Investigating the Overture Maps schemas from an ISO/TC 211 and OGC standards point of view

# Introduction

ISO/TC 211 and OGC have based their standardisation of information models for geospatial information on the Unified Modelling Language (UML) and a model-driven architecture (MDA) approach since 1998 when ISO/TC 211 decided to use UML as conceptual schema language instead of EXPRESS. These UML models lay the fundament for geospatial standards used worldwide, such as OGC CityGML, OGC IndoorGML, the INSPIRE and TN-ITS models in Europe, and national standards like OKSTRA in Germany and SOSI in Norway. Vast amounts of public geospatial data are collected, maintained and shared based on these standards. Database schemas and file implementation schemas are derived from the models through the MDA approach. The Geography Markup Language (GML) has been the official standardised format for information exchange.

Over the last few years, alternative representations have been introduced for information modelling and file exchange in OGC and ISO/TC 211. One example is JSON, which is used as an implementation format for, for example, CityGML (CityJSON) as an alternative to GML. Another example is Semantic Web technologies, especially the Web Ontology Language (OWL) for information modelling. The most prominent example is OGC GeoSPARQL, which is widely used in the Semantic Web.

The new player in this field is the Overture Maps Foundation, which develops schemas for its data themes. Unlike OGC and ISO/TC 211, Overture Maps are developing schemas in YAML, a JSON-related native format. The structure in the YAML files is based on JSON Schema (https://json-schema.org/).

# Methodology

This document aims to investigate the Overture Maps schemas from the perspective of ISO/TC 211 and OGC and find differences and similarities between their approaches.

The approach for the investigation is to apply Python scripts for mapping the Overture Maps YAML files to ISO/TC 211 conformant UML and OWL. The motivation is that such a mapping may provide a more familiar view of the Overture Maps schemas in the GIS domain and enable a mapping between Overture Maps schemas and standardised schemas based on ISO/TC 211 standards.

The work is done as part of the MODI project, with the intention of finding standardised ways of exchanging cross-border and harmonised navigable road networks for automated driving.

The Overture Maps schemas are developed and available on GitHub - <https://github.com/OvertureMaps/schema>.

Scripts and other results from this work are publicly available on GitHub - <https://github.com/jetgeo/OM2UML>.

# Overture Maps schemas

## Documentation

The OvertureMaps schemas are developed and available on GitHub - <https://github.com/OvertureMaps/schema>.

Explanations and a browsable representation of the schemas are available at <https://docs.overturemaps.org/>.

## Schema structure

The Overture Maps schemas are divided into five themes: Base, Administrative boundaries, Buildings, Places, and Transportation. Besides, common properties and types are defined for use across themes.

There is one YAML file for each object type within each theme, besides common properties and types in a separate file (“defs.yaml”).

Note: property definitions may also be described within an individual schema.

Et bilde som inneholder tekst, skjermbilde, Font

Automatisk generert beskrivelse

Figure 1 The Overture Maps schema folder structure and the common “defs.yaml” file

Et bilde som inneholder tekst, Font, skjermbilde

Automatisk generert beskrivelse

Figure 2. The file structure is within the Overture Maps “admins” folder.

## Common property types and property container types (“defs.yaml”)

The “def.yaml” files define properties and property containers for use across all schemas (the main “defs.yaml” in Figure 1) or schemas within a time (or, for example, the “defs.yaml” within the “admin” folder in Figure 2).

* Property types are defined under the heading “$defs/propertyDefinitions” with a name, followed by property type attributes such as description, type, format and more.
* Property container types are groups of property types and are defined under the heading “$defs/propertyContainers”

## Individual object type schemas

The individual object type schemas (for example, “administrativeBoundary.yaml” in Figure 2) define individual object types and their property types under the heading “properties”. Each property type is defined with a name and additional attributes such as a description, reference to a data type and more. Property types may also be complex, with property types within property types.

Besides, individual object type schemas may also contain property type definitions (“$defs/propertyDefinitions”) and property container types (“$defs/propertyContainers”) for use only on the object type.

# Transforming from YAML to ISO/TC 211 UML

## Basic principles

Scripts and other results from this work are publicly available on GitHub - <https://github.com/jetgeo/OM2UML>

The transformation is implemented in a Python script that connects with an Enterprise Architect repository.

The folders divide the Overture Maps schemas into themes. This is equivalent to packages in UML.

Conversion rule: Each Overture Maps folder (ref Figure 1) is transformed into a UML package with name = Title case (folder name). Besides, the root folder is transformed into a folder named “Common”.

Figure 3 shows the Overture Maps UML packages.

Et bilde som inneholder tekst, Font, skjermbilde

Automatisk generert beskrivelse

Figure 3. Overture Maps UML Packages

## Object and data types

The YAML schemas contain three main types of concepts:

1. Property types for the object type in question,
2. property definition types that are either for use in any Overture Maps schema, any schema within an Overture Maps theme, or the object type in question and
3. property container types for use in any Overture Maps schema, schema within an Overture Maps theme, or the object type in question.

An Overture Maps object type is considered equivalent to a UML class and, more specifically, a feature type according to ISO 19109.

Conversion rule: Each Overture Maps object type schema is transformed into a UML class with the stereotype “FeatureType” and with name = Upper Camel Case (schema name).

Property definitions can be considered a list of global property types for use in any class in any Overture Maps schema, any schema within an Overture Maps theme, or the object type in question. In UML, a class owns property types; they cannot exist independently. Property definitions must, therefore, be put into a UML class. For this purpose, the AttributeCatalogue construct from Jetlund et al. (2019) [1] is used, but with an adapted name scheme.

Conversion rule: One abstract UML class with stereotype “FeatureType” is created for each Overture Maps theme where property definitions are defined and one for the root folder (the Common package in UML). The name is set to the package name + “Defs”.

Figure 4 shows all Overture Maps Feature Types.

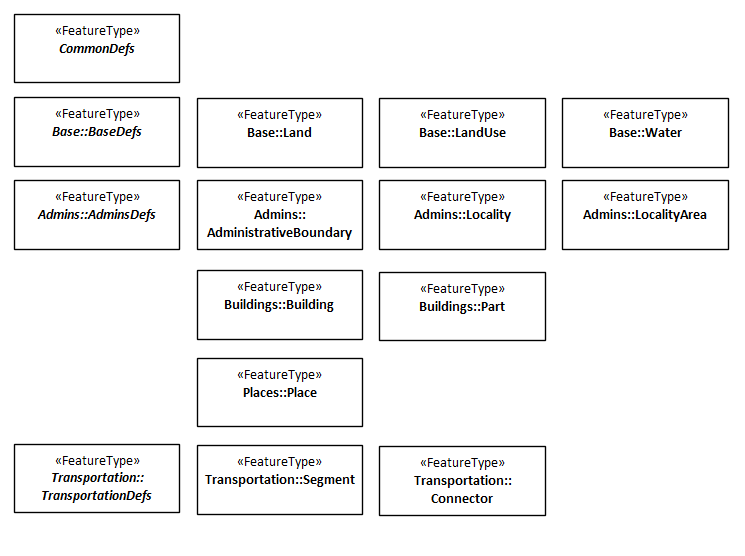


Figure 4. Overture Maps Feature Types, including abstract Def types

Property container types are groups of property types and can be considered equivalent to data types in UML.

Conversion rule: Each Overture Maps property container type is transformed into a UML Data Type with name = Upper Camel Case (container name).

Besides property container types, the Overture Maps schemas also contain complex property definition types or property types with value type “object”, where property types are defined within the property type. These kinds of value types are also considered equivalent to UML data types.

Conversion rule: For Property types with value type “object”, a data type is created with name = Upper Camel Case (property type name + “Type”)

Some property definition types have lists of “enum” values as valid value types. These are equivalent to UML Enumerations, which is a special kind of data type.

Conversion rule: For Property definition types with “enum” values, an enumeration is created with name = Upper Camel Case (property type name + “Enum”).

Besides property definition types with “enum” values, the Overture Maps schemas also contain “enum” values for property types within the specific schema. These must be given a prefix to avoid duplication.

Conversion rule: For Property types with “enum” values, within an object type, an enumeration is created with name = Upper Camel Case (Feature type name + property type name + “Enum”).

Figure 5 shows all classifiers (Feature Types, Data Types and Enumerations) in the Overture Maps Common package. Figure 6 shows all classifiers in the Overture Maps Base package, where different classes have different enumerations for the subtype and class property types.

Et bilde som inneholder tekst, skjermbilde, Font, line

Automatisk generert beskrivelse

Figure 5 All classifiers for the Overture Maps Common package

Et bilde som inneholder tekst, skjermbilde, Font, nummer

Automatisk generert beskrivelse

Figure 6. All classifiers for the Overture Maps Base package

## Property types

TODO: Fix plural in names?

Property types from YAML property definitions, property containers and object types are equivalent to UML attributes.

Conversion rule: Simple property definition types and simple property types are transformed into UML attributes under the abstract “Def” class in question. The name is not changed from YAML.

Conversion rule: property types under property types in complex property definition types and complex property types are transformed to UML attributes under the data type in question. The name is not changed from YAML.

Conversion rule: property types under property containers are transformed into attributes under the data type in question. The name is not changed from YAML.

Conversion of property type value types is the most complex part of the process since the Overture Maps YAML schemas contain more advanced constructs where properties may have one or all of the selected value types.

Value types for primitive property types and property definition types are, for the most, easily converted into ISO/TC 211 UML types.

Conversion rule: Transformation from simple YAML/JSON Schema types to ISO/TC 211 types:

YAML.string to UML.CharacterString

YAML.integer to UML.Integer

YAML.number to UML.Real

YAML.boolean to UML.Boolean

Alternatively, property types and property definition types in YAML can have references to a property type or a property definition type defined elsewhere (“$ref:”). Such a reference means that the property type shall have the same characteristics as the referenced type.

For example, each Overture Maps object type has an attribute “id”, with reference to “../defs.yaml#/$defs/propertyDefinitions/id”. This property definition type is defined in the core “defs.yaml” schema, with value type “string”, which is “CharacterString” in the UML model. Therefore, the attribute “id” in each schema also shall have value type “CharacterString”. Furthermore, the definiton and multiplicity of the referenced type is also inherited if it is not specifically defined for the type in question.

Conversion rule: For references to property definition types (“$ref:”), the following applies:

UML.Type = The type of the referenced property definition type

If UML cardinality is not specifically defined: The cardinality is set identical to the cardinality of the referenced property definition type.

If the property type does not have a definition: The definition is set identical to the definition of the referenced property definition type.

The value type “array” in YAML is used when there may be multiple items of a property type under one object type. In principle, this is equivalent to multiplicity (cardinality) with an upper bound of more than 1 in UML. For example, a property type with no upper limit has cardinality [0..\*], while a property type that shall have exactly two items has cardinality [2].

While UML can have multiple instances (items) of any property type defined by the upper bound, this is only relevant for arrays in YAML. Therefore, only property types with value type “array” can have an upper bound of more than one.

Like the value type for other property types, arrays can be simple or complex. A simple array can have specified a primitive value type (string, integer, etc.), or it can have a reference (“$ref”) to a property definition type.

Conversion rule: Transformation of value types for simple arrays.

Simple value types and simple references are transformed to ISO/TC 211 types as described earlier.

Example in Figure 7: The property type “connectors” in the “segment.yaml” schema in an array of stings, and is converted into the UML attribute “connectors” with value type “CharacterString” and multiplicity “0..\*”.

Et bilde som inneholder tekst, Font, line, skjermbilde

Automatisk generert beskrivelse

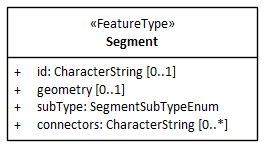


Figure 7. Example of a simple array

Complex arrays are similar to object value types, where each array item is a group of properties. For these, a data type is created in the same way as for object values, as shown in Figure 8.

Et bilde som inneholder tekst, skjermbilde, Font

Automatisk generert beskrivelse

Et bilde som inneholder tekst, skjermbilde, Font, nummer

Automatisk generert beskrivelse

Figure 8. Complex array for a property

Conversion rule:

If the array is not defined as value type for a property, but for the item within a container, a new property type with suffix “Item” is created, as illustrated in Figure 8.

Et bilde som inneholder tekst, skjermbilde, Font, nummer

Automatisk generert beskrivelse

Et bilde som inneholder tekst, skjermbilde, Font, line

Automatisk generert beskrivelse

Figure 9. Complex array in a container

Conversion rule:

PrefixItem: Array where each item has a different schema. [1..1] for the data type. *“prefixItems is an array, where each item is a schema that corresponds to each index of the document's array. That is, an array where the first element validates the first element of the input array, the second element validates the second element of the input array, etc.”*

DONE, describe!

TODO: OneOf, AllOf…

TODO:

* RoadType.flags 🡪 Still “Array” 🡪 oneOf, allOf
* AccessContainer 🡪 Still empty 🡪 oneOf, allOf
* VehicleType 🡪 no value type for attributes.
  + YAML type = relationalExpression (in TransportationDefs) 🡪 OneOf

AllOf = All shall have LowerBound and UpperBound >= 1

OneOf = All optional (LowerBound = 0) + constraint count = 1

Et bilde som inneholder tekst, skjermbilde, Font, nummer

Automatisk generert beskrivelse

For some object types and data types, the YAML schemas contain lists of required property types, following the keyword “required”. Specific lower and upper bounds in YAML are specified with the keywords “minItems” and “maxItems”. Property types without these settings are considered to have UML multiplicity [0..1], except for arrays, which are considered to have default multiplicity [0..\*]. Besides, “uniqueItems” can specify that duplicates are not allowed, and “default” can specify a default value for the property type.

NOTE: Some occurrences exists where a property type is not defined as required but has a “minItems” making it required. On this point, there seems to be some inconsistency in the schemas.

Conversion rule: Transformation of cardinality, uniqueness, and default value information from YAML to UML:

Default: UML.LowerBound = 0.

If YAML.**required** contains property type name: 1

If YAML.minItems is set: UML.LowerBound = YAML.minItems

Default if the value type is not “array”: UML.UpperBound = 1

Default if the value type is “array”: UML.UpperBound = \*

If YAML.maxItems is set: UML.UpperBound = YAML.maxItems

YAML.uniqueItems is transformed to to UML.AllowDuplicates (isUnique)

YAML.default is transformed to UML.Default

Restriction object both as property definition type and inside the lane property type? Minor differences – prohibitedTransitions vs minOccupancy.

TODO: Add both? laneRestrictions as specialisation?

Conversion rule: Transformation of value type reference to data type for items – properties.

What is the difference between properties and items?

*“The value of items must be a JSON Schema object or an array JSON Schema objects (let's ignore if it's an array for now). The array that the items key word is applicable to, passes validation if all items in that array validate against the schema*. *The value of properties must be an object. Each value in the object must be a json schema. The schemas in the object are applied to the instance object when the keys match.*”*.”(* [*https://stackoverflow.com/questions/51643224/what-is-the-difference-between-items-and-properties-in-jsonschema*](https://stackoverflow.com/questions/51643224/what-is-the-difference-between-items-and-properties-in-jsonschema)*)*

Minimum, maximum 🡪 item value

…geometry…

Remaining challenges…

## Enumeration values

Items in “enum” lists in the YAML schemas are equivalent to values in UML enumeration classifiers.

Conversion rule: Enumeration values are extracted from the YAML lists into values in UML enumeration classifiers.

Figure 7 shows enumeration classifiers and their values from the Overture Maps Common package. Figure 8 shows how the Overture Maps Base package has different enumerations for the property “subtype” in different Feature Types (Land, LandUse and Water).

Et bilde som inneholder tekst, skjermbilde, Font, nummer

Automatisk generert beskrivelse

Figure 10. Enumerations from the Overture Maps Common package

Et bilde som inneholder tekst, skjermbilde, Font, nummer

Automatisk generert beskrivelse

Figure 11. Different enumerations for the subtype property type in different Feature Types

# Complete UML diagrams

# Remaining challenges

# References

[1] Jetlund, K., Onstein, E., Huang, L. (2019). Adapted Rules for UML Modelling of Geospatial Information for Model-Driven Implementation as OWL Ontologies*.* *ISPRS International Journal of Geo-Information*, *8(9)*, p. 365,DOI: <https://doi.org/10.3390/ijgi8090365>.