to make, and such diagrams are presented as the main way to communicate OWL model structures for some well-known ontologies (see [_uml_owl_diagrams] for examples). For work within this testbed, basic UML diagramming was easily generated for a couple of modelling scenarios, see [_uml_owl_diagrams].

Tools that can utomatically generate UML diagrams from OWL include:

- OWLGrEd - http://owlgred.lumii.lv/ (http://owlgred.lumii.lv/)
- TopBraid Composer ontology editor - https://www.topquadrant.com/composer/featureClassDiagram.html (https://www.topquadrant.com/composer/featureClassDiagram.html)
- Protege ontology editor, via plugins - https://protegewiki.stanford.edu/wiki/OWL2UML (https://protegewiki.stanford.edu/wiki/OWL2UML)

It is likely, but unproved, that the UML diagrams for Domain Models similar to those in the ISO's Technical Committee 211's remit could be automatically produced from either GOM OWL ontologies or from slight variants of them.

Unlike UML-native objects: objects modelled initially in OWL but then expressed in UML would retain the original richer semantics of the OWL form allowing for better integration with other Semantic Web-based technologies such as SHACL [SHACL]-based data validators and SKOS [SKOS]-based vocabulary definitions. Following the reasoning of Jetlund *et al.* quoted above in UML to OWL regarding specifying a profile of UML to form OWL, a profile of OWL could easily be made that forms UML of the sort required by OGC or ISO stnadards. Such a profile could be implemented using [PROF].

Domain modelling

Regarding the specific modelling required for OGC/ISO standards: the ISO standard's "baseline", as referenced above, creates OWL classes of types recognisable to users of the UML models versions of their standards, given that the OWL classes are auto-generated from the UML. For instance, ISO 19160 (Addressing) [ISO_19160_2015]'s OWL version presents an `Address` class[19] and references classes from more fundamental models, such as ISO 19107's GM_Object[20].

It is straightforward to extend this baseline for other Standards, and where the examples in [_owl_diagramming] show GeoSPARQL's `geo:Feature` as a more abstract class (superclass) for their specific classes, links to ISO 19107's `GM_Object` exist, given that GeoSPARQL indicates `geo:Feature` is a subclass of `GM_Object` [21].

Other Models

Several elements not often seen in existing OGC Specifications are present in the ANZ 3D Cadastre Standard's Specification. Most prominant is a Use Cases Annex whose content is modelled with a simple Use Case Ontology created for this purpose[22].

Use Case modelling is well-known in UML but, as far as the author is aware, no Specifications other than the ANZ 3D Cadastre Standard's model Use Cases in such a way that Domain Model elements may be linked to the Use Cases that motivated their implementation.

While it would be possible to implement a UML-based, non-semantic, Use Case model within a Specification, use a Use Case ontology within the SW envelope of models allows for Domain Model / Use Case model element refereincing using SPAR Ontology referencing, as per References Model. Figure 7 shows a conceptual linking for this.

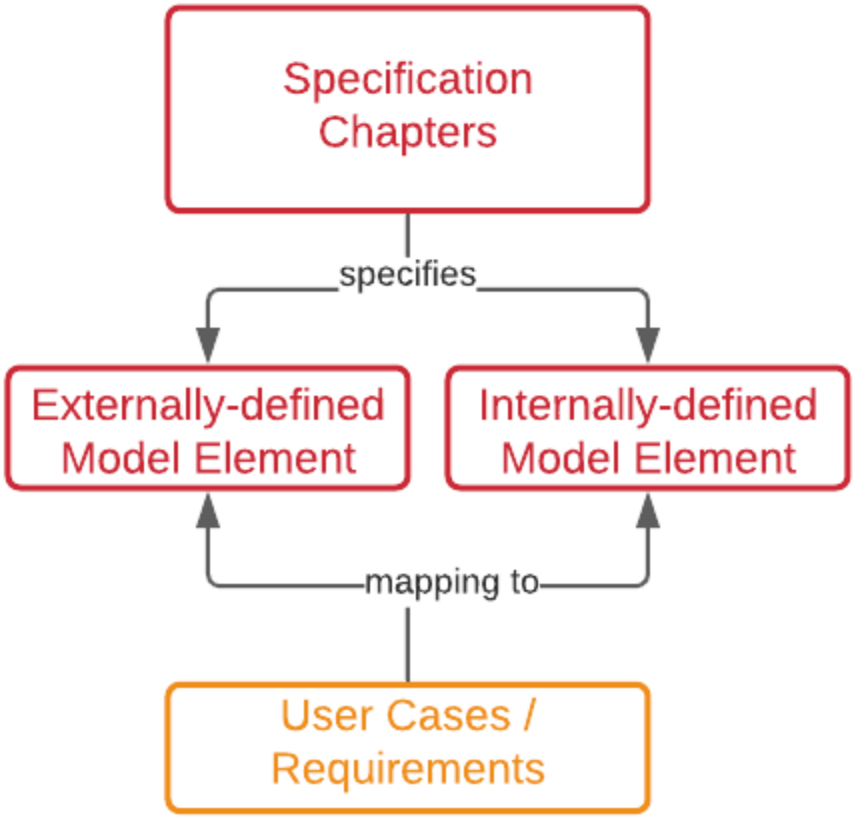
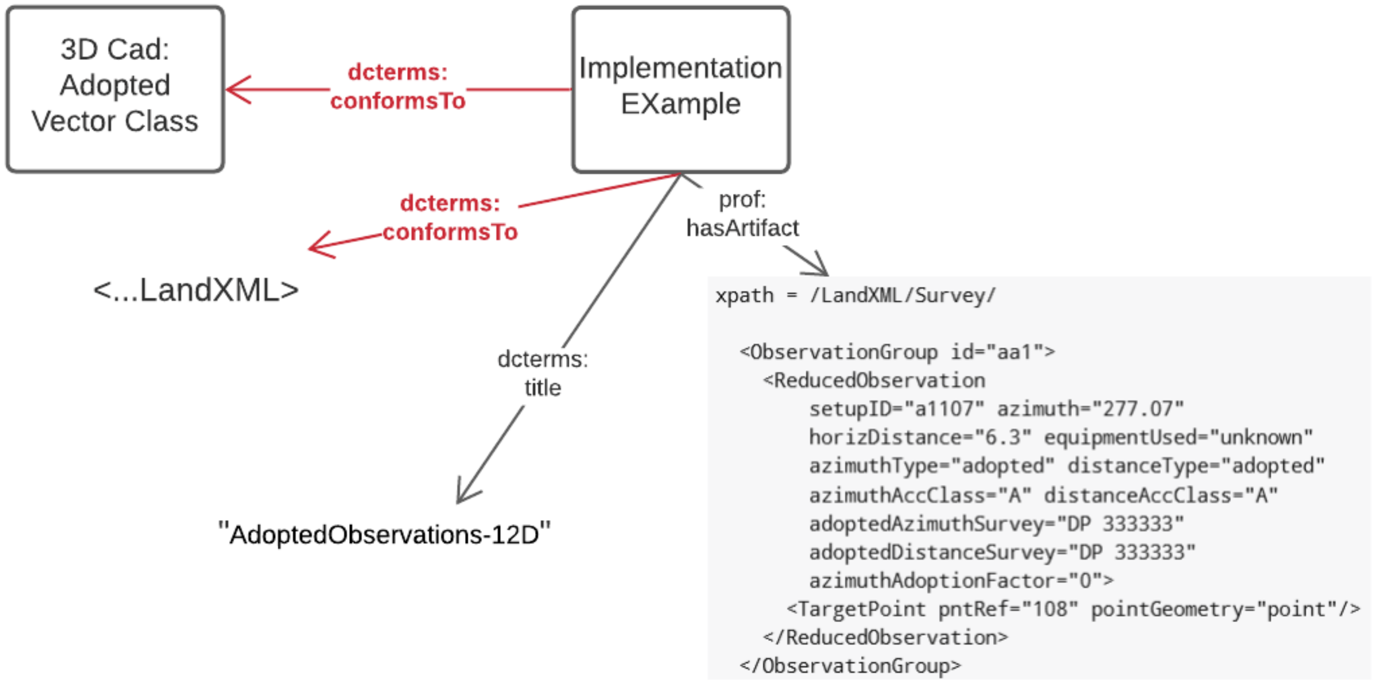

Specification Chapters
specifies
Externally-defined Model Element
Internally-defined Model Element
mapping to
User Cases / Requirements

Figure 9. Conceptual modelling of how a Specification's Domain Model elements relate to Use Case / Requirements section elements.

Implementation Examples Modelling

The ANZ 3D Cad Standard presents selectable variants of its Specification's content for different jurisdictional users. The jurisdictions all have different model terminology and implementation patterns but the different elements have been mapped to the Standard's core conceptual "canonical" model. A jurisdictional user of the Specification is able to turn on and off individual jurisdictionals' elements in the document to see only the forms they wish to. To facilitate this, the ANZ 3D Cad Specification contains multiple, different, *Implementation Examples* of the Standard's Domain Model for the jurisdictions. While it would be possible to simple link multiple Implementation Examples to Domain Model elements and then to tag individual ones as being relevant for one or more jurisdictions, the Specification actually contains mapping from Domain Model elements to Implementation Example object which are then described with "Resource Descriptor" properties. Such properties, taken from the *Profiles Vocabulary* [PROF], include `dcterms:conformsTo` which allows for the indication of Standards to which the Implementation Example object conforms. Clearly the Implementation Example must, and they all do, conform to the concepts in the Domain Model but then they also conform to other syntactic or diagrammatic or schema models, such as JSON, UML Package Diagram or others that then allow users to understand the *form* of the Implementation Example - what language, shcema, diagram etc. it uses to implement the Domain Model.

In Figure 8 and Figure 9 below, the `AdoptedVector` and `LandPropertyUnit` classes of the ANZ 3D Cad Standard's Domain Model are shown with Implementation Examples that conform to them and which also conform to other specifications - *LandXML* and *_ClassContextDiagram* respectively.



```
xpath = /LandXML/Survey/

  <ObservationGroup id="aa1">
    <ReducedObservation
        setupID="a1107" azimuth="277.07"
        horizDistance="6.3" equipmentUsed="unknown"
        azimuthType="adopted" distanceType="adopted"
        azimuthAccClass="A" distanceAccClass="A"
        adoptedAzimuthSurvey="DP 333333"
        adoptedDistanceSurvey="DP 333333"
        azimuthAdoptionFactor="0">
      <TargetPoint pntRef="108" pointGeometry="point"/>
    </ReducedObservation>
  </ObservationGroup>
```

Figure 10. The ANZ 3D Cad model's `AdoptedVector` *class with a LandXML Implementation Example*

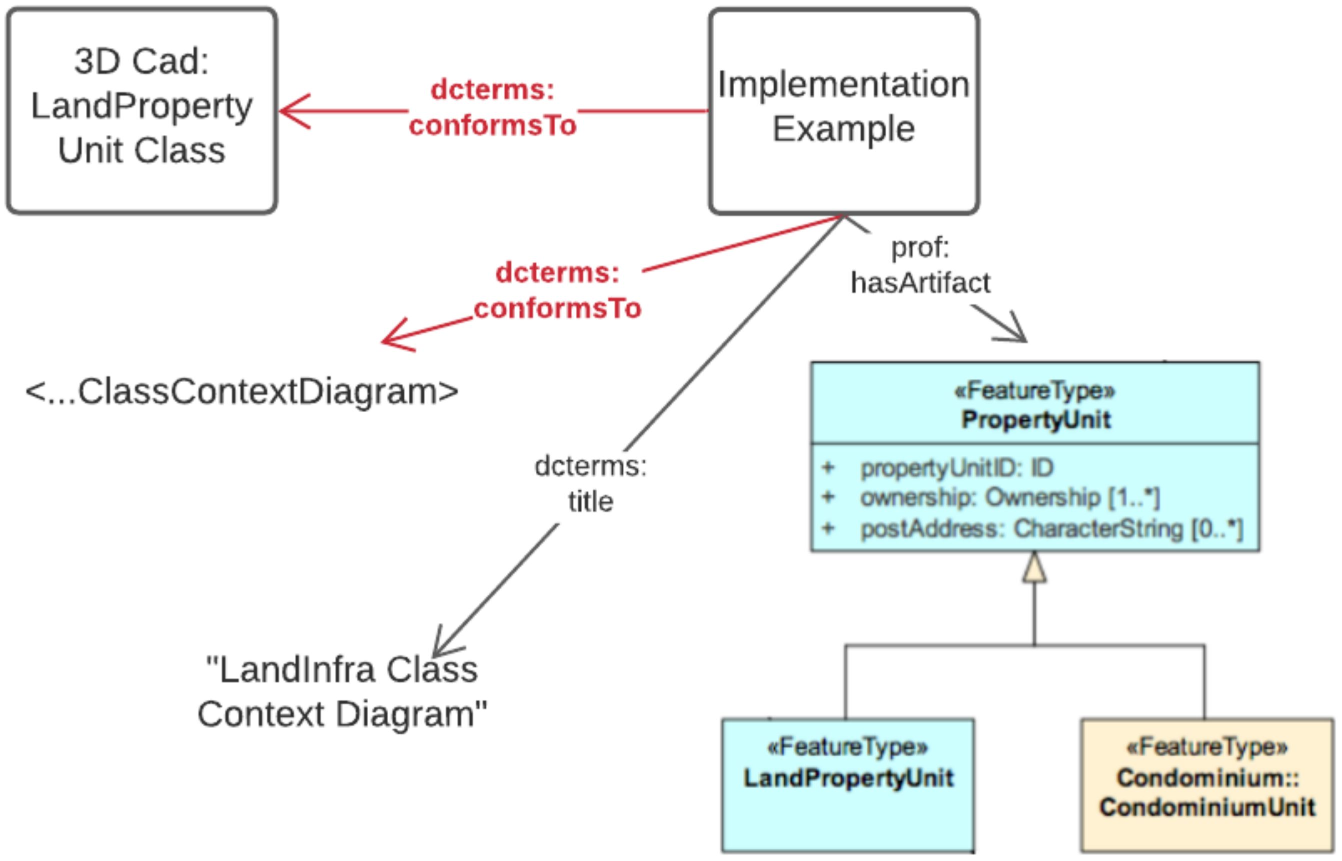

3D Cad: LandProperty Unit Class
dcterms: conformsTo
Implementation Example
dcterms: conformsTo
prof: hasArtifact
<...ClassContextDiagram>
dcterms: title
"LandInfra Class Context Diagram"
«FeatureType» PropertyUnit
+ propertyUnitID: ID
+ ownership: Ownership [1..*]
+ postAddress: CharacterString [0..*]
«FeatureType» LandPropertyUnit
«FeatureType» Condominium:: CondominiumUnit

Figure 11. The ANZ 3D Cad model's `LandPropertyUnit` *class with a documentation diagram conforming to a ClassContextDiagram Standard*

Differentiating types of conformance are not possible in the *Profiles Vocabulary* [PROF] thus no distinction is made between the way in which the Implementation Examples in the figures above conform to elements in the Domain Model and the other Standards that they conform to. Differentiating types of conformance, and if such a thing is sensible, has simply not been determined.

GeoSPARQL Standard Example

The GeoSPARQL 1.1 Specification is online at:

- https://opengeospatial.github.io/ogc-geosparql/geosparql11/spec.html (https://opengeospatial.github.io/ogc-geosparql/geosparql11/spec.html)

It is auto-built from ASCIIDOC source files located at https://github.com/opengeospatial/ogc-geosparql/tree/master/1.1/spec (https://github.com/opengeospatial/ogc-geosparql/tree/master/1.1/spec) using standard ASCIIDOC tooling. While it is conceptually a Model Driven Standard in that many parts of the Specification are backed by technical models, such as the Domain Model, Terms & Definitions vocabulary, Reqlirements listing etc., it is not auto-Model Driven.

It is planned that GeoSPARQL 1.2, to be created in 2022, will be auto-Model Driven with the document structure and many elements being auto built from data files (OWL ontologies).

The full listing of GeoSPARQL 1.1 elements, including the Specification document and the technical artifacts for the Terms & Definitions sections, etc., are listed in the GeoSPARQL 1.1 Profile Declaration at:

- https://opengeospatial.github.io/ogc-geosparql/geosparql11/profile.html (https://opengeospatial.github.io/ogc-geosparql/geosparql11/profile.html)

Note that this declaration is also specified in machine-readable form (RDF data, according to [PROF]) at https://opengeospatial.github.io/ogc-geosparql/geosparql11/profile.ttl (https://opengeospatial.github.io/ogc-geosparql/geosparql11/profile.ttl).

The automated build process for the Specification's ASCIIDOC to HTML conversion and for the generation of other documenation, such as the Profile Declaration, is specified in GeoSPARQL 1.1's infra-codeing scripts online at https://github.com/opengeospatial/ogc-geosparql/tree/master/.github/workflows (https://github.com/opengeospatial/ogc-geosparql/tree/master/.github/workflows).

ANZ 3D Cadastre Standard Example

The ANZ 3D Cadastre Specification is online in several forms at:

- http://cad.surroundaustralia.com/ (http://cad.surroundaustralia.com/)

It is auto-built from data files (OWL ontologies) with tooling converting OWL ontologies to ASCIIDOC intermediate files and then converting those to HTML files. This process is more fully detailed in [_omds_specification_production].

oMDS Specification Production

oMDS is an acronym for the *(ontology) Model-Driven Standards* tool used to create the ANZ 3D Cadastre Specification.

Specification variants

Several forms of the ANZ 3D Cadastre Specification are produced from model data and are online at:

- http://cad.surroundaustralia.com/ (http://cad.surroundaustralia.com/)

The multiple versions here are:

- a traditional Specification document
    - with normative/non-normative element selectors
    - with profile content selectors
- Specfciation document with embedded feedback forms
    - to allow non-technical readers of the Specification to supply feedback, per Specification element
- Specification with experimental elements
    - this version contains a 3D data viewer and other ontilie functionality extensions on top of what is usually see within a Specification

Build sequence

All version of the Specification are built in a similar way, which is:

1. OWL Ontology content stored in a database is read by the oMDS tool and converted to in-memory Python objects
    - each part of the AND 3D Cad model is stored in a separate *graph*, that is an isolated part of the overall ontology used. Graph here are akin to the Postgre relational databases' notion of a *schema*
2. In-memory objects are serialized to ASCIIDOC of a similar form to most OGC standards, including GeoSPARQL 1.1
    - mapping between ontology models for the Specifications various sections, e.g. Use Cases, Domain Model, Terms & Definitions are currently retained within the oMDS tool but may be externalised, for potential wider use, later
    - mappings take the form of Python classes, instances of which are generated by querying the ontologies using SPARQL queries and then serialized as needed by calling a `to_asciidoc()` function that templates Python object instance variables into ASCIIDOC
3. ASCIIDOC is converted to HTML or PDF using the standard of ASCIIDOCtor tool[23]
    - the next generation of the oMDS tool will incorporate a Python-based ASCIIDOC to HTML converter, rather than relying on the external ASCIIDOCtor tool
4. Build parameters determine which, of several, HTML post-processing steps are run on the ASCIIDOCtor-generated HTML
    - these post processing steps allow for the inclusion of functionality beyond the standard ASCIIDOC offering for features such as:
        - normative/non-normative element selectors, juristdctional profile selectors, embedded feedback forms, element highlighting (for incomplete development stages) etc.
5. Infracode tools place the auto-generated Specification documents online
    - at http://cad.surroundaustralia.com/ (http://cad.surroundaustralia.com/)

Much of the OWL-to-ASCIIDOC conversion approach was derived from the open source OWL Ontology documentation tool pyLODE[24].

Build element isolation

Each element of the oMDS-generated Specification can be built in isolation or in and ensemble so that, during testing, a single part may be worked on. This is partly enabled by the ability for ASCIIDOC documents to be generated either in isolation or via compilation through import statements. It is also partly enabled by the isolation of conceptual models for different parts, for example, the Terms & Definitions section, while referencing elements in other sections, forms a complete SKOS vocabulary by itself. Likewise, the Use Case information in the Specification forms complete instances of the SURROUND Use Case Ontology[25] even though the Use Case elements (Requirements etc.) are referenced by elements in the Domain Model.

oMDS next steps

Currently the oMDS tool is purpose-built for the ANZ 3D Cad Specification, however it has always been intended that the tool will be generalised for use with other (ontology) Model-Driven Standards. The follollowing developement spes are exected for the oMDS tool in the first hald of 2022 for at least the auto-generation of the GeoSPARQL 1.2 Specification and perhaps other Specifications too:

- ensure all *all* ASCIIDOC content is produced from OWL data using in-memory Python objects only
  - currently this is mostly the case, however some Python ASCIIDOC templating is still also used
- use of Python ASCIIDOC to HTML conversion
  - to 'inline' this conversion step, rather than rely on an external (to the oMDS tool) converter
- extraction of ASCIIDOC-generated CSS into separate resources from the output HTML files
  - for browser caching
- position the Specification elements according to a SPAR Ontologies document model
  - while the ANZ 3D Cad specification is modelled according to SPAR Ontologies, it is currently still produced from ordered ASCIIDOC documents which are themselves produced from models

OWL Diagramming

Early within Testbed 17, there was much discussion about the role of the visual element of UML diagrams in the creation of OGC standards. One set of thoughts related to the formality of the diagrams and how they did, or didn't, correspond to formal UML models. The general direction sought, going forward in the OGC, whas that while UML diagrams are useful or perhaps even necissary as vidual artifacts, they either should, or even must, be based on formal UML models that stnadards editors must supply in addition to the vidual representation of models to allow for model-driven stnadards. This set of thoughs is detailed most completely in the *UML Best Practice* document[26].

A parallel discussion was then entered into regarding the potential for visual dagramming in OWL with some thoughts being that OWL modelling would need to have some form of equivalence to UML diagramming for future Standards editors to be able to even consider it in place of UML.

This section relates a series of investidations into OWL Diagramming within the Testbed activities.

OWL Diagramming 'styles'

OWL diagramming is less well known and not as easy to overview as UML diagramming principlally because OWL is not first and foremost a visual design tool, as UML is, or at least was. OWL is a modelling system that is both system-independent and system-implementable. There are a series of OWL diagramming *styles* but no widely accepted *official* or *correct* form of OWL diagramming. The next section and subsections list several forms of OWL diagramming that were demnstrated during Testbed 17 with pros and cons, as they relate to OGC Standards, given.

An example piece of an OWL model, replicating CityGML's Tunnel class and its relations, is used for the next few diagram styles. The OWL code of that example, in the Turtle syntax[27] is given below. Snippets of that code will be explained in other code blocks, as needed.

TURTLE

```
@prefix cgml: <http://example.com/citygml/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix geo: <http://www.opengis.net/ont/geosparql#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .

cgml:HollowSpace a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:hasHollowSpace ] ;
            owl:someValuesFrom owl:Thing ],
        geo:Feature .

cgml:Tunnel a owl:Class,
        owl:Ontology ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:isTunnelPartOf ] ;
            owl:someValuesFrom owl:Thing ],
        cgml:TunnelPart .

cgml:TunnelConstructiveElement a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:hasTunnelConstructiveElement ] ;
            owl:someValuesFrom owl:Thing ] .

cgml:TunnelFurniture a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:hasTunnelFurniture ] ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty cgml:hasClass ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty cgml:hasFunction ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty cgml:hasUsage ;
            owl:someValuesFrom owl:Thing ] .

cgml:TunnelFurnitureUsage a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:hasUsage ] ;
            owl:someValuesFrom owl:Thing ],
        skos:Concept .

cgml:TunnelInstallation a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:hasTunnelInstallation ] ;
            owl:someValuesFrom owl:Thing ] .

cgml:TunnelPart a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onProperty cgml:hasClass ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty cgml:hasHollowSpace ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty cgml:hasTunnelConstructiveElement ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty cgml:hasTunnelFurniture ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty cgml:hasTunnelInstallation ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty cgml:isTunnelPartOf ;
            owl:someValuesFrom owl:Thing ],
        geo:Feature .

dcterms:isPartOf a owl:ObjectProperty ;
    rdfs:label "is part of" .

skos:Concept a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:hasClass ] ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:hasFunction ] ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:hasUsage ] ;
            owl:someValuesFrom owl:Thing ] .

cgml:hasFunction a owl:ObjectProperty .

cgml:hasHollowSpace a owl:ObjectProperty .

cgml:hasTunnelConstructiveElement a owl:ObjectProperty .

cgml:hasTunnelFurniture a owl:ObjectProperty .

cgml:hasTunnelInstallation a owl:ObjectProperty .

cgml:isTunnelPartOf a owl:ObjectProperty ;
    rdfs:subPropertyOf dcterms:isPartOf .

geo:Feature a owl:Class .

cgml:hasClass a owl:ObjectProperty .

cgml:hasUsage a owl:ObjectProperty .
```

Informal OWL diagrams

Many OWL ontologies, the equivalent of a UML domain model, have informal diagrams, that is diagrams following no strict specification. Such diagrams are common to see and appear in well-respected ontology documents such as:

- Epimorphics' *Registry Ontology*
- https://epimorphics.com/public/vocabulary/Registry.html (https://epimorphics.com/public/vocabulary/Registry.html)
- GeoSPARQL 1.1
- diagram by this author
- https://opengeospatial.github.io/ogc-geosparql/geosparql11/spec.html#_core (https://opengeospatial.github.io/ogc-geosparql/geosparql11/spec.html#_core)

Pros	**Cons**
Simple	Not 1:1 with technical model

Pros	Cons
Able to convey points of interest	

NOTE Even though informal OWL diagrams are not 1:1 with technical models, it's not possible, or at least it's not expected, that any OWL modeller could characterise an OWL model without a technical artifact, so there either cannot be, or at least there sofar has never been, an instance of a specification that has an informal OWL diagram and no reource that can be used for model-driven standards work.

Here is an informal diagram showing most of the elements of the CityGML `Tunnel` model:

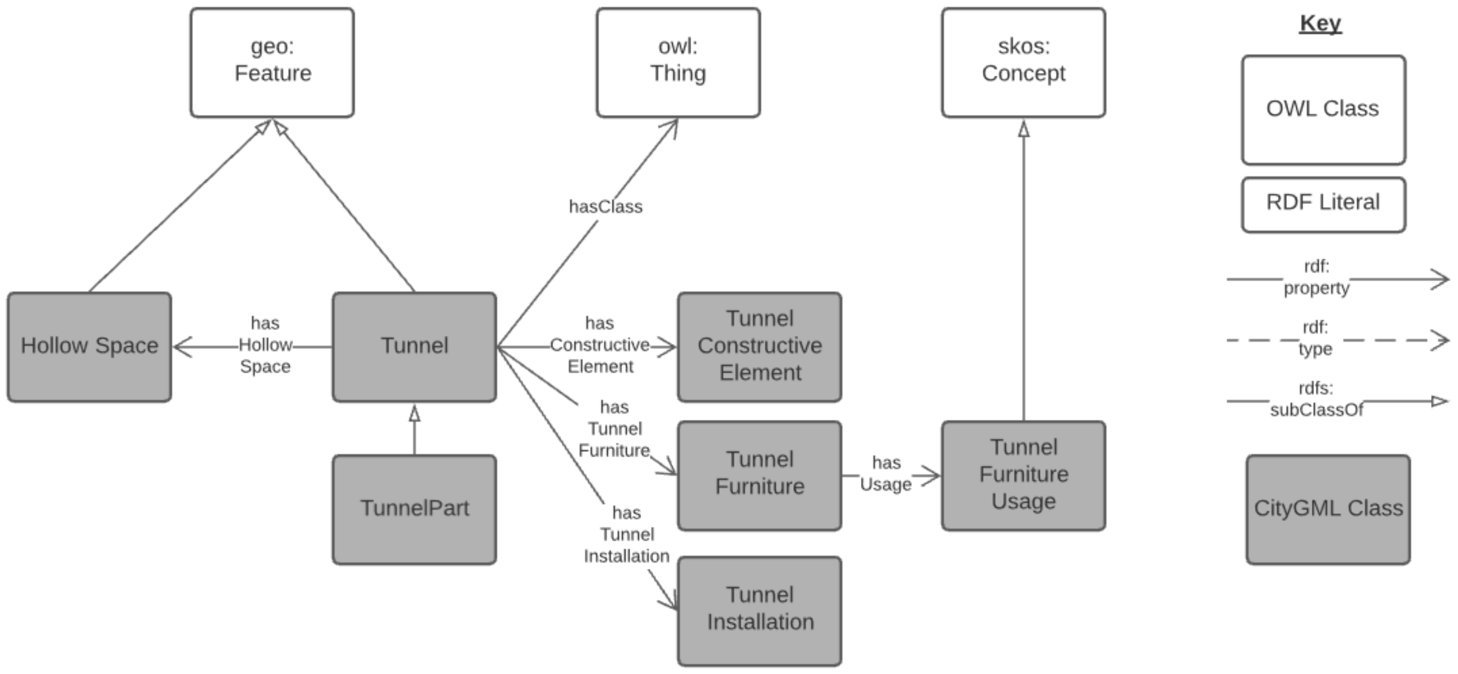


Figure 12. Part of CityGML's Domain Model focussed on the `Tunnel` *class represented with an informal OWL diagram. Note the use of some special property/association arrow styles. Other properies/associations have their type indicated on the instance*

Tooling

The above diagrma was drawn by hand using a generic diagramming tool. Many ontology editing systems provide auto informl diagraming, for example the well-known Protégé ontology editor[28] contains a plugin called *OntoGraph* that produces diagrams like the following from the same CityGML content used above:

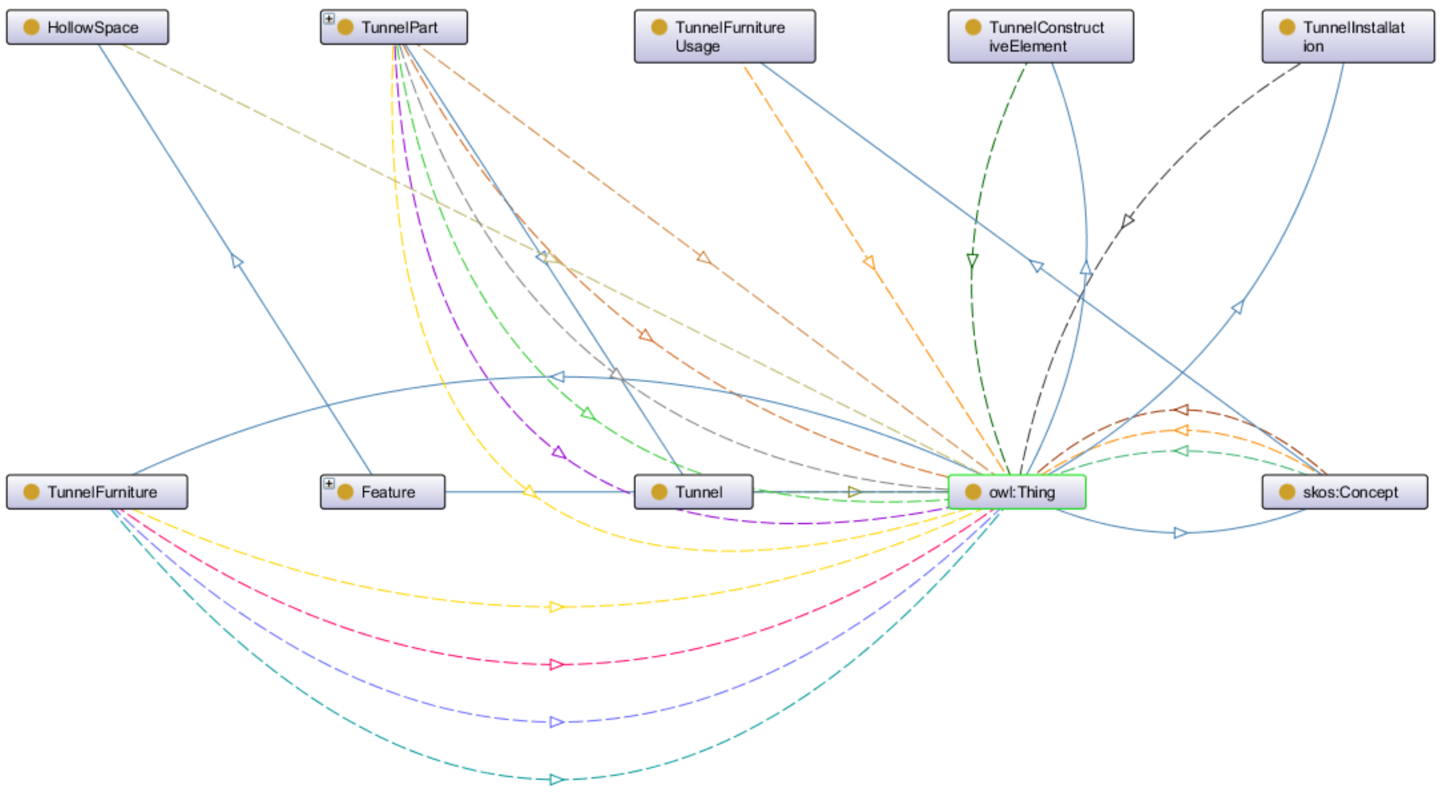
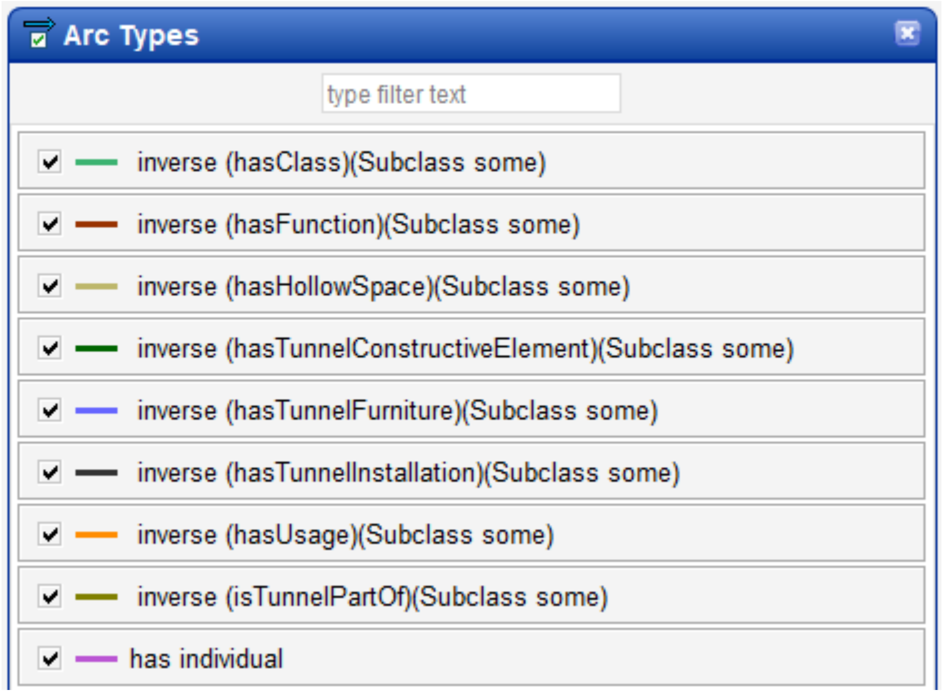


Figure 13. Informal OWL drawing of CityGML's `Tunnel` *class made by Protégé's OntoGraph plugin*

The relationships shown in the diagram above are typed and able to be shown or hidden using selectors built into Protégé, such as in the figure below:


Arc Types

type filter text

- ✓ inverse (hasClass)(Subclass some)
- ✓ inverse (hasFunction)(Subclass some)
- ✓ inverse (hasHollowSpace)(Subclass some)
- ✓ inverse (hasTunnelConstructiveElement)(Subclass some)
- ✓ inverse (hasTunnelFurniture)(Subclass some)
- ✓ inverse (hasTunnelInstallation)(Subclass some)
- ✓ inverse (hasUsage)(Subclass some)
- ✓ inverse (isTunnelPartOf)(Subclass some)
- ✓ has individual

Figure 14. Protégé's OntoGraph plugin's relationships dialogue

OntoGraph images, which can also be created outside Protégé, can be rearranged automatically or by hand. The following two images show firstly an auto-layout, 'hierarchical' and a layout specified by hand.

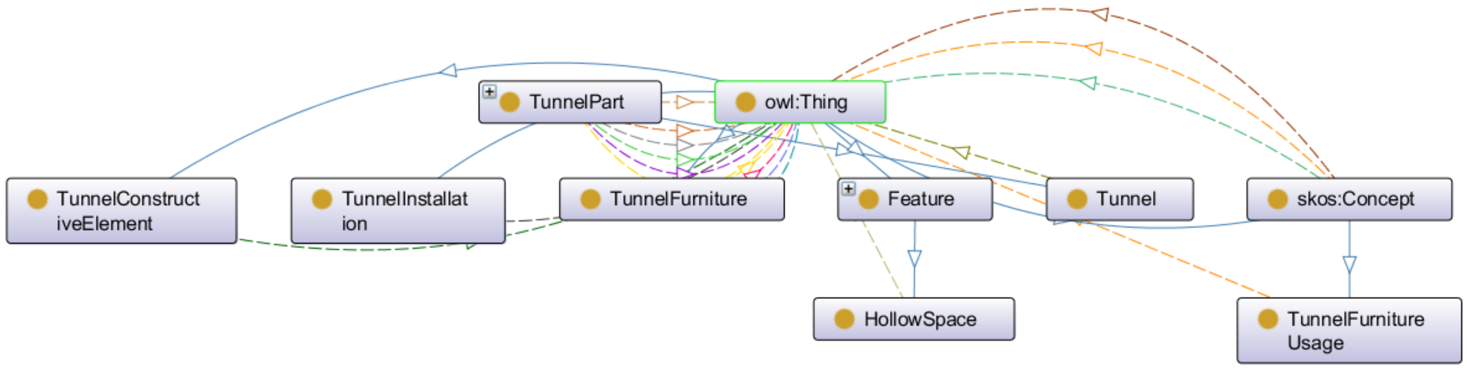


Figure 15. Informal OWL drawing of CityGML's `Tunnel` *class made by Protégé's OntoGraph plugin, hierarchical layout*

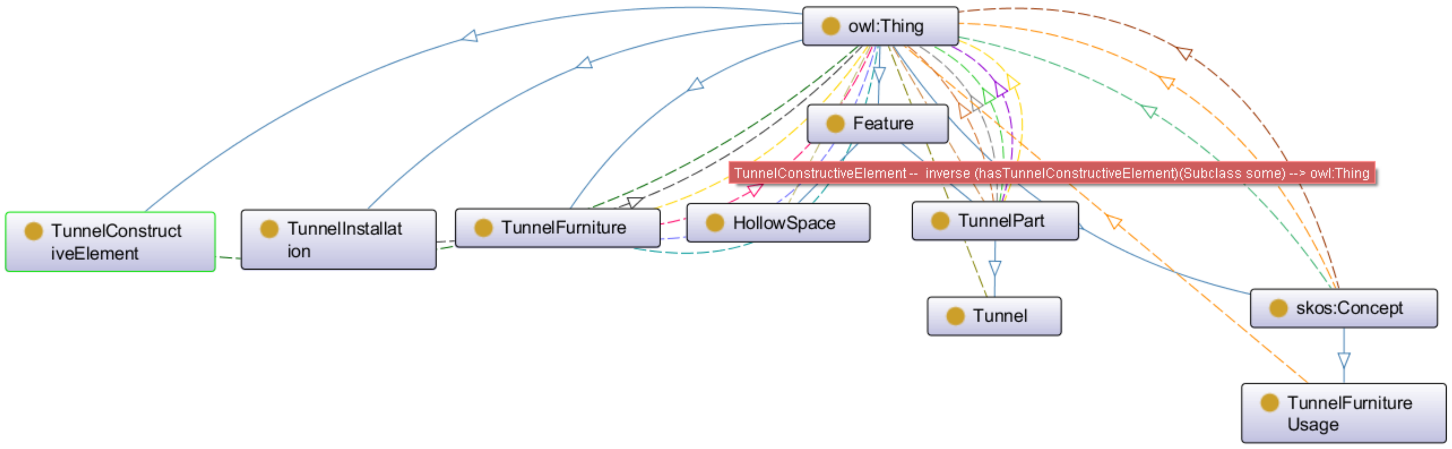

owl:Thing
Feature
TunnelConstructiveElement -- inverse (hasTunnelConstructiveElement)(Subclass some) --> owl:Thing
TunnelConstructiveElement
TunnelInstallation
TunnelFurniture
HollowSpace
TunnelPart
Tunnel
skos:Concept
TunnelFurnitureUsage

Figure 16. Informal OWL drawing of CityGML's `Tunnel` *class made by Protégé's OntoGraph plugin, layout by hand. Also shown are details of a selected relationship.*

UML OWL diagrams

Most formal OWL specifications use UML or UML-like diagrams. For example the W3C's Asset Description Metadata Schema (ADMS) ontology's domain model is as per the following figure that shows what it terms a "UML model of ADMS classes and properties":

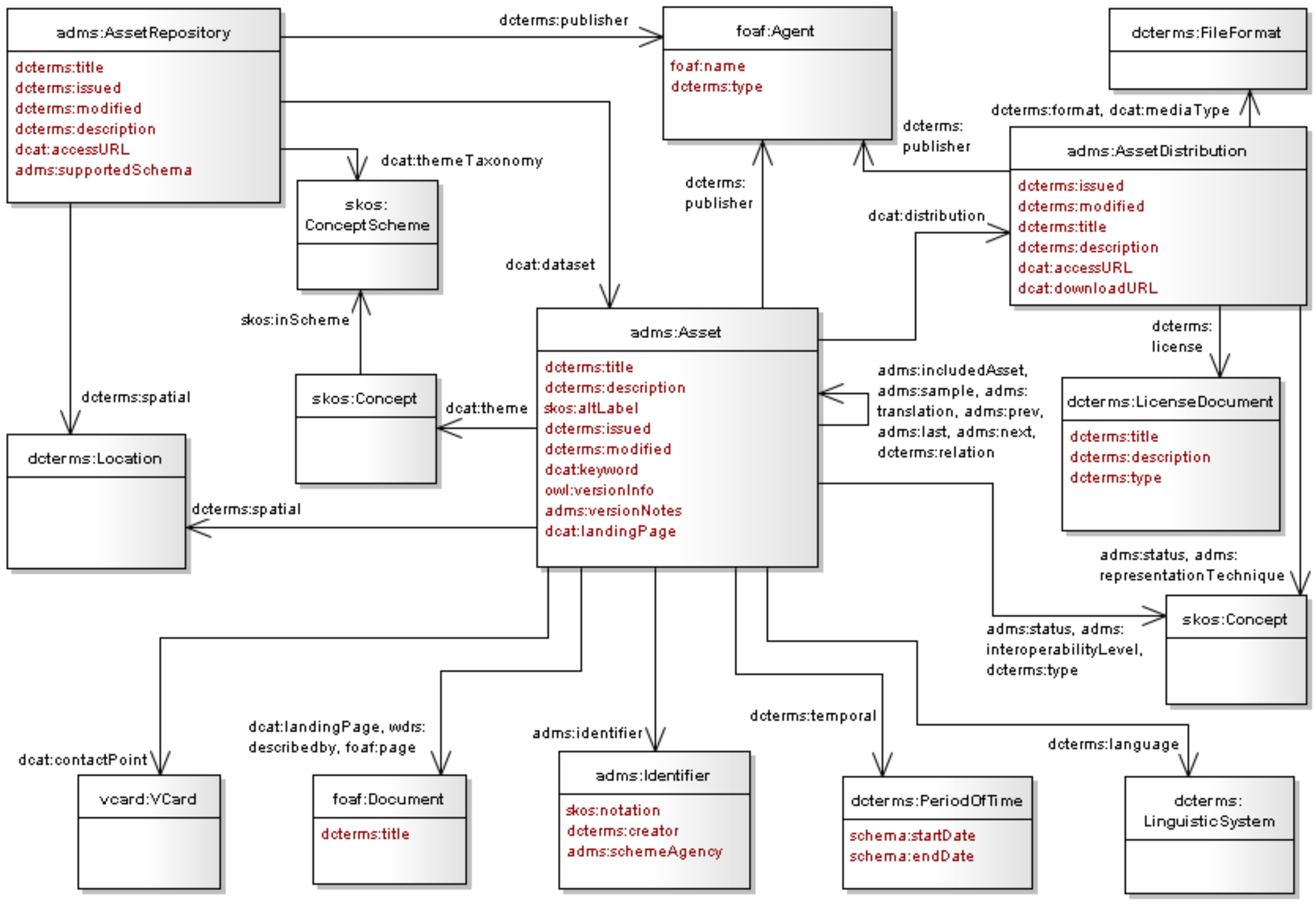


Figure 17. Asset Description Metadata Schema (ADMS) ontology's domain mode as a "UML model of ADMS classes and properties". After https://www.w3.org/TR/vocab-adms/#domainmodel (https://www.w3.org/TR/vocab-adms/#domainmodel)

It seems, based on the stying, that the UML diagram in the figure above was drawn using Sparx' _Enterprise Architect_[29], a UML diagramming tool well known to OGC modellers.

Other examples of UML OWL diagrams are given in the following W3C Specifications:

- Dataset Catalogue Vocabulary, Version 2 (DCAT2)
- Modern Sparx EA-style UML diagram
- https://www.w3.org/TR/vocab-dcat-2/#UML_DCAT_All_Attr (https://www.w3.org/TR/vocab-dcat-2/#UML_DCAT_All_Attr)
- DCAT Version 1
- Less formal UML diagram not drawn with Sparx's EA
- https://www.w3.org/TR/2014/REC-vocab-dcat-20140116/#overview (https://www.w3.org/TR/2014/REC-vocab-dcat-20140116/#overview)

Tooling

In addition to drawing UML diagrams for OWL ontologies in tools such as Sparx' EA, some OWL modelling tools can auto-generate UML or UML-like diagrams. For example, TopQuadrants' _TopBraid Composer_[30] can draw diagrams like the following, given the CityGML code in the listing above:

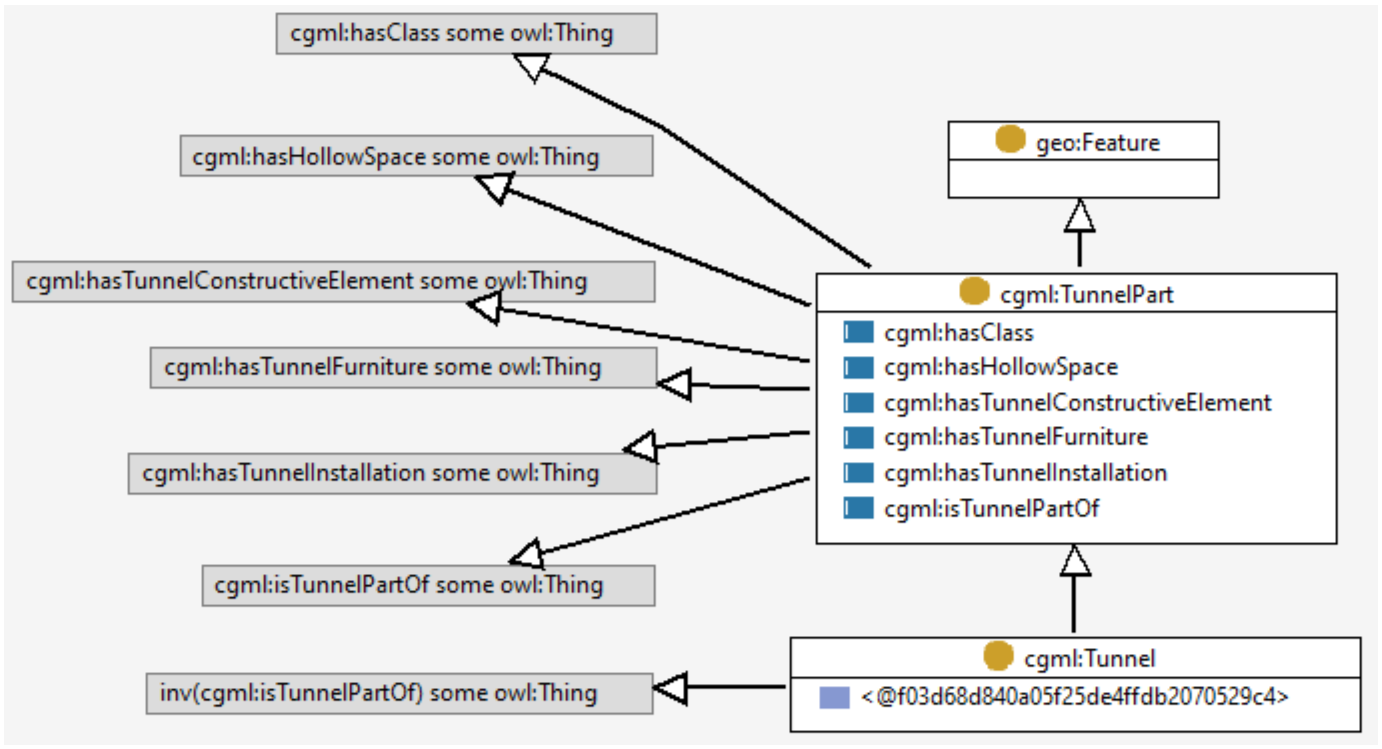

cgml:hasClass some owl:Thing
cgml:hasHollowSpace some owl:Thing
cgml:hasTunnelConstructiveElement some owl:Thing
cgml:hasTunnelFurniture some owl:Thing
cgml:hasTunnelInstallation some owl:Thing
cgml:isTunnelPartOf some owl:Thing
inv(cgml:isTunnelPartOf) some owl:Thing
geo:Feature
cgml:TunnelPart
cgml:hasClass
cgml:hasHollowSpace
cgml:hasTunnelConstructiveElement
cgml:hasTunnelFurniture
cgml:hasTunnelInstallation
cgml:isTunnelPartOf
cgml:Tunnel
<@f03d68d840a05f25de4ffdb2070529c4>

Figure 18. TopBraid Composer’s auto-drawn UML diagram of CityGML’s `Tunnel` *class ontology data*

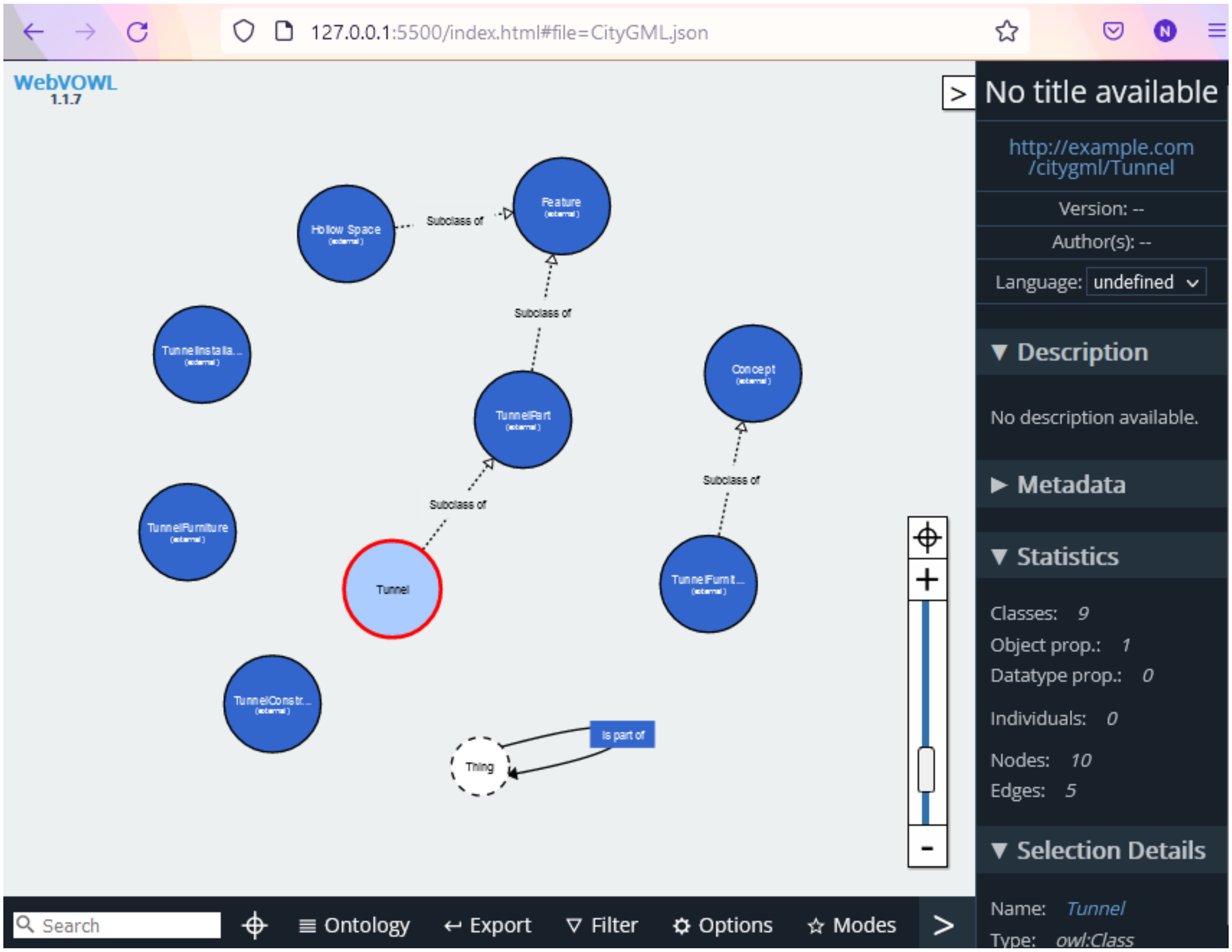
TopBraid Composer presents many options to the user for rearranging the diagram, hiding or showing elements and so on. What is shown above is a very simple rendering of only part of the CityGML `Tunnel` data, but mroe could be shown.

Native OWL diagrams

In addition to informal and UML diagrams, there are other kinds of formal OWL diagrams which are referred to here as 'native' OWL diagrams. These are diagrams that follow published specifications specific to OWL. There are several but the two demostrated here are:

1. Visual OWL (VOWL)[31] [VOWL]
2. Grapholfootnote:[] [GRAPHOL]

VOWL



No title available

http://example.com/citygml/Tunnel

Version: --

Author(s): --

Language: undefined

▼ Description

No description available.

▶ Metadata

▼ Statistics

Classes: 9
Object prop.: 1
Datatype prop.: 0
Individuals: 0
Nodes: 10
Edges: 5

▼ Selection Details

Name: Tunnel
Type: owl:Class

Figure 19. A Visual OWL (VOWL) representation of the CityGML's `Tunnel` *class ontology data. Auto-generated by WebVOWL*

VOWL diagrams are commonly seen in ontology documentation resources, for example the _I-ADOPT Framework ontology_[32]. VOWL documentation says that it "focuses on the visualization of the ontology schema (i.e. the classes, properties and datatypes, sometimes called TBox), while it also includes recommendations on how to depict individuals and data values (the ABox)".

In the quick visualisation of data above, a particulary useful diagram was not produced however it seems clear that VOWL can visualise all/most OWL constructs so more fiddeling with the tooling would likely produce reasonable results.

The VOWL developers used to provide an online free-to-use version of their tool but that is no-longer available, however desktop use is easy.

Graphol

Graphol is an OWL diagramming language which "builds on the Entity-Relationship model, but has a formal semantics and higher expressiveness. Notably, OWL 2 can be completely encoded in GRAPHOL". The following Graphol diagram was created based on the CityGML's `Tunnel` class ontology data but drawn by hand using the Eddy Graphol tool[33] supplied by a company providing commercial Graphol support.

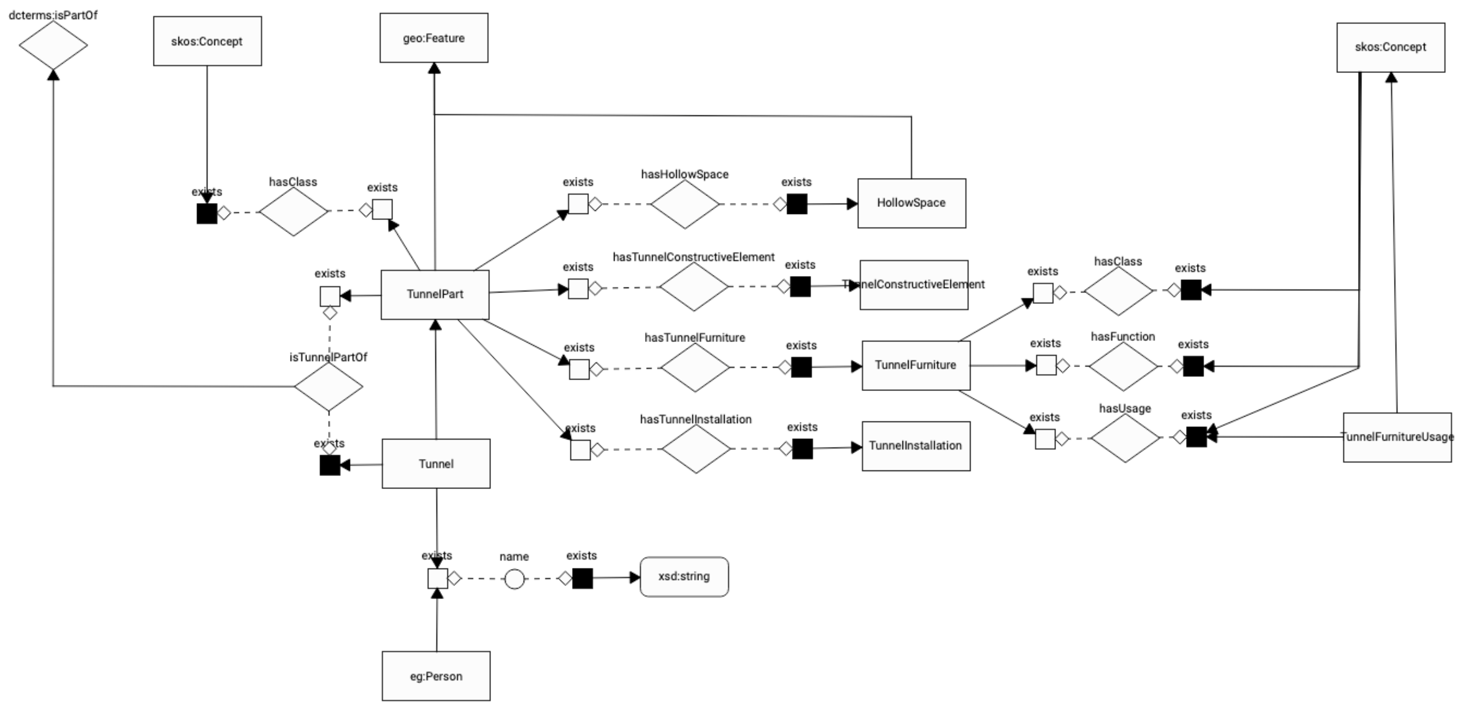


Figure 20. A Graphol representation of the CityGML's `Tunnel` *class ontology data. Hand drawn using the Eddy tool*

Note that Eddy can turn Graphol diagrams into stnadard OW ontology documents but not the other way around as layout information is lost. This prevents the loading of pre-existing ontologies into Eddy and diagram rendering.

The above Graphol diagram is "OWL complete" in that it claims to represent all elements of the OWL ontology it is diagramming with figure elements. Simplified versions of Graphol diagrams can be auto-produced by supplimentary Graphol tooling. Such a 'Graphol Lite' diagram is given below:

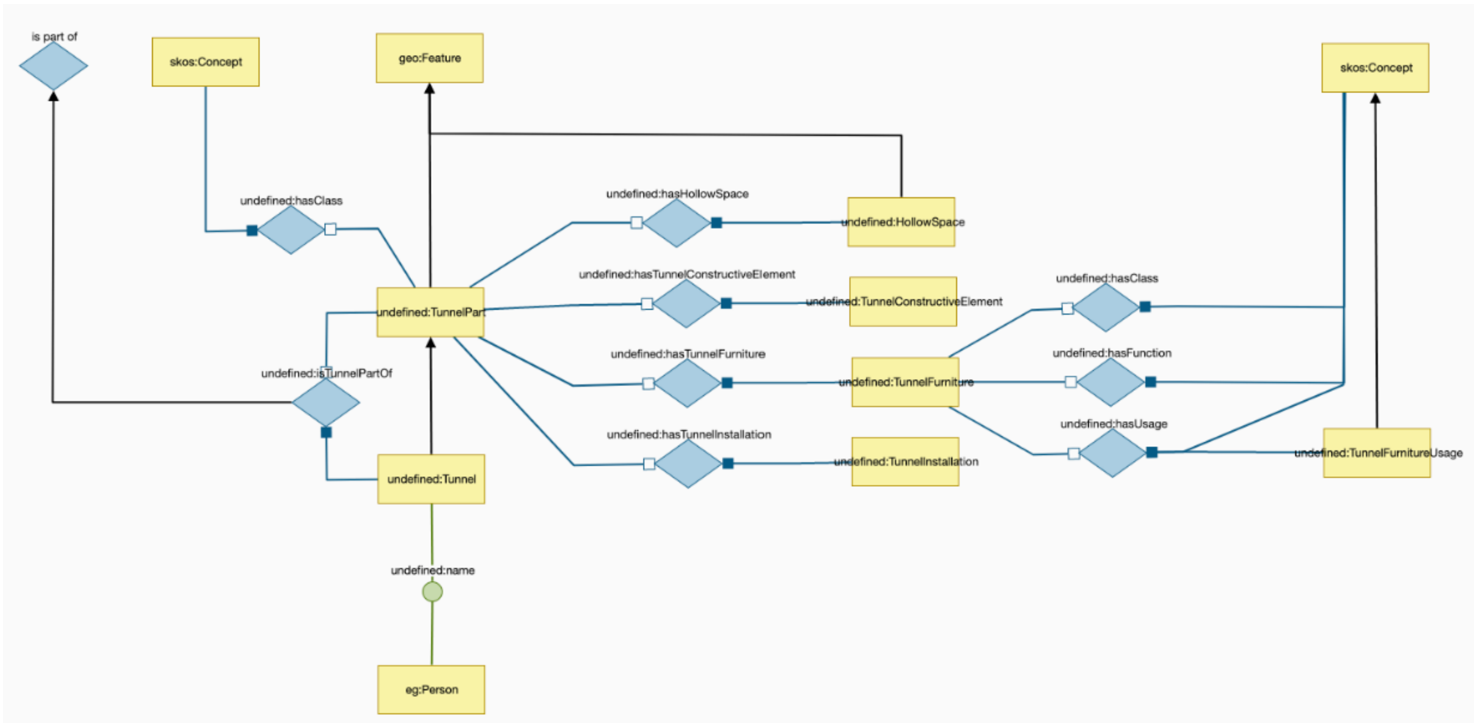


Figure 21. A Graphol Lite representation of the CityGML's `Tunnel` *class ontology data auto-produced from the original Graphol diagram*

OWL Diagramming 'style' Summary

There are quite a number of ways to visualise OWL ontologies ranging from hand-drawn UML diagrams to auto-generated, OWL-specialised, graph diagrams.

Round-tripping (diagram → code → diagram or *vice versa*) is possible in some systems however layouts are lost in the tools tested. It is suspected that there are diagramming round-tripping capabilities in some tools, such as *topBraid Composer* but these capabilities weren’t followed up in Testbed 17.

There seems to be quite an apetite for OWL diagramming, given the range of options and the fact that there formal specifications and whole companies, such as the authors of *Eddy*, dedicated to it. If the OGC were interested in pursuing visual OWL modelling, more investigations here would probably be able to address most queries.

Diagrams within Standards' *Specification* documents

In work for the Australian/New Zealand 3D Cadastre Standard, per-element diagrams were placed into the *Specification* document that was generated from the Standard’s ontology data. The diagrams where generated by hand and using a range of diagramming tools, such as Sparx' *Enterprise Architect* and TopQuadrant’s *EDG*'s inbuilt class visualiser[34]

These diagrams were placed *per element* within the *Specification*. The following three diagrams show informal, UML *Class* and UML *Package* diagrams.

7.3.1. Class: Cadastral Parcel

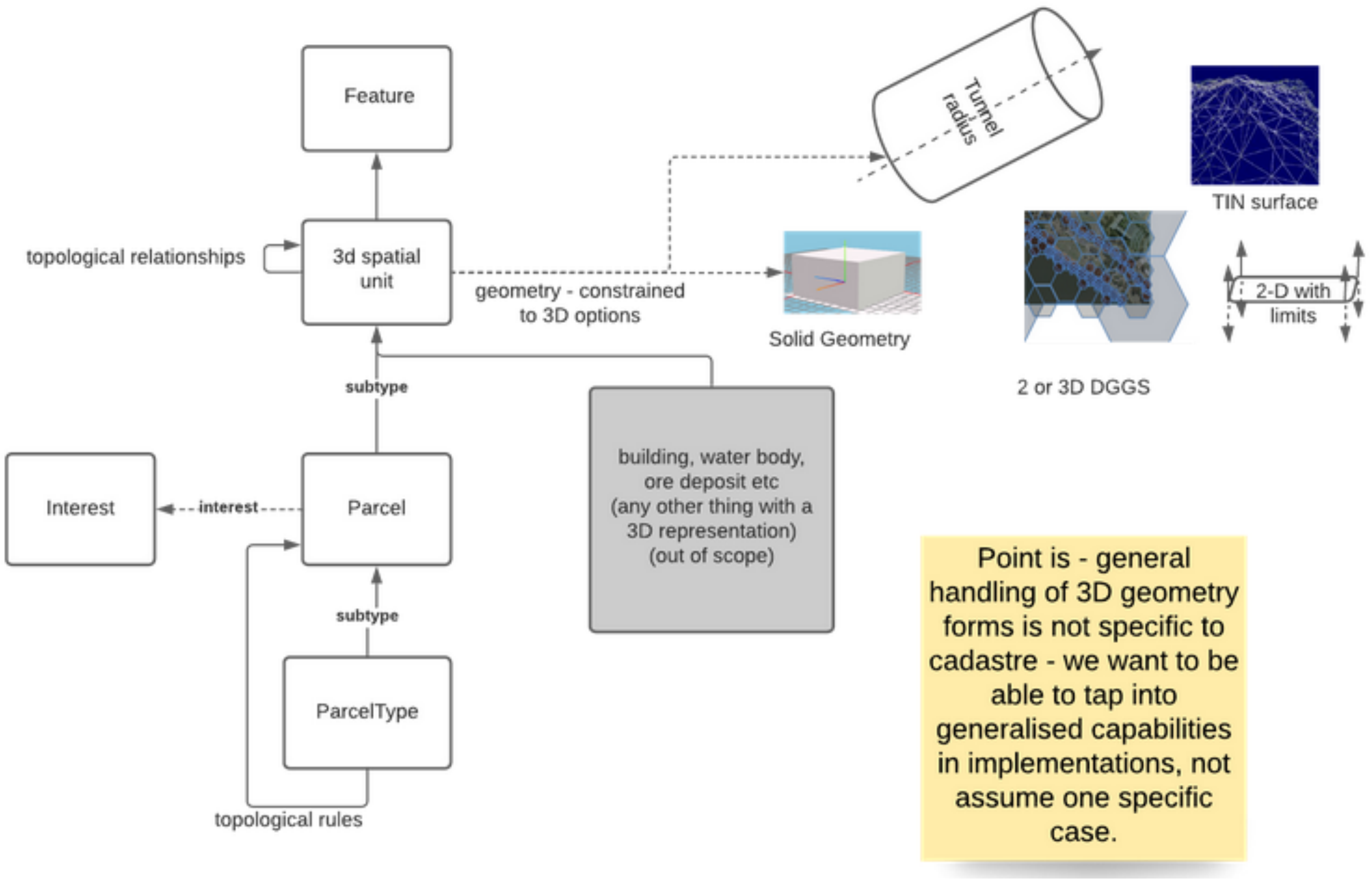
A single or multi area, or solid, above, or below the surface of the earth as specified through legislated process. A cadastral parcel can be described by a surface or solid and topological relationships with other parcels.

NOTE	Should a surface area only be mandatory for a primary cadastral parcel?

7.3.1.1. Subclass of

- 3D Spatial Unit
- No Title

7.3.1.2. Images



Feature
topological relationships
3d spatial unit
geometry - constrained to 3D options
Tunnel radius
TIN surface
Solid Geometry
2-D with limits
2 or 3D DGGS
subtype
building, water body, ore deposit etc (any other thing with a 3D representation) (out of scope)
Interest
interest
Parcel
subtype
ParcelType
topological rules
Point is - general handling of 3D geometry forms is not specific to cadastre - we want to be able to tap into generalised capabilities in implementations, not assume one specific case.

Figure 22. An informal Class diagram within the draft Australian/New Zealand 3D Cadastre Standard, darwn by hand

7.6.15. Class: Survey Mark

A SurveyMark is a physical object which by its form defines a point on the surface of the Earth and which is stable during surveying operations.

NOTE	Specific functions may be subclasses or attributes depending on both consistency of terminology and need to describe specific constraints per function. Functional subclasses may not be disjoint.

7.6.15.1. Subclass of

- Survey Point

7.6.15.2. Images

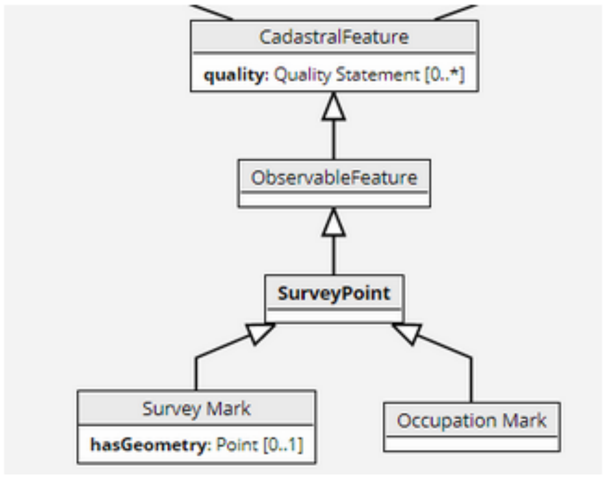


Figure 23. An informal UML class hierarchy diagram, drawn using TopQuadrant's EDG system

7.5. Conformance Class: Provenance Profile

urn:x-evn-master:provenance_profile

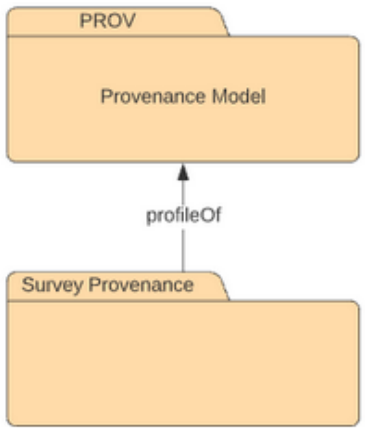


Figure 5. Package Dependencies Diagram

Format	image/png

Figure 24. A UML Package diagram for `Conformance Class` *objects in the draft Australian/New Zealand 3D Cadastre Standard, drawn by hand*

These diagrams where associated with the Domain Model elements they represented by use of the _Exemplification Ontology_[35] that provides "An ontology for the description of examples". The example descriptions for the above three diagrams were able to define:

- diagram metadata - title, description etc.
    - to be used for caption generation
- diagram resource technical descriptions
    - the format, file size & dimensions of images
    - used in auto-documentation generation
- example *role*
    - the role of the diagram v. the exemplified element
    - Roles of *canonical diagram* and *implementation diagram* are used in the ANZ 3D Cad model
- diagram Standard conformance
    - describes the conformance, if any, of the diagramming style to a diagramming specification, for exampl UML Class Diagram
    - using `dcterms:confromsTo`, conformance to ISO specifications such as ISO19109 cd cad *ApplicationSchema* [ISO_19109_2015] are indicated
    - specialised profiles of diagramming are currently being created by the OGC Naming Authority and diagrams could indicate conformance to them. They indicate a diagramming formalism and purpose and are the *Class Context* and *Package Dependencies* diagramming profiles.

This use of the *Exemplification Ontology*'s packaging of diagrams within a Standard's Domain Model is the same as its use for packaging code and other text data examples. See [Domain Model Examples]

Conclusion

This Testbed activity purposely explored the limits of what is currently possible regarding Model-Driven Standards and what may soon be possible with enhanced conceptual mdoelling of Stnadards & Specifications, updated tooling and new examples of OGC & ISO Standards to test modelling for.

It is clear that:

- various OWL models can be used to modell *all* of the information within a Standard
    - including all parts of a Specification
    - and Stnadards' parts' relations
- OWL models can be shown graphically in a number of ways, some whcih look similar to, or are, UML
- tooling is now available that can produce Standards documents from OWL models
    - next-generations of this tooling may be generic enough for many stnadards

What is not yet known is to what extend current standards editors find it possible to adopt OWL-based approaches to standards content modelling, both the Domain Models and other elements.

It is hoped that with the testing of oMDS approaches for GeoSPARQL 1.2 and perhaps other standarts, some OGC, and perhaps ISO TC 211, members will gain first-hand experience with developing ontology Model-Driven Standards and will be able to relate their pros and cons to their member organisations shortly, perhaps ithin 2022.

Annex A: Knowledge Graph v. Property Graphs

> *I want to have a relationship between classes, say Association, but I want to be able to add extra facts about that relationship. Is this a Property Graph thing?*

Property Graphs

Consist of:

- Nodes
- Edges
- Properties
  - key/value pairs fo information, or "tags"
  - can be associated with Nodes or Edges

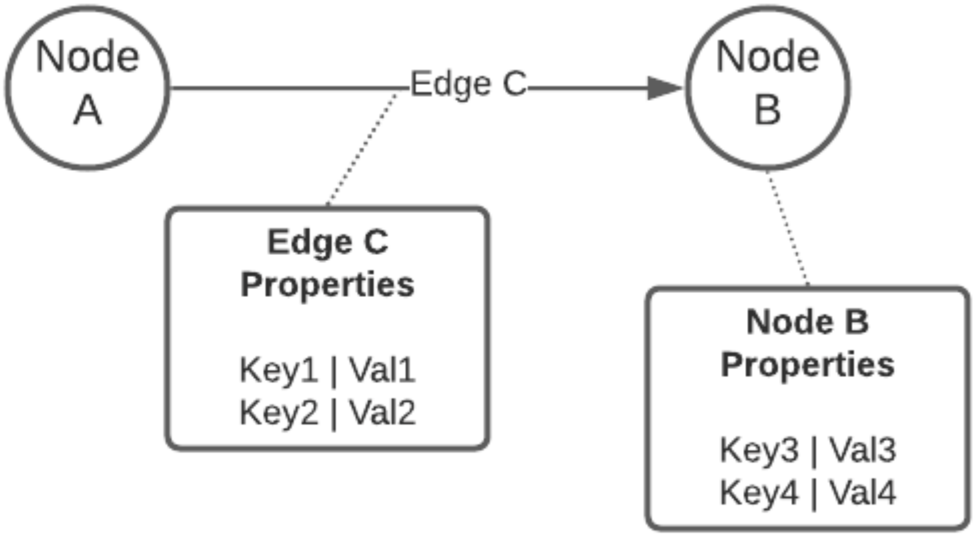


Figure 25. Basic components of a Property Graph

So yes, you can easily represent a relationship `-C→` as an Edge between things `(A)` & `(B)` (Nodes) in a Property Graph (PG) and associate anything you like with Edge `-C→` as PG Properties. However:

- How is the Properties information in the PG formalised?
  - PGs do not offer a consistent standard for doing this
  - Even when the main PG content - Nodes & Edges - are formally defined, the Properties (the keys & values) may not be
- How are instances of data, claiming conformance to the PG, implemented and tested?

◦ PGs do not offer a stnadard way of interpreting PG information as rules that can be used for data validation

Knowledge Graphs

Knowledge Graphs (KGs) can represent the information in Figure 1 in several ways with strong definitions and executable validation rules. There are two main ways to do this in RDF, called the *Qualified Relationship* and the *Reification* pattern. There is a 3rd way too which is a variant of Reification called *RDF**. These patterns are described below.

Qualified Relationship Pattern

RDF-based KGs are made of only Nodes and Edges and ensure that everything in them is a universally-defined Node or Edge. To emulate the `(A) -C→ (B)` structure in Figure 1 with more information about `C`, we could equate the relationship, a PG Edge, `-C→` to a KG Edge + Node + Edge like this:

```
-C-> == -X-> (Y) -Z->
```

So `(A) -C→ (B)` becomes `(A) -[-X→ (Y) -Z→]→ (B)`. Then we can associate anything we like with the intermediate Node `(Y)`. This is shown in Figure 2.

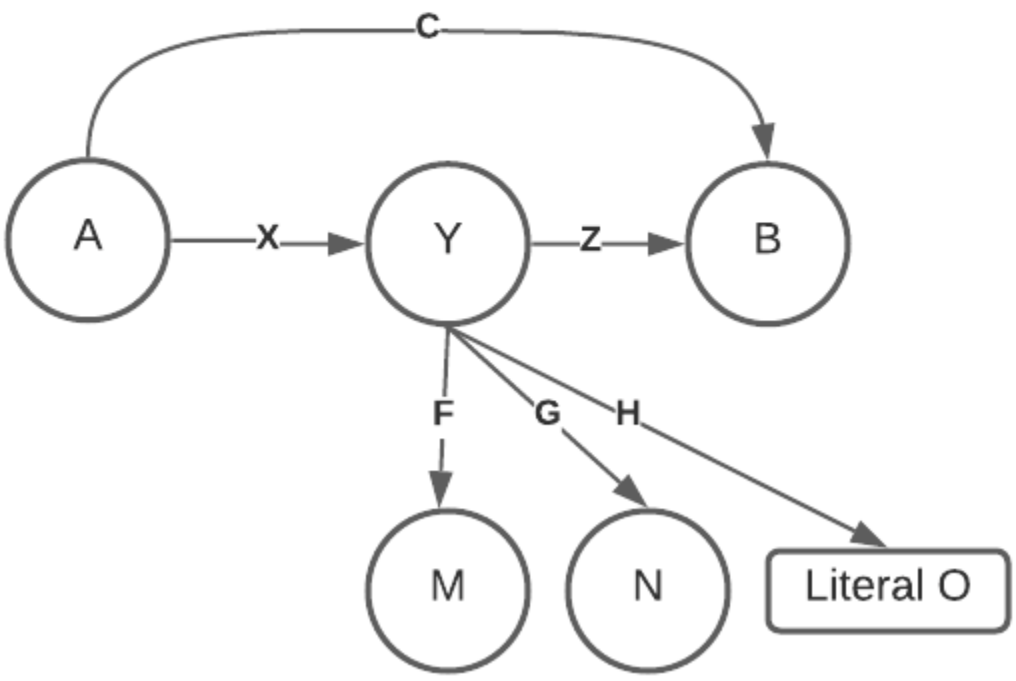


Figure 26. Qualified Relationship Pattern equivalent to Figure 1 with an intermediate Node, `(Y)` *being placed between* `(A)` *&* `(B)`*. Other things are associated with* `(Y)`*, here relationships,* `-F→`*,* `-G→` *&* `-H→` *to yet other nodes,* `(M)` *&* `(N)` *and to a literal (text, number etc.) to 'qualify' the original relationship* `-C→`*.*

This Qualified Relationship pattern is the normal/most common way that a KG uses to add more information to/about a relationship and, in RDF/OWL, any set of relationships, and the intermediate nodes, can be equated to another relationship formally using a *property chain axoim*. Using such, the relationship equivalence of `-C→ == -X→ (Y) -Z→` would formally be defined like this (Turtle syntax RDF):

```
example:C owl:propertyChainAxiom ( example:X example:Z ) .
```

To ensure that every such chain includes a node of type *Y* we then add:

```
example:X rdfs:range example:Y .
```

So now whenever we see `example:C` it's equal to `example:X` then `example:Z` with the `example:X` property requiring the intermediate node to be of type *Y*, and this would be required of either named or Blank (un-identified) intermediate nodes.

Finally, we can impose any RDF/OWL restrictions on nodes of type *Y* that we like, for instance insisting that they indicate properties to other nodes (classes or individuals) or simple data properties:

```
example:Y
    a owl:Class ;
    rdfs:subClassOf [
        a owl:Restriction ;
            owl:onProperty example:F ;
            owl:allValuesFrom example:M ;
            owl:cardinality 1 ;
        ] ;
.
```

The above OWL snippit says that "every instance of Class `Y` must have a property `F` that indicates one and only one instance of Class `M`.

Reification Pattern

Reification allows us to make meta statements about other statements in an RDF graph. With this pattern, we need not change the basic association `(A) -C→ (B)` but talk about it elsewhere.

Any triple in an RDF store is composed of a *subject*, *predicate* & *object* and the [fundamental structural ontology for RDF](http://www.w3.org/1999/02/22-rdf-syntax-ns# (http://www.w3.org/1999/02/22-rdf-syntax-ns#)) also defines a `Statement` class of object which has, as it's expected properties, `rdf:subject`, `rdf:predicate` & `rdf:object` which can be used to indicate the elements of any other triple. So, we can create a `Statement`, `S`, which indicates `(A) -C→ (B)` and then says anything else we want to about it, as per Figure 3.

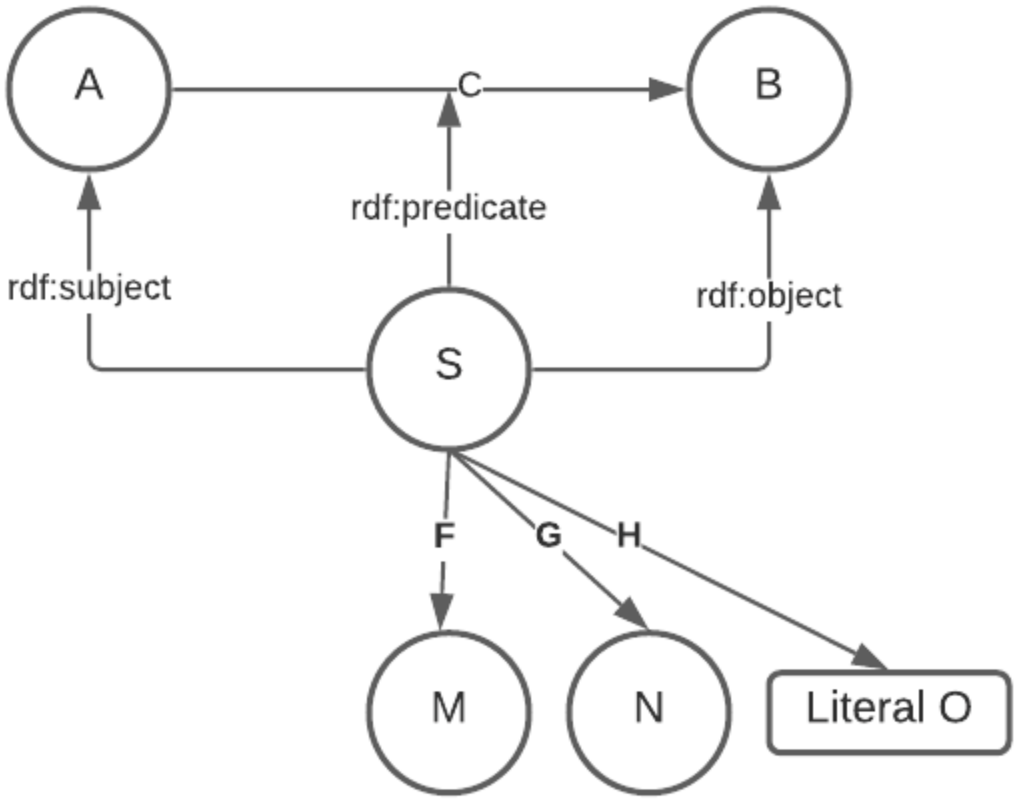


Figure 27. Reification of the elements in Figure 1 with Statement `S` *indicating* `A`*,* `B` *&* `C` *and other things, such as* `M`*,* `N`*, and literal* `O`*, just like in the qualified pattern in Figure 2.*

Reification is often used when primary facts need secondary facts/meta facts/provenance recorded about them semi-separately to the original data. However, if Statement `S` is stored in the same store as `(A) -C→ (B)` then we can query for the elements of the statement easily.

The full RDF code for Figure 3 is:

```
example:A example:C example:B .

example:S
    a rdf:Statement ;
    rdf:subject example:A ;
    rdf:predicate example:C ;
    rdf:object example:B ;
    example:F example:M ;
    example:G example:N ;
    example:H "Some Literal O" ;
.
```

…and any other statements for restrictions.

Reification is often used for things like uncertainty or quality *of the statement referred to*, e.g.:

```
example:S
    a rdf:Statement ;
    rdf:subject example:A ;
    rdf:predicate example:C ;
    rdf:object example:B ;
    example:certainty 0.75 ;
.
```

where 0.75 is the "certainty" of the statement `example:A example:C example:B .` being true.

RDF* Pattern

RDF* is an extension to RDF that is supported by a number of vendors' productions, including major triplestored like Jena and GraphDB. It is likely to be formalised in a W3C standard within the next 12 months.

RDF* really just proposes, per RDF syntax (JSON-LD, Turtle, RDF/XML, N-triples) a more compact way of showing reification. So, for the reification in Figure 3 where `(A) -C→ (B)` is represented by the RDF triple `example:A example:C example:B .` , you would write this, in Turtle*:

```
 <<example:A example:C example:B>>
    example:F example:M ;
    example:G example:N ;
    example:H "Some Literal O" ;
.
```

So RDF* does no more than Reification but presents it better!

Triplstores that support RDF* and SPARQL* allow compact notation using `<< & >>`, as above in data and queries and convert that, internally, to reified information.

Conclusion

Knowledge Graphs contain functional equivalents to Property Graph Properties for Edges which are in widspread use. RDF* was created recently specificaly to help people familiar with PGs start to use KGs.

The big KG benefit over PGs is the retention of semantics for the Properties, not just Nodes and Edges, which mean:

- no magic properties in KGs
  - what does *KeyX* in a PG Property table actually mean?
- standard query language
  - SPARQL can query *Qualified Patter* or *Reified Pattern* and extended PSARQL, SPARQL* can query *RDF* Pattern* data
- standard rules & validation
  - OWL rules, SPARQL `ASK` statements and Shapes languages such as SHACL all work happily with any of the 3 patterns

Bibliography

- [CONNEGP], World Wide Web Consortium, *Content Negotiation by Profile*, W3C Recommendation (25 August 2020). https://www.w3.org/TR/dx-connegp/ (https://www.w3.org/TR/dx-connegp/)
- [FEATURES], Open Geospatial Consortium, *OGC API - Features - Part 1: Core*, OGC® Implementation Standard (2019).

http://www.opengis.net/doc/IS/ogcapi-features-1/1.0 (http://www.opengis.net/doc/IS/ogcapi-features-1/1.0)

- [GRAPHOL] Console, Marco & Lembo, Domenico & Santarelli, Valerio & Savo, Domenico Fabio. (2014). Graphol: Ontology Representation Through Diagrams. 10.13140/2.1.3838.3363.
- [ISO 19150-2], *Geographic information — Ontology — Part 2: Rules for developing ontologies in the Web Ontology Language (OWL)*
- [ISO 19107:2019], *Geographic information — Spatial schema* https://www.iso.org/standard/66175.html (https://www.iso.org/standard/66175.html)
- [ISO 19109:2015], *Geographic information — Rules for application schema* https://www.iso.org/standard/59193.html (https://www.iso.org/standard/59193.html)
- [ISO 19160 2015], *ISO 19160-1:2015 Addressing — Part 1: Conceptual model*, https://www.iso.org/standard/61710.html (https://www.iso.org/standard/61710.html)
- [[[08-131r3]]], Open Geospatial Consortium, *The Specification Model — A Standard for Modular specifications*, Policy Standard (19 October 2009). https://www.ogc.org/standards/modularspec (https://www.ogc.org/standards/modularspec)
- [OWL2], World Wide Web Consortium, *OWL 2 Web Ontology Language Document Overview (Second Edition)*, W3C Recommendation (11 December 2012). https://www.w3.org/TR/owl2-overview/ (https://www.w3.org/TR/owl2-overview/)
- [PROF], World Wide Web Consortium, *The Profiles Vocabulary*, W3C Working Group Note (18 December 2019). https://www.w3.org/TR/dx-prof/ (https://www.w3.org/TR/dx-prof/)
- [RDF], World Wide Web Consortium, *RDF 1.1 Concepts and Abstract Syntax*, W3C Recommendation (25 February 2014). https://www.w3.org/TR/rdf11-concepts/ (https://www.w3.org/TR/rdf11-concepts/)
- [SKOS], World Wide Web Consortium, *SKOS Simple Knowledge Organization System Reference*, W3C Recommendation (18 August 2009). https://www.w3.org/TR/skos-reference/ (https://www.w3.org/TR/skos-reference/)
- [SHACL], World Wide Web Consortium, *Shapes Constraint Language (SHACL)*, W3C Recommendation (20 July 2017). https://www.w3.org/TR/shacl/ (https://www.w3.org/TR/shacl/)
- [SPARQL], World Wide Web Consortium, *SPARQL 1.1 Query Language*, W3C Recommendation (21 March 2013). https://www.w3.org/TR/sparql11-query/ (https://www.w3.org/TR/sparql11-query/)
- [TURTLE], World Wide Web Consortium, *RDF 1.1 Turtle - Terse RDF Triple Language*, W3C Recommendation (25 February 2014). https://www.w3.org/TR/turtle/ (https://www.w3.org/TR/turtle/)
- [VOWL] Lohmann, S., Negru, S., Haag F., Ertl, T.: Visualizing Ontologies with VOWL. *Semantic Web* 7(4): 399-419 (2016). http://www.semantic-web-journal.net/system/files/swj1114.pdf (http://www.semantic-web-journal.net/system/files/swj1114.pdf)
- [ISO19105], International Organization for Standardization, *ISO 19105: Geographic information – Conformance and testing* (2000)

---

1. See report respository, refs/T17-SOW-SURROUND.pdf
2. https://www.w3.org/ (https://www.w3.org/)
3. https://gitlab.ogc.org/ogc/T17-D144-MDA-Tool-2 (https://gitlab.ogc.org/ogc/T17-D144-MDA-Tool-2)
4. See next section for a description of this term
5. https://gitlab.ogc.org/ogc/T17-D022-Model-Driven-Standards-ER (https://gitlab.ogc.org/ogc/T17-D022-Model-Driven-Standards-ER)
6. https://www.w3.org/TR/dx-prof/#resource-roles-vocab (https://www.w3.org/TR/dx-prof/#resource-roles-vocab)
7. https://opengeospatial.github.io/ogc-geosparql/ (https://opengeospatial.github.io/ogc-geosparql/)
8. http://cad.surroundaustralia.com (http://cad.surroundaustralia.com)
9. https://www.w3.org/TR/dx-prof/#resource-roles-vocab (https://www.w3.org/TR/dx-prof/#resource-roles-vocab)

10. https://w3id.org/profile/vocpub (https://w3id.org/profile/vocpub)
11. https://www.w3.org/TR/skos-reference/ (https://www.w3.org/TR/skos-reference/)
12. http://www.sparontologies.net/ (http://www.sparontologies.net/)
13. http://defs.opengis.net/vocprez/ (http://defs.opengis.net/vocprez/)
14. https://github.com/opengeospatial/ogc-geosparql/blob/master/1.1/reqs.ttl (https://github.com/opengeospatial/ogc-geosparql/blob/master/1.1/reqs.ttl)
15. https://opengeospatial.github.io/ogc-geosparql/geosparql11/spec.html#_geosparql_standard_structure (https://opengeospatial.github.io/ogc-geosparql/geosparql11/spec.html#_geosparql_standard_structure)
16. https://github.com/ISO-TC211/GOM/ (https://github.com/ISO-TC211/GOM/)
17. Whole research communities are deveoted to Ontology Design Patterns and there are listings of such patterns online such as http://ontologydesignpatterns.org/wiki/Main_Page (http://ontologydesignpatterns.org/wiki/Main_Page). No formal rating of the GOM's ontologies has been conducted with respect to these, however, internally, the GOM has been able to show that some patterns are violated and the intention for GOM's operations in 2022/2023 is to show explicity violations and then to address them.
18. For a modern, succinct example, see https://henrietteharmse.com/uml-vs-owl/uml-class-diagram-to-owl-and-sroiq-reference/ (https://henrietteharmse.com/uml-vs-owl/uml-class-diagram-to-owl-and-sroiq-reference/)
19. https://github.com/ISO-TC211/GOM/blob/master/isotc211_GOM_harmonizedOntology/iso19160/-1/2015/Address.rdf#L77 (https://github.com/ISO-TC211/GOM/blob/master/isotc211_GOM_harmonizedOntology/iso19160/-1/2015/Address.rdf#L77)
20. https://github.com/ISO-TC211/GOM/blob/master/isotc211_GOM_harmonizedOntology/iso19160/-1/2015/Address.rdf#L1261 (https://github.com/ISO-TC211/GOM/blob/master/isotc211_GOM_harmonizedOntology/iso19160/-1/2015/Address.rdf#L1261) and many other places
21. Note that when GeosPARQL 1.0 was created (2012), there was no TC 211 ontology expression of `GM_Object` and thus no OWL link to one is present in GeoSPARQL.
22. ttps://data.surroundaustralia.com/def/uc
23. https://asciidoctor.org/ (https://asciidoctor.org/)
24. https://github.com/RDFLib/pyLODE (https://github.com/RDFLib/pyLODE)
25. https://data.surroundaustralia.com/def/uc (https://data.surroundaustralia.com/def/uc)
26. https://gitlab.ogc.org/ogc/T17-D023-OGC-UML-Modeling-Best-Practices (https://gitlab.ogc.org/ogc/T17-D023-OGC-UML-Modeling-Best-Practices)
27. https://www.w3.org/TR/turtle/ (https://www.w3.org/TR/turtle/)
28. https://protege.stanford.edu/ (https://protege.stanford.edu/)
29. https://sparxsystems.com/products/ea/ (https://sparxsystems.com/products/ea/)
30. https://www.topquadrant.com/products/topbraid-composer/ (https://www.topquadrant.com/products/topbraid-composer/)
31. http://vowl.visualdataweb.org (http://vowl.visualdataweb.org)
32. https://w3id.org/iadopt/ont/0.9.0 (https://w3id.org/iadopt/ont/0.9.0)
33. CityGML's `Tunnel` class ontology data
34. https://www.topquadrant.com/products/topbraid-enterprise-data-governance/ (https://www.topquadrant.com/products/topbraid-enterprise-data-governance/)
35. https://data.surroundaustralia.com/def/exem (https://data.surroundaustralia.com/def/exem)

Last updated 2021-11-29 18:31:48 01000