

March 17th, 2025



Presented by  
**Nataliya Keberle**  
**Arkadiusz Chądzyński**

# CityRDF

Webinar at W3C Linked Building Data Community Group




# Agenda

**01. Semantic  
Interoperability in AECO**

**02. Technical highlights**

**03. FAIR Principles**

The background features a light gray grid of dots on a white background. There are several abstract shapes: a large light blue circle on the right, a smaller light blue circle on the left, and a small blue sphere on the far left. The text is centered and reads:

# **Semantic interoperability in AECO**



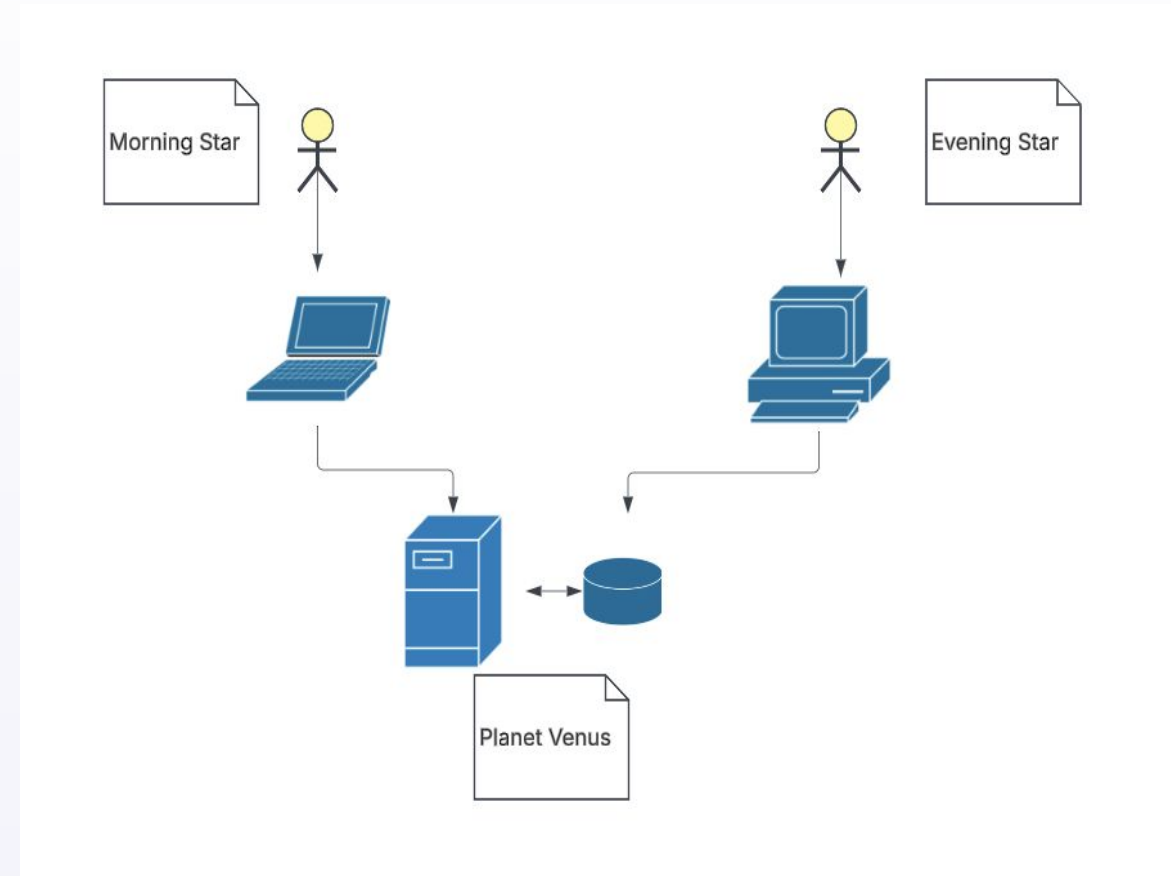
# What is semantic interoperability?



**Semantic interoperability** is the ability of computer systems to exchange data with unambiguous, shared meaning. It is a requirement to enable machine computable logic, inferencing, knowledge discovery, and data federation between information systems.

It is concerned not just with the packaging of data (syntax), but the **simultaneous transmission of the meaning with the data** (semantics). This is accomplished by adding data about the data (metadata), linking each data element to a controlled, shared vocabulary.

The meaning of the data is transmitted with the data itself, in one self-describing "information package" that is **independent of any information system**. It is this shared vocabulary, and its associated links to an **ontology**, which provides the foundation and capability of machine interpretation, inference, and logic.





# What are Data Silos?

An **information silo** is an insular management system in which one information system or subsystem is **incapable of reciprocal operation with others** that are, or should be, related.

In such scenarios, information is not adequately shared but rather remains contained within each system or subsystem, figuratively trapped within a container like grain is **trapped within a silo**: there may be much of it, and it may be stacked quite high and freely available within those limits, but it has no effect outside those limits. Such **data silos are proving to be an obstacle for businesses** wishing to use data mining to make productive use of their data.

Information silos occur whenever a data system is incompatible or not integrated with other data systems. This incompatibility may occur in the technical architecture, in the application architecture, or in the data architecture of any data system.

However, since it has been shown that **established data modelling methods are the root cause of the data integration problem**, most data systems are at least incompatible in the data architecture layer.



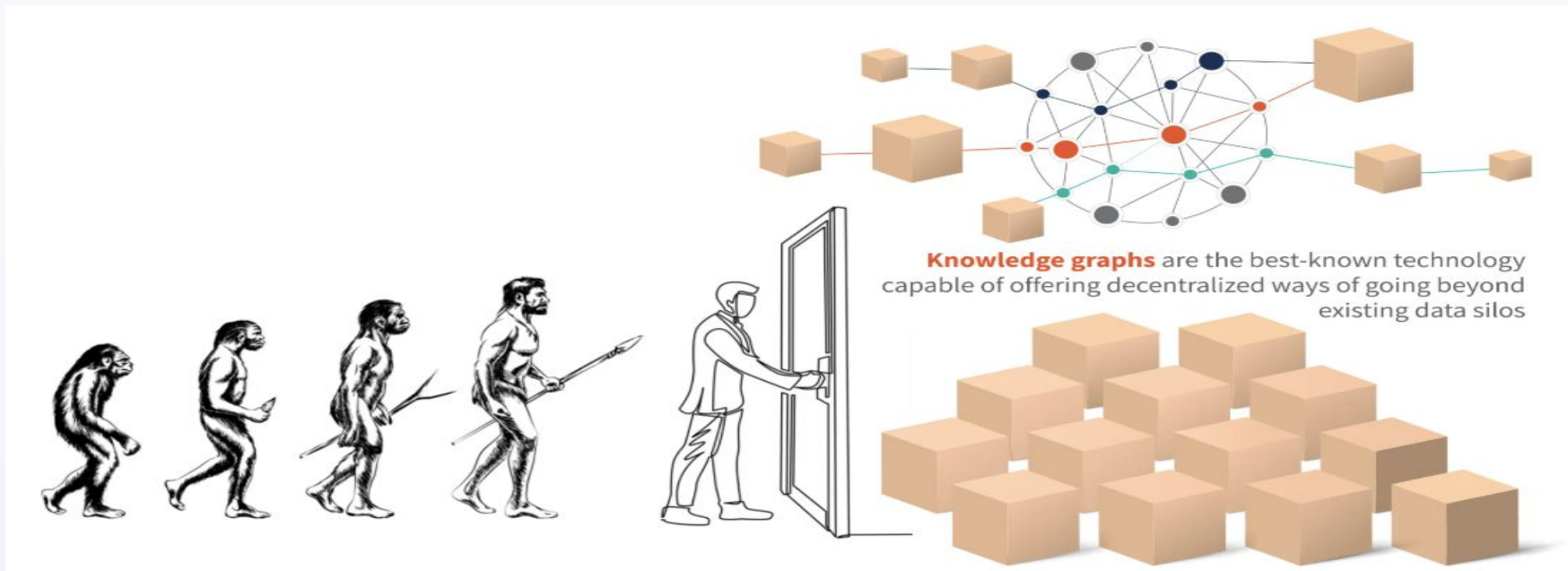


# Why is this important in AECO?



According to the World Bank, about **56% of the world's population (4.4 billion inhabitants) reside in cities**. This trend is expected to continue and **by 2050 the urban population is likely to be more than double its current size**, with nearly 7 of 10 people living in cities.

The rapid urbanization trend has serious implications for everyone inhabiting the planet and the rest of the biosphere. If we want to overcome the challenges of such transformations in sustainable ways, we need to look for solutions from ***multidimensional perspectives***. At present, **knowledge graphs** are the best-known technology capable of offering decentralized ways of going beyond existing data silos. They **enable the interlinking of various data sources and provide deeper insights, considering multiple points of interest**.

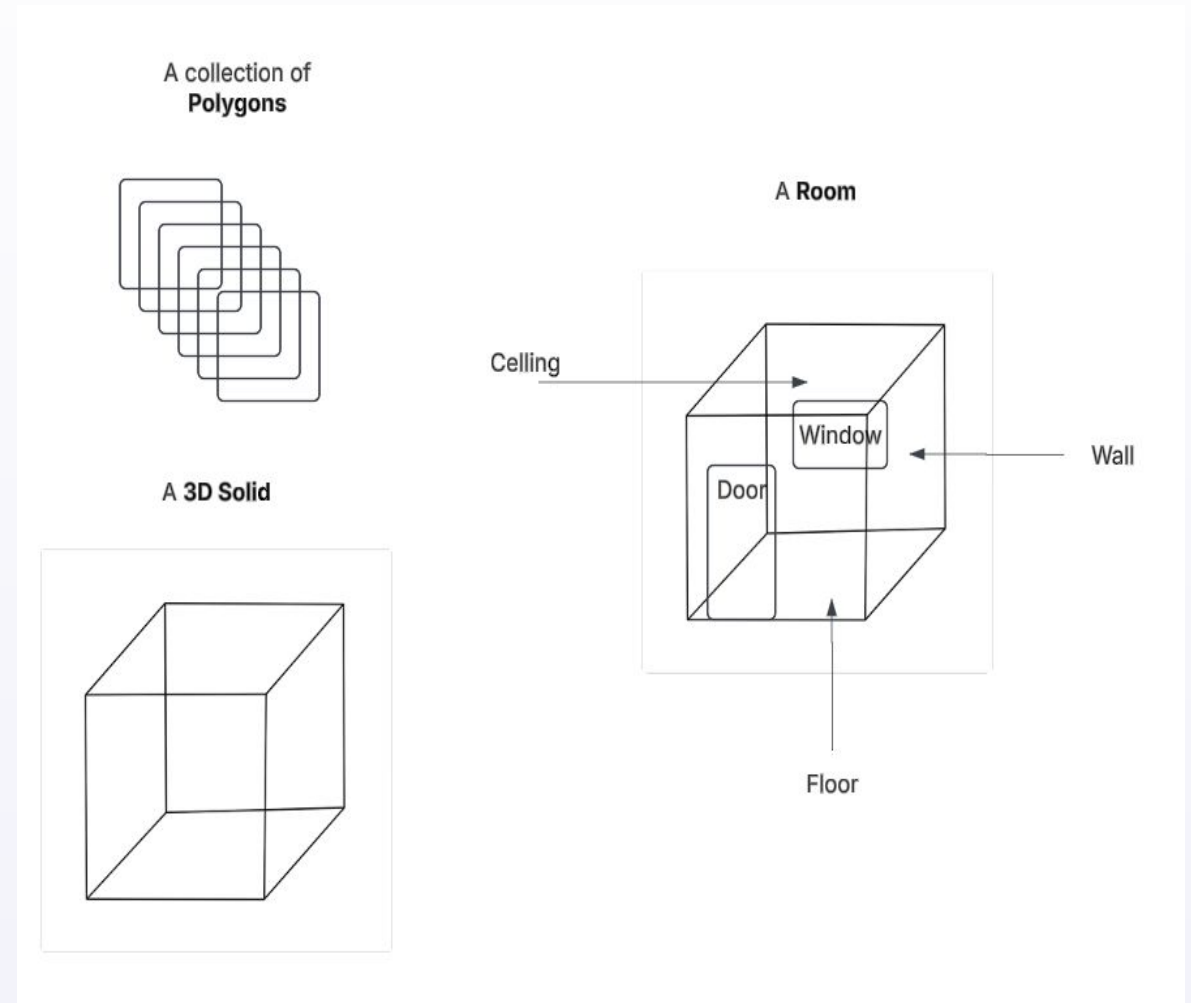


# ≡ Context To Keep Dimensions Together

**The spatial aspect of data** is a core characteristic. Spatial information provides the capability to provide both **location and shape as a context** to whatever it relates to.

**CityGML** is an open standardised data model and exchange format to store digital 3D models of cities and landscapes. It defines ways to describe most of the common 3D features and objects found in cities (such as buildings, roads, rivers, bridges, vegetation and city furniture) and the relationships between them.

CityGML allows to arrange spatial data with **semantics going beyond geometry while preserving location and shape**. This can provide context enabling interoperability with other standards concerned with city objects.



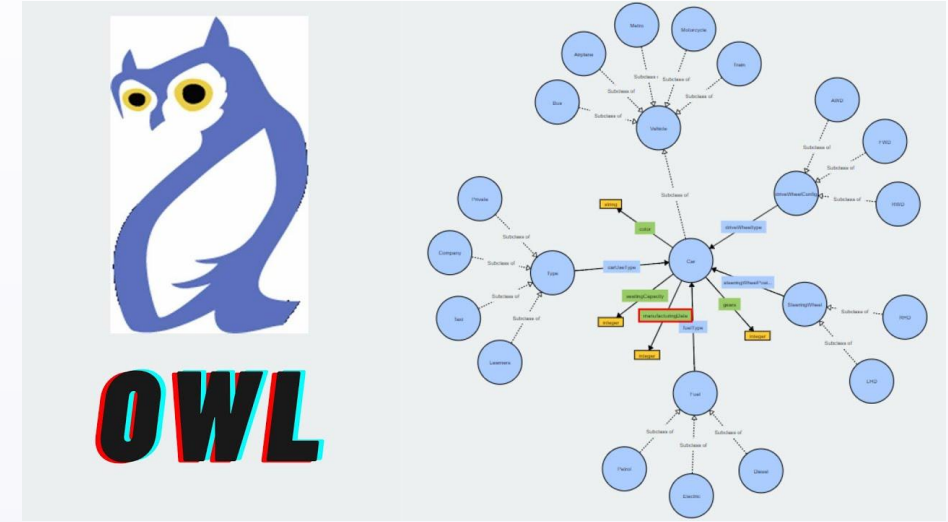
# What is an Ontology, in OWL?

OWL Ontologies are based on Description Logics, which distinguishes two knowledge components:

- TBox: terminological (theory)
- ABox: assertional (facts)

Knowledge Base is understood as a set of statements consisting of TBox and ABox combined.

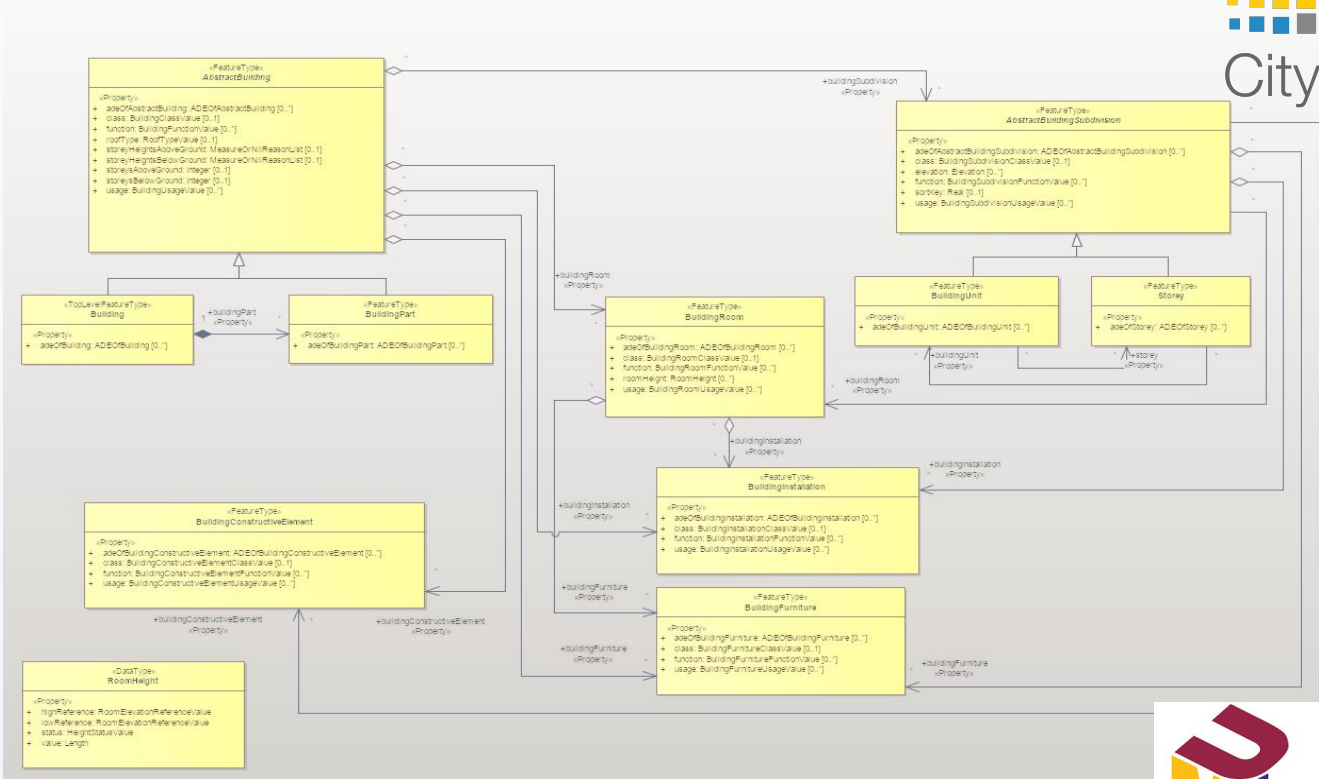
CityRDF TBox is a result of CityGML conceptual model transformation into OWL.



Concrete models describing actual city objects in CityRDF are ABoxes of this ontology.



Possibility of transformation between OGC CityGML conceptual model expressed in UML to the OWL TBox, expressed in RDF by W3C already shows that standardisation is crucial for semantic interoperability: same model and syntactically different languages.



CityGML

```
bldg:Building a owl:Class ;
rdfs:label "Building"@en ;
rdfs:subClassOf [ a owl:Restriction ;
    owl:allValuesFrom bldg:BuildingPart ;
    owl:onProperty bldg:buildingPart ],
    bldg:AbstractBuilding ;
owl:disjointWith bldg:BuildingPart .
```

```
bldg:BuildingClassValue a owl:Class ;
rdfs:label "BuildingClassValue"@en ;
rdfs:subClassOf skos:Concept ;
skos:definition "BuildingClassValue is a code list used to further classify a Building."@en .
```

```
bldg:BuildingConstructiveElementClassValue a owl:Class ;
rdfs:label "BuildingConstructiveElementClassValue"@en ;
rdfs:subClassOf skos:Concept ;
skos:definition "BuildingConstructiveElementClassValue is a code list used to further classify a BuildingConstructiveElement."@en .
```



# CityRDF ABox

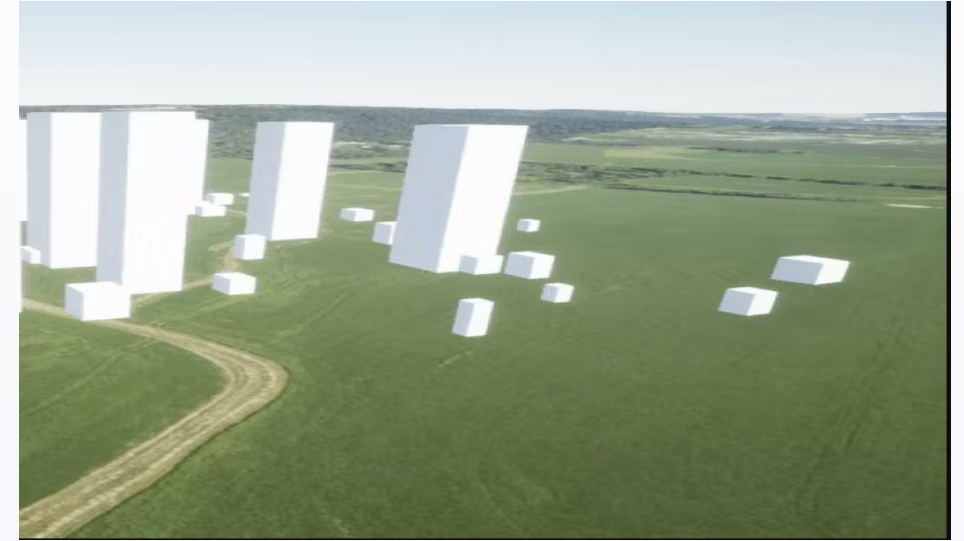
CityRDF ABox is a set of statements asserting something about one or more concrete city objects.

Depending on the level of detail (LOD1-4) those statements could simply say:

*At this location there is this 3D geometry and it is a building.*

Or even:

*At this location there are multiple 3D and 2D geometries which taken together are a buildings with walls, roofs, doors, windows and certain functions, etc.*





# How to build a city?

To start, we can choose location first.

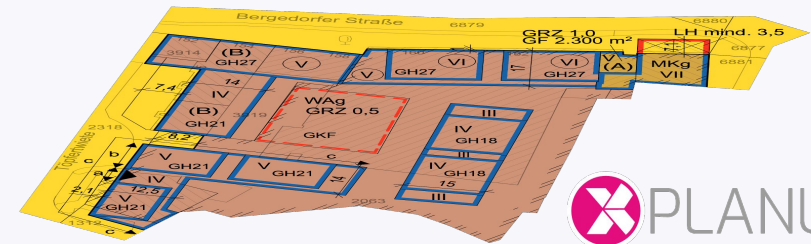
- The location is covered by GML and ontology for it is already provided by GeoSPARQL
- After that we get the land use plan.

In Germany, we can get GML based XPlan

- And our first building models.

They can be either in CityGML or other format convertible to it with regard to spatial design.

GeoSPARQL



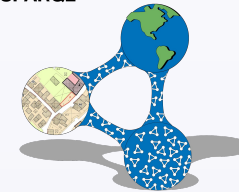
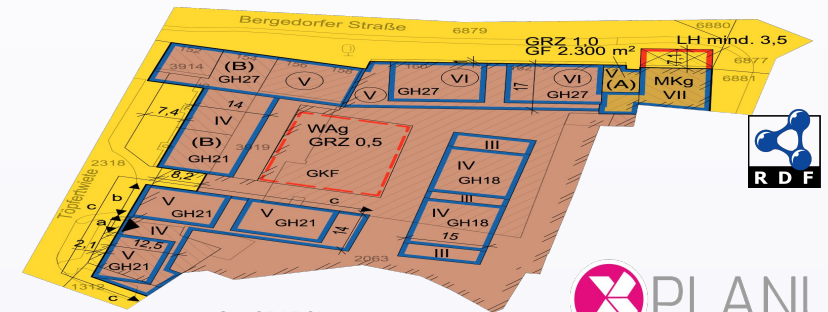


Land use standards, adding additional semantic descriptions to that what is regarded as a spatial feature in GML and GeoSPARQL, can be transformed into RDF.

Within ACCORD concrete XPlan land use design was transformed into RDF model.

Similarly to that, IFC BIM model was transformed into CityRDF.

In this way both models were made interoperable. This allowed to assess topology relationships between them (within, contains, intersects, crosses, etc.)



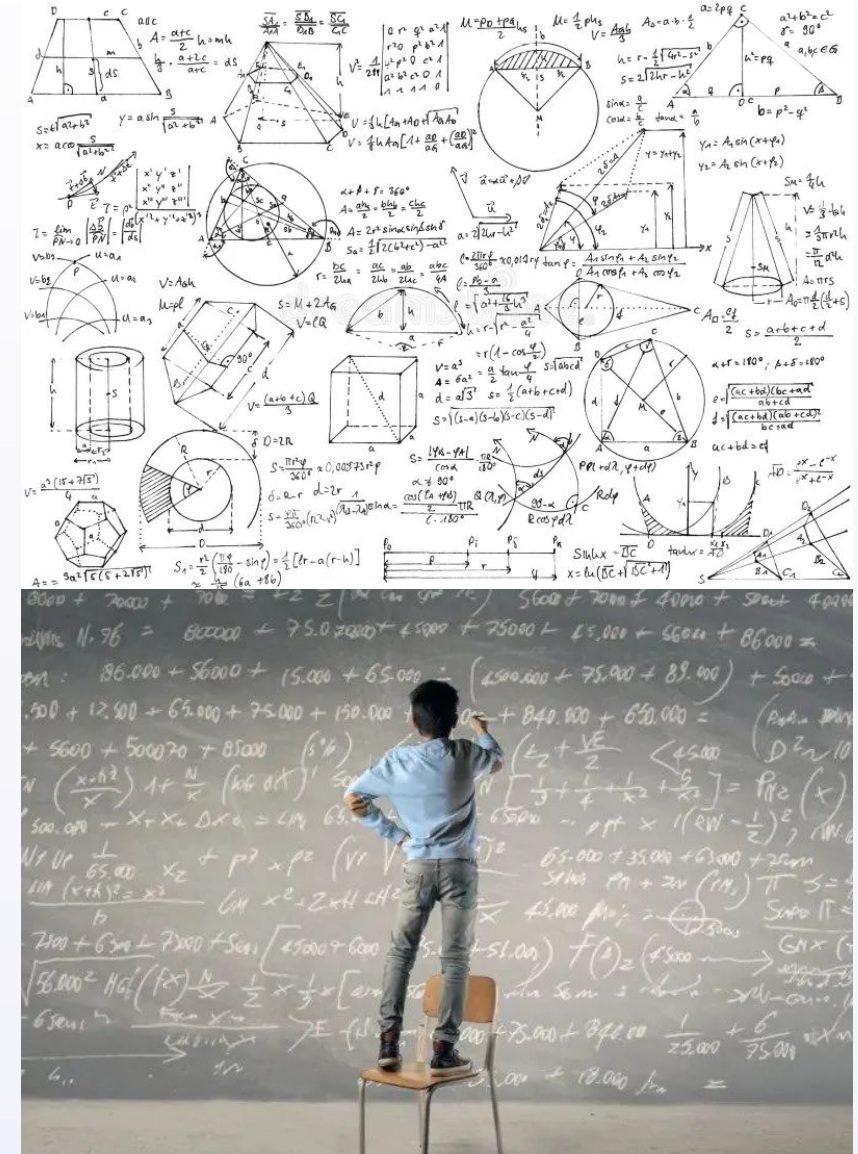


# Spatial Reasoning

Because of that, in RDF, land use model, city model as well as their location and topology relationships are expressed as sets of Subject-Predicate-Object statements, they can be logically reasoned about and reasoning could be assessed in terms of its truthfulness.

First of all, land use plans as well as concrete city models can be assessed with regard to consistency with axioms expressed in theoretical part (TBox).

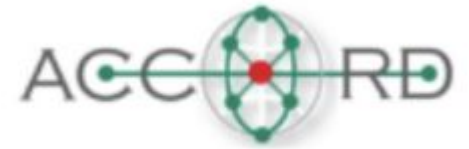
In addition to that it is possible to reason about the interconnected models and assess if particular building parts are conformant to the planned land use (ie. percentage of commercial use does not exceed certain threshold in residential sector.)





In ACCORD, we were able to answer following questions:

- Whether the building design is contained within Xplan.
- Whether all parts of the building are contained within permissible lot coverage.
- Whether use of all building rooms does not exceed 5% of residential use.
- Whether use of all building rooms is at least 60% of office use.
- Whether building floor space index does not exceed planned floor space index in XPlan.
- Whether building is not higher than permitted by Xplan.
- Whether the building is within heritage protection area.
- Whether the building does not cross building boundary.







# Large City Models and Reasoning



Knowledge Graph of Singapore with land parcels modeled as generic city objects and zones grouped via community detection algorithm.

Knowledge Graph of Berlin, assessed with regard to distances between city objects, solar panels locations, energy demands.



# Technical highlights





# The source



## VCity team

### UD-Graph project

- [CityGML 3.0 Conceptual Model](#) (Enterprise Architect file)
- [ShapeChange](#) conversion from .EA to .XML
- [a bash + Python + SPARQL workflow](#) to get .OWL files

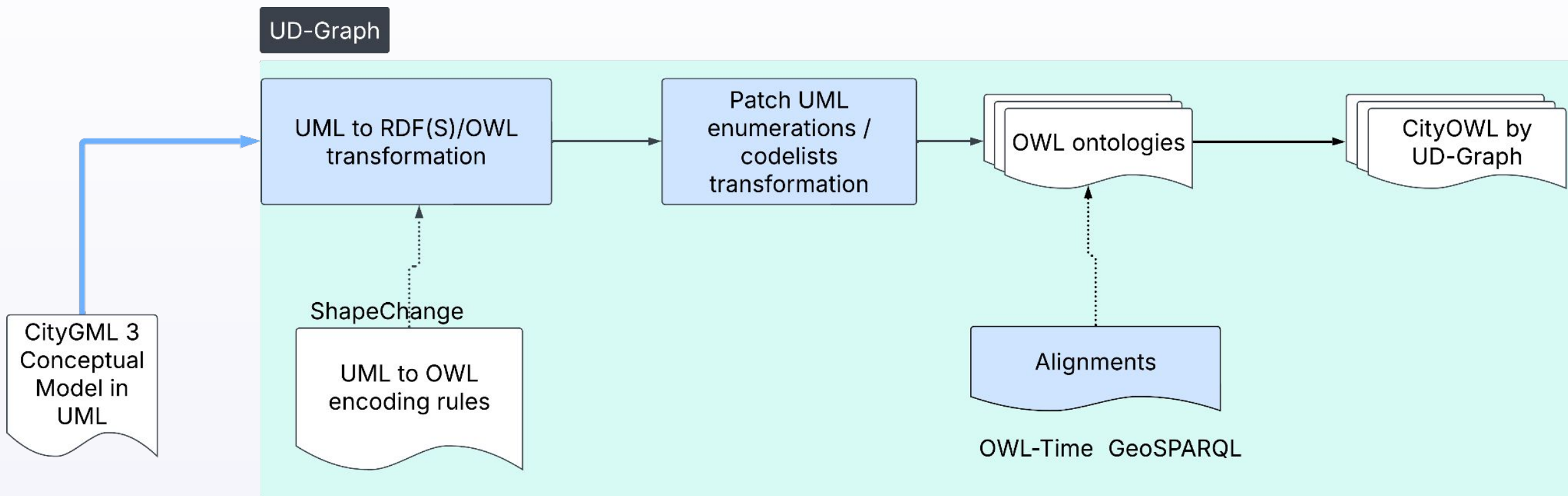
## Result

<https://dataset-dl.liris.cnrs.fr/rdf-owl-urban-data-ontologies/Ontologies/CityGML/3.0>





# CityOWL workflow (UD-Graph)



## Credits:

D. Vinasco-Alvarez, "Model-driven Integration of Heterogeneous n-Dimensional Urban Data," Ph.D. Thesis, Université Lyon 2 Lumière, Lyon, France, 2023. [Online]. Available: <https://www.theses.fr/s276081>  
<https://shapechange.github.io/ShapeChange/4.0.0>  
<https://github.com/VCityTeam/UD-Graph>



# Alignments

Alignments

GeoSPARQL

**Duplicate property name**  
**ns:Class.prop**

property names repeat in a package,  
across packages

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .  
@prefix geo: <http://www.opengis.net/ont/geosparql#> .  
@prefix time: <http://www.w3.org/2006/time#> .  
@prefix core: <https://www.opengis.net/ont/citygml/core/> .
```

```
# GeoSPARQL alignments for core
```

```
core:AbstractSpace.lod0MultiCurve rdfs:subPropertyOf geo:hasGeometry .  
core:AbstractThematicSurface.lod0MultiCurve rdfs:subPropertyOf geo:hasGeometry .  
core:AbstractSpace.lod0MultiSurface rdfs:subPropertyOf geo:hasGeometry .  
core:AbstractThematicSurface.lod0MultiSurface rdfs:subPropertyOf geo:hasGeometry .  
core:lod0Point rdfs:subPropertyOf geo:hasGeometry .
```

```
...
```



# Example code

```
bldg:AbstractBuilding.address a owl:ObjectProperty ;  
    rdfs:label "address"@en ;  
    rdfs:domain bldg:AbstractBuilding ;  
    rdfs:range core:Address ;  
    skos:definition "Relates the addresses to the Building or BuildingPart."@en .
```

```
bldg:AbstractBuilding.adeOfAbstractBuilding a owl:ObjectProperty ;  
    rdfs:label "adeOfAbstractBuilding"@en ;  
    rdfs:domain bldg:AbstractBuilding ;  
    rdfs:range bldg:ADEOfAbstractBuilding ;  
    skos:definition "Augments AbstractBuilding with properties defined in an  
ADE."@en .
```

```
bldg:ADEOfAbstractBuilding a owl:Class ;  
    rdfs:label "ADEOfAbstractBuilding"@en ;  
    iso19150-2:isAbstract true ;  
    skos:definition "ADEOfAbstractBuilding acts as a hook to define properties  
within an ADE that are to be added to AbstractBuilding."@en .
```

## Long property name

### ns:Class.prop

property names repeat in a package,  
across packages

## Dummy definitions of ADEs and relating properties

ADEs are given as placeholders or  
possible practical extensions. Keep them  
in CityRDF is unnecessary.





# Another example

```
bldg:AbstractBuilding.usage a owl:ObjectProperty ;
    rdfs:label "usage"@en ;
...
bldg:AbstractBuildingSubdivision.usage a owl:ObjectProperty ;
    rdfs:label "usage"@en ;
...
bldg:BuildingConstructiveElement.usage a owl:ObjectProperty ;
    rdfs:label "usage"@en ;
...
bldg:BuildingFurniture.usage a owl:ObjectProperty ;
    rdfs:label "usage"@en ;
...
bldg:BuildingInstallation.usage a owl:ObjectProperty ;
    rdfs:label "usage"@en ;
...
bldg:BuildingRoom.usage a owl:ObjectProperty ;
    rdfs:label "usage"@en ;
...
```

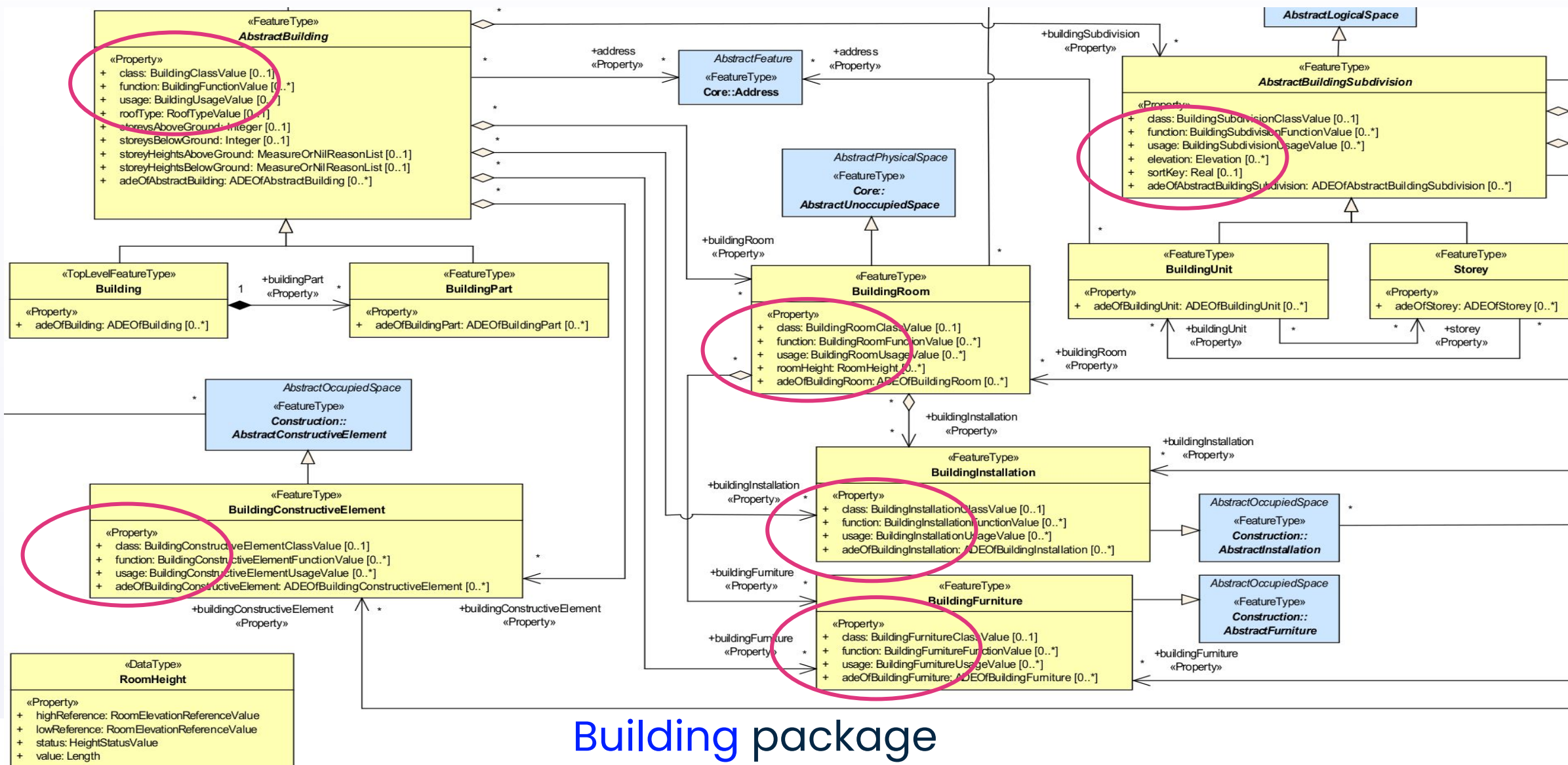
## Duplicate property name

### ns:Class.prop

property names repeat in a package,  
across packages



# CityGML 3.0 conceptual model



Building package



# Our Aim



Make CityRDF compact, more readable, yet following Semantic Web best practices

## Steps:

- Remove `ADE*-classes/ade*-properties` as optional
- deduplicate property names used more than once, global property reuse instead
  - create `common` namespace to keep all such properties with their stable parts of definitions
- avoid `ns:Class.property`, instead use `ns:property`





# Transformations



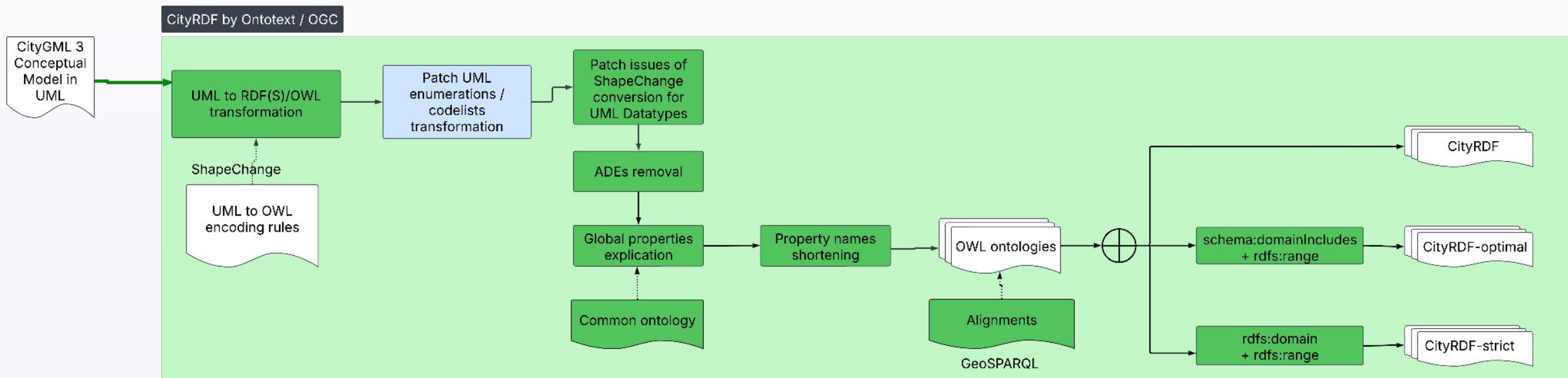
- Relay on UD-Graph ShapeChange conversion from .EA to .XML
- Take UD-Graph ShapeChange conversion from .XML to .RDF
- Extend it to have
  - [<<Union>>](#) stereotype conversion
  - global-scope attributes whenever possible (address, boundary, mimeType, etc.)
- [Extend a workflow](#) to get .OWL files







# CityRDF workflow (Ontotext/OGC)

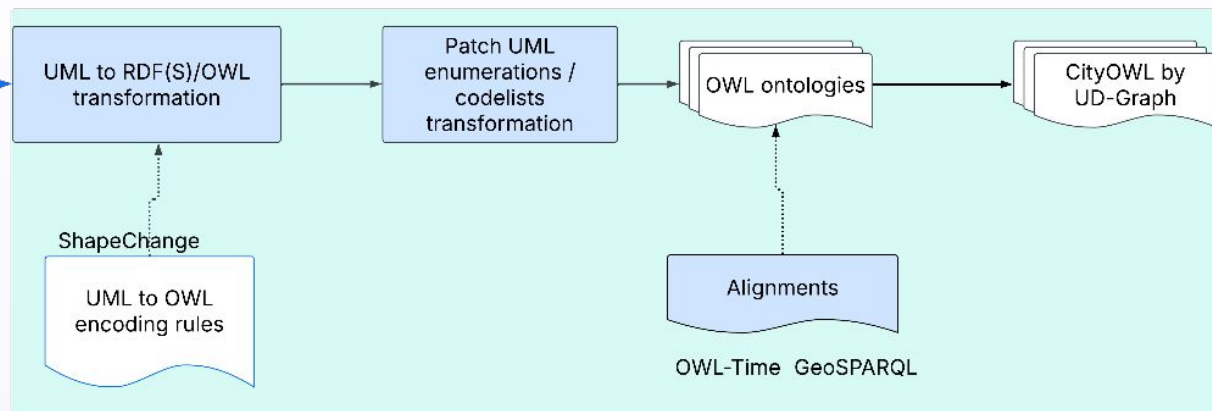


## Result

<https://github.com/ogcincubator/cityrdf/tree/master/citygml-owl/CityRDF>

# Comparison of workflows

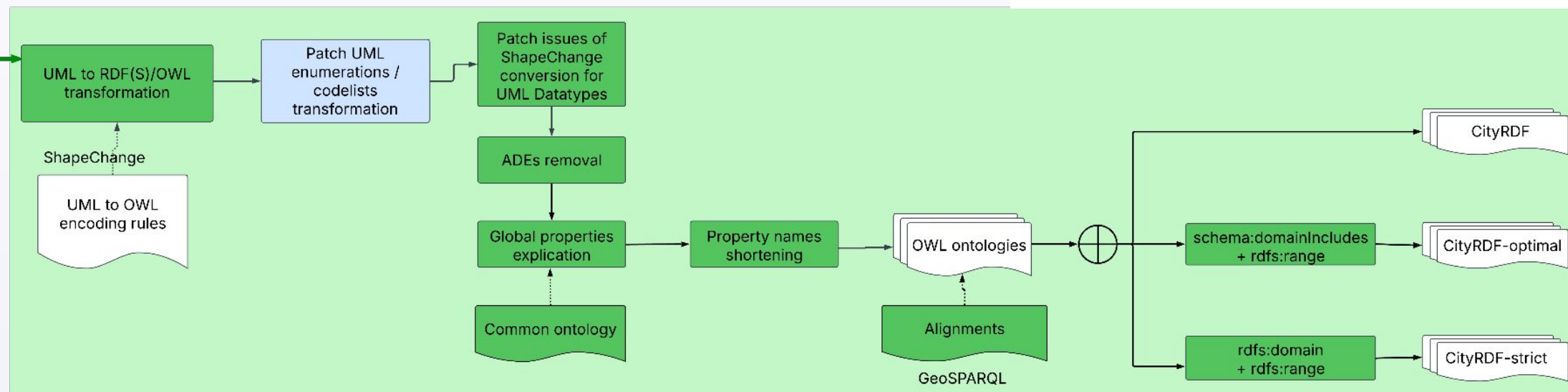
UD-Graph



**Diagram key**

- Activity
- document (XML, OWL, Turtle, UML)
- Ontology family
- UD-Graph transformation leading to CityOWL
- Ontotext/OGC transformation leading to CityRDF
- Distinguishing item

CityRDF by Ontotext / OGC





# ADE removal

CityGML 3 UML defines a set of Application Domain Extensions (ADEs). In fact, these are stubs to be used upon necessity.

We remove all ADE\* classes/ade\* properties from the ontologies, and leave their definitions and implementation at the user's responsibility.

To preserve the relatedness between classes, we will introduce an ADE ontology within CityRDF.



# Global properties explication



Made possible with ShapeChange.

- 1) ShapeChange configuration with [global scope attributes generation](#) allowed to select properties reused more than once.

All such properties are put to a new ontology called **common**. It supports definitions of object and datatype properties reusable several times across packages of CityGML.

- 2) ShapeChange was rerun *second time* with [localScopeAll generation](#), to disable *shuffling* of classes in subclass axioms.

Object Properties:

**"area", "class",  
"usage", "function",  
"mimeType",  
"occupancy",  
"elevation",  
"lowReference",  
"highReference";...**

Datatype Properties:

**"name",  
"description",  
"value", "status"**



# Shuffling of classes

```
<Warning message="Property mapping with potentially inconsistent ranges. Type of property 1 is
'BuildingRoomClassValue', while that of property 2 (to which 1 is mapped) is 'BuildingInstallationClassValue'.">
  <Detail message="--- Context - Property 1: Model::CityGML::Building::BuildingRoom::class"/>
  <Detail message="--- Context - Property 2: Model::CityGML::Building::BuildingInstallation::class"/>
</Warning>
<Warning message="Property mapping with potentially inconsistent definitions. Definition of property 1 is 'Indicates
the specific type of the BuildingRoom.', while that of property 2 (to which 1 is mapped) is 'Indicates the specific type
of the BuildingInstallation.'.">
  <Detail message="--- Context - Property 1: Model::CityGML::Building::BuildingRoom::class"/>
  <Detail message="--- Context - Property 2: Model::CityGML::Building::BuildingInstallation::class"/>
</Warning>
<Warning message="Property mapping with potentially inconsistent ranges. Type of property 1 is
'BuildingRoomFunctionValue', while that of property 2 (to which 1 is mapped) is 'BuildingInstallationFunctionValue'.">
  <Detail message="--- Context - Property 1: Model::CityGML::Building::BuildingRoom::function"/>
  <Detail message="--- Context - Property 2: Model::CityGML::Building::BuildingInstallation::function"/>
</Warning>
...
```



# Common ontology

## Common ontology

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix common: <https://www.opengis.net/ont/citygml/common/> .
@prefix geo: <http://www.opengis.net/ont/geosparql#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .

common:class rdf:type owl:ObjectProperty; rdfs:label "class"@en.
common:usage rdf:type owl:ObjectProperty; rdfs:label "usage"@en.
common:function rdf:type owl:ObjectProperty; rdfs:label "function"@en.
common:address rdf:type owl:ObjectProperty; rdfs:label "address"@en.
common:appearance rdf:type owl:ObjectProperty; rdfs:label "appearance"@en .
...
common:name rdf:type owl:DatatypeProperty; rdfs:label "name"@en
...
# GeoSPARQL alignments for common properties
common:lod0MultiCurve rdfs:subPropertyOf geo:hasGeometry .
common:lod0MultiSurface rdfs:subPropertyOf geo:hasGeometry .
common:boundary rdfs:subPropertyOf geo:hasGeometry .
...
```

# Property name shortening

```
bldg:Building.buildingPart a owl:ObjectProperty ;  
  rdfs:label "buildingPart"@en ;  
  rdfs:domain bldg:Building ; rdfs:range bldg:BuildingPart ;  
  skos:definition "Relates the building parts to the Building."@en .
```

```
bldg:Building a owl:Class ;  
  rdfs:label "Building"@en ;  
  rdfs:subClassOf [ a owl:Restriction ;  
    owl:allValuesFrom bldg:ADEOfBuilding ;  
    owl:onProperty bldg:Building.adeOfBuilding ],  
  [ a owl:Restriction ;  
    owl:allValuesFrom bldg:BuildingPart ;  
    owl:onProperty bldg:Building.buildingPart ],  
  bldg:AbstractBuilding ;  
  owl:disjointWith bldg:BuildingPart ;  
  skos:definition "A Building ..."@en .
```

```
bldg:AbstractBuilding a owl:Class ;  
  rdfs:label "AbstractBuilding"@en ; iso19150-2:isAbstract true ;  
  rdfs:subClassOf [ a owl:Restriction ;  
    owl:allValuesFrom core:Address ;  
    owl:onProperty bldg:AbstractBuilding.address ],...
```

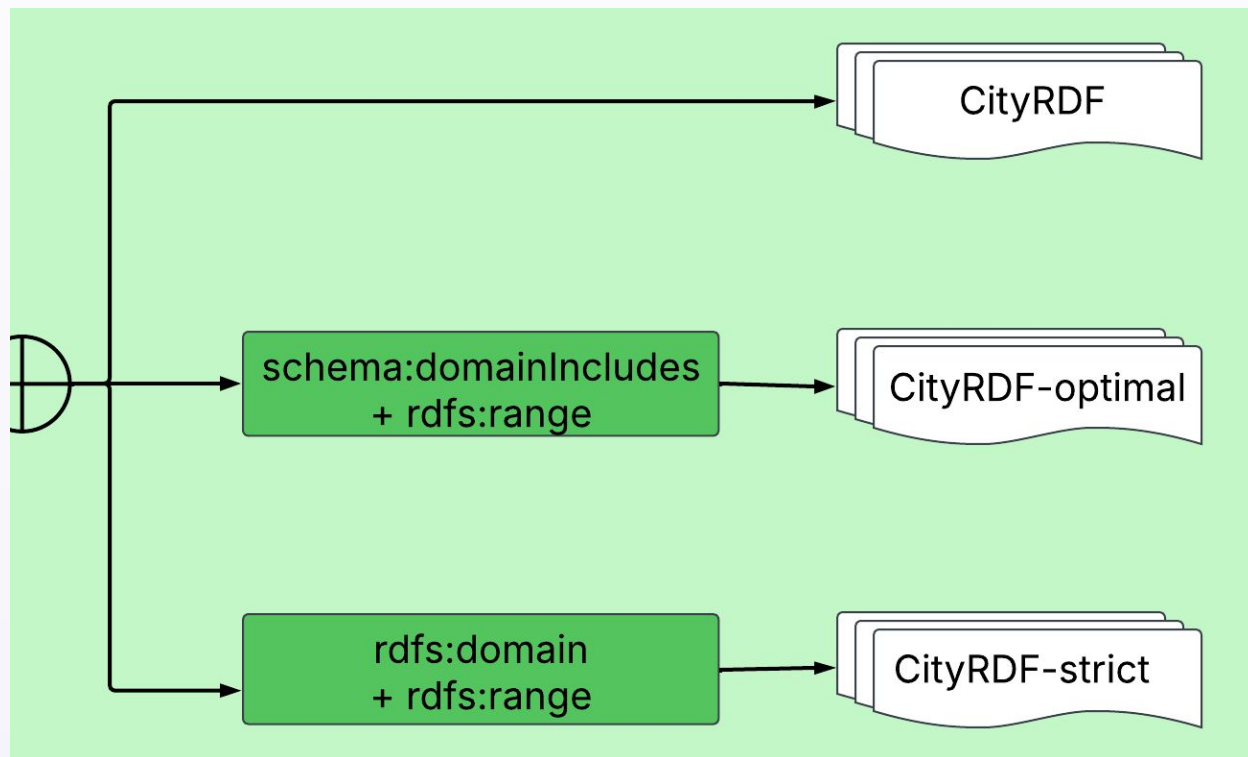
```
bldg:buildingPart a owl:ObjectProperty ;  
  rdfs:label "buildingPart"@en ;  
  rdfs:domain bldg:Building ; rdfs:range bldg:BuildingPart ;  
  skos:definition "Relates the building parts to the Building."@en .
```

```
bldg:Building a owl:Class ;  
  rdfs:label "Building"@en ;  
  rdfs:subClassOf  
    [ a owl:Restriction ;  
      owl:allValuesFrom bldg:BuildingPart ;  
      owl:onProperty bldg:buildingPart ],  
    bldg:AbstractBuilding ;  
  owl:disjointWith bldg:BuildingPart ;  
  skos:definition "A Building ..."@en .
```

```
bldg:AbstractBuilding a owl:Class ;  
  rdfs:label "AbstractBuilding"@en ; iso19150-2:isAbstract true ;  
  rdfs:subClassOf [ a owl:Restriction ;  
    owl:allValuesFrom core:Address ;  
    owl:onProperty common:address ],  
  ...
```



# Domain/range



Default behaviour

**schema:domainIncludes**  
+  
**schema:rangeIncludes**

gives maximum flexibility to extend CityRDF in applications, but limits reasoning.





# In Protege: Classes and Codelists

CityRDF (https://www.opengis.net/ont/citygml/CityRDF/3.0.0/)

Height

Active ontology x Entities x Individuals by class x DL Query x

Annotation properties Datatypes Individuals  
Classes Object properties Data properties

Class hierarchy: Height

Inferred

- owl:Thing
  - AbstractGenericAttribute
  - AbstractTextureParameterization
  - AnyFeature
  - CityModelMember
  - CityObjectRelation
  - ConstructionEvent
  - CV\_DiscreteGridPointCoverage
  - DoubleBetween0and1
  - DoubleBetween0and1List
  - DoubleList
  - DoubleOrNilReason
  - DoubleOrNilReasonList
  - Elevation
  - EngineeringCRS
  - ExternalReference
  - Geometry
  - GM\_TriangulatedSurface
  - Height**
  - ID
  - IntegerBetween0and3
  - MeasureOrNilReasonList
  - NilReason
  - Occupancy
  - QualifiedArea
  - QualifiedVolume
  - rdf:List
  - rdfs:Resource
  - Reference
  - Role
  - RoomHeight
  - SC\_CRS
  - SensorConnection
  - Sign
  - skos:Collection
  - skos:Concept
  - skos:ConceptScheme
  - skos:OrderedCollection
  - Tag
  - TextureAssociation
  - time:DayOfWeek

Annotations: Height

Annotations +

rdfs:label [language: en]  
Height

skos:definition [language: en]  
Height represents a vertical distance (measured or estimated high reference. [cf. INSPIRE])

Description: Height

Equivalent To +

SubClass Of +

- highReference **exactly** 1 ElevationReferenceValue
- highReference **only** ElevationReferenceValue
- lowReference **exactly** 1 ElevationReferenceValue
- lowReference **only** ElevationReferenceValue
- status **exactly** 1 HeightStatusValue
- status **only** HeightStatusValue
- value **exactly** 1 Measure
- value **only** Measure

General class axioms +

SubClass Of (Anonymous Ancestor)

Instances +

Target for Key +

Disjoint With +

Disjoint Union Of +

CityRDF (https://www.opengis.net/ont/citygml/CityRDF/3.0.0/)

skos:Concept > AuxiliaryTrafficAreaFunctionValue

Active ontology x Entities x Individuals by class x DL Query x

Annotation properties Datatypes Individuals  
Classes Object properties Data properties

Class hierarchy: AuxiliaryTrafficAreaFunctionValue

Inferred

- skos:Concept
  - AuthenticationTypeValue
  - AuxiliaryTrafficAreaClassValue
  - AuxiliaryTrafficAreaFunctionValue**
  - AuxiliaryTrafficAreaUsageValue
  - AuxiliaryTrafficSpaceClassValue
  - AuxiliaryTrafficSpaceFunctionValue
  - AuxiliaryTrafficSpaceUsageValue
  - BridgeClassValue
  - BridgeConstructiveElementClassValue
  - BridgeConstructiveElementFunctionValue
  - BridgeConstructiveElementUsageValue
  - BridgeFunctionValue
  - BridgeFurnitureClassValue
  - BridgeFurnitureFunctionValue
  - BridgeFurnitureUsageValue
  - BridgeInstallationClassValue
  - BridgeInstallationFunctionValue
  - BridgeInstallationUsageValue
  - BridgeRoomClassValue
  - BridgeRoomFunctionValue
  - BridgeRoomUsageValue
  - BridgeUsageValue
  - BuildingClassValue
  - BuildingConstructiveElementClassValue
  - BuildingConstructiveElementFunctionValue
  - BuildingConstructiveElementUsageValue
  - BuildingFunctionValue
  - BuildingFurnitureClassValue
  - BuildingFurnitureFunctionValue
  - BuildingFurnitureUsageValue
  - BuildingInstallationClassValue
  - BuildingInstallationFunctionValue
  - BuildingInstallationUsageValue
  - BuildingRoomClassValue
  - BuildingRoomFunctionValue
  - BuildingRoomUsageValue
  - BuildingSubdivisionClassValue
  - BuildingSubdivisionFunctionValue
  - BuildingSubdivisionUsageValue
  - BuildingUsageValue

Annotations: AuxiliaryTrafficAreaFunctionValue

Annotations +

rdfs:label [language: en]  
AuxiliaryTrafficAreaFunctionValue

skos:definition [language: en]  
AuxiliaryTrafficAreaFunctionValue is a code list that enumerates AuxiliaryTrafficArea.

Description: AuxiliaryTrafficAreaFunctionValue

Equivalent To +

SubClass Of +

- skos:Concept

General class axioms +

SubClass Of (Anonymous Ancestor)

Instances +

Target for Key +

Disjoint With +

Disjoint Union Of +



# In Protege: Datatypes, Object/Datatype properties

CityRDF (https://www.opengis.net/ont/citygml/CityRDF/3.0.0/)

Active ontology x Entities x Individuals by class x DL Query x

Annotation properties | Datatypes | Individuals  
Classes | Object properties | Data properties

Datatypes: ConditionOfConstructionValue

Annotations: ConditionOfConstructionValue

Annotations +

- rdfs:label [language: en]  
ConditionOfConstructionValue
- skos:definition [language: en]  
ConditionOfConstructionValue enumerates construction. [cf. INSPIRE]

Description: ConditionOfConstructionValue

Datatype Definitions +

- { "declined", "demolished", "functional", "ruin", "underConstruction" }

ConditionOfConstructionValue

- GranularityValue
- HeightStatusValue
- Measure
- owl:rational
- owl:real
- rdf:PlainLiteral
- rdf:XMLLiteral
- rdfs:Literal
- RelativeToConstruction
- RelativeToTerrain
- RelativeToWater
- Sign
- SpaceType
- TextureType
- time:generalDay
- time:generalMonth
- time:generalYear
- TimeseriesTypeValue
- TrafficDirectionValue
- TransactionTypeValue
- TransitionTypeValue
- WrapMode
- xsd:anyURI
- xsd:base64Binary
- xsd:boolean

CityRDF (https://www.opengis.net/ont/citygml/CityRDF/3.0.0/)

Active ontology x Entities x Individuals by class x

Annotation properties | Datatypes | Individuals  
Classes | Object properties | Data properties

Object property hierarchy:

Asserted

- owl:topObjectProperty
  - additionalGap
  - address
  - ambientIntensity
  - appearance
  - appearanceMember
  - appearanceValue
  - area
  - authType
  - auxiliaryTrafficSpace
  - borderColor
  - breaklines
  - bridgeConstructiveElement
  - bridgeFurniture
  - bridgeInstallation
  - bridgePart
  - bridgeRoom
  - buildingConstructiveElement
  - buildingFurniture
  - buildingInstallation
  - buildingPart
  - buildingRoom
  - buildingSubdivision
  - buildingUnit
  - cityModelMember
  - cityObjectMember
  - class
  - clearanceSpace
  - component

CityRDF (https://www.opengis.net/ont/citygml/CityRDF/3.0.0/)

Active ontology x Entities x Individuals by class x

Annotation properties | Datatypes | Individuals  
Classes | Object properties | Data properties

Data property hierarchy: attributeRef

Asserted

- owl:topDataProperty
  - attributeRef
  - averageHeight
  - baseURL
  - boolValue
  - clonePredecessor
  - codeSpace
  - conditionOfConstruction
  - count
  - creationDate
  - creator
  - crownDiameter
  - datastreamID
  - dateOfConstruction
  - dateOfDemolition
  - dateOfEvent
  - decimalSymbol
  - description
  - doubleValue
  - fieldSeparator
  - fileLocation
  - granularity
  - height
  - idColumnName
  - idColumnNo
  - idValue
  - imageURI
  - informationSystem
  - intValue



# CityRDF Summary



22 ontologies in RDF/Turtle:

21 – according to CityGML 3.0 Conceptual Model,

22th – [common](#) ontology – keeps global scope attributes declarations

possibly, 23rd – ADE ontology – will keep a superclass for all ADE\* classes mentioned in CityGML 3 UML

Codelists defined via `skos:ConceptSchema` and `skos:Concept` are stored separately, to avoid punning in OWL.

Ontology family is consistent (tested with Pellet and Hermit), openable in Protege 5.5



The background features a light gray grid of dots on a white background. Overlaid on this are several large, semi-transparent circles in shades of light blue and light gray. A small, solid blue circle is visible on the left edge.

# **FAIR Principles**





## Findability

F1: (Meta)data are assigned a globally unique and persistent identifier

+

F2: Data are described with rich metadata

partially , depends on CityGML 3.0

F3: Metadata clearly and explicitly include the identifier of the data they describe

+

F4: (Meta)data are registered or indexed in a searchable resource

ongoing (OGC)

## Accessibility

A1.1: The protocol of retrieving (meta)data is open, free, and universally implementable

+ (OGC)

A1.2: The protocol allows for an authentication and authorisation procedure, where necessary

+ (OGC)

A2: Metadata are accessible, even when the data are no longer available

ongoing (OGC)

## Interoperability

I1: (Meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation

+ (RDF)

I2: (Meta)data use vocabularies that follow FAIR principles

partially, depends on GML3.2, time, iso19150-2

I3: (Meta)data include qualified references to other (meta)data

partially, depends on GML3.2, time, iso19150-2

## Reuse

R1.1: (Meta)data are released with a clear and accessible data usage license

ongoing (OGC)

R1.2: (Meta)data are associated with detailed provenance

ongoing (OGC)

R1.3: (Meta)data meet domain-relevant community standards

ongoing (OGC)

**In total: 5 OK, 5 ongoing, 3 partially implementable**





# CityRDF Usage



- as the target ontology to run various transformations from CityGML to RDF/OWL  
e.g. xSPARQL, FME, others
- as a modeling language for Urban Data (with validation in SHACL shapes generated from CityRDF)
- in data profiling





# Future work

- Resolve possible errors – please report them in the [issues](#)
- Create ADE ontology
- Separate [Appearance ontology from CityRDF](#) to reuse in GeoSPARQL 1.3 proposal
- Adopt it further in projects



# Thank you for attention!

<https://github.com/ogcincubator/cityrdf/tree/master/citygml-owl>

If you have any questions, write us:

[nataliya.keberle@graphwise.ai](mailto:nataliya.keberle@graphwise.ai) and [arkadiusz.chadzynski@graphwise.ai](mailto:arkadiusz.chadzynski@graphwise.ai)