Open Geospatial Consortium

Submission Date: 2021-07-15

Approval Date: 2021-07-15

Publication Date: 2021-09-13

External identifier of this OGC® document: http://www.opengis.net/doc/UG/CityGML-user-guide/3.0

Internal reference number of this OGC® document: 20-066

Version: 1.0

Category: OGC® User Guide

Editor: Charles Heazel

OGC City Geography Markup Language (CityGML) 3.0 Conceptual Model Users Guide

Warning

This document is not an OGC Standard. This document provides guidance on the use of the OGC CityGML: 3.0 Conceptual Model Standard. This document is a non-normative resource and not an official position of the OGC membership. It is subject to change without notice and may not be referred to as an OGC Standard. Further, User Guides should not be referenced as required or mandatory technology in procurements.

Document type:	OGC® User Guide
Document subtype:	
Document stage:	Approved
Document language:	English

License Agreement

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications. This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

Table of Contents

i. Abstract

CityGML is an open conceptual data model for the storage and exchange of virtual 3D city models. It is defined through a Unified Modeling Language (UML) object model. This UML model extends the ISO Technical Committee 211 (TC211) conceptual model standards for spatial and temporal data. Building on the ISO foundation assures that the man-made features described in the City Models share the same spatial-temporal universe as the surrounding countryside within which they reside. The aim of the development of CityGML is to reach a common definition of the basic entities, attributes, and relations of a 3D city model. This is especially important with respect to the cost-effective sustainable maintenance of 3D city models, allowing the reuse of the same data in different application fields.

This Users Guide provides extended explanations and examples for the individual concepts that are defined in the CityGML 3.0 Conceptual Model Standard. Both documents, the Conceptual Model Standard and the Users Guide, are mutually linked to facilitate navigation between corresponding sections in these documents.

ii. Keywords

The following are keywords to be used by search engines and document catalogues.

ogcdoc, OGC document, CityGML, 3D city models

iii. Preface

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

iv. Submitting organizations

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

- Heazeltech LLC

v. Submitters

All questions regarding this submission should be directed to the editor or the submitters:

Table 1. Submission Contact Points

Name	Institution
Charles (Chuck) Heazel	HeazelTech LLC

vi. Contributors

The following individuals contributed content to the CityGML 3.0 Users Guide:

Table 2. Participants in the Users Guide development

Name	**Institution**
Emmanuel Devys	Institut national de l'information géographique et forestière (IGN), France
Charles (Chuck) Heazel	HeazelTech LLC
Tatjana Kutzner	Chair of Geoinformatics, Technical University of Munich, Germany

Chapter 1. Introduction

An increasing number of cities and companies are building virtual 3D city models for different application areas like urban planning, mobile telecommunication, disaster management, 3D cadastre, tourism, vehicle and pedestrian navigation, facility management and environmental simulations. Furthermore, in the implementation of the European Environmental Noise Directive (END, 2002/49/EC) 3D geoinformation and 3D city models play an important role.

In recent years, most virtual 3D city models have been defined as purely graphical or geometrical models, neglecting the semantic and topological aspects. Thus, these models could almost only be used for visualization purposes but not for thematic queries, analysis tasks, or spatial data mining. Since the limited reusability of models inhibits the broader use of 3D city models, a more general modeling approach had to be taken in order to satisfy the information needs of the various application fields.

CityGML is a common semantic information model for the representation of 3D urban objects that can be shared over different applications. The latter capability is especially important with respect to the cost-effective sustain-able maintenance of 3D city models, allowing the possibility of selling the same data to customers from different application fields. The targeted application areas explicitly include city planning, architectural design, tourist and leisure activities, environmental simulation, mobile telecommunication, disaster management, homeland security, real estate management, vehicle and pedestrian navigation, and training simulators.

CityGML is an open conceptual data model for the storage and exchange of virtual 3D city models. It is defined through a Unified Modeling Language (UML) object model. This UML model extends the ISO Technical Committee 211 (TC211) conceptual model standards for spatial and temporal data. Building on the ISO foundation assures that the man-made features described in the City Models share the same spatial-temporal universe as the surrounding countryside within which they reside.

CityGML defines the classes and relations for the most relevant topographic objects in cities and regional models with respect to their geometrical, topological, semantical, and appearance properties. “City” is broadly defined to comprise not just built structures, but also elevation, vegetation, water bodies, “city furniture”, and more. Included are generalization hierarchies between thematic classes, aggregations, relations between objects, and spatial properties. CityGML is applicable for large areas and small regions and can represent the terrain and 3D objects in different levels of detail simultaneously. Since either simple, single scale models without topology and few semantics or very complex multi-scale models with full topology and fine-grained semantical differentiations can be represented, CityGML enables lossless information exchange between different GI systems and users.

The CityGML 3.0 standard consists of several parts: 1) The CityGML 3.0 Conceptual Model standard that defines the conceptual model in UML and that is described in more detail within this Users Guide. 2) A separate Encoding standard for each Encoding to be defined. This will be the GML Encoding in the beginning, further encoding specifications (e.g., relational database schema, JSON-based representation) will follow in the future.

Chapter 2. How To Use This Resource

The Users Guide to the CityGML 3.0 Conceptual Model Standard is not intended to be read from start to finish. Rather, it is a resource structured to provide quick answers to questions which an implementer may have about the CityGML 3.0 Standard.

The CityGML 3.0 Standard includes hyperlinks which can be used to navigate directly to relevant sections of the Users Guide.

Some content in the Users Guide has been copied from the CityGML 3.0 Conceptual Model Standard to make the content more accessible to the user. In order to make clear which content in the Users Guide has been copied, the copied text is provided within grey boxes.

```
This text has been copied from the CityGML 3.0 Conceptual Model Standard.
```

All other texts are provided exclusively in this Users Guide.

Chapter 3. Scope

This document provides Engineering Guidance on the use of the CityGML 3.0 Conceptual Model Standard.

The OGC Conceptual Model Standard specifies the representation of virtual 3D city and landscape models. The CityGML 3.0 Conceptual Model is expected to be the basis for a number of future Encoding Standards in which subsets of the Conceptual Model can be implemented. These Encoding Standards will enable both storage and exchange of data.

The CityGML 3.0 Conceptual Model Standard was designed to be concise and easy to use. As a result, most non-normative content has been removed. The purpose of this Users Guide is to capture that non-normative content and make it easy to access if and when needed.

Chapter 4. References

The following documents contain provisions that, through reference in this text, constitute provisions of this Users Guide. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the document referred to applies.

- IETF: RFC 2045 & 2046, Multipurpose Internet Mail Extensions (MIME). (November 1996),
- IETF: RFC 3986, Uniform Resource Identifier (URI): Generic Syntax. (January 2005)
- INSPIRE: D2.8.III.2 Data Specification on Buildings – Technical Guidelines. European Commission Joint Research Centre.
- ISO: ISO 19101-1:2014, Geographic information - Reference model - Part 1: Fundamentals
- ISO: ISO 19103:2015, Geographic Information – Conceptual Schema Language
- ISO: ISO 19105:2000, Geographic information – Conformance and testing
- ISO: ISO 19107:2003, Geographic Information – Spatial Schema
- ISO: ISO 19108:2002/Cor 1:2006, Geographic information – Temporal schema — Technical Corrigendum 1
- ISO: ISO 19109:2015, Geographic Information – Rules for Application Schemas
- ISO: ISO 19111:2019, Geographic information – Referencing by coordinates
- ISO: ISO 19123:2005, Geographic information — Schema for coverage geometry and functions
- ISO: ISO 19156:2011, Geographic information – Observations and measurements
- ISO: ISO/IEC 19505-2:2012, Information technology — Object Management Group Unified Modeling Language (OMG UML) — Part 2: Superstructure
- ISO/IEC 19507:2012, Information technology — Object Management Group Object Constraint Language (OCL)
- ISO: ISO/IEC 19775-1:2013 Information technology — Computer graphics, image processing and environmental data representation — Extensible 3D (X3D) — Part 1: Architecture and base components
- Khronos Group Inc.: COLLADA – Digital Asset Schema Release 1.5.0
- OASIS: Customer Information Quality Specifications - extensible Address Language (xAL), Version v3.0
- OGC: The OpenGIS® Abstract Specification Topic 5: Features, OGC document 08-126
- OGC: The OpenGIS™ Abstract Specification Topic 8: Relationships Between Features, OGC document 99-108r2
- OGC: The OpenGIS™ Abstract Specification Topic 10: Feature Collections, OGC document 99-110

Chapter 5. Terms and Definitions

For the purposes of this document, the following additional terms and definitions apply.

2D data
geometry of features is represented in a two-dimensional space
NOTE In other words, the geometry of 2D data is given using (X,Y) coordinates.
[INSPIRE D2.8.III.2, definition 1]

2.5D data
geometry of features is represented in a three-dimensional space with the constraint that, for each (X,Y) position, there is only one Z
[INSPIRE D2.8.III.2, definition 2]

3D data
Geometry of features is represented in a three-dimensional space.
NOTE In other words, the geometry of 2D data is given using (X,Y,Z) coordinates without any constraints.
[INSPIRE D2.8.III.2, definition 3]

application schema
A set of conceptual schema for data required by one or more applications. An application schema contains selected parts of the base schemas presented in the ORM Information Viewpoint. Designers of application schemas may extend or restrict the types defined in the base schemas to define appropriate types for an application domain. Application schemas are information models for a specific information community.
OGC Definitions Register at http://www.opengis.net/def/glossary/term/ApplicationSchema

codelist
A value domain including a code for each permissible value.

conceptual model
model that defines concepts of a universe of discourse
[ISO 19101-1:2014, 4.1.5]

conceptual schema

1. formal description of a conceptual model
   [ISO 19101-1:2014, 4.1.6]
2. base schema. Formal description of the model of any geospatial information. Application schemas are built from conceptual schemas.
   OGC Definitions Register at http://www.opengis.net/def/glossary/term/ConceptualSchema

Implementation Specification
Specified on the OGC Document Types Register at http://www.opengis.net/def/doc-type/is

levels of detail
quantity of information that portrays the real world
NOTE The concept comprises data capturing rules of spatial object types, the accuracy and the types

of geometries, and other aspects of a data specification. In particular, it is related to the notions of scale and resolution.
[INSPIRE Glossary]

life-cycle information
set of properties of a spatial object that describe the temporal characteristics of a version of a spatial object or the changes between versions
[INSPIRE Glossary]

Platform (Model Driven Architecture)
the set of resources on which a system is realized.
[Object Management Group, Model Driven Architecture Guide rev. 2.0]

Platform Independent Model
a model that is independent of a spacific platform
[Object Management Group, Model Driven Architecture Guide rev. 2.0]

Platform Specific Model
a model of a system that is defined in terms of a specific platform
[Object Management Group, Model Driven Architecture Guide rev. 2.0]

Universally Unique Identifier
A 128-bit value generated in accordance with this Recommendation | International Standard, or in accordance with some historical specifications, and providing unique values between systems and over time. [ISO/IEC 9834-8:2014, Rec. ITU-T X.667 (10/2012)]

universe of discourse
view of the real or hypothetical world that includes everything of interest
[ISO 19101-1:2014, definition 4.1.38]

Chapter 6. Conventions

(Imported from the CityGML 3.0 Standard)

6.1. Identifiers

```
The normative provisions in this document are denoted by the URI
```

http://www.opengis.net/eg/CityGML-1/3.0

```
All requirements and conformance tests that appear in this document are denoted by
partial URIs relative to this base.
```

6.2. UML Notation

```
The CityGML Conceptual Model (CM) Standard is presented in this document through
diagrams using the Unified Modeling Language (UML) static structure diagram (see Booch
et al. 1997). The UML notations used in this standard are described in the diagram in
<<figure-1>>.
```

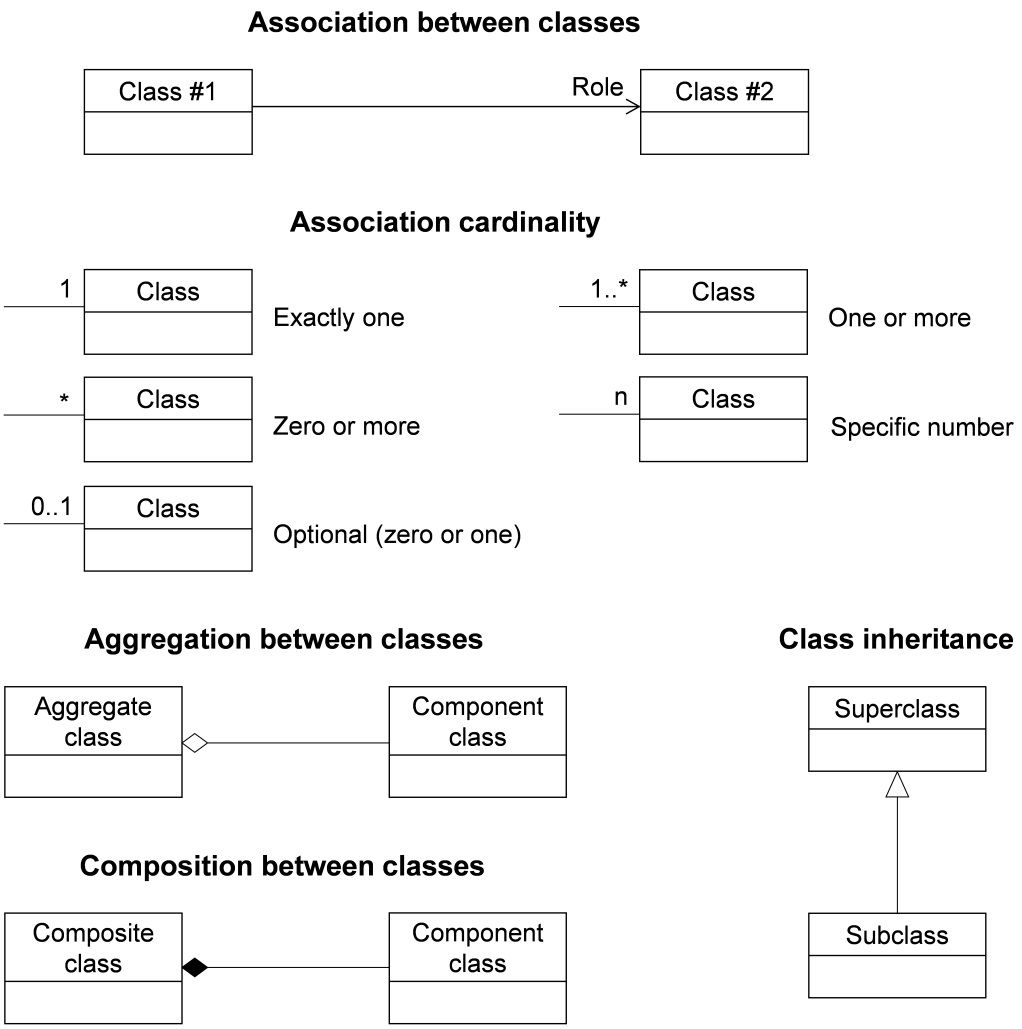


Figure 1. UML notation (see ISO TS 19103, Geographic information - Conceptual schema language).

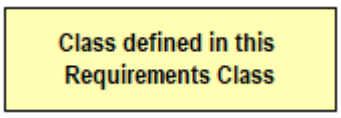
```
All associations between model elements in the CityGML Conceptual Model are uni-
directional. Thus, associations in the model are navigable in only one direction. The
direction of navigation is depicted by an arrowhead. In general, the context an
element takes within the association is indicated by its role. The role is displayed
near the target of the association. If the graphical representation is ambiguous
though, the position of the role has to be drawn to the element the association points
to.
```

```
The following stereotypes are used in this model:
```

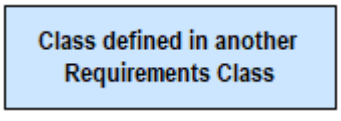
- «ApplicationSchema» denotes a conceptual schema for data required by one or more applications. In the CityGML Conceptual Model, every module is defined as a separate application schema to allow for modularization.
- «FeatureType» represents features that are similar and exhibit common characteristics. Features are abstractions of real-world phenomena and have an identity.
- «TopLevelFeatureType» denotes features that represent the main components of the conceptual model. Top-level features may be further semantically and spatially decomposed and substructured into parts.

- «Type» denotes classes that are not directly instantiable, but are used as an abstract collection of operation, attribute and relation signatures. The stereotype is used in the CityGML Conceptual Model only for classes that are imported from the ISO standards 19107, 19109, 19111, and 19123.
- «ObjectType» represents objects that have an identity, but are not features.
- «DataType» defines a set of properties that lack identity. A data type is a classifier with no operations, whose primary purpose is to hold information.
- «Enumeration» enumerates the valid attribute values in a fixed list of named literal values. Enumerations are specified in the CityGML Conceptual Model.
- «BasicType» defines a basic data type.
- «CodeList» enumerates the valid attribute values. In contrast to Enumeration, the list of values is open and, thus, not given inline in the CityGML UML Model. The allowed values can be provided within an external code list.
- «Union» is a list of attributes. The semantics are that only one of the attributes can be present at any time.
- «Property» denotes attributes and association roles. This stereotype does not add further semantics to the conceptual model, but is required to be able to add tagged values to the attributes and association roles that are relevant for the encoding.
- «Version» denotes that the value of an association role that ends at a feature type is a specific version of the feature, not the feature in general.

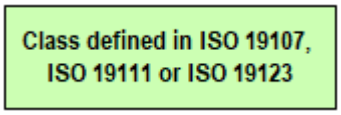
```
In order to enhance the readability of the CityGML UML diagrams, classes are
depicted in different colors. The following coloring scheme is applied:
```



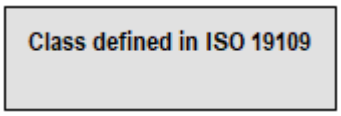

```
Classes painted in yellow belong to the Requirements Class which is subject of
discussion in that clause of the standard in which the UML diagram is given. For
example, in the context of <<ug-model-core-section>>, which introduces the _CityGML
Core_ module, the yellow color is used to denote classes that are defined in the
_CityGML Core_ Requirements Class. Likewise, the yellow classes shown in the UML
diagram in <<ug-model-building-section>> are associated with the _Building_
Requirements Class that is subject of discussion in that chapter.
```



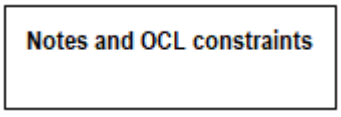
Classes painted in blue belong to a Requirements Class different to that associated with the yellow color. In order to explicitly denote to which Requirements Class these classes belong, their class names are preceded by the UML package name of that Requirements Class. For example, in the context of the _Building_ Requirements Class, classes from the _CityGML Core_ and the _Construction_ Requirements Classes are painted in blue and their class names are preceded by _Core_ and _Construction_, respectively.



Classes painted in green are defined in the ISO standards 19107, 19111, or 19123. Their class names are preceded by the UML package name, in which the classes are defined.



Classes painted in grey are defined in the ISO standard 19109. In the context of this standard, this only applies to the class _AnyFeature_. _AnyFeature_ is an instance of the metaclass _FeatureType_ and acts as super class of all classes in the CityGML UML model with the stereotype «FeatureType». A metaclass is a class whose instances are classes.



The color white is used for notes and <<iso19507,Object Constraint Language>> (OCL) constraints that are provided in the UML diagrams.

The example UML diagram in <<figure-2>> demonstrates the UML notation and coloring scheme used throughout this standard. In this example, the yellow classes are associated with the _CityGML Building_ module, the blue classes are from the _CityGML Core_ and _Construction_ modules, and the green class depicts a geometry element defined by ISO 19107.

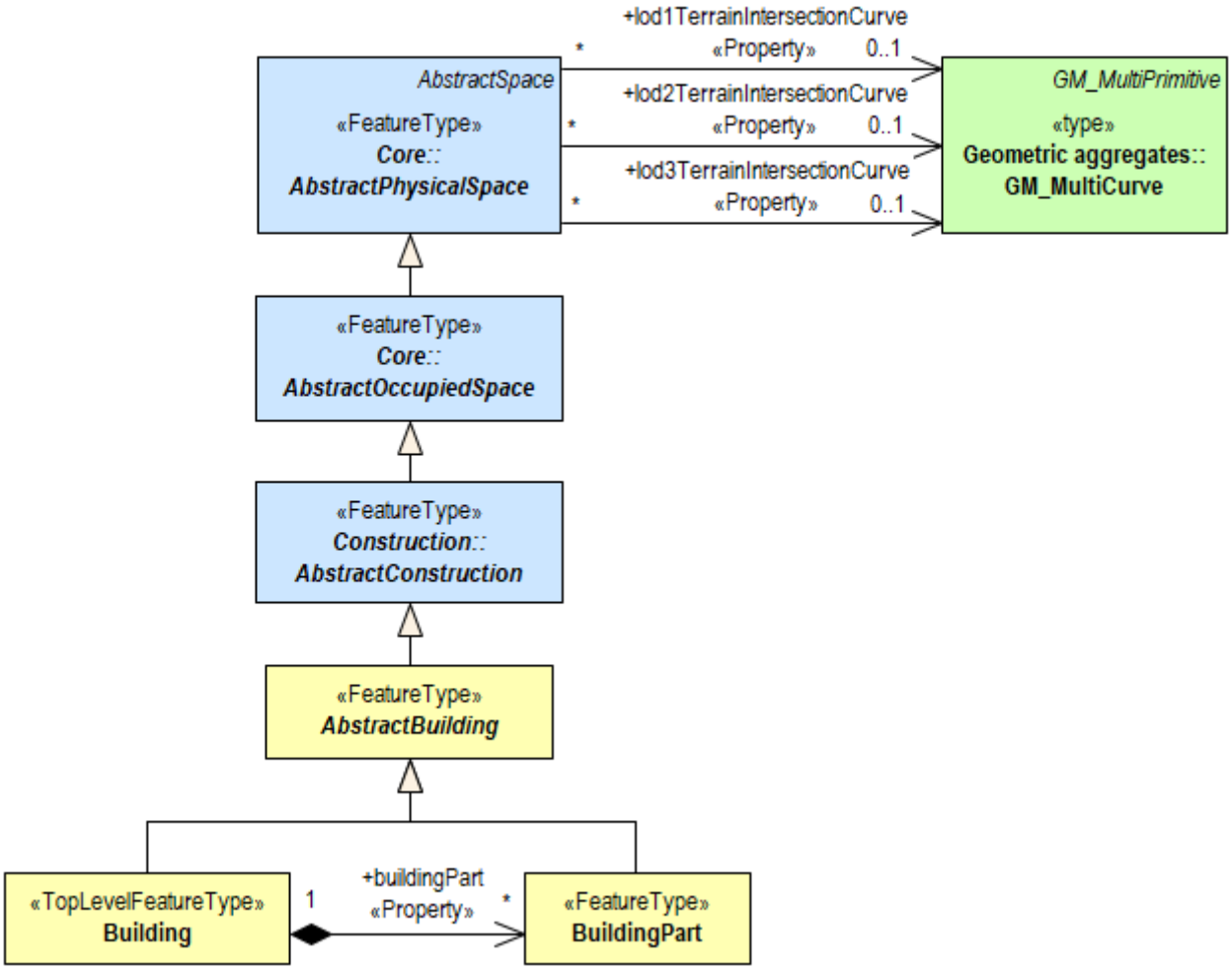


Figure 2. Example UML diagram demonstrating the UML notation and coloring scheme used throughout the CityGML Standard.

Chapter 7. CityGML Foundations

(Imported from the CityGML 3.0 Standard)

This standard defines an open CityGML Conceptual Model (CM) for the storage and exchange of virtual 3D city and landscape models. These models include the most relevant entities of the urban space like buildings, roads, railways, tunnels, bridges, city furniture, water bodies, vegetation, and the terrain. The conceptual schema specifies how and into which parts and pieces physical objects of the real world should be decomposed and classified. All objects can be represented with respect to their semantics, 3D geometry, 3D topology, appearances, and their changes over time. Different spatial representations can be provided for each object (outdoor and indoor) in four predefined Levels of Detail (LOD 0-3). The CityGML 3.0 Conceptual Model (<<ug-citygml-model-section>>) is formally specified using UML class diagrams, complemented by a data dictionary (<<data-dictionary-section>>) providing the definitions and explanations of the object classes and attributes. This Conceptual Model is the basis for multiple encoding standards, which map the concepts (or subsets thereof) onto exchange formats or database structures for data exchange and storage.

While the CityGML Conceptual Model can be used for 3D visualization purposes, its special merits lie in applications that go beyond visualization such as decision support, urban and landscape planning, urban facility management, Smart Cities, navigation (both indoor and outdoor), Building Information Modeling (especially for as-built documentation), integration of city and BIM models, assisted and autonomous driving, and simulations in general (cf. <<Kolbe2009>>). A comprehensive overview on the many different applications of virtual 3D city models is given in [<<Biljecki2015>>]. Many of the applications already use and some even require using CityGML.

In the CityGML CM, all 3D city objects can easily be enriched with thematic data. For example, street objects can be enriched with information about traffic density, speed limit, number of lanes etc., or buildings can be enriched by information on the heating and electrical energy demand, numbers of households and inhabitants, the appraised building value etc. Even building parts such as individual roof or wall surfaces can be enriched with information e.g., about solar irradiation and thermal insulation parameters. For many application domains specific extensions of the CityGML CM have already been created (cf. <<Biljecki2018>>).

7.1. Modularization

(Imported from the CityGML 3.0 Standard)

The CityGML Conceptual Model provides models for the most important types of objects within virtual 3D city and landscape models. These feature types have been identified to be either required or important in many different application areas. However, implementations are not required to support the complete CityGML model in order to be conformant to the standard. Implementations may employ a subset of constructs according to their specific information needs. For this purpose, modularization is applied to the CityGML CM.

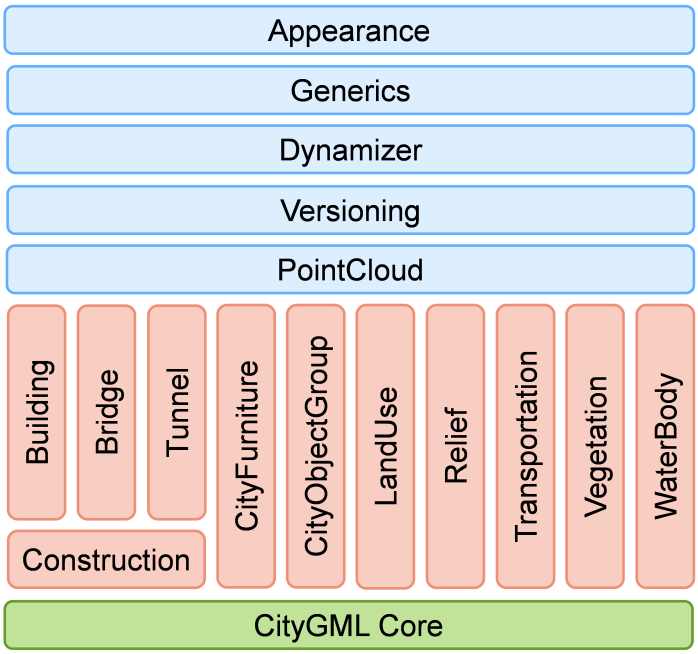


Figure 3. CityGML 3.0 module overview. The vertical boxes show the different thematic modules. Horizontal modules specify concepts that are applicable to all thematic modules.

The CityGML conceptual model is thematically decomposed into a _Core module_ and different kinds of _extension modules_ as shown in <<figure-moduleoverview>>. The Core module (shown in green) comprises the basic concepts and components of the CityGML CM and, thus, must be implemented by any conformant system. Each red colored module covers a specific thematic field of virtual 3D city models.

The CityGML CM introduces the following eleven thematic extension modules: _Building_, _Bridge_, _Tunnel_, _Construction_, _CityFurniture_, _CityObjectGroup_, _LandUse_, _Relief_, _Transportation_, _Vegetation_, and _WaterBody_. All three modules _Building_, _Bridge_, and _Tunnel_ model civil structures and share common concepts that are grouped within the _Construction_ module. The five blue colored extension modules add specific modeling aspects that can be used in conjunction with all thematic modules:

- The *Appearance* module contains the concepts to represent appearances (like textures and colors) of city objects.
- The *PointCloud* module provides concepts to represent the geometry of city objects by 3D point clouds.

- The *Generics* module defines the concepts for generic objects, attributes, and relationships.
- *Versioning* adds concepts for the representation of concurrent versions, real world object histories and feature histories.
- The *Dynamizer* module contains the concepts to represent city object properties by time series data and to link them with sensors, sensor data services or external files.

Each CityGML encoding can specify support for a subset of the CityGML modules only. If a module is supported by an encoding, then all concepts should be mapped. However, the encoding specification can define so-called _null mappings_ to restrict the use of specific elements of the conceptual model in an encoding. Null mappings can be expressed in an encoding specification for individual feature types, properties, and associations defined within a CityGML module. This means that the corresponding element will not be included in the respective encoding.

Note that also CityGML applications do not have to support all modules. Applications can also decide to only support a specific subset of CityGML modules. For example, when an application only has to work with building data, only the modules _Core_, _Construction_, and _Building_ would have to be supported.

7.2. General modeling Principles

(Imported from the CityGML 3.0 Standard)

7.2.1. Semantic modeling of Real-World Objects

Real-world objects are represented by geographic features according to the definition in ISO 19109. Geographic features of the same type (e.g., buildings, roads) are modeled by corresponding feature types that are represented as classes in the Conceptual Model (CM). The objects within a 3D city model are instances of the different feature types.

In order to distinguish and reference individual objects, each object has unique identifiers. In the CityGML 3.0 CM, each geographic feature has the mandatory _featureID_ and an optional _identifier_ property. The _featureID_ is used to distinguish all objects and possible multiple versions of the same real-world object. The _identifier_ is identical for all versions of the same real-world object and can be used to reference specific objects independent from their actual object version. The _featureID_ is unique within the same CityGML dataset, but it is generally recommended to use globally unique identifiers like UUID values or identifiers maintained by an organization such as a mapping agency. Providing globally unique and stable identifiers for the _identifier_ attribute is recommended. This means these identifiers should remain stable over the lifetime of the real-world object.

CityGML feature types typically have a number of spatial and non-spatial properties (also called attributes) as well as relationships with other feature or object types. Note that a single CityGML object can have different spatial representations at the same time. For example, different geometry objects representing the feature's geometry in different levels of detail or as different spatial abstractions.

Many attributes have simple, scalar values like a number or a character string. However, some attributes are complex. They do not just have a single property value. In CityGML the following types of complex attributes occur.

- *Qualified attribute values*: For example, a measure consists of the value and a reference to the unit of measure, or e.g., for relative and absolute height levels the reference level has to also be named.
- *Code list values*: A code list is a form of enumeration where the valid values are defined in a separate register. The code list values consist of a link or identifier for the register as well as the value from that register which is being used.
- Attributes consisting of a *tuple of different fields and values*: For example, addresses, space occupancy, and others.
- Attribute value consisting of a *list of numbers*: For example, representing coordinate lists or matrices.

  In order to support feature history, CityGML 3.0 introduces bitemporal timestamps for all objects. In CityGML 2.0, the attributes _creationDate_ and _terminationDate_ are supported. These refer to the time period in which a specific version of an object is an integral part of the 3D city model. In 3.0, all features can now additionally have the attributes _validFrom_ and _validTo_. These represent the lifespan a specific version of an object has in the real-world. Using these two time intervals a CityGML dataset could be queried both for how did the _city_ look alike at a specific point in time as well as how did the _city model_ look at that time.

  The combination of the two types of feature identifiers and bitemporal timestamps enables encoding not only the current version of a 3D city model, but also the model's entire history can be represented in CityGML and possibly exchanged within a single file.

7.2.2. Class Hierarchy and Inheritance of Properties and Relations

In CityGML, the specific feature types like __Building__, __Tunnel__, or _WaterBody_ are defined as subclasses of more general higher-level classes. Hence, feature types build a hierarchy along specialization / generalization relationships where more specialized feature types inherit the properties and relationships of all their superclasses along the entire generalization path to the topmost feature type __AnyFeature__.

Note: A superclass is the class from which subclasses can be created.

7.2.3. Relationships between CityGML objects

In CityGML, objects can be related to each other and different types of relations are distinguished. First of all, complex objects like buildings or transportation objects typically consist of parts. These parts are individual features of their own, and can even be further decomposed. Therefore, CityGML objects can form aggregation hierarchies. Some feature types are marked in the conceptual model with the stereotype _«TopLevelFeatureType»_. These constitute the main objects of a city model and are typically the root of an aggregation hierarchy. Only top-level features are allowed as direct members of a city model. The information about which feature types belong to the top level is required for software packages that want to filter imports, exports, and visualizations according to the general type of a city object (e.g., only show buildings, solitary vegetation objects, and roads). CityGML Application Domain Extensions should also make use of this concept, such that software tools can learn from inspecting their conceptual schema what are the main, i.e., the top-level, feature types of the extension.

Some relations in CityGML are qualified by additional parameters, typically to further specify the type of relationship. For example, a relationship can be qualified with a URI pointing to a definition of the respective relation type in an Ontology. Qualified relationships are used in CityGML, among others, for:

- General relationships between features – association *relatedTo* between city objects,
- User-defined aggregations using *CityObjectGroup*. This relation allows also for recursive aggregations,
- External references – linking of city objects with corresponding entities from external resources like objects in a cadastre or within a BIM dataset.

  The CityGML CM contains many relationships that are specifically defined between certain feature types. For example, there is the _boundary_ relationship from 3D volumetric objects to its thematically differentiated 3D boundary surfaces. Another example is the _generalizesTo_ relation between feature instances that represent objects on different generalization levels.

In the CityGML 3.0 CM there are new associations to express topologic, geometric, and semantic relations between all kinds of city objects. For example, information that two rooms are adjacent or that one interior building installation (like a curtain rail) is overlapping with the spaces of two connected rooms can be expressed. The CM also enables documenting that two wall surfaces are parallel and two others are orthogonal. Also distances between objects can be represented explicitly using geometric relations. In addition to spatial relations logical relations can be expressed.

7.2.4. Definition of the Semantics for all Classes, Properties, and Relations

The meanings of all elements defined in the CityGML conceptual model are normatively specified in the data dictionary in <<data-dictionary-section>>.

7.3. Representation of Spatial Properties

(Imported from the CityGML 3.0 Standard)

7.3.1. Geometry and Topology

Spatial properties of all CityGML feature types are represented using the geometry classes defined in ISO 19107. Spatial representations can have 0-, 1-, 2-, or 3-dimensional extents depending on the respective feature type and Levels of Detail (LOD). The LOD concept is discussed in <<ug-levels-of-detail-section>> and <<ug-geometry-lod-section>>. With only a few exceptions, all geometries must use 3D coordinate values. Besides primitive geometries like single points, curves, surfaces, and solids, CityGML makes use of different kinds of aggregations of geometries like spatial aggregates (_MultiPoint_, _MultiCurve_, _MultiSurface_, _MultiSolid_) and composites (_CompositeCurve_, _CompositeSurface_, _CompositeSolid_). Volumetric shapes are represented in ISO 19107 according to the so-called _Boundary Representation_ (B-Rep). For further explanation see <<Foley2002>>.

The CityGML Conceptual Model does not put any restriction on the usage of specific geometry types as defined in ISO 19107. For example, 3D surfaces could be represented in a dataset using 3D polygons or 3D meshes such as triangulated irregular networks (TINS) or by non-uniform rational B-spline surfaces (NURBS). However, an encoding may restrict the usage of geometry types. For example, curved lines like B-splines or clothoids, or curved surfaces like NURBS could be disallowed by explicitly defining _null encodings_ for these concepts in the encoding specification (c.f. <<ug-modularization-section>> above).

Note that the conceptual schema of ISO 19107 allows composite geometries to be defined by a recursive aggregation for every primitive type of the corresponding dimension. This aggregation schema allows the definition of nested aggregations (hierarchy of components). For example, a building geometry (_CompositeSolid_) can be composed of the house geometry (_CompositeSolid_) and the garage geometry (_Solid_), while the house's geometry is further decomposed into the roof geometry (_Solid_) and the geometry of the house body (_Solid_). This is illustrated in <<figure-recursiveaggregation>>.

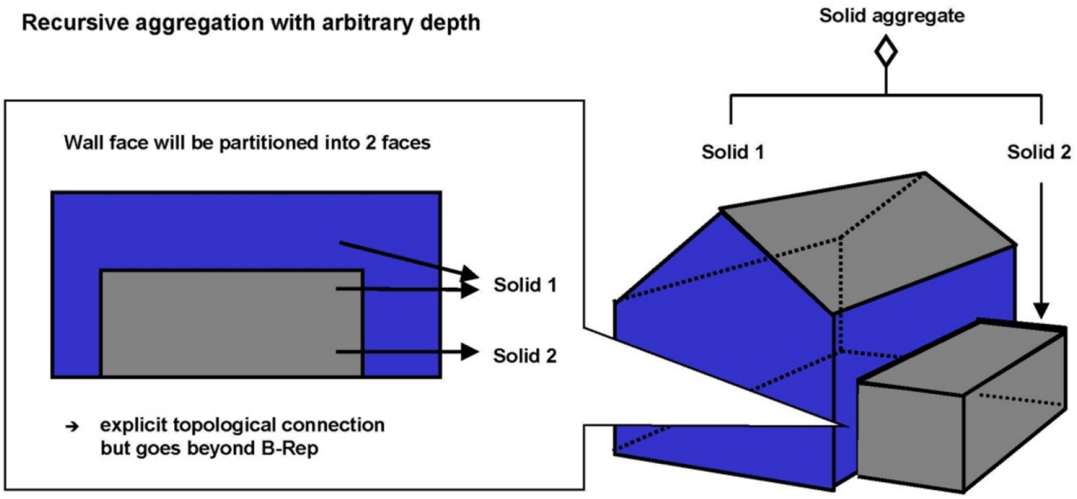


Figure 4. Recursive aggregation of objects and geometries in CityGML (graphic: IGG Uni Bonn).

While the CityGML Conceptual Model does not employ the topology classes from ISO 19107, topological relations between geometries can be established by sharing geometries (typically parts of the boundary) between different geometric objects. One part of real-world space can be represented only once by a geometry object and is referenced by all features or more complex geometries which are defined or bounded by this geometry object. Thus redundancy can be avoided and explicit topological relations between parts are maintained.

Basically, there are three cases for sharing geometries.

- First, two different semantic objects may be spatially represented by the same geometry object. For example, if a foot path is both a transportation feature and a vegetation feature, the surface geometry defining the path is referenced by both the transportation object and by the vegetation object.
- Second, a geometry object may be shared between a feature and another geometry. For example, a geometry defining a wall of a building may be referenced twice: By the solid geometry defining the geometry of the building, and by the wall feature.
- Third, two geometries may reference the same geometry, which is in the boundary of both. For example, a building and an adjacent garage may be represented by two solids. The surface describing the area where both solids touch may be represented only once and it is referenced by both solids. As it can be seen from Figure 4, this requires partitioning of the respective

surfaces.

In general, B-Rep only considers visible surfaces. However, to make topological adjacency explicit and to allow the possibility of deletion of one part of a composed object without leaving holes in the remaining aggregate, touching elements are included. Whereas touching is allowed, permeation of objects is not in order to avoid the multiple representation of the same space.

Another example of sharing geometry objects that are members of the boundaries in different higher-dimensional geometry objects is the sharing of point geometries or curve geometries, which make up the outer and inner boundaries of a polygon. This means that each point is only represented once, and different polygons could reference this point geometry. The same applies to the representation of curves for transportation objects like roads, whose end points could be shared such as between different road segments to topologically connect them.

Note that the use of topology in CityGML datasets by sharing geometries is optional. Furthermore, an encoding of the CityGML conceptual model might restrict the usage of shared geometries. For example, it might only be allowed to share identical (support) points from different 3D polygons or only entire polygons can be shared between touching solids (like shown in <<figure-recursiveaggregation>>).

7.3.2. Prototypic Objects / Scene Graph Concepts

In CityGML, objects of equal shape like trees and other vegetation objects, traffic lights and traffic signs can be represented as prototypes which are instantiated multiple times at different locations (see <<figure-prototypicshapes>>). The geometry of prototypes is defined in local coordinate systems. Every instance is represented by a reference to the prototype, a base point in the world coordinate reference system (CRS) and a transformation matrix that facilitates scaling, rotation, and translation of the prototype. The principle is adopted from the concept of scene graphs used in computer graphics standards. Since the ISO 19107 geometry model does not provide support for scene graph concepts, the CityGML class ImplicitGeometry has been introduced (for further description see <<ug-geometry-lod-section>>). The prototype geometry can be represented using ISO 19107 geometry objects or by referencing an external file containing the geometry in another data format.

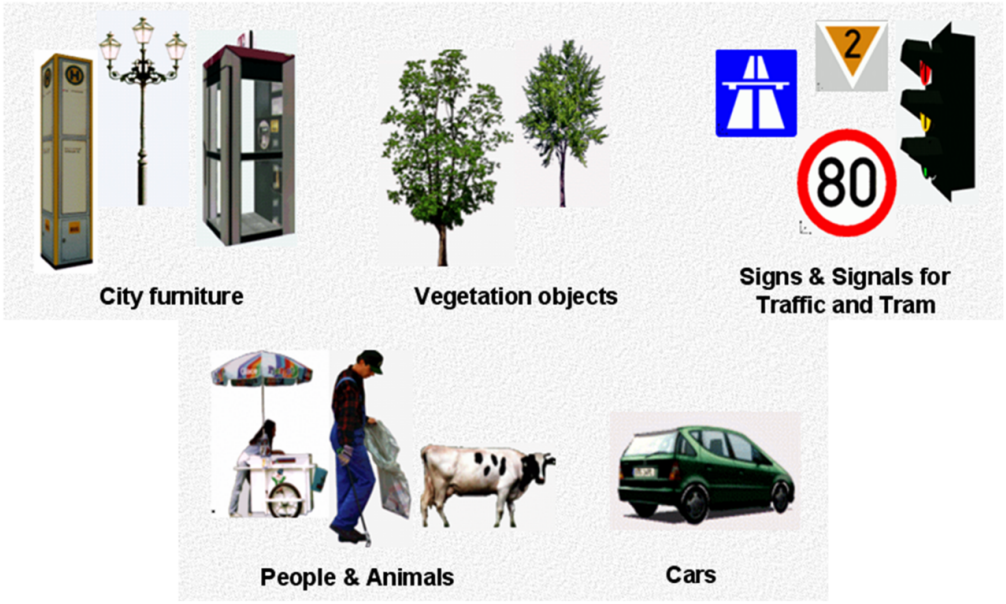


Figure 5. Examples of prototypic shapes (source: Rheinmetall Defence Electronics).

7.3.3. Point Cloud Representation

In addition to the spatial representations defined in the _Core_ module, the geometry of physical spaces and of thematic surfaces can now also be provided by 3D point clouds using MultiPoint geometry. This allows, for example, spatially representing the building hull, a room within a building or a single wall surface just by a point cloud. All thematic feature types including transportation objects, vegetation, city furniture, etc. can also be spatially represented by point clouds. In this way, the ClearanceSpace of a road or railway could, for instance, be modeled directly from the result of a mobile laser scanning campaign. Point clouds can either be included in a CityGML dataset or just reference an external file of some common types such as LAS or LAZ.

7.3.4. Coordinate Reference Systems (CRS)

CityGML is about 3D city and landscape models. This means that nearly all geometries use 3D coordinates, where each single point and also the points defining the boundaries of surfaces and solids have three coordinate values (x,y,z) each. Coordinates always have to be given with respect to a coordinate reference system (CRS) that relates them unambiguously with a specific position on the Earth. In contrast to CAD or BIM, each 3D point is absolutely georeferenced, which makes CityGML especially suitable to represent geographically large extended structures like airports, railways, bridges, dams, where the Earth curvature has a significant effect on the object□s geometry (for further explanations see <<Kaden2017>>).

In most CRS, the (x,y) coordinates refer to the horizontal position of a point on the Earth□s surface. The z coordinate typically refers to the vertical height over (or under) the reference surface. Note that depending on the chosen CRS, x and y may be given as angular values like latitude and longitude or as distance values in meters or feet. According to ISO 19111, numerous 3D CRS can be used. This includes global as well as national reference systems using geocentric, geodetic, or projected coordinate systems.

7.4. CityGML Core Model: Space Concept, Levels of Detail, Special Spatial Types

(Imported from the CityGML 3.0 Standard)

7.4.1. Spaces and Space Boundaries

In the CityGML 3.0 Conceptual Model, a clear semantic distinction of spatial features is introduced by mapping all city objects onto the semantic concepts of spaces and space boundaries. A _Space_ is an entity of volumetric extent in the real world. Buildings, water bodies, trees, rooms, and traffic spaces are examples for such entities with volumetric extent. A _Space Boundary_ is an entity with areal extent in the real world. Space Boundaries delimit and connect Spaces. Examples are the wall surfaces and roof surfaces that bound a building, the water surface as boundary between the water body and air, the road surface as boundary between the ground and the traffic space, or the digital terrain model representing the space boundary between the over- and underground space.

To obtain a more precise definition of spaces, they are further subdivided into physical spaces and logical spaces. Physical spaces are spaces that are fully or partially bounded by physical objects. Buildings and rooms, for instance, are physical spaces as they are bounded by walls and slabs. Traffic spaces of roads are physical spaces as they are bounded by road surfaces against the ground. Logical spaces, in contrast, are spaces that are not necessarily bounded by physical objects, but are defined according to thematic considerations. Depending on the application, logical spaces can also be bounded by non-physical, i.e., virtual boundaries, and they can represent aggregations of physical spaces. A building unit, for instance, is a logical space as it aggregates specific rooms to flats, the rooms being the physical spaces that are bounded by wall surfaces, whereas the aggregation as a whole is being delimited by a virtual boundary. Other examples are city districts which are bounded by virtual vertically extruded administrative boundaries, public spaces vs. Security zones in airports, or city zones with specific regulations stemming from urban planning. The definition of physical and logical spaces and of corresponding physical and virtual boundaries is in line with the discussion in [<<Smith2000>>] on the difference between bona fide and fiat boundaries to bound objects. Bona fide boundaries are physical boundaries; they correspond to the physical boundaries of physical spaces in the CityGML 3.0 CM. In contrast, fiat boundaries are man-made boundaries. They are equivalent to the virtual boundaries of logical spaces.

Physical spaces, in turn, are further classified into occupied spaces and unoccupied spaces. Occupied spaces represent physical volumetric objects that occupy space in the urban environment. Examples for occupied spaces are buildings, bridges, trees, city furniture, and water bodies. Occupying space means that some space is blocked by these volumetric objects. For instance, the space blocked by the building in <<figure-occupiedandunoccupiedspaces>> cannot be used any more for driving through this space or placing a tree on that space. In contrast, unoccupied spaces represent physical volumetric entities that do not occupy space in the urban environment, i.e., no space is blocked by these volumetric objects. Examples for unoccupied spaces are building rooms and traffic spaces. There is a risk of misunderstanding the term OccupiedSpace. However, we decided to use the term anyway, as it is established in the field of robotics for over three decades [<<Elfes1989>>]. The navigation of mobile robots makes use of a so-called occupancy map that marks areas that are occupied by matter and, thus, are not navigable for robots.

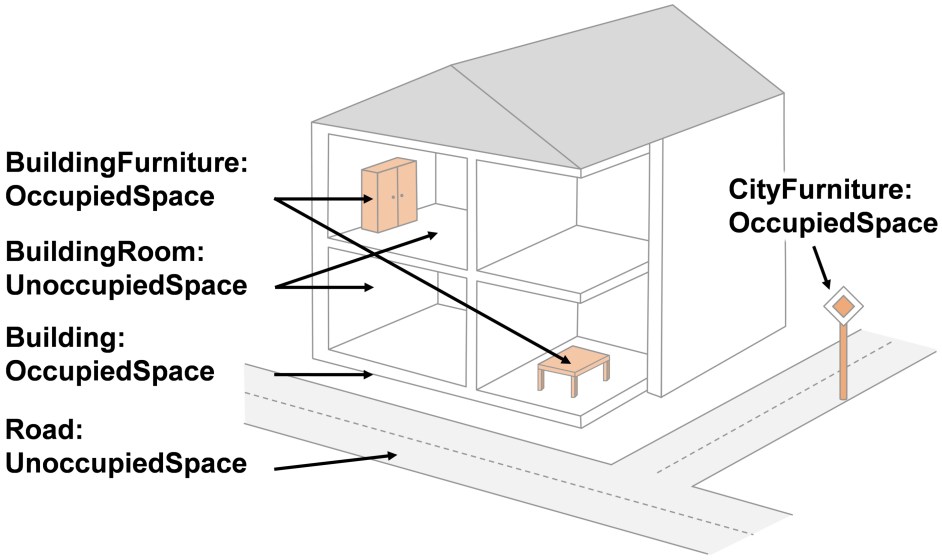


Figure 6. Occupied and unoccupied spaces

```
The new space concept offers several advantages.
```

- In the CityGML 3.0 Conceptual Model, all geometric representations are only defined in the *Core* module. This makes (a) models of the thematic modules simpler as they no longer need to be associated directly with the geometry classes, and (b) implementation easier as all spatial concepts have only to be implemented once in the *Core* module. All thematic modules like *Building*, *Relief*, *WaterBody*, etc. inherit their geometric representations from the *Core* module.
- The space concept supports the expression of explicit topological, geometrical, and thematic relations between spaces and spaces, spaces and space boundaries, and space boundaries and space boundaries. Thus, implementing the checking of geometric-topological consistency will become easier. That is because most checks can be expressed and performed on the CityGML *Core* module and then automatically applied to all thematic modules
- For the analysis of navigable spaces (e.g., to generate IndoorGML data from CityGML) algorithms can be defined on the level of the *Core* module. These algorithms will then work with all CityGML feature classes and also ADEs as they are derived from the *Core*. The same is true for other applications of 3D city models listed in [Biljecki et al. 2015] such as visibility analyses including shadow casting or solar irradiation analyses.
- Practitioners and developers do not see much of the space concept. That is because the space and space boundary classes are just abstract classes. Only elements representing objects from concrete subclasses such as Building, BuildingRoom, or TrafficSpace will appear in CityGML data sets.

7.4.2. Modeling City Objects by the Composition of Spaces

```
Semantic objects in CityGML are often composed of parts, i.e., they form multi-level
aggregation hierarchies. This also holds for semantic objects representing occupied
and unoccupied spaces. In general, two types of compositions can be distinguished.
```

1. **Spatial partitioning**: Semantic objects of either the space type OccupiedSpace or

UnoccupiedSpace are subdivided into different parts that are of the same space type as the parent object. Examples are Buildings that can be subdivided into BuildingParts, or Buildings that are partitioned into ConstructiveElements. Buildings as well as BuildingParts and constructiveElements represent OccupiedSpaces. Similarly, Roads can be subdivided into TrafficSpaces and AuxiliaryTrafficSpaces, all objects being UnoccupiedSpaces.

2. **Nesting of alternating space types**: Semantic objects of one space type contain objects that are of the opposite space type as the parent object. Examples are Buildings (OccupiedSpace) that contain BuildingRooms (UnoccupiedSpace), BuildingRooms (UnoccupiedSpace) that contain Furniture (OccupiedSpace), and Roads (UnoccupiedSpace) that contain CityFurniture (OccupiedSpace). The categorization of a semantic object into occupied or unoccupied takes place at the level of the object in relation to the parent object. A building is part of a city model. Thus, in the first place the building occupies urban space within a city. As long as the interior of the building is not modeled in detail, the space covered by the building needs to be considered as occupied and only viewable from the outside. To make the building accessible inside, voids need to be added to the building in the form of building rooms. The rooms add free space to the building interior. In other words, the OccupiedSpace now contains some UnoccupiedSpace. The free space inside the building can, in turn, contain objects that occupy space again, such as furniture or installations. In contrast, roads also occupy urban space in the city. However, this space is initially unoccupied as it is accessible by cars, pedestrian, or cyclists. Adding traffic signs or other city furniture objects to the free space results in specific sections of the road becoming occupied by these objects. Thus, one can also say that occupied spaces are mostly filled with matter; whereas, unoccupied spaces are mostly free of matter and, thus, realize free spaces.

7.4.3. Rules for Surface Orientations of OccupiedSpaces and UnoccupiedSpaces

```
The classification of feature types into OccupiedSpace and UnoccupiedSpace also
defines the semantics of the geometries attached to the respective features. For
OccupiedSpaces, the attached geometries describe volumes that are (mostly) physically
occupied. For UnoccupiedSpaces, the attached geometries describe (or bound) volumes
that are (mostly) physically unoccupied. This also has an impact on the required
orientation of the surface normal (at point _P_ this is a vector perpendicular to the
tangent plane of the surface at __P__) for attached thematic surfaces. For
OccupiedSpaces, the normal vectors of thematic surfaces must point in the same
direction as the surfaces of the outer shell of the volume. For UnoccupiedSpaces, the
normal vectors of thematic surfaces must point in the opposite direction as the
surfaces of the outer shell of the volume. This means that from the perspective of an
observer of a city scene, the surface normals must always be directed towards the
observer. In the case of OccupiedSpaces (e.g., Buildings, Furniture), the observer
must be located outside the OccupiedSpace for the surface normals being directed
towards the observer; whereas in the case of UnoccupiedSpaces (e.g., Rooms, Roads),
the observer is typically inside the UnoccupiedSpace.
```

7.4.4. Levels of Detail (LOD)

The CityGML Conceptual Model differentiates four consecutive Levels of Detail (LOD 0-3), where objects become more detailed with increasing LOD with respect to their geometry. CityGML datasets can - but do not have to - contain multiple geometries for each object in different LODs simultaneously. The LOD concept facilitates multi-scale modeling; i.e., having varying degrees of spatial abstractions that are appropriate for different applications or visualizations.

The classification of real-world objects into spaces and space boundaries is solely based on the semantics of these objects and not on their used geometry type, as the CityGML 3.0 CM allows various geometrical representations for objects. A building, for instance, can be spatially represented by a 3D solid (e.g., in LOD1), but at the same time, the real-world geometry can also be abstracted by a single point, footprint or roof print (LOD0), or by a 3D mesh (LOD3). The outer shell of the building may also be semantically decomposed into wall, roof, and ground surfaces. <<figure-buildinglods>> shows different representations of the same real-world building object in different geometric LODs (and appearances).

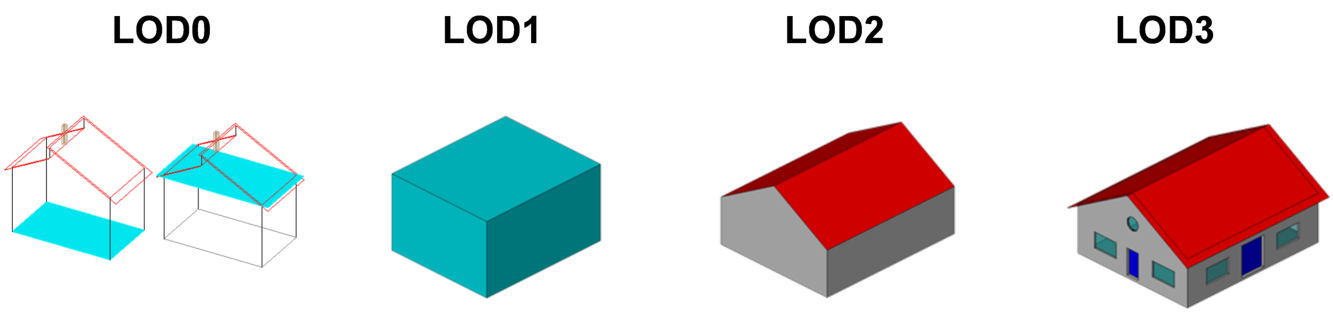


Figure 7. Representation of the same real-world building in the Levels of Detail 0-3.

The biggest changes between CityGML 3.0 and earlier versions are as follows.

1. LOD4 was dropped, because now all feature types can have outdoor and indoor elements in LODs 0-3 (for those city objects where it makes sense like buildings, tunnels, or bridges). This means that the outside shell such as of a building, could be spatially represented in LOD2 and the indoor elements like rooms, doors, hallways, stairs etc. in LOD1. CityGML can now be used to represent building floor plans, which are LOD0 representations of building interiors (cf. Konde et al. 2018). It is even possible to model the outside shell of a building in LOD1, while representing the interior structure in LOD2 or 3. Figure 8 shows different indoor/outdoor representations of a building. Details on the changes to the CityGML LOD concept are provided in [Löwner et al. 2016].
2. Levels of Detail are no longer associated with the degree of semantic decomposition of city objects and refer to the spatial representations only. This means that, for example, buildings can have thematic surfaces (like WallSurface, GroundSurface) also in LODs 0 and 1 and windows and doors can be represented in all LODs 0-3. In CityGML 2.0 or earlier thematic surfaces were only allowed starting from LOD2, openings like doors and windows starting from LOD3, and interior rooms and furniture only in LOD4.
3. In the CityGML 3.0 Conceptual Model the geometry representations were moved from the

thematic modules to the *Core* module and are now associated with the semantic concepts of *Spaces* and *Space Boundaries*. This led to a significant simplification of the models of the thematic modules. Since all feature types in the thematic modules are defined as subclasses of the space and space boundary classes, they automatically inherit the geometry classes and, thus, no longer require direct associations with them. This also led to a harmonized LOD representation over all CityGML feature types.

4. If new feature types are defined in Application Domain Extensions (ADEs) based on the abstract Space and Space Boundary classes from the Core module, they automatically inherit the spatial representations and the LOD concept.

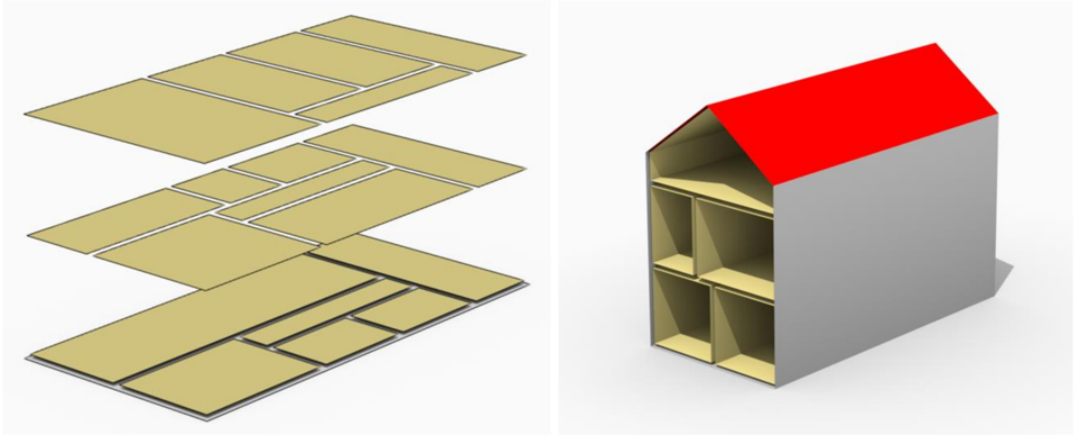


Figure 8. Floor plan representation (LOD0) of a building (left), combined LOD2 indoor and outdoor representation (right). Image adopted from Löwner et al. 2016.

```
_Spaces_ and all its subclasses like _Building_, _Room_, and _TrafficSpace_ can now be
spatially represented by single points in LOD0, multi-surfaces in LOD0/2/3, solids in
LOD1/2/3, and multi-curves in LOD2/3. _Space Boundaries_ and all its subclasses such
as _WallSurface_, _LandUse_, or _Relief_ can now be represented by multi-surfaces in
LOD0/2/3 and as multi-curves in LOD2/3. See <<geometry-lod-section>> for further
details on the different Levels of Detail.
```

7.4.5. Closure Surfaces

```
Objects, which are not spatially represented by a volumetric geometry, must be
virtually closed in order to compute their volume (e.g., pedestrian underpasses or
airplane hangars). They can be sealed using a specific type of space boundary called a
ClosureSurface. These are virtual surfaces. They are used when a closed surface is
needed to compute volumes or perform similar 3D operations. Since they do not actually
exist, they are neglected when they are not needed or not appropriate. For example,
ClosureSurfaces would not be used in visualizations.
```

The concept of ClosureSurface can also be employed to model the entrances of subsurface objects. Those objects like tunnels or pedestrian underpasses have to be modeled as closed solids in order to compute their volume. An example would be for use in flood simulations. The entrances to subsurface objects also have to be sealed to avoid holes in the digital terrain model (see <<figure-closuresurfaces>>). However, in close-range visualizations the entrance should be treated as open. Thus, closure surfaces are an adequate way to model those entrances.

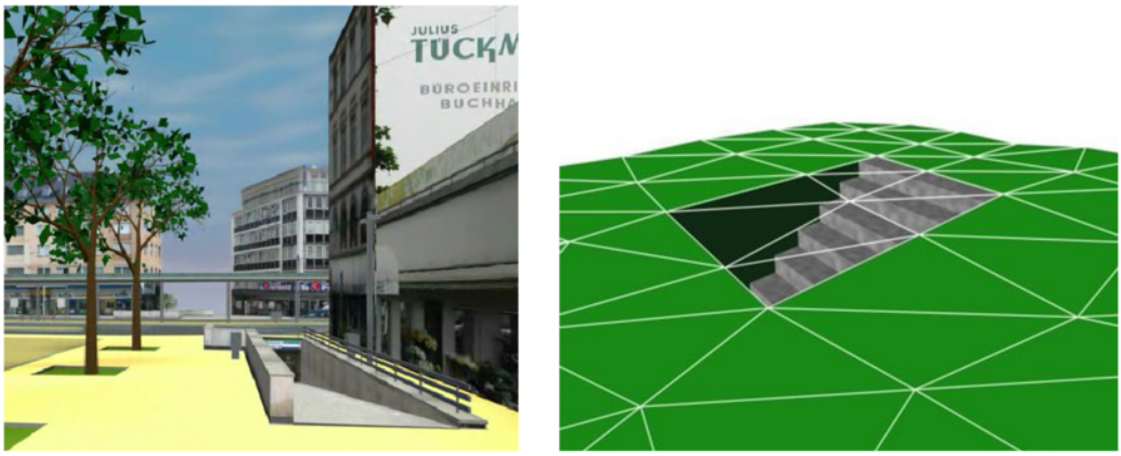


Figure 9. Closure surfaces to seal open structures. Passages are subsurface objects (left). The entrance is sealed by a virtual ClosureSurface feature, which is both part of the DTM and the subsurface object (right) (graphic: IGG Uni Bonn).

7.4.6. Terrain Intersection Curves

An important issue in city modeling is the integration of 3D objects and the terrain. Problems arise if 3D objects float over or sink into the terrain. This is particularly the case when terrains and 3D objects in different LODs are combined, when the terrain and 3D models are updated independently from each other, or when they come from different data providers [<<Kolbe2003>>]. To overcome this problem, the TerrainIntersectionCurve (TIC) of a 3D object is introduced. These curves denote the exact position where the terrain touches the 3D object (see <<figure-terrainintersectioncurves>>). TICs can be applied to all CityGML feature types that are derived from AbstractPhysicalSpace such as buildings, bridges, tunnels, but also city furniture, vegetation, and generic city objects.

If, for example, a building has a courtyard, the TIC consists of two closed rings: One ring representing the courtyard boundary, and one which describes the building's outer boundary. This information can be used to integrate the building and a terrain by "pulling up" or "pulling down" the surrounding terrain to fit the TerrainIntersectionCurve. The digital terrain model (DTM) may be locally warped to fit the TIC. By this means, the TIC also ensures the correct positioning of textures or the matching of object textures with the DTM. Since the intersection with the terrain may differ depending on the LOD, a 3D object may have different TerrainIntersectionCurves for all LODs.

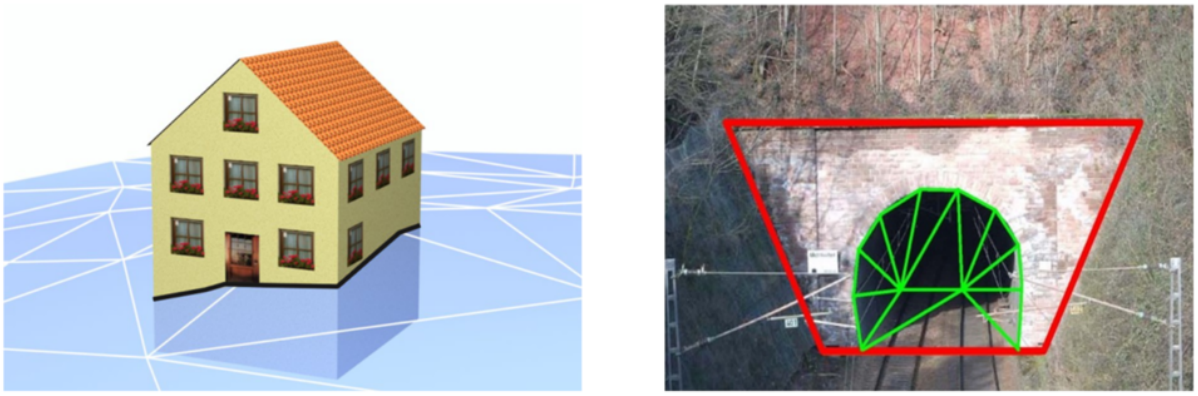


Figure 10. TerrainIntersectionCurve for a building (left, black) and a tunnel object (right, red). The tunnel's hollow space is sealed by a triangulated ClosureSurface (graphic: IGG Uni Bonn).

7.4.7. Coherent Semantical-Geometrical modeling

An important design principle for CityGML is the coherent modeling of semantic objects and their spatial representations. At the semantic level, real-world entities are represented by features, such as buildings, walls, windows, or rooms. The description also includes attributes, relations and aggregation hierarchies (part-whole-relations) between features. Thus the part-of-relationship between features can be derived at the semantic level only, without considering geometry. However, at the spatial level, geometry objects are assigned to features representing their spatial location, shape, and extent. So the model consists of two hierarchies: The semantic and the geometrical in which the corresponding objects are linked by relationships (cf. <<Stadler2007>>). The advantage of this approach is that it can be navigated in both hierarchies and between both hierarchies arbitrarily, for answering thematic and/or geometrical queries or performing analyses.

If both hierarchies exist for a specific object, they must be coherent (i.e., it must be ensured that they match and fit together). For example, if a building is semantically decomposed into wall surfaces, roof surfaces and so forth, the polygons representing these thematic surfaces (in a specific LOD) must be part of the solid geometry representing the entire building (for the same LOD).

7.5. Appearances

(Imported from the CityGML 3.0 Standard)

In addition to semantics and geometry, information about the appearance of surfaces, i.e., observable properties of the surface, is considered an integral part of virtual 3D city and landscape models. Appearance relates to any surface-based theme such as infrared radiation or noise pollution, not just visual properties like RGB texture images. Consequently, data provided by appearances can be used as input for both, presentation of and analysis in virtual 3D city models.

The CityGML Conceptual Model supports feature appearances for an arbitrary number of themes per city model. Each LOD of a feature can have an individual appearance. Appearances can represent among others textures and georeferenced textures. CityGML s appearance model is packaged within the Appearance module (cf. <<ug-model-appearance-section>>).

7.6. modeling Dynamic Data

(Imported from the CityGML 3.0 Standard)

In general, city objects can have properties related to their geometry, topology, semantics, and appearance. All of these properties may change over time. For example, a construction event leads to the change in geometry of a building (i.e., addition of a new building floor or demolition of an existing door). The geometry of an object can be further classified according to its shape, location, and extent, which can also change over time. A moving car object involves changing only the location of the car object. However, a flood incident involves variations in the location and shape of water. There might be other properties, which change with respect to thematic data of city objects such as hourly variations in energy or gas consumption of a building or changing the building usage from residential to commercial. Some properties involve changes in appearances over a time period, such as building textures changing over years or traffic cameras recording videos of moving traffic over definite intervals. 3D city models also represent interrelationships between objects and relations may change over time as well. Hence, it is important to consider that the representation of time-varying data is required to be associated with these different properties. A detailed discussion on the requirements of city model applications regarding the support of dynamic data is given in [<<Chaturvedi2019>>].

The CityGML 3.0 Conceptual Model introduces two concepts to manage dynamic, time-dependent, properties of city models. The _Versioning_ module manages changes that are slower in nature. Examples are (1) the history or evolution of cities such as construction or demolition of buildings, and (2) managing multiple versions of the city models.

The _Dynamizer_ module manages higher-frequency or dynamic variations of object properties, including variations of (1) thematic attributes such as changes of physical quantities (energy demands, temperature, solar irradiation levels), (2) spatial properties such as change of a feature s geometry, with respect to shape and location (moving objects), and (3) real-time sensor observations. The Dynamizer module allows establishing explicit links from city objects to sensors and sensor data services.

7.6.1. Versioning and Histories

As described in <<ug-semantic-modeling-section>>, the bitemporal timestamps of all CityGML feature types allow representing the evolution of the real city and its model over time. The new _Versioning_ module extends this concept by the possibility of representing multiple, concurrent versions of the city model. For that purpose, the module defines two new feature types: 1) _Version_, which can be used to explicitly define named states of the 3D city model and denote all the specific versions of objects belonging to such states. 2) _VersionTransition_, which allows to explicitly link different versions of the 3D city model by describing the reason of change and the modifications applied. Details on the versioning concept are given in [<<Chaturvedi2015>>].

This approach not only facilitates the explicit representation of different city model versions, but also allows distinguishing and referring to different versions of city objects in an interoperable exchange format. All object versions could be stored and exchanged within a single dataset. Software systems could use such a dataset to visualize and work with the different versions simultaneously. The conceptual model also takes into account the management of multiple histories or multiple interpretations of the past of a city, which is required when looking at historical city developments and for archaeological applications. In addition, the Versioning module supports collaborative work. All functionality to represent a tree of workspaces as version control systems like _git_ or _SVN_ is provided. The Versioning module handles versions and version transitions as feature types, which allows the version management to be completely handled using the standard OGC Web Feature Service [<<Vretanos2010>>]. No extension of the OGC Web Feature Service standard is required to manage the versioning of city models.

7.6.2. Dynamizers: Using Time-Series Data for Object Attributes

The new Dynamizer module improves the usability of CityGML for different kinds of simulations as well as to facilitate the integration of devices from the Internet-of-Things (IoT) like sensors with 3D city models. Both, simulations and sensors provide dynamic variations of some measured or simulated properties such as the electricity consumption of a building or the traffic density within a road segment. The variations of the value are typically represented using time-series data. The data sources of the time-series data could be either sensor observations (e.g., from a smart meter), pre-recorded load profiles (e.g., from an energy company), or the results of some simulation run.

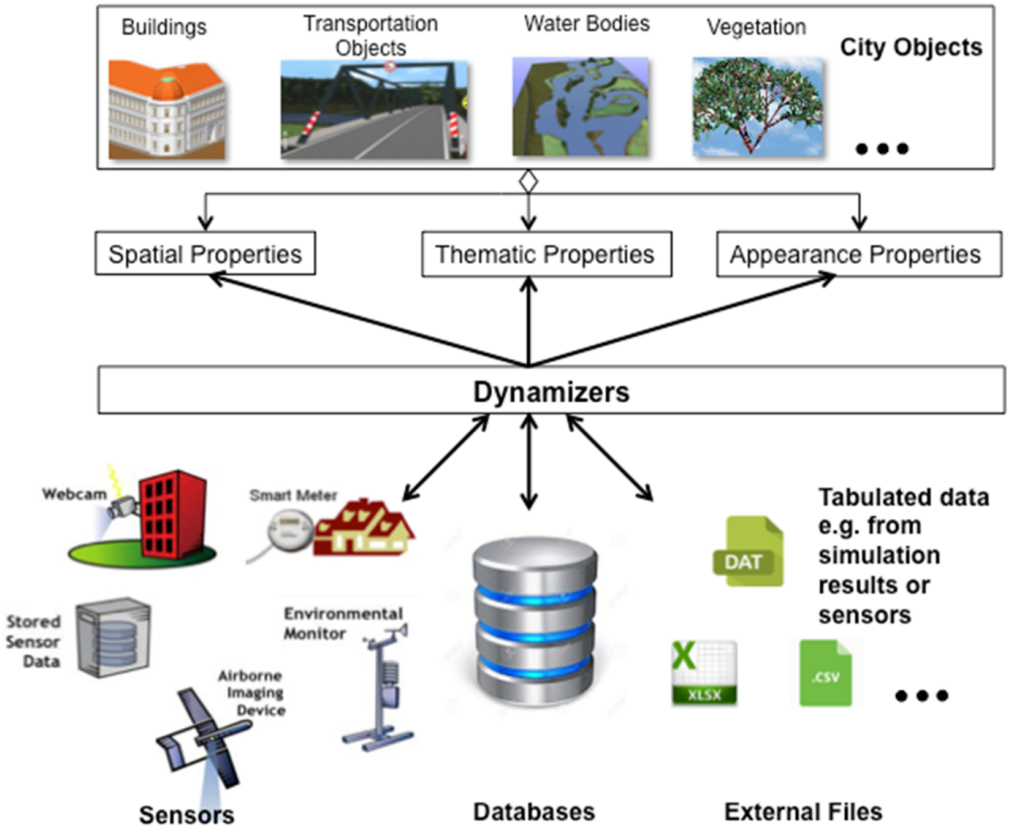


Figure 11. Dynamizers link timeseries data coming from different sources to specific properties of individual city objects.

```
As shown in <<figure-dynamizers>>, Dynamizers serve three main purposes.
```

1. Dynamizer is a data structure to represent dynamic values in different and generic ways. Such dynamic values may be given by (1) tabulation of time/value pairs using its *AtomicTimeseries* class, (2) patterns of time/value pairs based on statistical rules using its *CompositeTimeseries* class, and (3) retrieving observations directly from external sensor/IoT services using its *SensorConnection* class. The values can be obtained from sensor services such as the OGC Sensor Observation Service or OGC SensorThings API, simulation specific databases, and also external files such as CSV or Excel sheets.
2. Dynamizer delivers a method to enhance static city models by adding dynamic property values. A Dynamizer references a specific property (e.g., spatial, thematic or appearance properties) of a specific object within a 3D city model providing dynamic values overriding the static value of the referenced object attribute.
3. Dynamizer objects establish explicit links between sensor/observation data and the respective properties of city model objects that are measured by them. By making such explicit links with city object properties, the semantics of sensor data become implicitly defined by the city model.

```
Dynamizers are used to inject dynamic variations of city object properties into an
otherwise static representation. The advantage in following such an approach is
that it allows only selected properties of city models to be made dynamic. If an
application does not support dynamic data, the application simply does not
allow/include these special types of features.
```

Dynamizers have already been implemented as an Application Domain Extension (ADE) for CityGML 2.0 and were employed in the OGC Future City Pilot Phase 1. More details about Dynamizers are given in [<<Chaturvedi2017>>].

Chapter 8. CityGML Model

8.1. Structural Overview

Contributors
C. Heazel - first draft

Under Construction

8.2. Core

Contributors
C. Heazel - Author

Under Construction

8.2.1. Key Concepts

The following is a summary of the key concepts described by the Core Module. This is not an exhaustive listing of all of the Core concepts. Rather, it is an introduction those concepts which are essential for understanding the role of the Core Module in the CityGML Conceptual Model.

CityModel: (Discussion) The CityModel class is the root class of every CityGML conceptual model. It's primary purpose is to aggregate CityModelMembers.

CityModelMember: (Discussion) CityModelMember is a Union of all possible classes which can be included in a city model. Those classes are summarized in Table 3.

Table 3. City Model Members

AbstractAppearance	Rules for rending the associated objects
AbstractCityObject	Physical objects with a location and geometry
AbstractFeature	Others
AbstractVersion	Version information
AbstractVersionTransition	Version information

AbstractFeature: (Discussion) AbstractFeature is a subclass of the AnyFeature class from the ISO General Feature Model. Every «FeatureType» class in a CityGML model is descended from this class. `AbstractFeature` also defines identifying attributes which are common across the model. Finally, `AbstractFeature` allows an ADE to be associated with any «FeatureType» class.

AbstractFeatureWithLifespan: (Discussion)This class extends AbstractFeature with temporal properties for the classes. Most Feature classes in the CityGML model are descended from this class. As such they contain the following attributes:

- featureId
- identifier
- name
- description
- creationDate
- terminationDate
- validFrom
- validTo
- adeOfAbstractFeatureWithLifespan

AbstractAppearance: (Discussion) The root class for all classes related to the rendering of the model including textures and materials.

AbstractVersion: (Discussion) a defined state of a city model consisting of the dedicated versions of all city object instances that belong to the respective city model version.

AbstractVersionTransition: (Discussion) describes the change of the state of a city model from one version to another.

AbstractCityObject: (Discussion) The superclass of all thematic classes within the CityGML conceptual model. Significant subclasses of `AbstractCityObject` are AbstractSpace and AbstractSpaceBoundary. These subclasses provide CityGML objects with location, geometry and boundary properties.

AbstractSpace: (Discussion) Classes which have a volumetric extent in the real world.

AbstractPhysicalSpace: (Discussion) AbstractPhysicalSpace is the abstract superclass for all types of physical spaces. Physical space refers to spaces that are fully or partially bounded by physical objects.

AbstractOccupiedSpace: (Discussion) Classes which describe spaces that are partially or entirely filled with matter.

AbstractUnoccupiedSpace: (Discussion) Classes which describe spaces that are entirely or mostly free of matter.

AbstractLogicalSpace: (Discussion) Classes which describe spaces that are not bounded by physical surfaces but are defined according to thematic considerations.

AbstractSpaceBoundary: (Discussion) Classes which bound a space.

AbstractThematicSurface: (Discussion) The superclass for all thematic surfaces. A type of Space Boundary.

ImplicitGeometry: (Discussion) A geometry representation where the shape is stored only once as a prototypical geometry, for example a tree or other vegetation object, a traffic light or a traffic sign. This prototypic geometry object can be re-used or referenced many times, wherever the

corresponding feature occurs in the 3D city model. A type of Abstract Feature.

AbstractGenericAttribute: AbstractGenericAttribute is the abstract superclass for all types of generic attributes.

8.2.2. City Models and City Objects

Under Construction

CityModel

Under Construction

CityModelMember

Under Construction

AbstractFeature

Under Construction

AbstractFeatureWithLifespan

Under Construction

AbstractAppearance

Under Construction

AbstractVersion

Under Construction

AbstractVersionTransition

Under Construction

AbstractCityObject

Under Construction

8.2.3. Space Concept

Under Construction

8.3. Appearance

Contributors
C. Heazel - first draft

Under Construction

8.4. City Furniture

Contributors
C. Heazel: first draft

Under Construction

8.5. City Object Group

Contributors
C. Heazel: first draft

Under Construction

8.6. Dynamizer

Contributors
C. Heazel - first draft

Under Construction

8.7. Generics

Contributors
C. Heazel - first draft
T. Kutzner - update content from v2 to v3

8.7.1. Synopsis

The *Generics* module supports application-specific extensions to the CityGML conceptual model. These extensions may be used to model and exchange additional attributes and features not covered by the predefined thematic classes of CityGML. Generic extensions shall only be used if appropriate thematic classes or attributes are not provided by any other CityGML module.

8.7.2. Key Concepts

GenericLogicalSpace: A space that is not represented by any explicitly modeled AbstractLogicalSpace subclass within CityGML. (See Discussion for further details.)
A type of AbstractLogicalSpace.

GenericOccupiedSpace: A space that is not represented by any explicitly modeled AbstractOccupiedSpace subclass within CityGML. (See Discussion for further details.)

A type of AbstractOccupiedSpace.

GenericUnoccupiedSpace: A space that is not represented by any explicitly modeled AbstractUnoccupiedSpace subclass within CityGML. (See Discussion for further details.)
A type of AbstractUnoccupiedSpace.

GenericThematicSurface: A surface that is not represented by any explicitly modeled AbstractThematicSurface subclass within CityGML. (See Discussion for further details.)
A type of AbstractThematicSurface.

GenericAttributeSet: A named collection of generic attributes. (See Discussion for further details.)
A type of AbstractGenericAttribute.

StringAttribute, **IntAttribute**, **DoubleAttribute**, **DateAttribute**, **UriAttribute**, **MeasureAttribute**, **CodeAttribute**: Data types used to define generic attributes of the types *String*, *Int*, *Double*, *Date*, *URI*, *Measure*, or *Code*, respectively. (See Discussion for further details.)
A type of AbstractGenericAttribute.

8.7.3. Discussion

The *Generics* module provides the representation of generic city objects. These are city objects that are not covered by any explicitly modeled thematic class within the CityGML conceptual model. The Generics module also provides the representation of generic attributes, i.e., attributes that are not explicitly represented in the CityGML conceptual model.

In order to avoid problems concerning semantic interoperability, generic city objects and generic attributes shall only be used if appropriate thematic classes and attributes are not provided by any other CityGML module.

In accordance with the CityGML Space concept defined in the Core module, generic city objects can be represented as generic logical spaces, generic occupied spaces, generic unoccupied spaces, and generic thematic surfaces. In this way, spaces and surfaces can be defined that are not represented in the CityGML Conceptual Model by any explicitly modeled subclass of the classes AbstractLogicalSpace, AbstractOccupiedSpace, AbstractUnoccupiedSpace or AbstractThematicSurface. Generic city objects are represented in the UML model by the top-level feature types *GenericLogicalSpace*, *GenericOccupiedSpace*, *GenericUnoccupiedSpace* and *GenericThematicSurface*.

Generic attributes are defined as name-value pairs and are always associated with a city object. Generic attributes can be of type *String*, *Integer*, *Double*, *Date*, *URI*, *Measure*, and *Code*. In addition, generic attributes can be grouped under a common name as generic attribute sets. In the Core module, the class AbstractCityObject is related to the data type *AbstractGenericAttribute* via the role name *genericAttribute*. This means that each thematic subclass of AbstractCityObject inherits this relation and, thus, the possibility that an arbitrary number of generic attributes can be assigned to each thematic subclass in order to represent additional properties of features not represented by the properties that are explicitly defined in the UML model. This also means that the *Generics* module has a deliberate impact on all CityGML modules that define thematic subclasses of the class AbstractCityObject.

GenericLogicalSpace, GenericOccupiedSpace, GenericUnoccupiedSpace, and GenericThematicSurface

These classes are used to define generic city objects by representing them as generic spaces and surfaces in accordance with the CityGML 3 Space concept. All classes represent top level feature types and they are derived from corresponding abstract classes in the Core module:

- *GenericLogicalSpace* is to be used for generic city objects that represent *logical* spaces. The class is derived from the class AbstractLogicalSpace.
- *GenericOccupiedSpace* is to be used for generic city objects that represent *physically occupied* spaces. The class is derived from the class AbstractOccupiedSpace.
- *GenericUnoccupiedSpace* is to be used for generic city objects that represent *physically unoccupied* spaces. This class is derived from AbstractUnoccupiedSpace.
- *GenericThematicSurface* is to be used for representing surfaces that bound a space. This class is derived from the class AbstractThematicSurface.

Generic spaces and surfaces provide the properties *class*, *function* and *usage* whose values are defined in external code lists:

- The *class* property indicates the specific type of the space or surface, i.e., it allows for object classification within a thematic area (e.g., water pipes in the field of infrastructure or city zones in the field of city planning). This property is optional and can only be provided once.
- The *function* property specifies the intended purpose(s) of the space or surface (e.g., water supply or residential zones). This property is optional and can be provided multiple times.
- The *usage* property specifies the actual usage of the space or surface in cases where the way the object is actually used differs from the intended purpose of the object. This property is optional and can occur multiple times.

The geometry of a generic space or surface can be any of the geometries defined in the Core module for the classes AbstractSpace, AbstractPhysicalSpace, and AbstractThematicSurface. The generic spaces and surfaces inherit the geometries defined for these abstract classes, since they are subclasses thereof. This means:

- All generic spaces can be spatially represented as single point in LOD0, as multi-surface in LOD0/2/3, as solid in LOD1/2/3, and as multi-curve in LOD2/3. Absolute coordinates according to the reference system of the city model must be given for these geometries.
- Generic occupied and unoccupied spaces can, in addition, be complemented with a *TerrainIntersectionCurve* in LOD1/2/3, in order to specify the exact intersection of the DTM with the 3D geometry of these spaces. This is important for 3D visualization but also for certain applications like driving simulators. For instance, when a city wall (e.g., the Great Wall of China) is to be represented as a *GenericOccupiedSpace*, a smooth transition between the DTM and the city wall needs to be ensured. Above that, both spaces can also be represented by point clouds.
- Generic occupied spaces can, in addition, be spatially represented by an ImplicitGeometry. In this case, a reference point (anchor point) of the object and optionally a transformation matrix must be given. In order to compute the actual location of the object, the transformation of the local coordinates into the reference system of the city model must be processed and the anchor point coordinates must be added. The shape of an ImplicitGeometry can be given as an external

resource with a proprietary format, e.g., a VRML or DXF file from a local file system or an external web service. Alternatively the shape can be specified as a 3D geometry with local cartesian coordinates using the property *relativeGeometry*.

StringAttribute, IntAttribute, DoubleAttribute, DateAttribute, UriAttribute, MeasureAttribute, and CodeAttribute

These data types are used to define generic attributes of the types *String*, *Integer*, *Double*, *Date*, *URI*, *Measure*, or *Code*, respectively. The data types are derived from the abstract data type *AbstractGenericAttribute* that is defined in the Core module.

Each generic attribute has two mandatory properties:

1. *name* for specifying the name of the generic attribute. The name can be freely chosen.
2. *value* for specifying the value of the generic attribute. The type of the value may be *String*, *Integer*, *Double* (floating point number), *Date*, *URI*, *Measure*, or *Code*, depending on the selected subclass.

A *MeasureAttribute* facilitates the representation of measured values. Its value is of the structured type *Measure* which can additionally provide the units of measure via an optional attribute *uom* (units of measure).

GenericAttributeSet

Generic attributes can be grouped under a common name using a *GenericAttributeSet*. *GenericAttributeSet* is a data type that is derived from the data type *AbstractGenericAttribute* defined in the Core module; thus, generic attribute sets represent generic attributes themselves. This means that generic attribute sets may also be contained in other generic attribute sets, facilitating a recursive nesting of arbitrary depth.

The value of a generic attribute set is implicitly the set of contained generic attributes to which *GenericAttributeSet* relates via the role name *genericAttribute*. The *name* property specifies the name for the entire generic attribute set. The optional *codeSpace* property (of type *URI*) can be used to associate the generic attribute set with an authority, e.g., the organisation or community that defines and maintains the generic attribute set and its contained attributes. In this way, generic attribute sets can be clearly distinguished from each other even if they share the same name.

8.7.4. UML Model

The UML diagram of the *Generics* module is depicted in Figure 12.

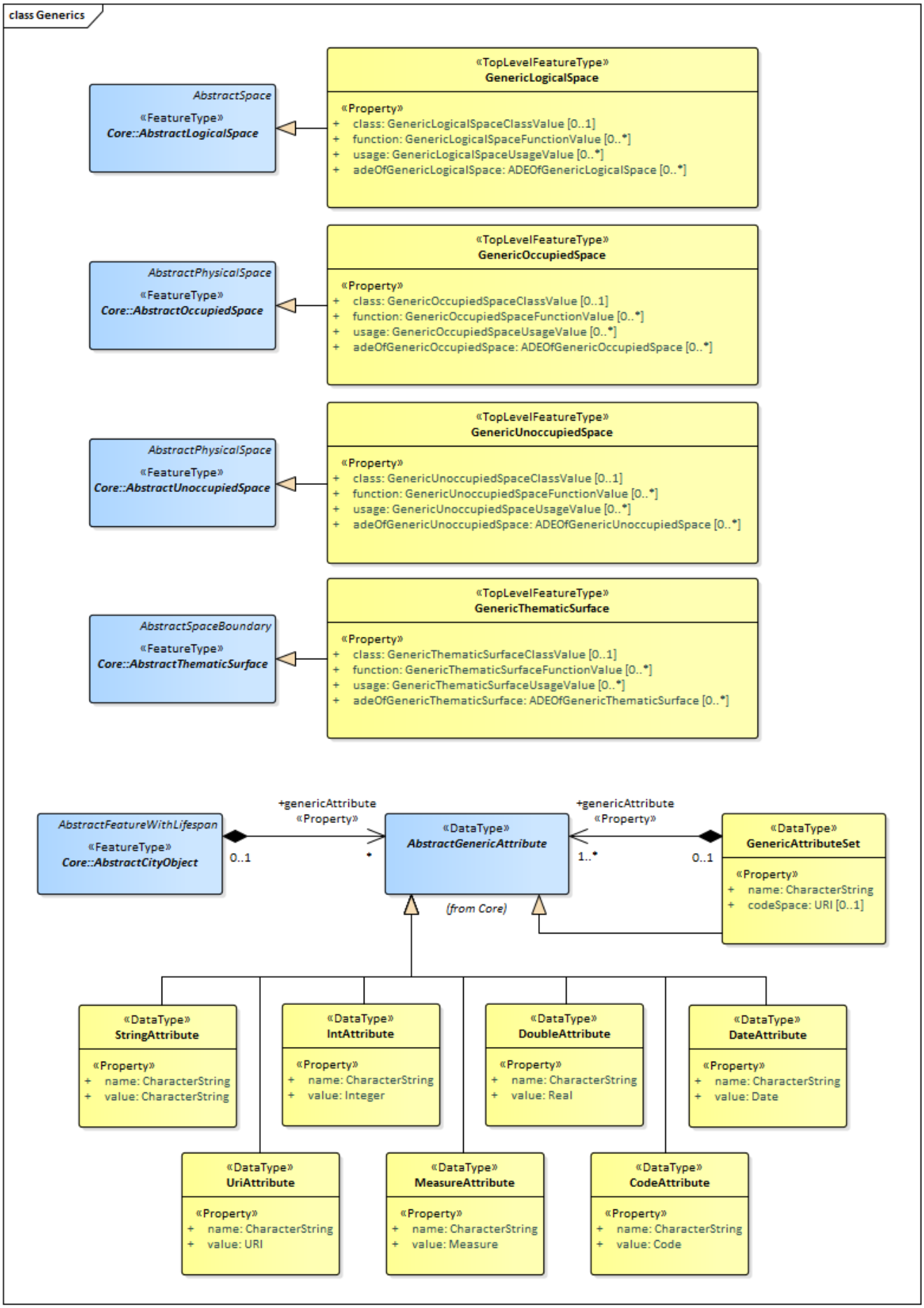


Figure 12. UML diagram of the Generics Model.

The ADE data types provided for the *Generics* module are illustrated in Figure 13.

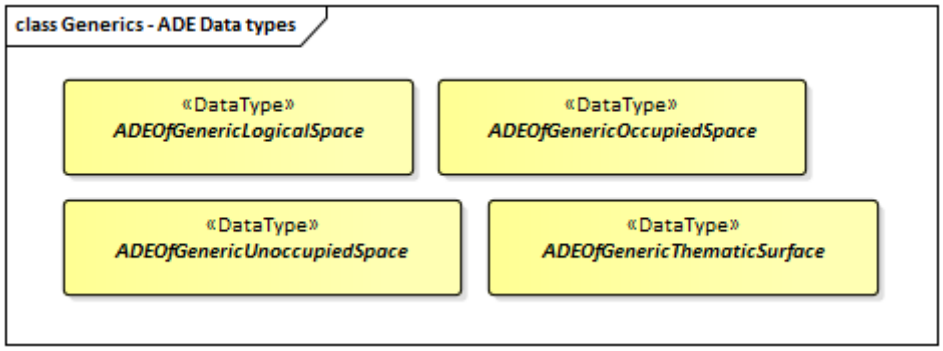


Figure 13. ADE classes of the CityGML Generics module.

The Code Lists provided for the *Generics* module are illustrated in Figure 14.

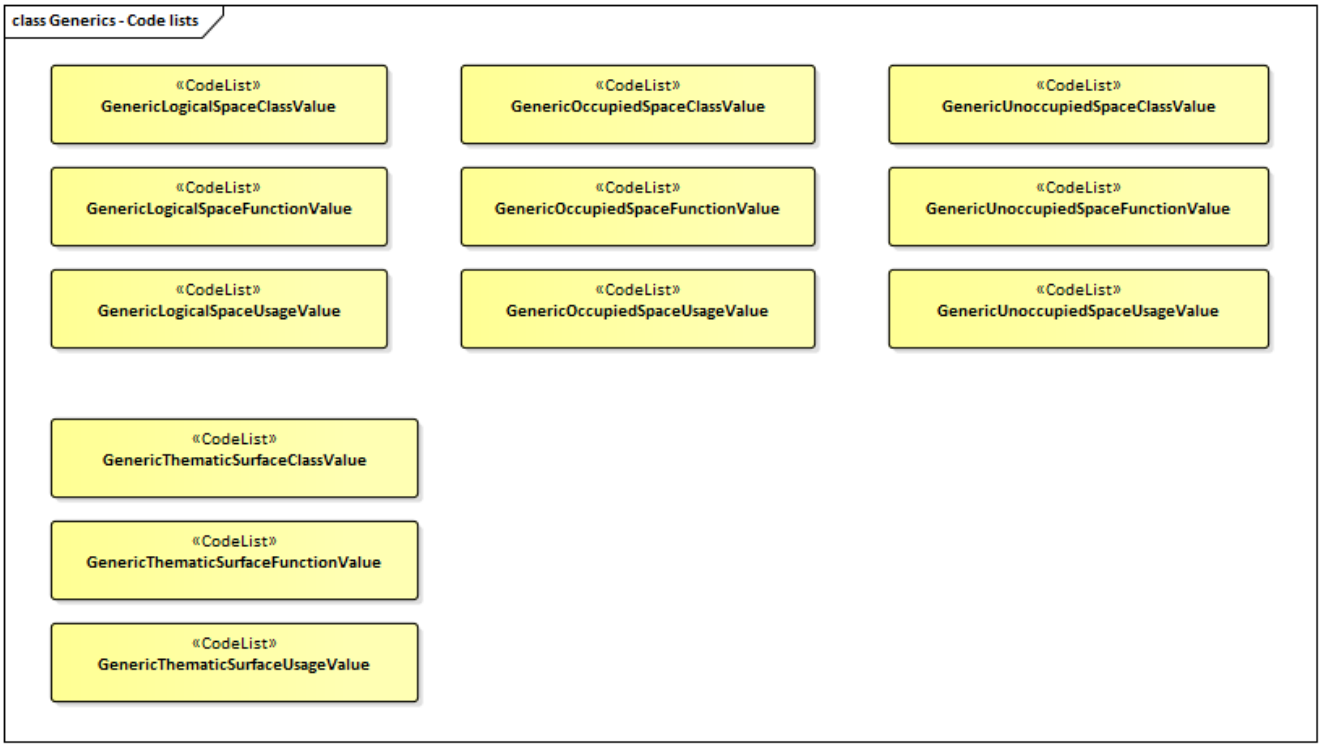


Figure 14. Codelists from the CityGML Generics module.

8.7.5. Examples

Under Construction

8.8. Land Use

Contributors
Chuck Heazel - first draft

Under Construction

8.9. Point Cloud

Contributors

C. Heazel - first draft

Under Discussion

8.10. Relief

Contributors
C. Heazel - first draft

Under Construction

8.11. Transportation

Contributors
C. Heazel - first draft

Under Construction

8.12. Vegetation

Contributors
C. Heazel - first draft

Under Construction

8.13. Versioning

Contributors
C. Heazel - first draft
T. Kutzner - added further explanations

8.13.1. Synopsis

The *Versioning* module supports the representation of multiple versions of CityGML features within a single CityGML model. In addition, also the version transitions and transactions that lead to the different versions can be represented.

8.13.2. Key Concepts

Version: *Version* represents a defined state of a city model consisting of the dedicated versions of all city object instances that belong to the respective city model version. Versions can have names, a description and can be labeled with an arbitrary number of user defined tags. (See Version section for further details.)
A type of AbstractVersion.

VersionTransition: *VersionTransition* describes the change of the state of a city model from one version to another. Version transitions can have names, a description and can be further qualified by a type and a reason. (See Version Transition section for further details.)
A type of AbstractVersionTransition.

VersionTransaction: *VersionTransaction* indicates the specific type of modification applied to a city object in a transition from one city model version to another. (See Version Transaction section for further details.)

8.13.3. Discussion

The Versioning module defines the concepts that enable representing multiple versions of a city model. A specific version represents a defined state of a city model consisting of the dedicated versions of all city object instances that belong to the respective city model version. Each version can be complemented by version transitions that describe the change of the state of a city model from one version to another and that give the reason for the change and the modifications applied.

By using the Versioning module, slow changes over a long time period with respect to cities and city models can be represented. This includes the creation and termination of objects (e.g., construction or demolition of sites, planting of trees, construction of new roads), structural changes of objects (e.g., extension of buildings), and changes in the status of an object (e.g., change of building owner, change of the traffic direction of a road to a one-way street). In this way, the history or evolution of cities and city models can be modeled, parallel or alternative versions of cities and city models can be managed, and changes of geometries and thematic properties of individual city objects over time can be tracked.

This approach not only facilitates the explicit representation of different city model versions, but also allows distinguishing and referring to different versions of city objects in an interoperable exchange format. All object versions could be stored and exchanged within a single dataset. Software systems could use such a dataset to visualize and work with the different versions simultaneously. The conceptual model also takes into account the management of multiple histories or multiple interpretations of the past of a city, which is required when looking at historical city developments and for archaeological applications. In addition, the Versioning module supports collaborative work. All functionality to represent a tree of workspaces as version control systems like git or SVN is provided. The Versioning module handles versions and version transitions as feature types, which allows the version management to be completely handled using the standard OGC Web Feature Service [Vrenatos 2010]. No extension of the OGC Web Feature Service standard is required to manage the versioning of city models.

In order to allow for representing the evolution of real cities and their 3D city models over time, the class AbstractFeatureWithLifespan introduces bitemporal timestamps for all features by defining the following four attributes:

- *creationDate* and *terminationDate*: These attributes refer to the time period a specific version of a feature is an integral part of the 3D city model. The attributes can be used to query how did the city model look at a specific point in time.
- *validFrom* and *validTo*: These attributes represent the lifespan a specific version of a feature has in the real-world. The attributes can be used to query how did the actual city look alike at a specific point in time.

The class AbstractFeatureWithLifespan is defined in the Core module and acts as base class of all CityGML features, i.e., all features that are represented by subclasses of AbstractFeatureWithLifespan inherit the bitemporal timestamps. This includes not only all thematic features defined in the thematic extension modules, such as Building, CityFurniture, or Transportation, but also all features defined in the general extension modules Appearance, Dynamizer, Generics, PointCloud, and Versioning.

The *Versioning* module extends this concept of bitemporal timestamps by the possibility of representing multiple, concurrent versions of the city model. For that purpose, the module defines two new feature types, *Version* and *VersionTransition*, which are explained in more detail below.

Version

The class *Version* can be used to explicitly define states of the 3D city model and denote all the specific versions of objects belonging to such states. Versions can have the following properties.

- *name* for providing the version with a specific name. This property is inherited from the class AbstractFeature defined in the Core module.
- *description* for describing the version in more detail. This property is inherited from the class AbstractFeature as well.
- *tag* for adding user-defined tags to the version. With the help of such tag attributes, the user can, e.g., search for a version developed by a specific worker. The property is of the type *CharacterString* which means that the user can provide any textual value describing the specific state in the most appropriate way. The property can be provided multiple times.
- *versionMember* for referencing the objects that that belong to the version. Any object that is a direct or indirect subclass of AbstractFeatureWithLifespan can be part of a version. The objects within each version are referenced via the *featureID* property that all objects inherit from the class AbstractFeature defined in the Core module.

In this context, a *snapshot* represents the state of all objects of the entire city model at a specific point in time. It explicitly links to all objects in their versions that belong to the corresponding city model version.

In CityGML, features can have aggregated sub-features. For instance, a building feature may consist of the sub-features roof surface and wall surface, which, in turn, may contain the sub-features window surface or door surface. In case of a change in any of the sub-features, the model would require a change in all the parent features in the aggregation levels above as well, because the aggregate object points to its parts. If a part is replaced by a new version with a new *featureID*, the corresponding pointer of the aggregate object also needs to be updated. This will create a new object version also for the aggregate object, and so on, following the aggregation hierarchy.

Version Transition

The class *VersionTransition* allows to explicitly link different versions of the 3D city model, referred to as predecessor and successor versions, by describing the reason of change and the modifications applied. Version transitions represent the causal relationships between the version snapshots.

Version transitions are represented by the following properties.

- *reason*: Specifies why the predecessor version has been changed. This property is optional.
- *clonePredecessor*: Indicates how to obtain the set of objects belonging to the successor version of the city model. The value *false* indicates that the set of objects is explicitly enumerated within the successor version. The value *true* indicates that that the set of objects has to be derived from the modifications of the city model that are provided as a list of transactions on the city object versions in the predecessor version. This property is mandatory.
- *type*: Indicates the specific type of the transition, i.e., whether the transition is *planned*, *realized*, a *historicalSuccession*, a *fork*, or a *merge*. Only these five values are allowed; they are defined in the Enumeration *TransitionTypeValue*. *planned* and *fork* should be used in cases when from one city model version multiple successor versions are being created. *realized* and *merge* should be used when different city model versions are converging into a common successor version. This property is optional.
- *transaction*: Relates to the specific transactions that have been applied to the individual objects in the version transition. A version transition can comprise several transactions.
- *from*: Relates to the predecessor version of the city model that is linked by the version transition to its successor version. This property is optional.
- *to*: Relates to the successor version of the city model that is linked by the version transition to its predecessor version. This property is optional.

Advantages of the version transition approach are that it requires low memory or storage requirements, that it is similar to full back-up or incremental back-up, and that it may also be used to stream dynamic changes.

However, a limitation with this approach is that it may be necessary to determine the actual members of a version by going back via the predecessors. One important aspect is that the merging of two different versions may lead to possible conflicts. For all such convergence situations it must be ensured that the members of the converged version/state can be determined unambiguously. The easiest and safest way is to require that at maximum one of the incoming transitions has transactions. Such transitions are required to be able to detect who has changed an object and whether there are any conflicts. Several methods for detecting changes in CityGML files have already been identified, such as in [Redweik & Becker 2015] and [Pédrinis et al. 2015]. These methods may be useful tools to resolve conflicts in a similar way as with the "diff" command in SVN.

Version Transaction

This data type provides information on what type of modification has been applied to an individual city object that is part of a specific version. Modifications can be the creation, termination, or replacement of a specific city object. While the creation of a city object also marks its first object version, the termination marks the end of existence of a real world object and, hence, also terminates the final version of a city object. The replacement of a city object means that a specific version of it is replaced by a new version.

Transactions have the following properties.

- *type*: Indicates the specific type of the transaction. Allowed values are *insert* (creation), *delete* (termination), or *update* (replacement); they are defined in the Enumeration

TransactionTypeValue. This property is mandatory.

- *oldFeature*: Relates to the predecessor version of the city object that is modified by the transaction. This property is optional and it is always empty for the value *insert*.
- *newFeature*: Relates to the successor version of the city object that is modified by the transaction. This property is optional and it is always empty for the value *delete*.

8.13.4. UML Model

The UML diagram of the *Versioning* module is depicted in Figure 15.

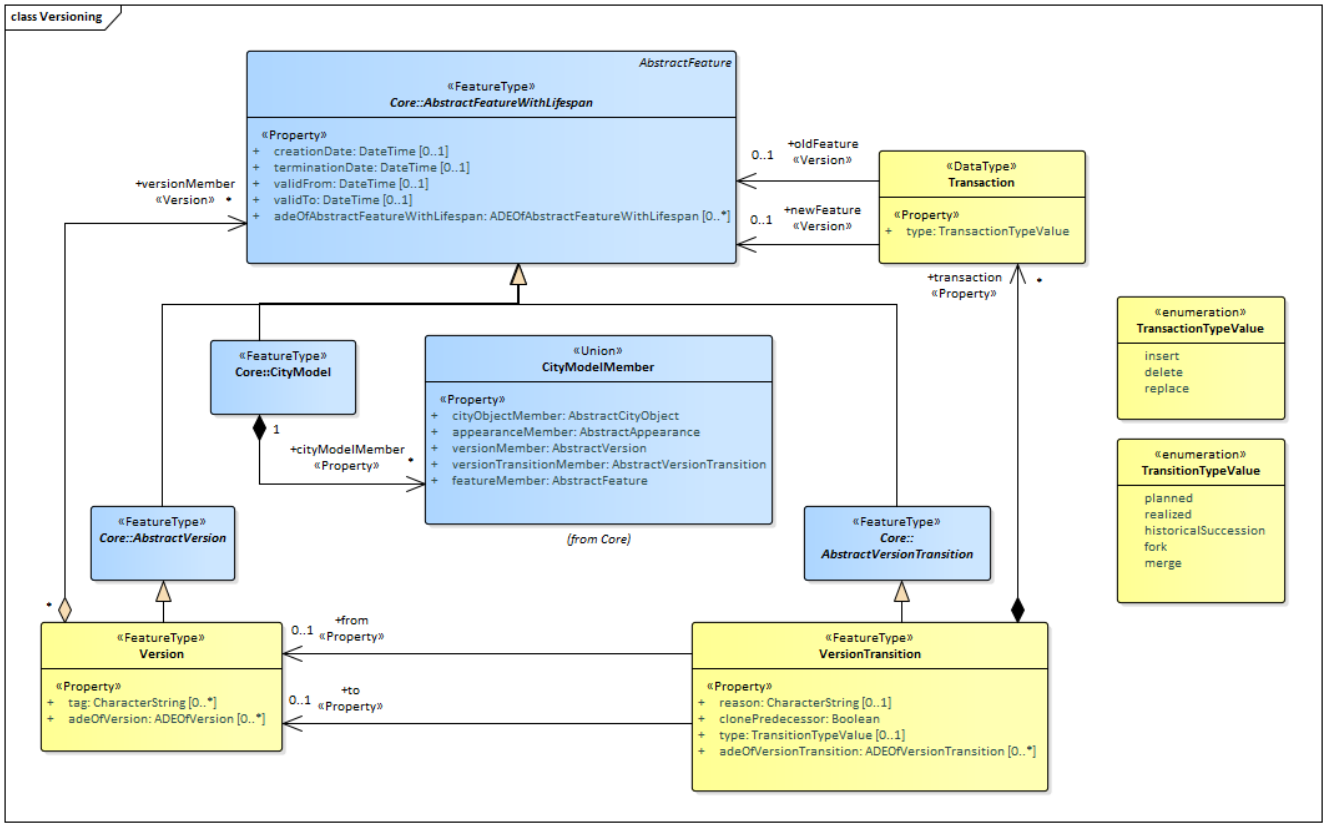


Figure 15. UML diagram of the Versioning Model.

The ADE data types provided for the *Versioning* module are illustrated in Figure 16.

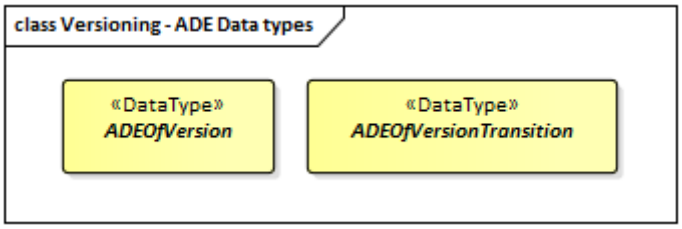


Figure 16. ADE classes of the CityGML Versioning module.

8.13.5. Examples

NOTE

To Be Provided

8.14. Water Body

Contributors
C. Heazel - first draft

Under Construction

8.15. Construction

Contributors
C. Heazel - first draft

Under Construction

8.16. Bridge

Contributors
C. Heazel - first draft

Under Construction

8.17. Building

Contributors
C. Heazel - first draft

Under Construction

8.18. Tunnel

Contributors
C. Heazel - first draft

Under Construction

Chapter 9. CityGML Extensions

9.1. Extending CityGML

(Imported from the CityGML 3.0 Standard)

```
CityGML is designed as a universal information model that defines object types and
attributes which are useful for a broad range of applications. In practical
applications, the objects within specific 3D city models will most likely contain
attributes which are not explicitly modeled in CityGML. Moreover, there might be 3D
objects which are not covered by the CityGML CM thematic classes. The CityGML CM
provides three different concepts to support the exchange of such data:
```

1. Generic objects and attributes,
2. Application Domain Extensions, and
3. Code lists.

```
The concept of generic objects and attributes allows application-specific concepts
to be represented in CityGML at runtime. This means that any city object may be
augmented by additional attributes and relations, whose names, data types, and
values can be provided by a running application without requiring extensions to the
CityGML conceptual schema and the respective encodings. Similarly, features not
represented by the predefined thematic classes of the CityGML conceptual model may
be modeled and exchanged using generic objects. The generic extensions of CityGML
are provided by the _Generics_ module (cf. <<ug-generics-section>>).
```

```
Application Domain Extensions (ADE) specify additions to the CityGML conceptual
model. Such additions comprise the introduction of new properties to existing
CityGML feature types such as the energy demand of a building or the definition of
additional feature types. The difference between ADEs and generic objects and
attributes is, that an ADE has to be defined in an extra conceptual schema
(provided in UML) with its own namespace. Encodings have to be extended
accordingly. The advantage of this approach is that the extension is formally
specified. Extended CityGML datasets can be validated against the CityGML CM and
the respective ADE schema. ADEs can be defined (and even standardized) by
information communities which are interested in specific application fields. More
than one ADE can be used simultaneously in the same dataset. Examples for popular
ADEs are the Utility Network ADE [<<Becker2011>>; <<Kutzner2018>>] and the Energy
ADE [<<Nouvel2015>>; <<Agugiaro2018>>]. A comprehensive overview of CityGML ADEs is
given in [<<Biljecki2018>>]. Further details on ADEs are given in
<<ug_ade_section>>.
```

CityGML can also be extended with regard to the allowed values specified in code lists. Many attributes of CityGML types use a code list as a data type such as, for instance, the attributes _class_, _usage_, and _function_ of city objects. A code list defines a value domain including a code for each permissible value. In contrast to fixed enumerations, modifications and extensions to the value domain become possible with code lists. The values for all code lists in CityGML have to be defined externally. This could, for example, be by adopting classifications from global, national, or industrial standards.

9.2. Application Domain Extensions

Contributors
TBD

Under Construction

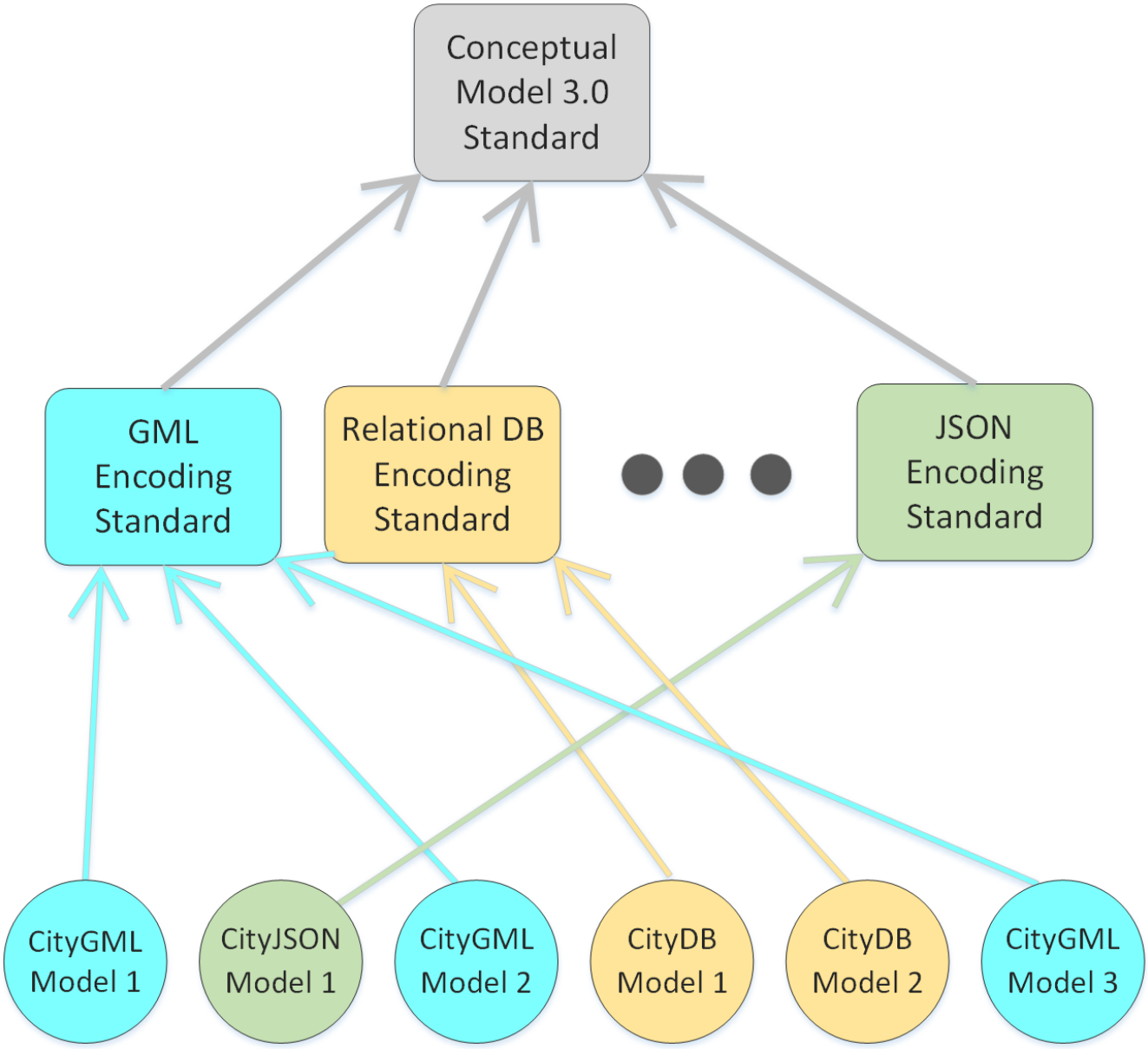
9.3. codelists

Contributors
C. Heazel - first draft

Under Construction

Chapter 10. Implementation Specifications

CityGML version 3.0 consists of a single Conceptual Model and multiple implementations of the Conceptual Model using different data definition and data storage methodologies. Each storage methodology is designated as a named encoding. This scope of this repository is encodings of the Conceptual Model.



CityGML model instances are created according to CityGML Encoding standards, which in turn are implementations of the abstract CityGML Conceptual Model

These Implementation Specifications are being developed on the CityGML-3.0 Encodings GitHub repository.

10.1. GML Encoding Standard

Work is well underway on the **GML Encoding Standard for CityGML 3.0** Implementation Specification.

Draft versions of the Standard are availble as PDF and HTML documents.

10.2. JSON Encoding Standard

Work has not yet begun on this Implementation Specification.

10.3. Relational Database Encoding Standard

Work has not yet begun on this Implementation Specification.

Annex A: Glossary

conformance test class
set of conformance test modules that must be applied to receive a single certificate of conformance
[OGC 08-131r3, definition 4.4]

feature
abstraction of real world phenomena
[ISO 19101-1:2014, definition 4.1.11]

feature attribute
characteristic of a feature
[ISO 19101-1:2014, definition 4.1.12]

feature type
class of features having common characteristics
[ISO 19156:2011, definition 4.7]

measurement
set of operations having the object of determining the value of a quantity
[ISO 19101-2:2018, definition 3.21] / [VIM:1993, 2.1]

model
abstraction of some aspects of reality
[ISO 19109:2015, definition 4.15]

observation
act of measuring or otherwise determining the value of a property
[ISO 19156:2011, definition 4.11]

observation procedure
method, algorithm or instrument, or system of these, which may be used in making an observation
[ISO 19156:2011, 4.12]

observation result
estimate of the value of a property determined through a known observation procedure
[ISO 19156:2011, 4.14]

property
facet or attribute of an object referenced by a name.
[ISO 19143:2010, definition 4.21]

requirements class
aggregate of all requirement modules that must all be satisfied to satisfy a conformance test class
[OGC 08-131r3, definition 4.19]

schema
formal description of a model
[ISO 19101-1:2014, definition 4.1.34]

sensor
type of observation procedure that provides the estimated value of an observed property at its output
[OGC 08-094r1, definition 4.5]

Standardization Target
TBD

timeseries
sequence of data values which are ordered in time
[OGC 15-043r3]

universe of discourse
view of the real or hypothetical world that includes everything of interest
[ISO 19101-1:2014, definition 4.1.38]

version
Particular variation of a spatial object
[INSPIRE Glossary]

A.1. ISO Concepts

The following concepts from the ISO TC211 Harmonized UML model are referenced by the CityGML Conceptual UML model but do not play a major role in its' definition. They are provided here to support a more complete understanding of the model.

Area
The measure of the physical extent of any topologically 2-D geometric object. Usually measured in "square" units of length.
[ISO 19103:2015]

Boolean
boolean is the mathematical datatype associated with two-valued logic
[ISO 19103:2015]

CC_CoordinateOperation
mathematical operation on coordinates that transforms or converts coordinates to another coordinate reference system.
[ISO 19111:2019]

Character
symbol from a standard character-set.
[ISO 19103:2015]

CharacterString
Characterstring is a family of datatypes which represent strings of symbols from standard character-sets.
[ISO 19103:2015]

CRS

Coordinate reference system which is usually single but may be compound.
[ISO 19111:2019]

CV_DiscreteCoverage
A subclass of CV_Coverage that returns a single record of values for any direct position within a single geometric object in its spatiotemporal domain.
[ISO 19123:2005]

CV_DomainObject
[ISO 19123:2005]

CV_GridPointValuePair
[ISO 19123:2005]

CV_GridValuesMatrix
The geometry represented by the various offset vectors is in the image plane of the grid.
[ISO 19123:2005]

CV_ReferenceableGrid
[ISO 19123:2005]

Date
Date gives values for year, month and day. Representation of Date is specified in ISO 8601. Principles for date and time are further discussed in ISO 19108.
[ISO 19103:2015]

DateTime
A DateTime is a combination of a date and a time types. Representation of DateTime is specified in ISO 8601. Principles for date and time are further discussed in ISO 19108.
[ISO 19103:2015]

Distance
Used as a type for returning distances and possibly lengths.
[ISO 19103:2015]

Engineering CRS
A contextually local coordinate reference system which can be divided into two broad categories:

1. earth-fixed systems applied to engineering activities on or near the surface of the earth;
2. CRSs on moving platforms such as road vehicles, vessels, aircraft or spacecraft.
   [ISO 19111:2019]

Generic Name
Generic Name is the abstract class for all names in a NameSpace. Each instance of a GenericName is either a LocalName or a ScopedName.
[ISO 19103:2015]

Geometry
[ISO 19107:2003]

GM_CompositePoint
[ISO 19107:2003]

GM_CompositeSolid
set of geometric solids adjoining one another along common boundary geometric surfaces
[ISO 19107:2003]

GM_GenericSurface
GM_Surface and GM_SurfacePatch both represent sections of surface geometry, and therefore share a number of operation signatures. These are defined in the interface class GM_GenericSurface.
[ISO 19107:2003]

GM_LineString
consists of sequence of line segments, each having a parameterization like the one for GM_LineSegment
[ISO 19107:2003]

GM_MultiPrimitive
[ISO 19107:2003]

GM_OrientableSurface
a surface and an orientation inherited from GM_OrientablePrimitive. If the orientation is "+", then the GM_OrientableSurface is a GM_Surface. If the orientation is "-", then the GM_OrientableSurface is a reference to a GM_Surface with an upNormal that reverses the direction for this GM_OrientableSurface, the sense of "the top of the surface".
[ISO 19107:2003]

GM_PolyhedralSurface
a GM_Surface composed of polygon surfaces (GM_Polygon) connected along their common boundary curves.
[ISO 19107:2003]

GM_Position
a union type consisting of either a DirectPosition or of a reference to a GM_Point from which a DirectPosition shall be obtained.
[ISO 19107:2003]

GM_Primitive
The abstract root class of the geometric primitives. Its main purpose is to define the basic "boundary" operation that ties the primitives in each dimension together.
[ISO 19107:2003]

Integer
An exact integer value, with no fractional part.
[ISO 19103:2015]

Internet of Things
The network of physical objects--“things”--that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the Internet.

Wikipedia

IO_IdentifiedObjectBase
[ISO 19103:2015]

Length
The measure of distance as an integral, i.e., the limit of an infinite sum of distances between points on a curve.
[ISO 19103:2015]

Measure
The result from performing the act or process of ascertaining the extent, dimensions, or quantity of some entity.
[ISO 19103:2015]

Number
The base type for all number data, giving the basic algebraic operations.
[ISO 19103:2015]

Point
GM_Point is the basic data type for a geometric object consisting of one and only one point.
[ISO 19107:2003]

Real
The common binary Real finite implementation using base 2.
[ISO 19103:2015]

RS_ReferenceSystem
Description of a spatial and temporal reference system used by a dataset.
[ISO 19111:2019]

Scoped Name
ScopedName is a composite of a LocalName for locating another NameSpace and a GenericName valid in that NameSpace. ScopedName contains a LocalName as head and a GenericName, which might be a LocalName or a ScopedName, as tail.
[ISO 19103:2015]

Solid
GM_Solid, a subclass of GM_Primitive, is the basis for 3-dimensional geometry. The extent of a solid is defined by the boundary surfaces.
[ISO 19107:2003]

Time
Time is the designation of an instant on a selected time scale, astronomical or atomic. It is used in the sense of time of day.
[ISO 19103:2015]

TM_Duration
[ISO 19108:2006]

TM_TemporalPosition
The position of a TM_Instant relative to a TM_ReferenceSystem.
[ISO 19108:2006]

Unit of Measure
Any of the systems devised to measure some physical quantity such distance or area or a system devised to measure such things as the passage of time.
[ISO 19103:2015]

URI
Uniform Resource Identifier (URI), is a compact string of characters used to identify or name a resource
[ISO 19103:2015]

Volume
Volume is the measure of the physical space of any 3-D geometric object.
[ISO 19103:2015]

A.2. Abbreviated Terms

- 2D Two Dimensional
- 3D Three Dimensional
- AEC Architecture, Engineering, Construction
- ALKIS German National Standard for Cadastral Information
- ATKIS German National Standard for Topographic and Cartographic Information
- BIM Building Information Modeling
- B-Rep Boundary Representation
- bSI buildingSMART International
- CAD Computer Aided Design
- COLLADA Collaborative Design Activity
- CSG Constructive Solid Geometry
- DTM Digital Terrain Model
- DXF Drawing Exchange Format
- EuroSDR European Spatial Data Research Organisation
- ESRI Environmental Systems Research Institute
- FM Facility Management
- GDF Geographic Data Files
- GDI-DE Spatial Data Infrastructure Germany (Geodateninfrastruktur Deutschland)
- GDI NRW Geodata Infrastructure North-Rhine Westphalia
- GML Geography Markup Language

- IAI International Alliance for Interoperability (now buildingSMART International (bSI))
- IETF Internet Engineering Task Force
- IFC Industry Foundation Classes
- IoT Internet of Things
- ISO International Organization for Standardisation
- ISO/TC211 ISO Technical Committee 211
- LOD Levels of Detail
- MQTT Message Queuing Telemetry Transport
- NBIMS National Building Information Model Standard
- OASIS Organisation for the Advancement of Structured Information Standards
- OGC Open Geospatial Consortium
- OSCRE Open Standards Consortium for Real Estate
- SIG 3D Special Interest Group 3D of the GDI-DE
- TIC Terrain Intersection Curve
- TIN Triangulated Irregular Network
- UML Unified Modeling Language
- URI Uniform Resource Identifier
- VRML Virtual Reality Modeling Language
- W3C World Wide Web Consortium
- W3DS OGC Web 3D Service
- WFS OGC Web Feature Service
- X3D Open Standards XML-enabled 3D file format of the Web 3D Consortium
- XML Extensible Markup Language
- xAL OASIS extensible Address Language

Annex B: Bibliography

- Open Geospatial Consortium: **The Specification Model — A Standard for Modular specifications**, OGC 08-131
- Agugiaro, G., Benner, J., Cipriano, P., Nouvel, R., 2018: **The Energy Application Domain Extension for CityGML: enhancing interoperability for urban energy simulations**. Open Geospatial Data, Software and Standards, Vol. 3. https://doi.org/10.1186/s40965-018-0042-y
- Becker, T., Nagel, C., Kolbe, T. H., 2011: **Integrated 3D Modeling of Multi-utility Networks and their Interdependencies for Critical Infrastructure Analysis**. In: T. H. Kolbe, G. König, C. Nagel (Eds.): Advances in 3D Geoinformation Sciences. LNG&C, Springer, Berlin. https://doi.org/10.1007/978-3-642-12670-3_1
- Beil, C., Kolbe, T. H., 2017: **CityGML and the streets of New York - A proposal for detailed street space modelling**. In: Proceedings of the 12th International 3D GeoInfo Conference 2017, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. IV-4/W5, ISPRS. http://doi.org/10.5194/isprs-annals-IV-4-W5-9-2017
- Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., Çöltekin, A., 2015: **Applications of 3D City Models: State of the Art Review**. ISPRS International Journal of Geo-Information, 4(4). https://doi.org/10.3390/ijgi4042842
- Biljecki, F., Kumar, K., Nagel, C., 2018: **CityGML Application Domain Extension (ADE): overview of developments**. Open Geospatial Data, Software and Standards, 3(1). https://doi.org/10.1186/s40965-018-0055-6
- Billen, R., Zaki, C. E., Servières, M., Moreau, G., Hallot, P., 2012: **Developing an ontology of space: Application to 3D city modeling**. In: Leduc, T., Moreau, G., Billen, R. (eds): Usage, usability, and utility of 3D city models — European COST Action TU0801, EDP Sciences, Nantes, Vol. 02007. https://hal.archives-ouvertes.fr/hal-01521445
- Chaturvedi, K., Smyth, C. S., Gesquière, G., Kutzner, T., Kolbe, T. H., 2015: **Managing versions and history within semantic 3D city models for the next generation of CityGML**. In: Selected papers from the 10th International 3DGeoInfo Conference 2015 in Kuala Lumpur, Malaysia, Springer LNG&C, Berlin. https://doi.org/10.1007/978-3-319-25691-7_11
- Chaturvedi, K., Kolbe, T. H., 2016: **Integrating Dynamic Data and Sensors with Semantic 3D City Models in the context of Smart Cities**. In: Proceedings of the 11th International 3D Geoinfo Conference, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. IV-2/W1, ISPRS. https://doi.org/10.5194/isprs-annals-IV-2-W1-31-2016
- Chaturvedi, K., Kolbe, T. H., 2017: **Future City Pilot 1 Engineering Report**, Open Geospatial Consortium. OGC Doc. 19-098
- Chaturvedi, K., Kolbe, T. H., 2019: **A Requirement Analysis on Extending Semantic 3D City Models for Supporting Time-dependent Properties**. In: Proceedings of the 4th International Conference on Smart Data and Smart Cities, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. IV-4/W9, ISPRS. https://doi.org/10.5194/isprs-annals-IV-4-W9-19-2019
- Elfes, A., 1989: **Using occupancy grids for mobile robot perception and navigation**. Computer 22(6):46–57. https://doi.org/10.1109/2.30720

- Foley, J., van Dam, A., Feiner, S., Hughes, J., 2002: **Computer Graphics: Principles and Practice**. 2nd ed., Addison Wesley
- Gröger, G., Plümer, L., 2012: **CityGML – Interoperable semantic 3D city models**. ISPRS Journal of Photogrammetry and Remote Sensing, Vol. 71, July 2012. https://dx.doi.org/10.1016/j.isprsjprs.2012.04.004
- Gröger, G., Kolbe, T. H., Nagel, C., Häfele, K.-H., 2012: **OGC City Geography Markup Language (CityGML) Encoding Standard, Version 2.0.0**, Open Geospatial Consortium. OGC Doc. 12-019
- Jensen, Christian S. and Dyreson, Curtis E.: **The Consensus Glossary of Temporal Database Concepts**. February 1998 Version. In: Temporal Databases: Research and Practice [online]. Springer Berlin Heidelberg, 1998. p. 367–405. Lecture Notes in Computer Science. Available from: 10.1007/BFb0053710
- Jensen, Christian S. and Snodgrass, Richard T., eds.: **TR-90, Temporal Database Entries for the Springer Encyclopedia of Database Systems**. Technical Report. TimeCenter, 22 May 2008. Available from: http://timecenter.cs.aau.dk/TimeCenterPublications/TR-90.pdf
- Johnson, Tom: **Bitemporal Data**. Elsevier, 2014. ISBN 978-0-12-408067-6. Available from: 10.1016/C2012-0-06609-4
- Kaden, R., Clemen, C., 2017: **Applying Geodetic Coordinate Reference Systems within Building Information Modeling (BIM)**. In: Proceedings of the FIG Working Week 2017, Helsinki, Finland. https://www.fig.net/resources/proceedings/fig_proceedings/fig2017/papers/ts06h/TS06H_kaden_clemen_8967.pdf
- Kolbe, T. H., Gröger, G., 2003: **Towards unified 3D city models**. In: Proceedings of the Joint ISPRS Commission IV Workshop on Challenges in Geospatial Analysis, Integration and Visualization II, Stuttgart, Germany. https://mediatum.ub.tum.de/doc/1145769/
- Kolbe, T. H., 2009: **Representing and Exchanging 3D City Models with CityGML**. In: J. Lee, S. Zlatanova (Eds.), 3D Geo-Information Sciences, Selected Papers of the 3rd International Workshop on 3D Geo-Information in Seoul, Korea. Springer, Berlin. https://doi.org/10.1007/978-3-540-87395-2_2
- Konde, A., Tauscher, H., Biljecki, F., Crawford, J., 2018: **Floor plans in CityGML**. In: Proceedings of the 13th 3D GeoInfo Conference 2018, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. IV-4/W6, 25–32, ISPRS. https://doi.org/10.5194/isprs-annals-IV-4-W6-25-2018
- Kutzner, T., Hijazi, I., Kolbe, T. H., 2018: **Semantic modelling of 3D Multi-utility Networks for Urban Analyses and Simulations – The CityGML Utility Network ADE**. International Journal of 3-D Information Modeling (IJ3DIM) 7(2), 1-34. https://dx.doi.org/10.4018/IJ3DIM.2018040101
- Kutzner, T., Chaturvedi, K. & Kolbe, T. H., 2020: **CityGML 3.0: New Functions Open Up New Applications**. PFG - Journal of Photogrammetry, Remote Sensing and Geoinformation Science, 88, 43–61. https://doi.org/10.1007/s41064-020-00095-z
- Labetski, A., van Gerwen, S., Tamminga, G., Ledoux, H., Stoter, J., 2018: **A proposal for an improved transportation model in CityGML**. In: Proceedings of the 13th 3D GeoInfo Conference 2018, ISPRS Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. XLII-4/W10, 89–96. https://doi.org/10.5194/isprs-archives-XLII-4-W10-89-2018
- Liu, Ling and Özsu, M. Tamer, eds.: **Encyclopedia of Database Systems**. New York, NY :

Springer New York, 2018. ISBN 978-1-4614-8266-6. Available from: 10.1007/978-1-4614-8265-9

- Löwner, M.-O., Gröger, G., Benner, J., Biljecki, F., Nagel, C., 2016: **Proposal for a new LOD and multi-representation concept for CityGML**. In: Proceedings of the 11th 3D Geoinfo Conference 2016, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. IV-2/W1, 3–12. https://doi.org/10.5194/isprs-annals-IV-2-W1-3-2016
- Nouvel, R., Bahu, J. M., Kaden, R., Kaempf, J., Cipriano, P., Lauster, M., Haefele, K.-H., Munoz, E., Tournaire, O, Casper, E., 2015: **Development of the CityGML Application Domain Extension Energy for Urban Energy Simulation**. In: Proceedings of Building Simulation 2015 - 14th Conference of the International Building Performance Simulation Association, IBPSA, 559-564. http://www.ibpsa.org/proceedings/BS2015/p2863.pdf
- Pédrinis, F., Morel, M., Gesquière, G., 2015. Change Detection of Cities. In: Breunig, M., Al-Doori, M., Butwilowski, E., Kuper, P., Benner, J., Haefele, K.-H. (eds.): 3D Geoinformation Science. Lecture Notes in Geoinformation and Cartography. Springer, Cham. 123–139. https://doi.org/10.1007/978-3-319-12181-9_8
- Redweik, R., Becker, T., 2015. Change Detection in CityGML Documents. In: Breunig, M., Al-Doori, M., Butwilowski, E., Kuper, P., Benner, J., Haefele, K.-H. (eds.): 3D Geoinformation Science. Lecture Notes in Geoinformation and Cartography. Springer, Cham. 107–121. https://doi.org/10.1007/978-3-319-12181-9_7
- Smith, B., Varzi, A. C., 2000: **Fiat and Bona Fide Boundaries**. Philosophy and Phenomenological Research, Vol. 60, No. 2, 401-420. https://doi.org/10.2307/2653492
- Snodgrass, Richard T: **Developing time-oriented database applications in SQL**. San Francisco, California : Morgan Kaufmann Publishers, July 1999. ISBN 1-55860-436-7. Available from: http://www.cs.arizona.edu/rts/tdbbook.pdf[http://www.cs.arizona.edu/rts/tdbbook.pdf]
- Stadler, A., Kolbe, T. H., 2007: **Spatio-semantic Coherence in the Integration of 3D City Models**. In: Proceedings of the 5th International ISPRS Symposium on Spatial Data Quality ISSDQ 2007 in Enschede. http://www.isprs.org/proceedings/XXXVI/2-C43/Session1/paper_Stadler.pdf
- Vretanos, P. A. 2010: **OpenGIS Web Feature Service 2.0 Interface Standard**, Open Geospatial Consortium. OGC Doc. 09-025r1
- OASIS MQTT Technical Committee: **MQTT Version 5.0 Standard**, OASIS, March 7, 2019, Available from OASIS.
- Reed, C., Belayneh T.: **OGC Indexed 3d Scene Layer (I3S) and Scene Layer Package Format Specification**, Open Geospatial Consortium, Available from OGC Doc. 17-014r7
- [[3dtiles_citation, OGC 3D Tiles]]Cozzi, P., Lilley, S., Getz, G. **OGC 3D Tiles Specification 1.0** Open Geospatial Consortium, Available from OGC Doc. 18-053r2
- Burggraf, D.: **OGC KML 2.3**, Open Geospatial Consortium, Available from OGC Doc. 12-007r2
- Bröring, A., Stasch, C., Echterhoff, J.: **OGC® Sensor Observation Service Interface Standard**, Open Geospatial Consortium, Available from OGC Doc. 12-006
- Liang, S., Huang, C., Khalafbeigi, T.: **OGC SensorThings API Part 1: Sensing**, Open Geospatial Consortium, Available from OGC Doc. 15-078r6
- [[3dps_citation, OGC 3D Portrayal Service]]Hagedorn, B., Thum, S., Reitz, T., Coors, V., Gutbell, R.: **OGC® 3D Portrayal Service 1.0**, Open Geospatial Consortium, Available from OGC Doc. 15-

001r3.

- Bhatia, S.,Cozzi, P., Knyazev, A., Parisi, T.: **The GL Transmission Format (glTF)**, The Khronos Group, Available from https://www.khronos.org/gltf.