Open Geospatial Consortium

Submission Date: <yyyy-mm-dd>

Approval Date: <yyyy-mm-dd>

Publication Date: <2020-12-22>

External identifier of this OGC® document: http://www.opengis.net/doc/EG/CityGML/3.0

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

Version: 0.1

Category: OGC® Engineering Guidance

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 Engineering Guidance on the use of the PGC 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, Engineering Guidance should not be referenced as required or mandatory technology in procurements.

Document type:	OGC®Engineering Guidance
Document subtype:	
Document stage:	Draft
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 fundation 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.

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.

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 visualisation 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 modelling 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 securi-ty, 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 fundation 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 generalisation 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 differenti-ations can be represented, CityGML enables lossless information exchange between different GI systems and users.

Chapter 2. 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 3. References

The following normative 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 4. 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]

Abbreviated Terms

The following abbreviated terms are used in this document:

The list of acronyms needs to be reviewed once all sections have been updated.

- 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
- 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
- ISO International Organization for Standardisation
- LOD Level of Detail
- 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
- TC211 ISO Technical Committee 211
- TIC Terrain Intersection Curve
- TIN Triangulated Irregular Network
- UML Unified Modeling Language
- URI Uniform Resource Identifier
- UUID Universally Unique 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

Chapter 5. General Principles

5.1. General Modeling Principles

Contributors
TBD

NOTE This content was copied from Section 7.2 of the Standard.

5.1.1. 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 modelled 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.

5.1.2. 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.

5.1.3. 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 generalisation 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.

5.2. Spatial-Temporal Fundamentals

Contributors
TBD

NOTE Still contains GML-specific content

Spatial properties of CityGML features are represented by objects of GML3's geometry model. This model is based on the standard ISO 19107 'Spatial Schema' (Herring 2001), representing 3D geometry according to the well-known Boundary Representation (cf. Foley et al. 1995). CityGML actually uses only a subset of the GML3 geometry package, defining a profile of GML3. This subset is depicted in Fig. 9 and Fig. 10. Furthermore, GML3's explicit Boundary Representation is extended by scene graph concepts, which allow the representation of the geometry of features with the same shape implicitly and thus more space efficiently (see Implicit geometries, prototypic objects, scene graph concepts).

5.2.1. Geometric-topological model

The geometry model of GML3 consists of primitives, which may be combined to form complexes, composite geometries or aggregates. For each dimension, there is a geometrical primitive: a zero-dimensional object is a Point, a one-dimensional a _Curve, a two-dimensional a _Surface, and a three-dimensional a _Solid (Fig. 9). Each geometry can have its own coordinate reference system. A solid is bounded by surfaces and a surface by curves. In CityGML, a curve is restricted to be a straight line, thus only the GML3 class LineString is used. Surfaces in CityGML are represented by Polygons, which define a planar geometry, i.e. the boundary and all interior points are required to be located in one single plane.

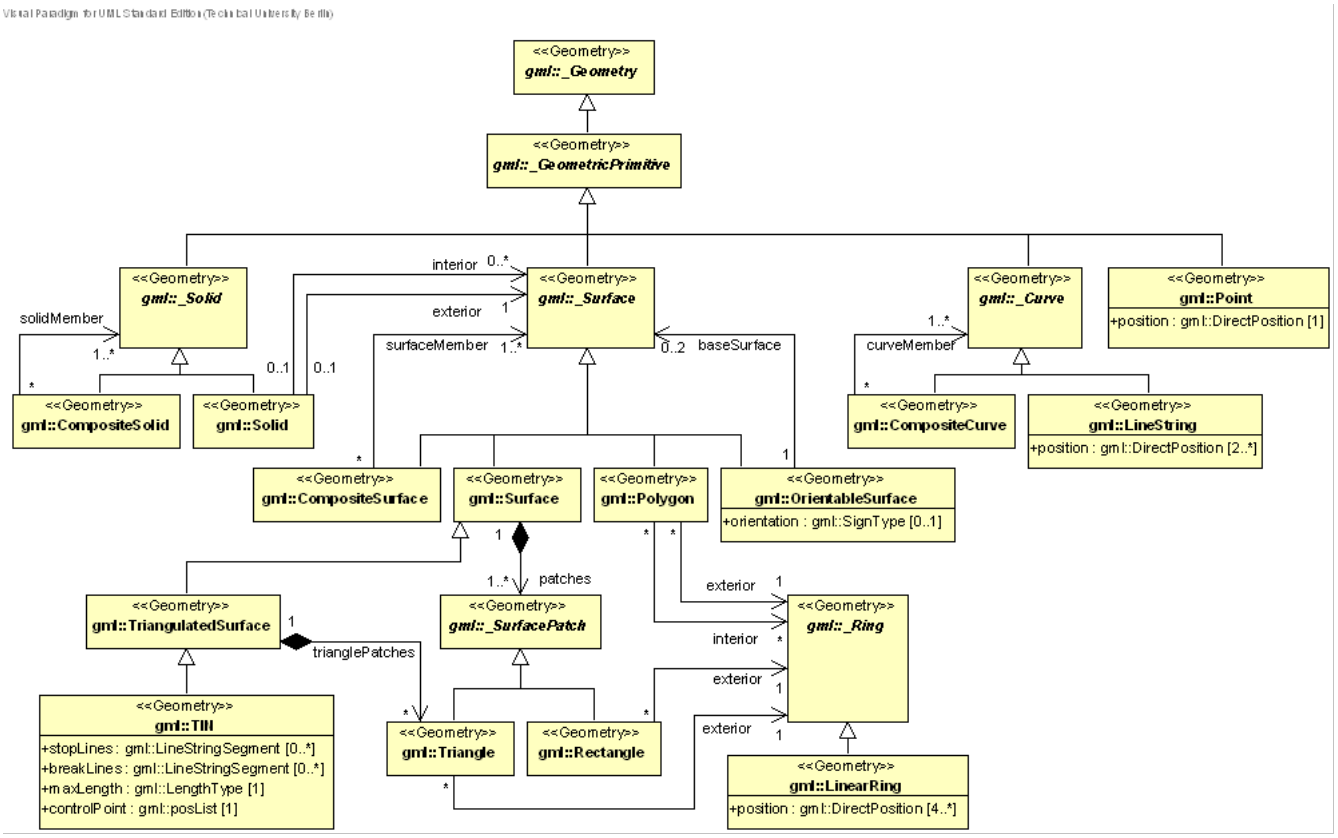


Figure 1. UML diagram of CityGML's geometry model (subset and profile of GML3): Primitives and Composites.

Combined geometries can be aggregates, complexes or composites of primitives (see illustration in Fig. 11). For an aggregate, the spatial relationship between components is not restricted. They may be disjoint, overlapping, touching, or disconnected. GML3 provides a special aggregate for each dimension, a MultiPoint, a MultiCurve, a MultiSurface, and a MultiSolid (see Fig. 10). In contrast to aggregates, a complex is topologically structured: its parts must be disjoint, must not overlap and are allowed to touch, at most, at their boundaries or share parts of their boundaries. A composite is a special complex provided by GML3. It can only contain elements of the same dimension. Its elements must be disjoint as well, but they must be topologically connected along their boundaries. A composite can be a CompositeSolid, a CompositeSurface, or CompositeCurve. (cf. Fig. 9).

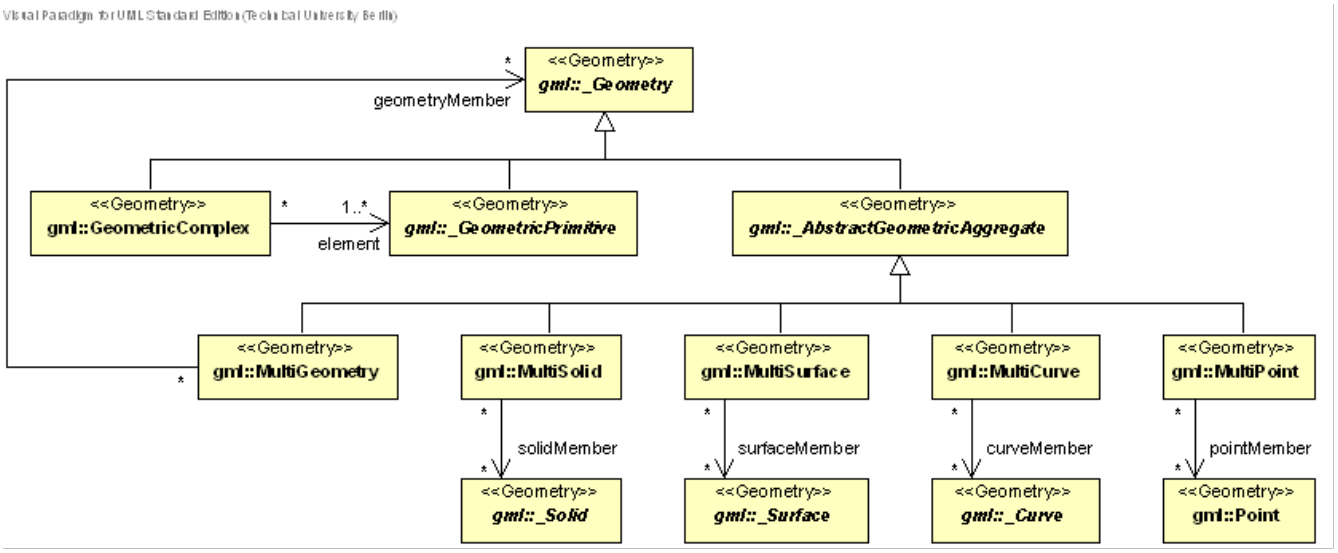


Figure 2. UML diagram of CityGML's geometry model: Complexes and Aggregates

An OrientableSurface is a surface with an explicit orientation, i.e. two sides, front and back, can be distinguished. This may be used to assign textures to specific sides of a surface, or to distinguish the exterior and the interior side of a surface when bounding a solid. Please note, that curves and surfaces have a default orientation in GML which results from the order of the defining points. Thus, an OrientableSurface only has to be used, if the orientation of a given GML geometry has to be reversed.

TriangulatedSurfaces are special surfaces, which specify triangulated irregular networks often used to represent the terrain. While a TriangulatedSurface is a composition of explicit Triangles, the subclass TIN is used to represent a triangulation in an implicit way by a set of control points, defining the nodes of the triangles. The triangulation may be reconstructed using standard triangulation methods (Delaunay triangulation). In addition, break lines and stop lines define contour characteristics of the terrain.

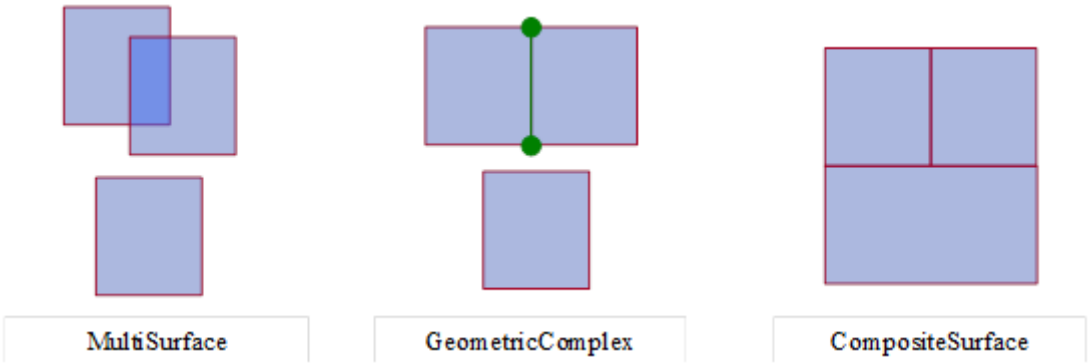


Figure 3. Combined geometries

The GML3 composite model realises a recursive aggregation schema 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).

CityGML provides the explicit modelling of topology, for example the sharing of geometry objects between features or other geometries. One part of space is represented only once by a geometry object and is reference by all features or more complex geometries which are defined or bounded by this geometry object. Thus redundancy is avoided and explicit topological relations between parts are maintained. Basically, there are three cases. First, two features may be defined spatially by the same geometry. For example, if a path is both a transportation feature and a vegetation feature, the surface geometry defining the path is referenced both by the transportation object and by the vegetation object. Second, geometry may be shared between a feature and another geometry. 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 Fig. 12, this requires partitioning of the respective surfaces. In general, Boundary Representation 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. However, the use of topology in

CityGML is optional.

NOTE The following paragraph uses XML techniques.

In order to implement topology, CityGML uses the XML concept of XLinks provided by GML. Each geometry object that should be shared by different geometric aggregates or different thematic features is assigned an unique identifier, which may be referenced by a GML geometry property using a href attribute. CityGML does not deploy the built-in topology package of GML3, which provides separate topology objects accompanying the geometry. This kind of topology is very complex and elaborate. Nevertheless, it lacks flexibility when data sets, which might include or neglect topology, should be covered by the same data model. The XLink topology is simple and flexible and nearly as powerful as the explicit GML3 topology model. However, a disadvantage of the XLink topology is that navigation between topologically connected objects can only be performed in one direction (from an aggregate to its components), not (immediately) bidirectional as it is the case for GML's built-in topology.

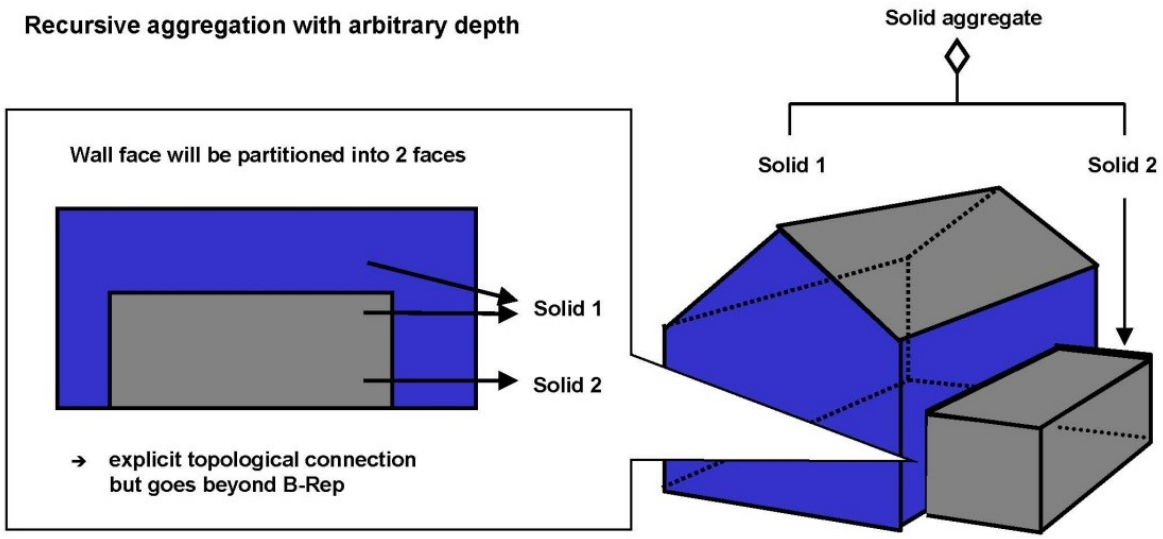


Figure 4. Recursive aggregation of objects and geometries in CityGML (graphic: IGG Uni Bonn).

The following excerpt of a CityGML example file defines a gml:Polygon with a gml:id wallSurface4711, which is part of the geometry property lod2Solid of a building. Another building being adjacent to the first building references this shared polygon in its geometry representation.

NOTE replace GML? Or retain it as an example.

5.2.2. Spatial reference system

When dealing with geoinformation and virtual 3D city models in particular, the exact spatial reference is of utmost importance and a key requirement for the integration of different spatial datasets in a single 3D city model. CityGML inherits GML3's spatial capabilities of handling Coordinate Reference Systems (CRS) which is the usual way of denoting the spatial reference in GML 3.1.1. As CityGML is a true 3D standard, geometry elements are associated with a 3D CRS. There are only few exceptions to this rule where CityGML allows a 2D geometry element.

In general, a geometry may point to the CRS definition used by this geometry through the attribute srsName which is inherited from the abstract GML superclass gml:_Geometry. This may be a reference to a well-known CRS definition provided by an authority organization such as the European Petroleum Survey Group (EPSG), but may also be a pointer to a CRS that is locally defined within the same CityGML instance document. The OGC document "Definition identifier URNs in OGC namespace" (cf. Whiteside 2009; OGC Doc. No. 07-092r3) provides best practices for the URN encoding of CRS references. Amongst others, it describes how to reference a single well-known 3D CRS definition (such as a 3D geographic CRS) as well as a compound CRS which combines two or more well-known CRS definitions (e.g., a projected CRS for the planimetry with a vertical CRS for the height reference). Examples for denoting a compound CRS for a CityGML instance document are given in Annex G.

GML3 also supports the definition of engineering CRSs which are used in a contextually local sense. For example, this might be a local 3D Cartesian coordinate system that is essentially based on a flat-earth approximation of the earth's surface, and thus ignores the effect of earth curvature on feature geometry (cf. chapter 12.1.4.4 of the GML 3.1.1 specification document). Local engineering CRSs are commonly applied in the AEC/FM domain and thus are useful when integrating CAD data or BIM models into a 3D city model. Annex G.9 provides an example demonstrating the definition of an engineering CRS within a CityGML instance document and the use of local coordinate values for the feature geometry. The definition of an engineering CRS requires an anchor point which relates the origin of the local coordinate system to a point on the earth's surface in order to facilitate the transformation of coordinates from the local engineering CRS.

According to GML 3.1.1, if no srsName attribute is given on a geometry element, then the CRS shall be specified as part of the larger context this geometry element is part of, e.g. a geometric aggregate. For convenience in constructing feature and feature collection instances, the value of the srsName attribute on the gml:Envelope (or gml:Box) which is the value of the gml:boundedBy property of the feature shall be inherited by all directly expressed geometries in all properties of the feature or members of the collection, unless overruled by the pres-ence of a local srsName. Thus it is not necessary for a geometry to carry an srsName attribute if it uses the same CRS as given on the gml:boundedBy property of its parent feature. Inheritance of the CRS continues to any depth of nesting, but if overruled by a local srsName declaration, then the new CRS is inherited by all its chil-dren in turn (cf. chapter 8.3 of the GML 3.1.1 specification document).

It is strongly recommended that any CityGML instance document explicitly specifies the CRS for all contained geometry elements. This is especially important if the instance document is to be exchanged externally with third parties or is to be integrated with other spatial datasets. A mixed usage of different CRSs within the same dataset is possible and conformant with GML 3.1.1, whereas a single CRS reference given on the embracing CityModel feature collection (cf. chapter 10.1) simplifies the processing of the dataset by software systems. As for CityGML 2.0, this

recommendation is non-normative and thus not accompanied by a conformance class. The main reason for this is to maintain backwards compatibility with CityGML 1.0.

5.2.3. Implicit geometries, prototypic objects, scene graph concepts

The concept of implicit geometries is an enhancement of the geometry model of GML3. It is, for example, used in CityGML's building, bridge, tunnel, and vegetation model as well as for city furniture and generic objects. Implicit geometries may be applied to features from different thematic fields of CityGML in order to geometrically represent the features within a specific level of detail (LOD). Thus, each extension module may define spatial properties providing implicit geometries for its thematic classes. For this reason, the concept of implicit geometries is defined within the CityGML core module (cf. chapter 10.1). However, its description is drawn here since implicit geometries are part of CityGML's spatial model. The UML diagram is depicted in Fig. 13. The corresponding XML schema definition is provided in annex A.1.

An implicit geometry is a geometric object, where the shape is stored only once as a prototypical geometry, for example a tree or other vegetation objects, a traffic light or a traffic sign. This prototypic geometry object is re-used or referenced many times, wherever the corresponding feature occurs in the 3D city model. Each occurrence is represented by a link to the prototypic shape geometry (in a local cartesian coordinate system), by a transformation matrix that is multiplied with each 3D coordinate of the prototype, and by an anchor point denoting the base point of the object in the world coordinate reference system. This reference point also defines the CRS to which the world coordinates belong after the application of the transformation. In order to determine the absolute coordinates of an implicit geometry, the anchor point coordinates have to be added to the matrix multiplication results. The transformation matrix accounts for the intended rotation, scaling, and local translation of the prototype. It is a 4x4 matrix that is multiplied with the prototype coordinates using homogeneous coordinates, i.e. (x,y,z,1). This way even a projection might be modeled by the transformation matrix.

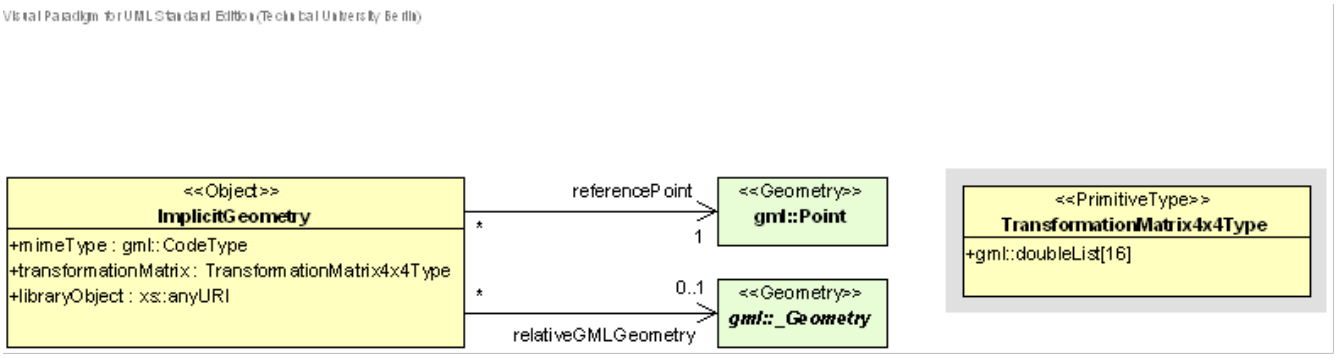


Figure 5. UML diagram of ImplicitGeometries. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Core module.

The reason for using the concept of implicit geometries in CityGML is space efficiency. Since the shape of, for example, trees of the same species can be treated as identical, it would be inefficient to model the detailed geometry of each of the large number of trees explicitly. The concept of implicit geometries is similar to the well known concept of primitive instancing used for the representation of scene graphs in the field of computer graphics (Foley et al. 1995).

The term implicit geometry refers to the principle that a geometry object with a complex shape can be simply represented by a base point and a transformation, implicitly unfolding the object's shape

at a specific location in the world coordinate system.

The shape of an ImplicitGeometry can be represented in an external file with a proprietary format, e.g. a VRML file, a DXF file, or a 3D Studio MAX file. The reference to the implicit geometry can be specified by an URI pointing to a local or remote file, or even to an appropriate web service. Alternatively, the shape can be defined by a GML3 geometry object. This has the advantage that it can be stored or exchanged inline within the CityGML dataset. Typically, the shape of the geometry is defined in a local coordinate system where the origin lies within or near to the object's extent. If the shape is referenced by an URI, also the MIME type of the denoted object has to be specified (e.g. "model/vrml" for VRML models or "model/x3d+xml" for X3D models).

The implicit representation of 3D object geometry has some advantages compared to the explicit modeling, which represents the objects using absolute world coordinates. It is more space-efficient, and thus more extensive scenes can be stored or handled by a system. The visualisation is accelerated since 3D graphics cards support the scene graph concept. Furthermore, the usage of different shape versions of objects is facilitated, e.g. different seasons, since only the library objects have to be exchanged (see example in Fig. 65).

NOTE	include GML example "ImplicitGeometryType, ImplicitRepresentationPropertyType" or replace with UML

Example CityGML datasets

An example for an implicit geometry is given by the following city furniture object (cf. chapter 10.9), which is represented by a geometry in LOD2:

```
  <frn:CityFurniture>
    <!-- class "traffic"; as specified in the code list proposed by the SIG 3D (cf.
annex C.4) -->
    <frn:class
codeSpace="http://www.sig3d.org/codelists/standard/cityfurniture/2.0/CityFurniture_cla
ss.xml">1000</frn:class>
    <!-- function "traffic light"; as specified in the code list proposed by the SIG 3D
(cf. annex C.4) -->
    <frn:function
codeSpace="http://www.sig3d.org/codelists/standard/cityfurniture/2.0/CityFurniture_fun
ction.xml">1080</frn:function>
    <frn:lod2ImplicitRepresentation>
      <core:ImplicitGeometry>
        <core:mimeType>model/vrml</core:mimeType>
        <core:libraryObject>
          http://www.some-3d-library.com/3D/furnitures/TrafficLight434.wrl
        </core:libraryObject>
        <core:referencePoint>
          <gml:Point srsName="urn:ogc:def:crs,crs:EPSG:6.12:31467,crs:EPSG:6.12:5783">
            <gml:pos srsDimension="3">5793898.77 3603845.54 44.8</gml:pos>
          </gml:Point>
        </core:referencePoint>
      </core:ImplicitGeometry>
    </frn:lod2ImplicitRepresentation>
  </frn:CityFurniture>
```

The shape of the geometry of the traffic light (city furniture with class “1000” and function “1080” according to the code lists proposed in annex C.4) is defined by a VRML file which is specified by a URL. This library object, which is defined in a local coordinate system, is transformed to its actual location by adding the coordi-nates of the reference point.

The following clip of a CityGML file provides a more complex example for an implicit geometry:

```
  <frn:CityFurniture>
    <!-- class "traffic"; as specified in the code list proposed by the SIG 3D (cf.
annex C.4) -->
    <frn:class>1000</frn:class>
    <!-- function "traffic light"; as specified in the code list proposed by the SIG 3D
(cf. annex C.4) -->
    <frn:function>1080</frn:function>
    <frn:lod2ImplicitRepresentation>
      <core:ImplicitGeometry>
        <core:mimeType>model/vrml</core:mimeType>
        <core:transformationMatrix>
          0.866025 -0.5 0 0.7
          0.5 0.866025 0 0.8
          0 0 1 0
          0 0 0 1
        </core:transformationMatrix>
        <core:libraryObject>
          http://www.some-3d-library.com/3D/furnitures/TrafficLight434.wrl
        </core:libraryObject>
        <core:referencePoint>
          <gml:Point srsName="urn:ogc:def:crs,crs:EPSG:6.12:31467,crs:EPSG:6.12:5783">
            <gml:pos srsDimension="3">5793898.77 3603845.54 44.8</gml:pos>
          </gml:Point>
        </core:referencePoint>
      </core:ImplicitGeometry>
    </frn:lod2ImplicitRepresentation>
  </frn:CityFurniture>
```

In addition to the first example, a transformation matrix is specified. It is a homogeneous matrix, serialized in a row major fashion, i.e. the first four entries in the list denote the first row of the matrix, etc. The matrix combines a translation by the vector (0.7, 0.8, 0) – the origin of the local reference system is not the center of the object – and a rotation around the z-axis by 30 degrees (cos(30) = 0.866025 and sin(30) = 0.5). This rotation is necessary to align the traffic light with respect to a road. The actual position of the traffic light is computed as follows:

1. each point of the VRML file (with homogeneous coordinates) is multiplied by the transformation matrix;
2. for each resulting point, the reference point (5793898.77, 3603845.54, 44.8, 1)T is added, yielding the actual geometry of the city furniture.

5.3. Metadata

Contributors
Chuck Heazel

CityGML version 2.0 provided an explicit element that implementors could use to insert metadata. This element was inherited from the GML classes upon which version 2.0 was based. Version 3.0 of

CityGML is no longer based on GML. It does not explicitly support that element. The result is that version 3.0 of the CityGML Conceptual Model is agnostic of metadata.

That does not mean that metadata cannot be supported. The CityGML extension mechanisms, specifically ADEs and Generics, can be used for metadata elements. An advantage to this approach is that metadata can be added where it is needed. It provides a degree of flexibility which is absent the dedicated element approach.

Another alternative is to insert metadata elements through the Implementation Specification. For example, an Implementation Specification which builds on GML would inherit the same metadata element as CityGML 2.0.

The CityGML Standards Working Group will continue the metadata discussion. Additional metadata-related requirements may be added to future versions of this Users Guide as they are identified and validated.

5.4. Modularization

Contributors
TBD

CityGML is a rich standard both on the thematic and geometric-topological level of its data model. On its thematic level CityGML defines classes and relations for the most relevant topographic objects in cities and regional models comprising built structures, elevation, vegetation, water bodies, city furniture, and more. In addition to geometry and appearance content these thematic components allow to employ virtual 3D city models for sophisticated analysis tasks in different application domains like simulations, urban data mining, facility management, and thematic inquiries.

CityGML is to be seen as a framework giving geospatial 3D data enough space to grow in geometrical, topological and semantic aspects over its lifetime. Thus, geometry and semantics of city objects may be flexibly structured covering purely geometric datasets up to complex geometric-topologically sound and spatio-semantically coherent data. By this means, CityGML defines a single object model and data exchange format applicable to consecutive process steps of 3D city modelling from geometry acquisition, data qualification and refinement to preparation of data for specific end-user applications, allowing for iterative data enrichment and lossless information exchange (cf. Kolbe et al. 2009).

According to this idea of a framework, applications are not required to support all thematic fields of CityGML in order to be compliant to the standard, but may employ a subset of constructs corresponding to specific relevant requirements of an application domain or process step. The use of logical subsets of CityGML limits the complexity of the overall data model and explicitly allows for valid partial implementations. As for version 2.0 of the CityGML standard, possible subsets of the data model are defined and embraced by so called CityGML modules. A CityGML module is an aggregate of normative aspects that must all be implemented as a whole by a conformant system. CityGML consists of a core module and thematic extension modules.

The CityGML core module defines the basic concepts and components of the CityGML data model. It

is to be seen as the universal lower bound of the overall CityGML data model and a dependency of all thematic extension modules. Thus, the core module is unique and must be implemented by any conformant system. Based on the CityGML core module, each extension module contains a logically separate thematic component of the CityGML data model. The extensions to the core are derived by vertically slicing the overall CityGML data model. Since the core module is contained (by reference) in each extension module, its general concepts and components are universal to all extension modules. The following thirteen thematic extension modules each cover the corresponding thematic field of CityGML:

- Appearance,
- City Furniture,
- City Object Group,
- Construction,
- Dynamizer,
- Generics,
- Land Use,
- Point Cloud,
- Relief,
- Transportation,
- Vegetation,
- Versioning, and
- Water Body

The Construction module is further extended by three additional thematic extension modules:

- Bridge,
- Building, and
- Tunnel

The thematic decomposition of the CityGML data model allows for implementations to support any combination of extension modules in conjunction with the core module in order to be CityGML conformant. Thus, the extension modules may be arbitrarily combined according to the information needs of an application or application domain. A combination of modules is called a CityGML profile. The union of all modules is defined as the CityGML base profile. The base profile is unique at any given time and forms the upper bound of the overall CityGML data model. Any other CityGML profile must be a valid subset of the base profile. By following the concept of CityGML modules and profiles, valid partial implementations of the CityGML data model may be realised in a well-defined way.

As for future development, each CityGML module may be further developed independently from other modules by expert groups and information communities. Resulting proposals and changes to modules may be introduced into future revisions of the CityGML standard without affecting the validity of other modules. Furthermore, thematic components not covered by the current CityGML

data model may be added to future revisions of the standard by additional thematic extension modules. These additional extensions may establish dependency relations to any other existing CityGML module but shall at least be dependent on the CityGML core module. Consequently, the CityGML base profile may vary over time as new extensions are added. However, if a specific application has information needs to be modelled and exchanged which are beyond the scope of the CityGML data model, this application data can also be incorporated within the existing modules using CityGML's Application Domain Extension mechanism or by employing the concepts of generic city objects and attributes.

The introduced modularisation approach supports CityGML's versatility as a data modelling framework and exchange format addressing various application domains and different steps of 3D city modelling. For sake of clarity, applications should announce the level of conformance to the CityGML standard by declaring the em-ployed CityGML profile. Since the core module is part of all profiles, this should be realised by enumerating the implemented thematic extension modules. For example, if an implementation supports the Building module, the Relief module, and the Vegetation module in addition to the core, this should be announced by "CityGML [Building, Relief, Vegetation]". In case the base profile is supported, this should be indicated by "CityGML [full]".

5.5. CityGML Extension Mechanisms

Contributors
TBD

NOTE The following text needs to be reviewed and updated.

CityGML defines a rich and general-purpose information model for 3D city and landscape models. While CityGML provides a common basis for a multitude of different use cases and applications, specific applications might have modelling requirements that go beyond the predefined elements of the CityGML conceptual model.

CityGML affords this flexibility by providing a slim base model with the following distinct extension mechanisms:

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

These extension mechanisms have been introduced with CityGML 1.0 and are since widely used and accepted in the CityGML community.

5.5.1. Code Lists

When an attribute may have only one value selected from a small and fixed set of values, CityGML 3.0 specifies those as an enumeration. When the set of possible values is a priori unknown but predefined and fixed for a specific application domain or user organization, the values may be modelled as classes with the stereotype «CodeList».

Many attributes of CityGML types use a code list as 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 to the value domain become possible with code lists.

The feasible values for attributes with code lists may substantially vary for different applications, uses cases, information communities or even countries (e.g., due to national law or regulations). For this reason, code lists are modelled as empty classes without predefined normative content. The governance of code lists is rather decoupled from the governance of the CityGML conceptual model, and the contents may be defined and managed outside this International Standard by any organization or information community according to their information needs. Code lists should be made available as publicly accessible resource, either globally or within an application domain.

Rules and constraints for the encoding of both code lists and attributes having a code list as value are not subject of this document but are defined in a corresponding CityGML Encoding Specification. It is recommended, though, that attributes should be encoded such that they can take a value from a code list and, optionally, provide an identifier that references the code list in a unique way.

Next

CityGML feature types often include attributes whose values can be enumerated in a list of discrete values. An example is the attribute roof type of a building, whose attribute values typically are saddle back roof, hip roof, semi-hip roof, flat roof, pent roof, or tent roof. If such an attribute is typed as string, misspellings or different names for the same notion obstruct interoperability.

If the list of values is fixed, the allowed attribute values are specified in and enforced by the CityGML schema using an enumeration as attribute type. Attributes of an enumerated type may only take values from the prede-fined list. Examples for such attributes are relativeToTerrain and relativeToWater of the abstract base class core:_CityObject (CityGML Core module, cf. chapter 10.1) as well as wrapMode of the abstract class app:_Texture (Appearance module, cf. chapter 9.4).

In case a fixed enumeration of possible attribute values is not suitable, the attribute type is specified as gml:CodeType and the allowed attribute values can be provided in a code list which is specified outside the CityGML schema. Examples for such attributes are class, function, and usage which are available for almost all CityGML feature types. A code list contains coded attribute values which hinder misspellings and ensure that the same code is used for the same notion or concept. If a code list is provided for an attribute of type gml:CodeType, then any conformant attribute shall only take values from the given code list. This allows appli-cations to validate the attribute values and thus facilitates semantic and syntactic interoperability. The optional codeSpace attribute declared for gml:CodeType is used to associate an attribute with a code list. If a codeSpace is present, then its value shall be a persistent URI identifying the code list. If no codeSpace is given, then the attribute value can only be interpreted as a simple text token and validation requires additional knowledge.

The governance of code lists is decoupled from the governance of the CityGML schema and specification. Accordingly, code lists can be specified by any organisation or information community according to their infor-mation needs. There shall be one authority per codeSpace and hence per code list who is in charge of the con-tents and the maintenance of the code list. As a

result, the code list values are managed outside the CityGML schema. Thus, in contrast to a fixed enumeration enforced by the CityGML schema, changes to a code list do not require a revision of the CityGML schema and specification.

The contents of code lists may substantially vary for different countries (e.g., due to national law or regulations) and for different information communities. For this reason, this International standard does not specify norma-tive code lists for any of the attributes of type gml:CodeType. However, Annex C provides non-normative code lists for selected attributes which are proposed and maintained by the SIG 3D. These code lists can be directly referenced in CityGML instance documents and serve as an example for the definition of code lists. The code lists given in Annex C comprise the non-normative code lists which are included in the previous version 1.0 of this International standard in order to ensure backwards compatibility.

It is recommended that code lists are implemented as simple dictionaries following the GML 3.1.1 Simple Dictionary Profile (cf. Whiteside 2005). An example for a code list implemented as simple dictionary is given below. It shows an excerpt of the code list proposed by the SIG 3D for the attribute roofType of the class _AbstractBuilding (Building module, cf. chapter 10.3).

NOTE Issue - should we use GML in the examples?

```
<gml:Dictionary xmlns:gml="http://www.opengis.net/gml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/gml
http://schemas.opengis.net/gml/3.1.1/profiles/SimpleDictionary/1.0.0/gmlSimpleDictiona
ryProfile.xsd" gml:id="roofType">
  <gml:name>roofType</gml:name>
  <gml:dictionaryEntry>
    <gml:Definition gml:id="id357">
      <gml:description>flat roof</gml:description>
      <gml:name>1000</gml:name>
    </gml:Definition>
  </gml:dictionaryEntry>
  <gml:dictionaryEntry>
    <gml:Definition gml:id="id358">
      <gml:description>monopitch roof</gml:description>
      <gml:name>1010</gml:name>
    </gml:Definition>
  </gml:dictionaryEntry>
  …
</gml:Dictionary>
```

Example of a code list implemented as simple dictionary following the GML 3.1.1 Simple Dictionary Profile.

In the simple dictionary concept, the code list itself is represented by a gml:Dictionary element. The allowed attribute values are listed as gml:Definition entries contained in the gml:Dictionary. For each definition entry, the coded attribute value is specified by the gml:name subelement. Any attribute referencing this code list in a CityGML instance document may only take values which are specified by a gml:name element of one of the definition entries. If the attribute value is not specified by one of the definition entries, then the attribute value is invalid. The gml:description

subelement of a definition entry provides an additional textual description for the coded attribute value. This description can be used, for example, as human readable substitute for the coded attribute value.

The following excerpt of a CityGML instance document illustrates the usage of the code list mechanism. The document contains a bldg:Building object whose roofType value is taken from the code list shown in Listing 2. The codeSpace attribute of the roofType element identifies the code list through the globally unique URL http://www.sig3d.org/codelists/standard/building/2.0/_AbstractBuilding_roofType.xml which is managed and maintained by the SIG 3D. According to this code list, the coded attribute value 1000 denotes a flat roof for this building.

```
<bldg:Building>
  <bldg:roofType

codeSpace="http://www.sig3d.org/codelists/standard/building/2.0/_AbstractBuilding_roof
Type.xml">1000</bldg:roofType>
  ...
</bldg:Building>
```

5.5.2. Generic Objects and Attributes

NOTE Fix link to generics section of the standard.

Generic objects and attributes are available from the Generics module of the CityGML Conceptual Model (cf. [rc_generics_section]). A generic object is intended to be used as proxy for a real-world feature that is not mapped by a specific class in CityGML. Depending on the type and geometric characteristics of the real-world feature, CityGML offers the proxy classes *GenericLogicalSpace*, *GenericOccupiedSpace*, *GenericUnoccupiedSpace* and *GenericThematicSurface* to capture the feature. Each proxy can have a semantic meaning defined by the attributes *class*, *function*, and *usage*.

Generic attributes are name-value pairs that can be assigned to any city object (i.e., an instance of *core:AbstractCityObject*) to augment it with application data not covered by the predefined attributes. The attribute *name* can be freely chosen to identify the piece of information represented by the generic attribute. A fixed list of simple data types is offered as possible domains for the attribute *value*. Generic attributes can be grouped into named collections using the *GenericAttributeSet* data type.

The main advantage of generic objects and attributes is that they are simple and easy-to-use to represent application-specific content. Since this extension mechanism is built into the conceptual model of CityGML, it provides the capability of ad-hoc data enrichment (“at run-time”) without the need for modifying the conceptual model. This flexibility also faces disadvantages though:

- Generic objects are “flat” and cannot be decomposed into sub-features and feature hierarchies like other CityGML features such as, for instance, buildings or transportation features. However, they may be related to other city objects through the inherited *relatedTo* association.
- Names, data types, and multiplicities of generic attributes cannot be specified in a formal way. Consequently, there is no guarantee for an application that a generic attribute of a specific

name and type is available a minimum or maximum number of times for a given city object.

- Name clashes between generic attributes from different applications are possible and cannot be avoided in a formal way, which might impede semantic interoperability.
- There is only a limited number of predefined simple data types available for generic attributes.

To avoid semantic interoperability issues, generic objects and attributes shall only be used if a more specific feature class or attribute is not available from the CityGML conceptual model.

5.5.3. Application Domain Extension (ADE)

An *Application Domain Extension* (ADE) is a formal and systematic extension of CityGML for a specific application or domain in the form of a conceptual UML model. The application data is mapped to a set of additional classes, attributes, and relations. ADEs may use elements from CityGML, for instance, to derive application-specific subclasses, to inject additional properties, to associate application data with predefined CityGML content, or to define value domains for attributes.

The ADE mechanism allows application-specific information to be aligned with the conceptual model of CityGML in a well-structured and systematic way. By this means, CityGML can be extended to meet the information needs of an application while at the same time preserving its concepts and semantic structures. Moreover, and in contrast to generic city objects and attributes, application data can be validated against the formal definition of an ADE to ensure semantic interoperability.

Previous versions of CityGML defined the ADE mechanism solely on the level of the XML Schema encoding. With CityGML 3.0, ADEs become platform-independent models on a conceptual level that can be mapped to multiple and different target encodings.

ADEs have successfully been implemented in practice and enable a wide range of applications and use cases based on CityGML. An overview and discussion of existing ADEs is provided in [[Biljecki2018]].

NOTE	fix uml notation section reference and Biljecki2018 citation.

General Rules for ADEs

An ADE shall be defined as conceptual model in UML in accordance with the conceptual modelling framework of the ISO 19100 series of International Standards and by adhering to the General Feature Model and the rules and constraints for application schemas as specified in ISO 19109 and ISO/TS 19103. The UML notations and stereotypes used in the CityGML conceptual model should also be applied to corresponding model elements in an ADE.

Every ADE shall be organized into one or more UML packages having globally unique namespaces and containing all UML model elements defined by the ADE. An ADE may additionally import and use predefined classes from external conceptual UML models such as the CityGML modules or the standardized schemas of the ISO 19100 series of International Standards.

Defining New ADE Model Elements

Following ISO 19109, features are the primary view of geospatial information and the core elements of application schemas. ADEs therefore typically extend CityGML by defining new feature types appropriate to the application area together with additional content such as object types, data types, code lists, and enumerations.

Every feature type in an ADE shall be derived either directly or indirectly from the CityGML root feature type *core:AbstractFeature* or, depending on its type and characteristics, from a more appropriate subclass thereof. According to the general space concept of CityGML, features representing spaces or space boundaries shall be derived either directly or indirectly from *core:AbstractSpace* or *core:AbstractSpaceBoundary* respectively. UML classes representing top-level feature types shall use the *«TopLevelFeatureType»* stereotype.

In contrast to feature types, object types and data types are not required to be derived from a predefined CityGML class unless explicitly stated otherwise.

ADE classes may have an unlimited number of attributes and associations in addition to those inherited from their parents. Attributes can be modelled with either simple or complex data types. To ensure semantic interoperability, the predefined types from CityGML or the standardized schemas of the ISO 19100 series of International Standards should be used wherever appropriate. This includes, amongst others, basic types from ISO/TS 19103, geometry and topology objects from ISO 10107, and temporal geometry and topology objects from ISO 19108.

If a predefined type is not available, ADEs can either define their own data types or import data types from external conceptual models. This explicitly includes the possibility to define new geometry types not offered by ISO 19107. Designers of an ADE should however note that software might not be able to properly identify and consume such geometry types.

A feature type capturing a real-world feature with geometry should be derived either directly or indirectly from *core:AbstractSpace* or *core:AbstractSpaceBoundary*. By this means, the predefined spatial properties and the associated LOD concept of CityGML are inherited and available for the feature type. If, however, these superclasses are either inappropriate or lack a spatial property required to represent the feature, an ADE may define new and additional spatial properties. If such a spatial property should belong to one of the predefined LODs, then the property name shall start with the prefix “lod*X*”, where *X* is to be replaced by an integer value between 0 and 3 indicating the target LOD. This enables software to derive the LOD of the geometry.

Constraints on model elements should be expressed using a formal language such as the Object Constraint Language (OCL). The ADE specifies the manner of application of constraints. However, following the CityGML conceptual model, constraints should at least be expressed on ADE subclasses of *core:AbstractSpace* to limit the types of space boundaries (i.e., instances of *core:AbstractSpaceBoundary*) that may be used to model the boundary of a space object.

NOTE	add ADE examples

Illustrative examples for ADEs can be found in the CityGML 3.0 User Guide.

Augmenting CityGML Feature Types with Additional ADE Properties

If a predefined CityGML feature type lacks one or more properties required for a specific application, a feasible solution is to derive a new ADE feature type as subclass of the CityGML class and to add the properties to this subclass. While conceptually clean, this approach also faces drawbacks. If multiple ADEs require additional properties for the same CityGML feature type, this will lead to many subclasses of this feature type in different ADE namespaces. Information about the same real-world feature might therefore be spread over various instances of the different feature classes in an encoding making it difficult for software to consume the feature data.

For this reason, CityGML provides a way to augment the predefined CityGML feature types with additional properties from the ADE domain without the need for subclassing. Each CityGML feature type has an extension attribute of name "adeOf*FeatureTypeName*" and type "*ADEOfFeatureTypeName*", where *FeatureTypeName* is replaced by the class name in which the attribute is defined. For example, the *bldg:Building* class offers the attribute *bldg:adeOfBuilding* of type *bldg:ADEOfBuilding*. Each of these extension attributes can occur zero to unlimited times, and the attribute types are defined as abstract and empty data types.

If an ADE augments a specific CityGML feature type with additional ADE properties, the ADE shall create a subclass of the corresponding abstract data type associated with the feature class. This subclass shall also be defined as data type using the stereotype *«DataType»*. The additional application-specific attributes and associations are then modelled as properties of the ADE subclass. This may include, amongst others, attributes with simple or complex data type, spatial properties or associations to other object and feature types from the ADE or external models such as CityGML.

The predefined "*ADEOfFeatureTypeName*" data types are called "hooks" because they are used as the head of a hierarchy of ADE subclasses attaching application-specific properties. When subclassing the "hook" of a specific CityGML feature type in an ADE, the properties defined in the subclass can be used for that feature type as well as for all directly or indirectly derived feature types, including feature types defined in the same or another ADE.

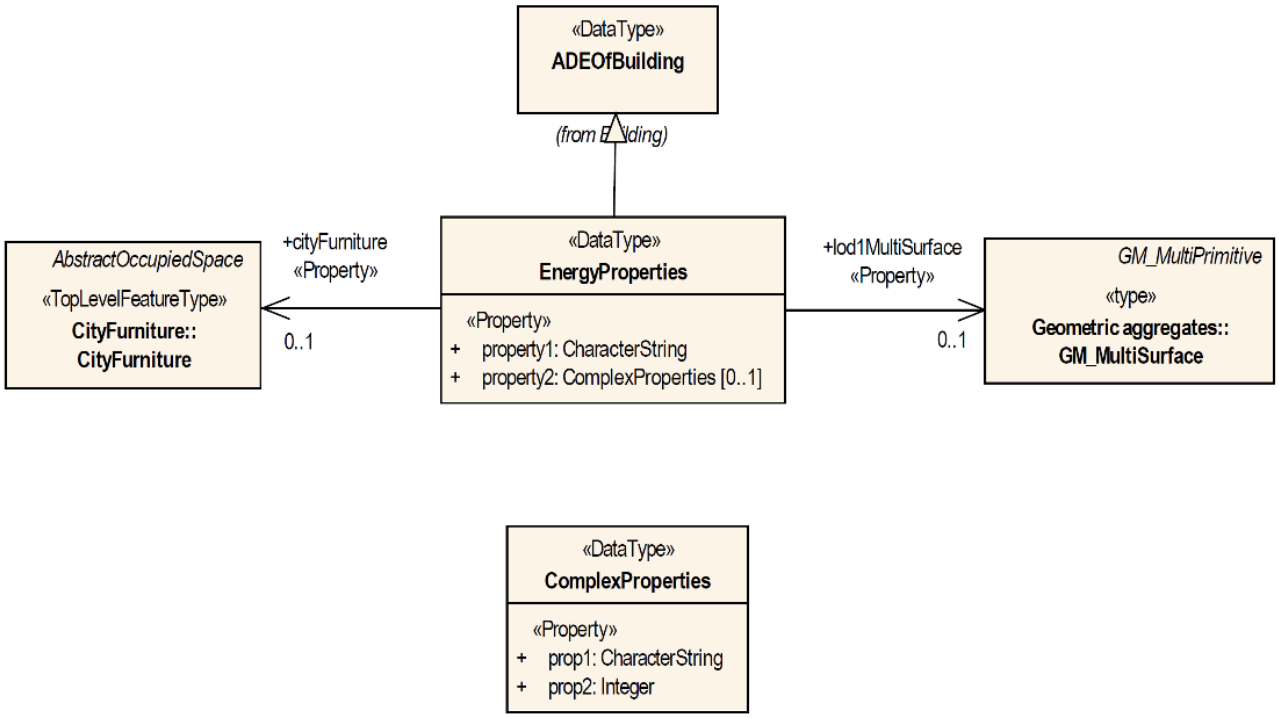
Multiple distinct ADEs can use the "hook" mechanism to define additional ADE properties for the same CityGML feature type. Since the "adeOf*FeatureTypeName*" attribute may occur multiple times, the various ADE properties can be exchanged as part of the same CityGML feature instance in an encoding. Software can therefore easily consume the default CityGML feature data plus the additional properties from the different ADEs.

Content from unknown or unsupported ADEs may be ignored by an application or service consuming an encoded CityGML model.

Designers of an ADE should favor using this "hook" mechanism over subclassing a CityGML feature type when possible. If an ADE must enable other ADEs to augment its own feature types (so-called ADE of an ADE), then it shall implement "hooks" for its feature types following the same schema and naming concept as in the CityGML conceptual model.

NOTE	Reference to the user guide.

The following UML fragment shows an example for using the "hook" mechanism. For more details on this and other example ADEs, please see the CityGML 3.0 User Guide for an example ADE.



Encoding of ADEs

This document only addresses the conceptual modelling of ADEs. Rules and constraints for mapping a conceptual ADE model to a target encoding are expected to be defined in a corresponding CityGML Encoding Standard. If supported, an ADE may provide additional mapping rules and constraints in conformance with a corrresponding CityGML Encoding Standard.

5.6. CityGML profiles

Contributors
TBD

NOTE	Old content which needs to be replaced

A CityGML profile is a combination of thematic extension modules in conjunction with the core module of CityGML. Each CityGML instance document shall employ the CityGML profile appropriate to the provided data. In general, two approaches to employ a CityGML profile within an instance document can be differentiated:

1. CityGML profile definition embedded inline the CityGML instance document A CityGML profile can be bound to an instance document using the schemaLocation attribute defined in the XML Schema instance namespace, http://www.w3.org/2001/XMLSchema-instance (commonly associated with the prefix xsi). The xsi:schemaLocation attribute provides a way to locate the XML Schema definition for namespaces defined in an XML instance document. Its value is a whitespace-delimited list of pairs of Uniform Resource Identifiers (URIs) where each pair consists of a namespace followed by the location of that namespace's XML Schema definition, which is typically a .xsd file.

By this means, the namespaces of the respective CityGML modules shall be defined within a CityGML instance document. The xsi:schemaLocation attribute then shall be used to provide the location to the respective XML Schema definition of each module. All example instance documents given in Annex G follow this first approach.

2. CityGML profile definition provided by a separate XML Schema definition file The CityGML profile may also be specified by its own XML Schema file. This schema file shall com-bine the appropriate CityGML modules by importing the corresponding XML Schema definitions. For this purpose, the import element defined in the XML Schema namespace shall be used, http://www.w3.org/2001/XMLSchema (commonly associated with the prefix xs). For the xs:import el-ement, the namespace of the imported CityGML module along with the location of the namespace's XML Schema definition have to be declared. In order to apply a CityGML profile to an instance docu-ment, the profile's schema has to be bound to the instance document using the xsi:schemaLocation at-tribute. The XML Schema file of the CityGML profile shall not contain any further content.

   The targetNamespace of the profile's schema shall differ from the namespaces of the imported CityGML modules. The namespace associated with the profile should be in control of the originator of the instance document and must be given as a previously unused and globally unique URI. The profile's XML Schema file must be available (or accessible on the internet) to everybody parsing the associated CityGML instance document.

The second approach is illustrated by the following example XML Schema definition for the base profile of CityGML. Since the base profile is the union of all CityGML modules, the corresponding XML Schema defini-tion imports each and every CityGML module. By this means, all components of the CityGML data model are available in and may be exchanged by instance documents referencing this example base profile. The schema definition file of the base profile is shipped with the CityGML schema package, and is accessible at http://schemas.opengis.net/citygml/profiles/base/2.0/CityGML.xsd.

NOTE replace XML with UML if feasible.

The following excerpt of a CityGML dataset exemplifies how to apply the base profile schema CityGML.xsd to a CityGML instance document. The dataset contains two building objects and a city object group. The base profile defined by CityGML.xsd is referenced using the xsi:schemaLocation attribute of the root element. Thus, all CityGML modules are employed by the instance document and no further references to the XML Schema documents of the CityGML modules are necessary.

NOTE replace XML with UML if feasible

Chapter 6. CityGML UML Model

6.1. Overview

Contributors
TBD

NOTE The following text needs to be reviewed and updated.

6.1.1. Conceptual Modelling

ISO 19101 defines universe of discourse to be a view of the real or hypothetical world that includes everything of interest. That standard then defines conceptual model to be a model that defines concepts of a universe of discourse.

The scope of this CityGML Conceptual Model Standard establishes the limits of the universe of discourse for this Standard. The next task is to discover and standardize the concepts within this scope. CityGML will potentially support numerous diverse application software packages covering multiple disciplines and facility life cycle phases. Each conceivably can have its own universe of discourse and their own set of concepts.

The goal of this CityGML Conceptual Model Standard is to establish and document a common set of concepts that spans the applications supported. This does not attempt to redefine application concepts, but merely present a common set of concepts from and to which their concepts can be understood and mapped.

GML and JSON encodings are planned and other encodings are anticipated. Each encoding addresses a specific information community and set of application software packages. However, with the increasing desire to share information between communities and applications having a common conceptual model across all of these encodings is highly advantageous.

An added benefit of the development of a conceptual model results from the rigor involved in achieving consensus. After numerous iterations, the end result is consistent, cohesive, and complete. Updating a conceptual model is far easier than rewriting software code. Further, the iterations help to flesh out details as well as to unearth differences in individual conceptualizations.

Perhaps the greatest benefit of the standards activity is the ability to communicate the resultant model. This is in part due to using a standardized conceptual modelling language like UML and the agreed OGC and ISO/TC211 conventions for using UML. The eventual outcome of being able to provide formal documentation for what is meant by each concept is invaluable in understanding the subsequent encodings and applications.

This will be the first OGC conceptual model standard without accompanying encodings. Yet the model is presented in a manner consistent with the formalisms adopted for writing OGC standards. This standard follows the OGC Specification Model standard for modular specifications and is consistent with the OGC Naming Authority conventions and recommendations. The target of this Standard are the encoding standards which will follow and not the application software that will

implement these encodings. Requirements for the encodings are explicit and grouped into Requirements Classes. Accompanying Conformance Classes are included to determine if an encoding conforms to the conceptual model.

UML has been used as the conceptual modelling language in this Standard. Class Diagrams have been created and inserted as Figures. The boxes in these diagrams (officially “Classifiers” in UML) typically represent classes, data types, enumerations, code lists, unions, etc. and this terminology is used throughout the Standard. However, since this is a Conceptual Model, these should all be interpreted to be “concepts”. For each Requirements Class, an introductory diagram is included which contains all of the concepts relevant to that Requirements Class.

Though redundant with the UML diagrams, all of the classes, class attributes, and associations are repeated in the Data Dictionary in [data-dictionary-section]. If these differ, the UML takes precedence.

6.1.2. From CityGML 2

CityGML is an open data model and XML-based format for the storage and exchange of virtual 3D city models. It is an application schema for the Geography Markup Language version 3.1.1 (GML3), the extendible interna-tional standard for spatial data exchange issued by the Open Geospatial Consortium (OGC) and the ISO TC211.

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 mainte-nance of 3D city models, allowing the reuse of the same data in different application fields.

CityGML not only represents the graphical appearance of city models but specifically addresses the representa-tion of the semantic and thematic properties, taxonomies and aggregations. CityGML includes a geometry model and a thematic model. The geometry model allows for the consistent and homogeneous definition of geometrical and topological properties of spatial objects within 3D city models (chapter 8). The base class of all objects is _ *CityObject* which is a subclass of the GML class _ *Feature*. All objects inherit the properties from _ *CityObject*.

The thematic model of CityGML employs the geometry model for different thematic fields like Digital Terrain Models, sites (i.e. buildings, bridges, and tunnels), vegetation (solitary objects and also areal and volumetric biotopes), land use, water bodies, transportation facilities, and city furniture (chapter 10). Further objects, which are not explicitly modelled yet, can be represented using the concept of generic objects and attributes (chapter 6.11). In addition, extensions to the CityGML data model applying to specific application fields can be realised using the Application Domain Extensions (ADE) (chapter 6.12). Spatial objects of equal shape which appear many times at different positions like e.g. trees, can also be modelled as prototypes and used multiple times in the city model (chapter 8.2). A grouping concept allows the combination of single 3D objects, e.g. buildings to a building complex (chapter 6.8). Objects which are not geometrically modelled by closed solids can be virtually sealed in order to compute their volume (e.g. pedestrian underpasses, tunnels, or airplane hangars). They can be closed using *ClosureSurfaces* (chapter 6.4). The concept of the *TerrainIntersectionCurve* is introduced to inte-grate 3D objects with the Digital Terrain Model at their correct positions in order to prevent e.g. buildings from floating over or sinking into the terrain (chapter 6.5).

CityGML differentiates five consecutive Levels of Detail (LOD), where objects become more detailed with increasing LOD regarding both their geometry and thematic differentiation (chapter 6.2). CityGML files can - but do not have to - contain multiple representations (and geometries) for each object in different LOD simulta-neously. Generalisation relations allow the explicit representation of aggregated objects over different scales.

In addition to spatial properties, CityGML features can be assigned appearances. Appearances are not limited to visual data but represent arbitrary observable properties of the feature's surface such as infrared radiation, noise pollution, or earthquake-induced structural stress (chapter 9).

Furthermore, objects can have external references to corresponding objects in external datasets (chapter 6.7). The possible attribute values of enumerative object attributes can be enumerated in code lists defined in external, redefinable dictionaries (chapter 6.6).

6.2. Core

Contributors
TBD

NOTE The following text needs to be reviewed and updated.

The CityGML Core module defines the basic concepts and components of the overall CityGML data model. It forms the universal lower bound of the CityGML data model and, thus, is a dependency of all extension modules. Consequently, the core module has to be implemented by any conformant system. Primarily, the core module provides the abstract base classes from which thematic classes within extension modules are (transitively) derived. Besides abstract type definitions, the core also contains non-abstract content, for example basic data types and thematic classes that may be used by more than one extension module. The UML diagram in Fig. 21 illustrates CityGML's core module, for the XML Schema definition see below and annex A.1.

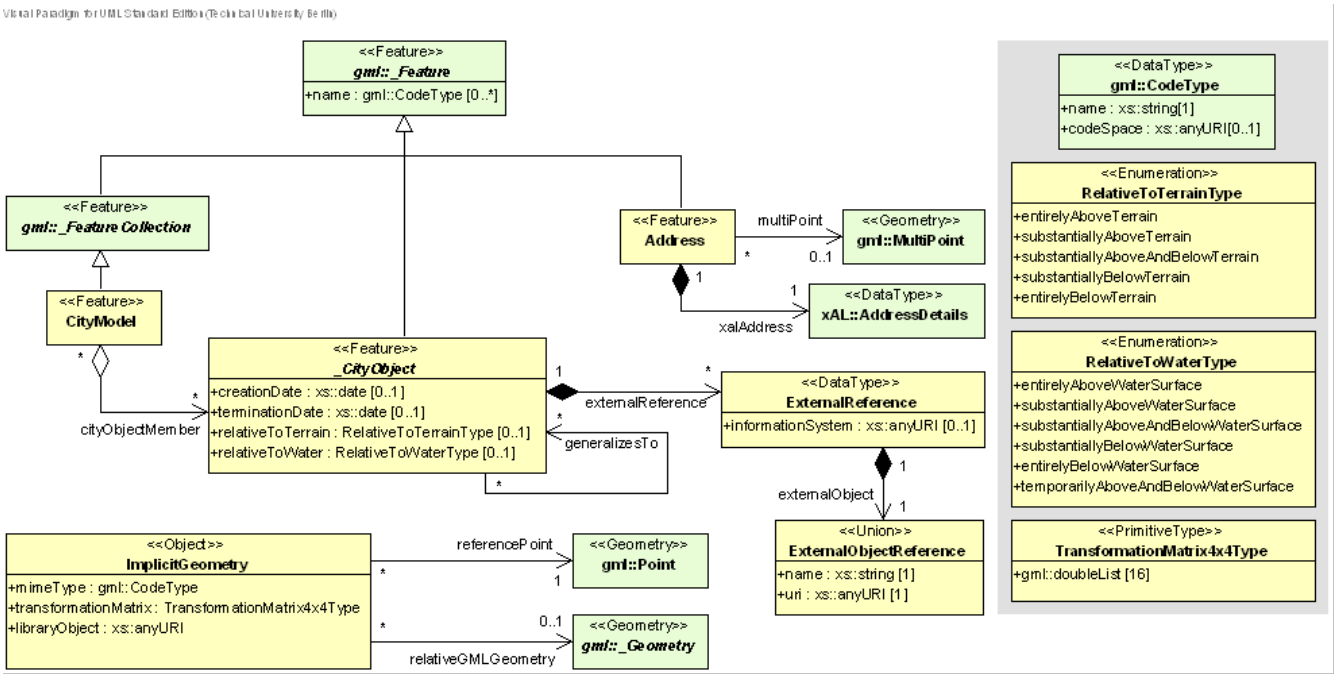


Figure 6. UML diagram of CityGML's core module. The bracketed numbers following the attribute names denote the attribute's multiplicity: the minimal and maximal number of occurrences of the attribute per object. For example, a name is optional (0) in the class _Feature or may occur multiple times (star symbol), while a _CityObject has none or at most one creationDate. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Core module.

The base class of all thematic classes within CityGML's data model is the abstract class _CityObject. _CityObject provides a creation and a termination date for the management of histories of features as well as the possibility to model external references to the same object in other data sets. Furthermore, two qualitative attributes relativeToTerrain and relativeToWater are provided which enable to specify the feature's location with respect to the terrain and water surface. The possible topological relations are illustrated in Fig. 22. Both attributes facilitate simple and efficient queries like for the number of subsurface buildings (entirelyBelowTerrain) without the need for an additional digital terrain model or a model of the water body.

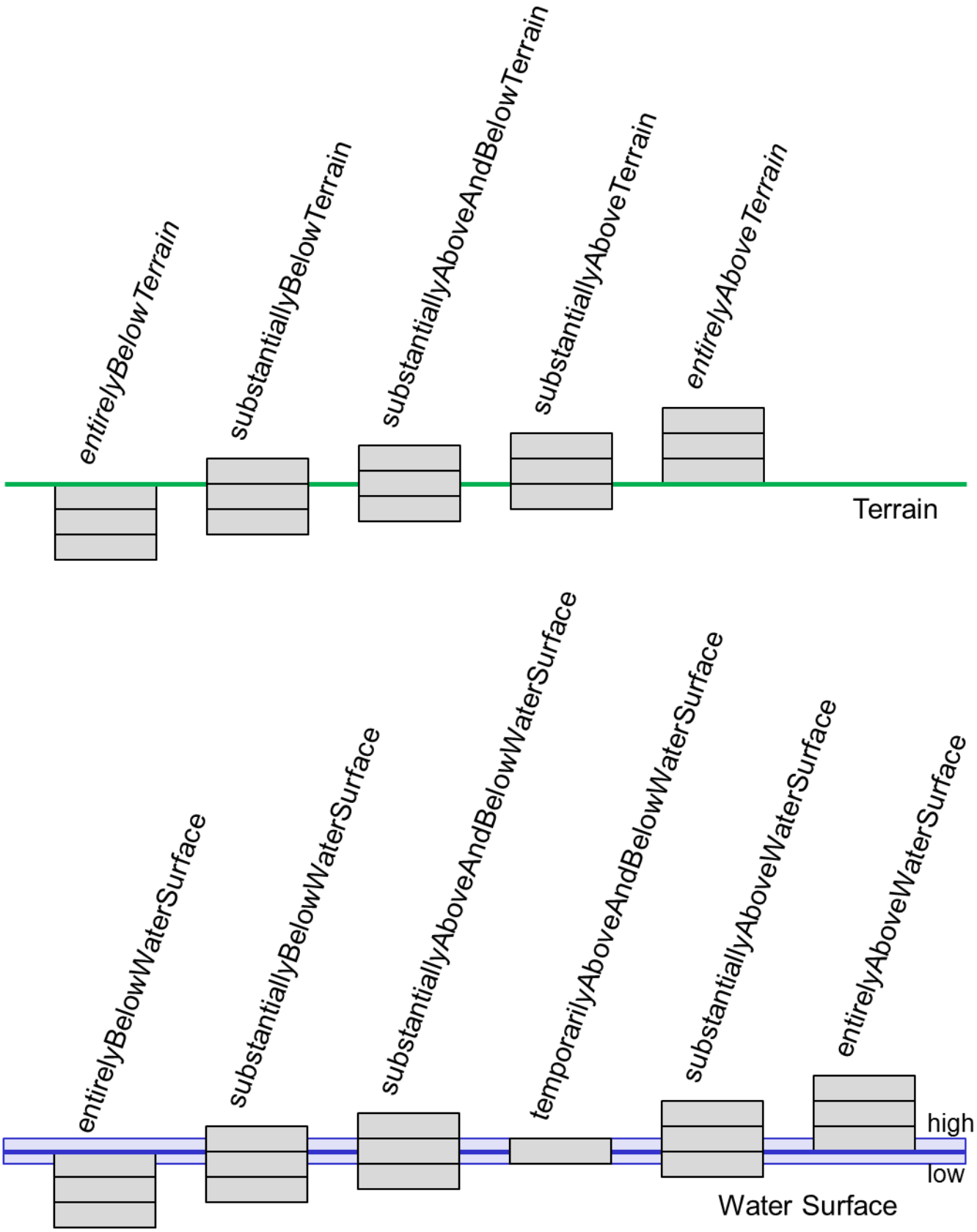


Figure 7. Topological relations of a CityGML object with respect to a) the terrain and b) the water surface.

_CityObject is a subclass of the GML class _Feature, thus it inherits the metadata property (which can be e.g. information about the lineage, quality aspects, accuracy, local CRS) and name property from the superclass _GML. A _CityObject may have multiple names, which are optionally qualified

by a codeSpace. This enables the differentiation between, for example, an official name and a popular name or of names in different languages (cf. the name property of GML objects, Cox et al. 2004). The generalisation property generalizesTo of _CityObject may be used to relate features, which represent the same real-world object in different Levels-of-Detail, i.e. a feature and its generalised counterpart(s). The direction of this relation is from the feature to the corresponding generalised feature.

Thematic classes may have further subclasses with relations, attributes and geometry. Features of the specialized subclasses of _CityObject may be aggregated to a single CityModel, which is a feature collection with optional metadata. Generally, each feature has the attributes class, function, and usage, unless it is stated otherwise. The class attribute can occur only once, while the attributes usage and function can be used multiple times. The class attribute allows for the classification of features beyond the thematic class hierarchy of _CityObject. For exam-ple, a building feature is represented by the thematic subclass bldg:Building of _CityObject in the first place (this subclass is defined within CityGML's Building module, cf. chapter 10.3). A further classification, e.g. as resi-dential or administration building, may then be modelled using the class attribute of the class bldg:Building. The attribute function normally denotes the intended purpose or usage of the object, such as hotel or shopping centre for a building, while the attribute usage normally defines its real or actual usage. Possible values for the attrib-utes class, function, and usage can be specified in code lists which are recommended to be implemented as simple dictionaries following the Simple Dictionary Profile of GML 3.1.1 (cf. chapter 6.6 and 10.14). Annex C provides code lists proposed and maintained by the SIG 3D which contain feasible attribute values and which may be extended or redefined by users.

In addition to thematic content, the core module also provides the concept of implicit geometries as an enhance-ment of the geometry model of GML3. Since this concept is strongly related to the spatial model of CityGML it has already been introduced in chapter 8.2.

The top level class hierarchy of the thematic model in CityGML is presented in Fig. 23. The subclasses of _CityObject comprise the different thematic fields of a city model covered by separate CityGML extension modules: the terrain, buildings, bridges, tunnels, the coverage by land use objects, water bodies, vegetation, generic city objects, city furniture objects, city object groups, and transportation. To indicate the extension module defining a respective subclass of _CityObject, the class names in Fig. 23 are preceded by prefixes. Each prefix is associated with one CityGML extension module (see chapter 4.3 and chapter 7 for a list of CityGML's extension modules and the corresponding prefixes).

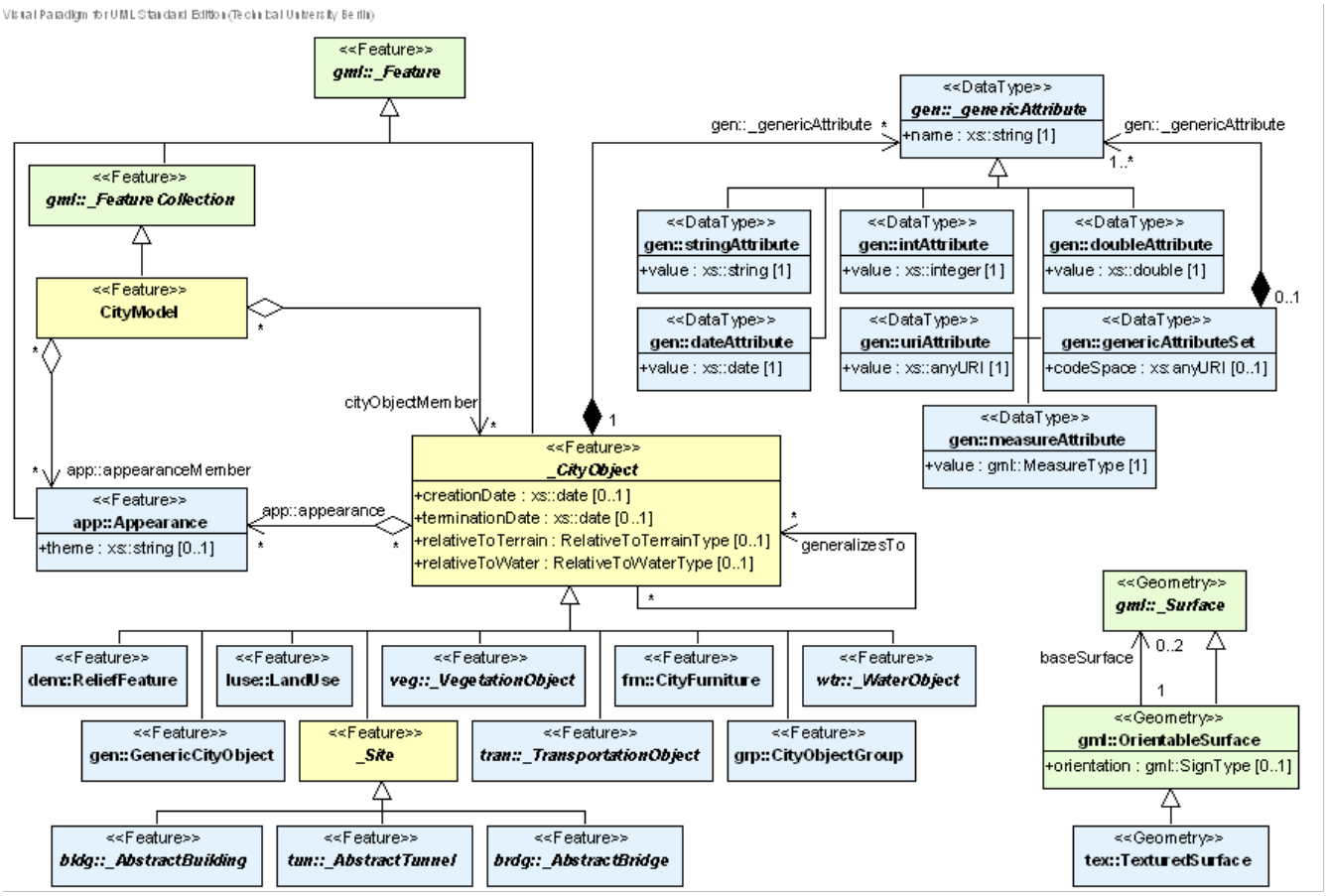


Figure 8. CityGML's top level class hierarchy. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Core module.

The classes GenericCityObject and _genericAttribute defined within CityGML's Generics module (cf. chapters 6.11 and 10.12) allow for modelling and exchanging of 3D objects which are not covered by any other thematic class or which require attributes not represented in CityGML. For example, in the future, sites derived from the abstract class _Site of the core module may be completed by further subclasses like excavation, city wall or embankment. At present, the class GenericCityObject should be used in order to represent and exchange these features. However, the concept of generic city objects and attributes may only be used if appropriate thematic classes or attributes are not provided by any other CityGML module.

If the Generics module is employed, each CityObject may be assigned an arbitrary number of generic attributes in order to represent additional properties of features. For this purpose, the Generics module augments the abstract base class _CityObject by the property element _genericAttribute. The additional property _genericAttribute is injected into _CityObject using CityGML's Application Domain Extension mechanism (cf. chapter 10.13). By this means, each thematic subclass of _CityObject inherits this property and, thus, the possi-bility to contain generic attributes. Therefore, the Generics module has a deliberate impact on all CityGML extension modules defining thematic subclasses of _CityObject.

Appearance information about a feature's surfaces can be represented by the class Appearance provided by CityGML's Appearance module (cf. chapter 9). In contrast to the other thematic extensions to the core, Appear-ance is not derived from _CityObject but from the GML class _Feature. _CityObject features and Appearance features may be embraced within a single CityModel feature collection in an arbitrary or even mixed sequence using the cityObjectMember and appearanceMember elements, both being members of the substitution group gml:featureMember

(cf. chapter 9 and chapter 10.1.1). Furthermore, feature appearances may be stored inline the _CityObject itself. In order to enable city objects to store appearance information, the Appearance module augments the abstract base class _CityObject by the property element appearance using CityGML's Application Domain Extension mechanism (cf. chapter 10.13). Consequently, the appearance property is only available for _CityObject and its thematic subclasses if the Appearance module is supported. Therefore, like the Generics module, the Appearance module has a deliberate impact on any other extension module.

For sake of completeness, the class TexturedSurface is also illustrated in Fig. 23. This approach of appearance modelling of previous versions of CityGML has been deprecated and is expected to be removed in future CityGML versions. Since the information covered by TexturedSurface can be losslessly converted to the Ap-pearance module, the use of TexturedSurface is strongly discouraged.

6.2.1. Base elements

AbstractCityObjectType, _CityObject

AbstractCityObjectType, _CityObject

NOTE	insert AbstractCityObjectType, _CityObject UML

CityModelType, CityModel

NOTE	insert CityModelType, CityModel UML

cityObjectMember

NOTE	insert cityObjectMember UML

AbstractSiteType, _Site

NOTE	insert AbstractSiteType, _Site UML

The abstract class _Site is intended to be the superclass for buildings, bridges, tunnels, facilities, etc. Future extension of CityGML (e.g. excavations, city walls or embankments) would be modelled as subclasses of _Site. As subclass of _CityObject, a _Site inherits all attributes and relations, in particular the id, names, external references, and generalisation relations.

6.2.2. Generalisation relation, RelativeToTerrainType and RelativeToWaterType

GeneralizationRelationType

NOTE	insert GeneralizationRelationType UML

RelativeToTerrainType, RelativeToWaterType

NOTE	insert RelativeToTerrainType, RelativeToWaterType UML

6.2.3. External references

An ExternalReference defines a hyperlink from a _CityObject to a corresponding object in another information system, for example in the German cadastre (ALKIS), the German topographic information system (ATKIS), or the OS MasterMap®. The reference consists of the name of the external information system, represented by an URI, and the reference of the external object, given either by a string or by an URI. If the informationSystem element is missing in the ExternalReference, the ExternalObjectReference must be an URI.

ExternalReferenceType, ExternalObjectReferenceType

NOTE	insert ExternalReferenceType, ExternalObjectReferenceType UML

6.2.4. Address information

The CityGML core module provides the means to represent address information of real-world features within virtual city models. Since not every real-world feature is assigned an address, a correspondent address property is not defined for the base class _CityObject, but has to be explicitly modelled for a thematic subclass. For example, the building model declares address properties for its classes _AbstractBuilding and Door. Both classes are referencing the corresponding data types of the core module to represent address information (cf. chapter 10.3).

Addresses are modelled as GML features having one xalAddress property and an optional multiPoint property. For example, for a building feature the multiPoint property allows for the specification of the exact positions of the building entrances that are associated with the corresponding address. The point coordinates can be 2D or 3D. Modelling addresses as features has the advantage that GML3's method of representing features by refer-ence (using XLinks) can be applied. This means, that addresses might be bundled as an address FeatureCollec-tion that is stored within an external file or that can be served by an external Web Feature Service. The address property elements within the CityGML file then would not contain the address information inline but only references to the corresponding external features.

The address information is specified using the xAL address standard issued by the OASIS consortium (OASIS 2003), which provides a generic schema for all kinds of international addresses. Therefore, child elements of the xalAddress property of Address have to be structured according to the OASIS xAL schema.

AddressPropertyType, AddressType, Address

NOTE	insert AddressPropertyType, AddressType, Address UML

The following two excerpts of a CityGML dataset contain examples for the representation of German and British addresses in xAL. The address information is attached to building objects (bldg:Building) according to the CityGML Building module (cf. chapter 10.3). Generally, if a CityGML

instance document contains address information, the namespace prefix “xAL” should be declared in the root element and must refer to “urn:oasis:names:tc:ciq:xsdschema:xAL:2.0”. An example showing a complete CityGML dataset including a building with an address element is provided in annex G.1.

NOTE insert examples here if appropriate.

6.3. Appearance

Contributors
TBD

NOTE The following text needs to be reviewed and updated.

In addition to spatial properties, CityGML features have appearances – observable properties of the feature’s surface. Appearances are not limited to visual data but represent arbitrary categories called themes such as infrared radiation, noise pollution, or earthquake-induced structural stress. Each LOD can have an individual appearance for a specific theme. An appearance is composed of data for each surface geometry object, i.e. surface data. A single surface geometry object may have surface data for multiple themes. Similarly, surface data can be shared by multiple surface geometry objects (e.g. road paving). Finally, surface data values can either be constant across a surface or depend on the exact location within the surface.

CityGML’s appearance model is defined within the extension module Appearance (cf. chapter 7). The UML diagram of the appearance model is illustrated in Fig. 14, for the XML Schema definition see annex A.2.

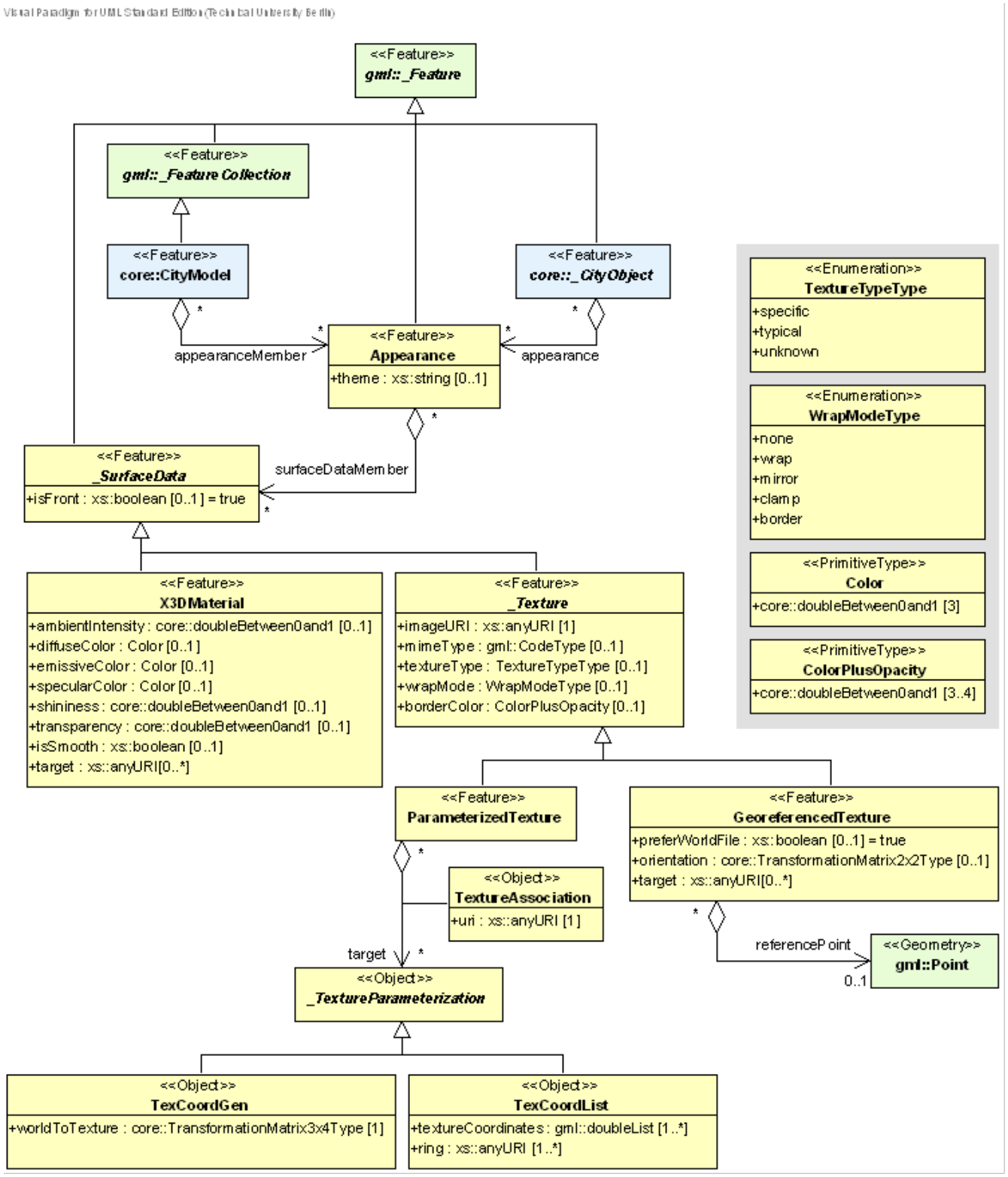


Figure 9. UML diagram of CityGML's appearance model. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Appearance module.

In CityGML's appearance model, themes are represented by an identifier only. The appearance of a city model for a given theme is defined by a set of Appearance objects referencing this theme. Thus, the Appearance objects belonging to the same theme compose a virtual group. They may be included in different places within a CityGML dataset. Furthermore a single CityGML dataset may contain several themes. An Appearance object collects surface data relevant for a specific theme either for individual features or the whole city model in any LOD. Surface data is represented by objects of class _SurfaceData and its descendents with each covering the whole area of a surface

geometry object. The relation between surface data and surface geometry objects is expressed by an URI (Uniform Resource Identifier) link from a _SurfaceData object to an object of type gml:AbstractSurfaceType or type gml:MultiSurface.

A constant surface property is modelled as material. A surface property, which depends on the location within the surface, is modelled as texture. Each surface geometry object can have both a material and a texture per theme and side. This allows for providing both a constant approximation and a complex measurement of a surface's property simultaneously. An application is responsible for choosing the appropriate property represen-tation for its task (e.g. analysis or rendering). A specific mixing is not defined since this is beyond the scope of CityGML. If a surface geometry object is to receive multiple textures or materials, each texture or material requires a separate theme. The mixing of themes or their usage is not defined within CityGML and left to the application.

XML namespace

The XML namespace of the CityGML Appearance module is identified by the Uniform Resource Identifier (URI) http://www.opengis.net/citygml/appearance/2.0. Within the XML Schema definition of the Appearance module, this URI is also used to identify the default namespace.

6.3.1. Relation between appearances, features and geometry

Despite the close relation between surface data and surface, surface data is stored separately in the feature to preserve the original GML geometry model. Instead of surface data being an attribute of the respective target surface geometry object, each surface data object maintains a set of URIs specifying the gml:ids of the target surface geometry objects (of type gml:AbstractSurfaceType or gml:MultiSurface). In case of a composite or aggregate target surface, the surface data object is assigned to all contained surfaces. Other target types such as features, solids, or gml:AbstractSurfacePatchType (which includes gml:Triangle) are invalid, even though the XML schema language cannot formally express constrains on URI target types. For the exact mapping function of surface data values to a surface patch refer to the respective surface data type description.

The limitation of valid target types to gml:AbstractSurfaceType and gml:MultiSurface excluding gml:Abstract-SurfacePatchType is based on the GML geometry model and its use in CityGML. In general, GML surfaces are represented using subclasses of gml:AbstractSurfaceType. Such surfaces are required to be continuous. A gml:MultiSurface does not need to fulfill this requirement and consequently is no gml:AbstractSurfaceType (cf. 8.1). Since captured real-world surfaces often cannot be guaranteed to be continuous, CityGML allows for gml:MultiSurface to represent a feature's boundary in various places as an alternative to a continuous surface. To treat such surfaces similarly to a gml:CompositeSurface, surface data objects are allowed to link to gml:Multi-Surface objects. gml:AbstractSurfacePatchType is no valid target type since it is not derived from gml:AbstractGMLType. Thus, a gml:AbstractSurfacePatchType (which includes gml:Triangle and gml:Rectangle) cannot receive a gml:id and cannot be referenced.

Each surface geometry object can have per theme at most one active front-facing material, one active back-facing material, one active front-facing texture, and one active back-facing texture. If multiple surface data objects of the same category and theme are assigned to a surface geometry object, one is chosen to become active. Multiple indirect assignments due to nested surface

definitions are resolved by overwriting, e.g. the front-facing material of a gml:Polygon becomes active by overwriting the front-facing material of the parental gml:CompositeSurface. Multiple direct assignments, i.e. a surface geometry object's gml:id is referenced multi-ple times within a theme, are not allowed and are resolved implementation-dependently by choosing exactly one of the conflicting surface data objects. Thus, multiple direct assignments within a theme need to be avoided.

Each _CityObject feature can store surface data. Thus, surface data is arranged in the feature hierarchy of a CityGML dataset. Surface data then links to its target surface using URIs. Even though the linking mechanism permits arbitrary links across the feature hierarchy to another feature's surface, it is recommended to follow the principle of locality: Surface data should be stored such that the linked surfaces only belong to the containing _CityObject feature and its children. "Global" surface data should be stored with the city model. Adhering to the locality principle also ensures that CityObjects retrieved from a WFS will contain the respective appearance information.

The locality principle allows for the following algorithm to find all relevant _SurfaceData objects referring to a given surface geometry object (of type gml:AbstractSurfaceType or gml:MultiSurface) in a given _CityObject:

```
---------------------------
  function findSurfaceData
  in: gmlSurface, cityObject
  out: frontMaterial, frontTexture, backMaterial, backTexture
---------------------------
  1: frontMaterial := empty
  2: frontTexture := empty
  3: backMaterial := empty
  4: backTexture := empty
  5: flip := false
  6:
  7: while (gmlSurface) { // traverse the geometry hierarchy from inner to outer
  8:   cObj := cityObject // start from the innermost cityobject
  9:
  10:   while (cObj) { // traverse the cityobject hierarchy for the current geometry
object 11: // search all surfaceData objects in all appearance containers
  12:     foreach (appearance in cObj) {
  13:       foreach (surfaceData in appearance) {
  14:         if (surfaceData refers to gmlSurface) { // if a surfaceData object
refers to the geometry object, check its category
  15:           if (flip) { // consider flipping
  16: // only pick the first surfaceData for a particular category
  17:             if (surfaceData is frontside material AND backMaterial is empty) {
  18:               backMaterial := surfaceData
  19:             }
  20:             if (surfaceData is frontside texture AND backTexture is empty) {
  21:               backTexture := surfaceData
  22:             }
  23:             if (surfaceData is backside material AND frontMaterial is empty) {
  24:               frontMaterial := surfaceData
  25:             }
```

```
  26:             if (surfaceData is backside texture AND frontTexture is empty) {
  27:               frontTexture := surfaceData
  28:             }
  29:           } else {
  30:             // only pick the first surfaceData for a particular category
  31:             if (surfaceData is frontside material AND frontMaterial is empty) {
  32:               frontMaterial := surfaceData
  33:             }
  34:             if (surfaceData is frontside texture AND frontTexture is empty) {
  35:               frontTexture := surfaceData
  36:             }
  37:             if (surfaceData is backside material AND backMaterial is empty) {
  38:               backMaterial := surfaceData
  39:             }
  40:             if (surfaceData is backside texture AND backTexture is empty) {
  41:               backTexture := surfaceData
  42:             }
  43:           }
  44:
  45:           // shortcut: could stop here if all 4 categories have been found
  46:         }
  47:       }
  48:     }
  49:     cObj := cObj.parent // this also includes the global CityModel
  50:   }
  51:   gmlSurface := gmlSurface.parent // this also includes a root gml:MultiSurface
  52:   if (gmlSurface isA gml:OrientableSurface AND gmlSurface.orientation is
negative) {
  53:     negate flip
  54:   }
  55: }
```

Listing 1: Algorithm to find all relevant _SurfaceData objects referring to a given surface geometry object (of type gml:AbstractSurfaceType or gml:MultiSurface) in a given _CityObject.

The evaluation of the isFront property of a _SurfaceData object needs to take gml:OrientableSurfaces into account, as those can flip the orientation of a surface. Assume a gml:OrientableSurface os, which flips its base surface bs. A front side texture t targeting bs will appear on the actual front side of bs. If t targets os, it will appear on the back side of bs. If t targets both os and bs, it appears on both sides of bs since it becomes the front and back side texture.

XLinks influence the hierarchy traversal in the pseudocode. In general, the separation of surface data and geome-try objects requires the reevaluation of the surface data assignment for each occurrence of a geometry object in the context of the respective _CityObject. Stepping up the (geometry or _CityObject) hierarchy in the algorithm takes XLinks into account, i.e., for the purpose of this algorithm, referenced objects are conceptually copied to the location of the referring XLink. In particular, this applies to ImplicitGeometry objects. If an ImplicitGeome-try object contains GML geometry (in the relativeGMLGeometry property), the surface data assignment needs to be reevaluated in the context of each referring _CityObject. Thus, the appearance (but not the relative geometry) of a given ImplicitGeometry can differ between its occurrences. A consistent appearance

results if all required surface data objects are placed in Appearance objects and the latter are stored either

1. in the _CityObject containing the original ImplicitGeometry with XLinks referencing the same Appear-ance objects in all _CityObjects that refer to the ImplicitGeometry or
2. in the global CityModel.

6.3.2. Appearance and SurfaceData

The feature class Appearance defines a container for surface data objects. It provides the theme that all contained surface data objects are related to. All appearance objects with the same theme in a CityGML file are considered a group. Surface data objects are stored in the surfaceDataMember property. They can be used in multiple themes simultaneously as remote properties.

The feature class _SurfaceData is the base class for materials and textures. Its only element is the boolean flag isFront, which determines the side a surface data object applies to. Please note, that all classes of the appearance model support CityGML's ADE mechanism (cf. chapters 6.12 and 10.13). The hooks for application specific extensions are realized by the elements "_GenericApplicationPropertyOf…".

AppearanceType, Appearance, AppearancePropertyType

NOTE	insert UML

appearanceMember, appearance

NOTE	insert UML

The definition of appearanceMember allows for an arbitrary or even mixed sequence of _CityObject features and Appearance features within a CityModel feature collection (cf. chapter 10.1).

In order to store appearance information within a single _CityObject feature, the corresponding abstract class _CityObject of the core module is augmented by the property element appearance. The additional property appearance is injected into _CityObject using CityGML's Application Domain Extension mechanism (cf. chapter 10.13). By this means, each thematic subclass of _CityObject inherits this property. Thus, the Appear-ance module has a deliberate impact on each extension module defining thematic subclasses of _CityObject.

AbstractSurfaceDataType, _SurfaceData, SurfaceDataPropertyType

NOTE	insert UML

6.3.3. Material

Materials define light reflection properties being constant for a whole surface geometry object. The definition of the class X3DMaterial is adopted from the X3D and COLLADA specification (cf. X3D, COLLADA specifica-tion). diffuseColor defines the color of diffusely reflected light. specularColor

defines the color of a directed reflection. emissiveColor is the color of light generated by the surface. All colors use RGB values with red, green, and blue between 0 and 1. Transparency is defined separately using the transparency element where 0 stands for fully opaque and 1 for fully transparent. ambientIntensity defines the minimum percentage of dif-fuseColor that is visible regardless of light sources. shininess controls the sharpness of the specular highlight. 0 produces a soft glow while 1 results in a sharp highlight. isSmooth gives a hint for normal interpolation. If this boolean flag is set to true, vertex normals should be used for shading (Gouraud shading). Otherwise, normals should be constant for a surface patch (flat shading).

Target surfaces are specified using target elements. Each element contains the URI of one target surface geome-try object (of type gml:AbstractSurfaceType or gml:MultiSurface).

X3DMaterialType, X3DMaterial

NOTE	insert UML

6.3.4. Texture and texture mapping

The abstract base class for textures is _Texture. Textures in CityGML are always raster-based 2D textures. The raster image is specified by imageURI using a URI and can be an arbitrary image data resource, even a prefor-matted request for a web service. The image data format can be defined using standard MIME types in the mimeType element.

Textures can be qualified by the attribute textureType. The textureType differentiates between textures, which are specific for a certain object (specific) and prototypic textures being typical for that object surface (typical). Textures may also be classified as unknown.

The specification of texture wrapping is adopted from the COLLADA standard. Texture wrapping is required when accessing a texture outside the underlying image raster. wrapMode can have one of five values (Fig. 15 illustrates the effect of these wrap modes):

1. none – the resulting color is fully transparent
2. wrap – the texture is repeated
3. mirror – the texture is repeated and mirrored
4. clamp – the texture is clamped to its edges
5. border – the resulting color is specified by the borderColor element (RGBA)

In wrap mode mirror, the texture image is repeated both in horizontal and in vertical direction to fill the texture space similar to wrap mode wrap. Unlike wrap, each repetition results from flipping the previous texture part along the repetition direction. This behaviour removes the edge correspondence constraint for wrapped textures and always results in a seamless texture.

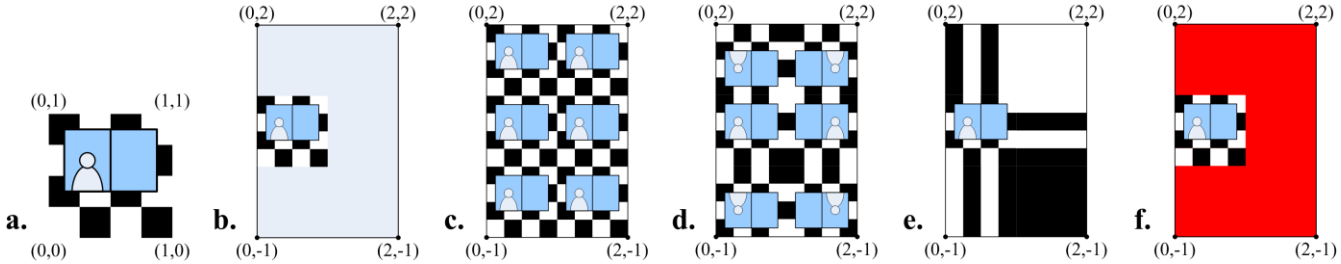


Figure 10. A texture (a) applied to a facade using different wrap modes: (b) none, (c) wrap, (d) mirror, (e) clamp and (f) border. The border color is red. The numbers denote texture coordinates (image: Hasso-Plattner-Institute).

AbstractTextureType, _Texture, WrapModeType, TextureTypeType

NOTE	insert UML

_Texture is further specialised according to the texture parameterisation, i.e. the mapping function from a loca-tion on the surface to a location in the texture image. CityGML uses the notion of texture space, where the texture image always occupies the region $[0,1]^2$ regardless of the actual image size or aspect ratio. The lower left image corner is located at the origin (some graphics APIs may use other conventions and require texture coordi-nate conversion). The mapping function must be known for each surface geometry object to receive texture.

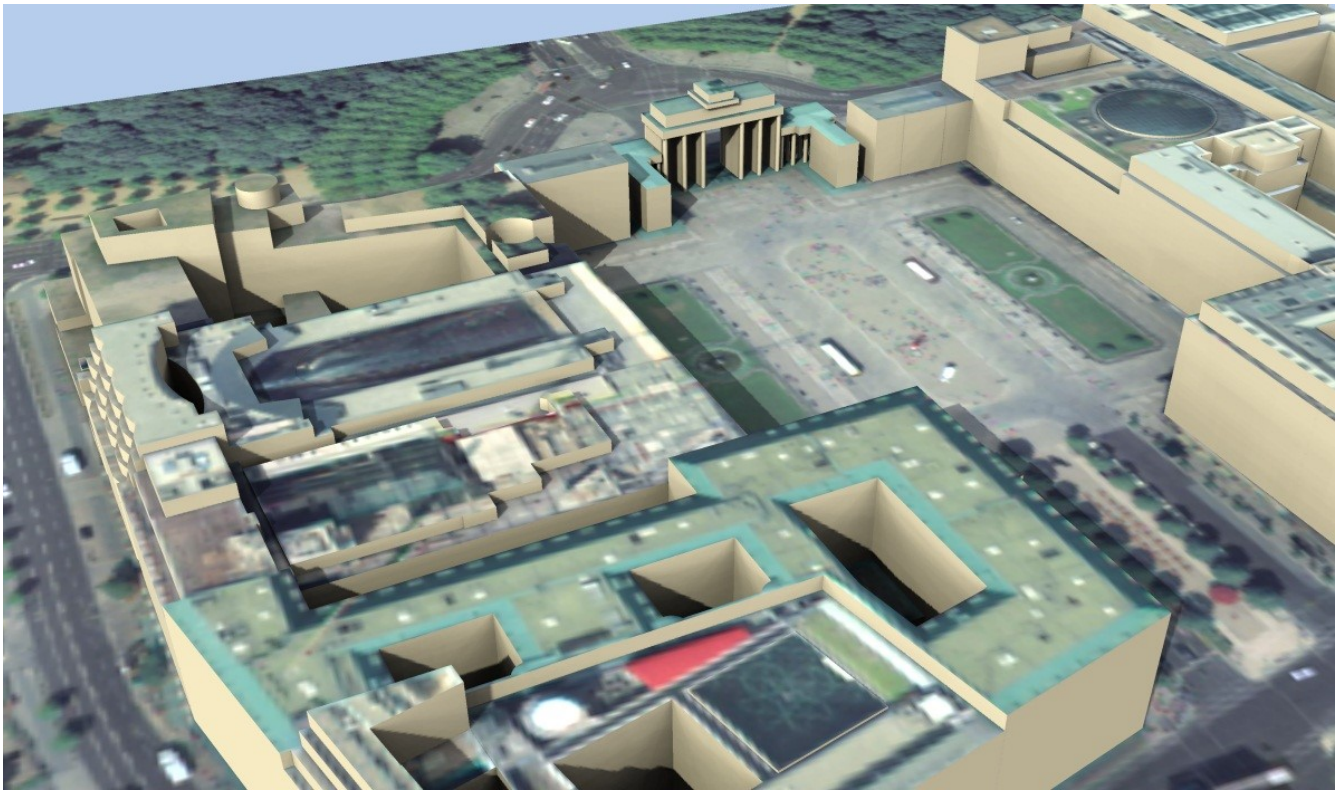


Figure 11. A georeferenced texture applied to ground and roof surfaces (source: Senate of Berlin, Hasso-Plattner-Institute).

The class GeoreferencedTexture describes a texture that uses a planimetric projection. Consequently, it does not make sense to texture vertical surfaces using a GeoreferencedTexture. Such a texture has a unique mapping function which is usually provided with the image file (e.g. georeferenced TIFF) or as a separate ESRI world file1. The search order for an external georeference is determined by the boolean flag preferWorldFile. If this flag is set to true (its default

value), a world file is looked for first and only if it is not found the georeference from the image data is used. If preferWorldFile is false, the world file is used only if no georeference from the image data is available.

Alternatively, CityGML allows for inline specification of a georeference similar to a world file. This internal georeference specification always takes precedence over any external georeference. referencePoint defines the location of the center of the upper left image pixel in world space and corresponds to values 5 and 6 in an ESRI world file. Since GeoreferencedTexture uses a planimetric projection, referencePoint is two-dimensional. orien-tation defines the rotation and scaling of the image in form of a 2x2 matrix (a list of 4 doubles in row-major order corresponding to values 1, 3, 2, and 4 in an ESRI world file). The CRS of this transformation is identical to the referencePoint's CRS. A planimetric point □ □T x, y in that CRS is transformed to a point □ □T s,t in texture space using the formula:

NOTE insert equation

with M denoting orientation, PR denoting referencePoint., w the image's width in pixels, and h the image's height in pixels. This transformation compensates for the difference between the image coordinate system used in ESRI world files (origin in upper left corner, positive x-axis rightwards, and positive y-axis downwards) and texture space in CityGML (origin in lower left corner, positive x-axis rightwards, and positive y-axis upwards).

If neither an internal nor an external georeference is given the GeoreferencedTexture is invalid. Each target surface geometry object is specified by an URI in a target element. All target surface geometry objects share the mapping function defined by the georeference. No other mapping function is allowed. Please note, that the gml:boundedBy property inherited from gml:AbstractFeatureType could be set to the bounding box of valid image data to allow for spatial queries. Fig. 16 shows a georeferenced texture applied to the ground and all roof surfaces.

GeoreferencedTextureType, GeoreferencedTexture

NOTE insert UML

The class ParameterizedTexture describes a texture with target-dependent mapping function. The mapping is defined by subclasses of class _TextureParameterization as a property of the link to the target surface geometry object. Each target surface geometry object is specified as URI in the uri attribute of a separate target element. Since target implements gml:AssociationAttributeGroup, it allows referencing to a remote _TextureParameterization object (using the xlink:href attribute), e.g. for sharing a mapping function between targets or textures in different themes. The mapping function can either use the concept of texture coordinates (through class TexCoordList) or a transformation matrix from world space to texture space (through class TexCoordGen).

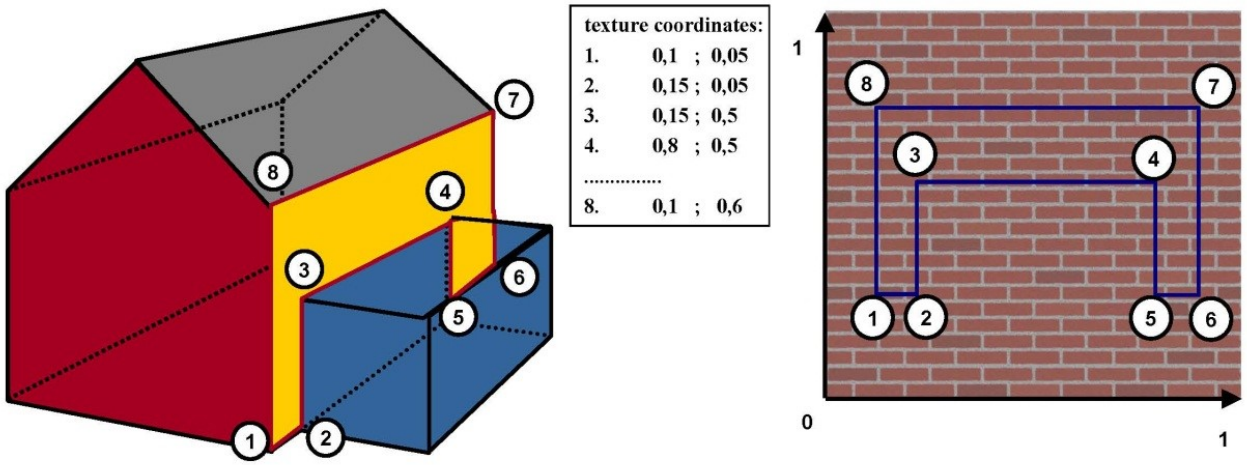


Figure 12. Positioning of textures using texture coordinates (image: IGG Uni Bonn).

Texture coordinates are applicable only to polygonal surfaces, whose boundaries are described by gml:LinearRing (e.g., gml:Triangle, gml:Polygon, or a gml:MultiSurface consisting of gml:Polygons). They define an explicit mapping of a surface's vertices to points in texture space, i.e. each vertex including interior ring vertices must receive a corresponding coordinate pair in texture space (for the notion of coordinates, refer to ISO 19111). These coordinates are not restricted to the [0,1] interval. Texture coordinates for interior surface points are planarly interpolated from the vertices' texture coordinates. Fig. 16 shows an example.

Texture coordinates for a target surface geometry object are specified using class TexCoordList as a texture parameterization object in the texture's target property. Each exterior and interior gml:LinearRing composing the boundary of the target surface geometry object (which also might be a gml:CompositeSurface, gml:MultiSurface, or gml:TriangulatedSurface) requires its own set of texture coordinates. A set of texture coordinates is specified using the textureCoordinates element of class TexCoordList. Thus, a TexCoordList contains as many textureCoordinate elements as the target surface geometry object contains gml:LinearRings. textureCoordinate's mandatory attribute ring provides the gml:id of the respective ring. The content is an ordered list of double values where each two values define a T s,t texture coordinate pair with s denoting the horizontal and t the vertical texture axis. The list contains one pair per ring point with the pairs' order corresponding to the ring points' order in the CityGML document (regardless of a possibly flipped surface orientation). If any ring point of a target surface geometry object has no texture coordinates assigned, the mapping is incomplete and the respective surface cannot be textured. In case of aggregated target geometry objects, mapping completeness is determined only for leaf geometry objects.

NOTE combine figures 18

Figure 13. Projecting a photograph (a) onto multiple facades (b) using the worldToTexture transformation. The photograph does not cover the left facade completely. Thus, the texture appears to be clipped. Texture wrapping is set to "none" (source: Senate of Berlin, Hasso-Plattner-Institute).

Alternatively, the mapping function can comprise a 3x4 transformation matrix specified by class TexCoordGen. The transformation matrix, specified by the worldToTexture element, defines a linear transformation from a spatial location in homogeneous coordinates to texture space. The use of homogeneous coordinates facilitates perspective projections as transformation, e.g. for projecting a photograph into a city model (cf. Fig. 18). Texture coordinates □ □T s,t are calculated from a space location □ □T x, y, z as □ □ □ □T T s,t □ s□ q□,t□ q□ with □ □ □ □T T s□,t□,q□ □M□ x, y, z,1 . M denotes the 3x4 transformation matrix. Compared to a general 4x4 transformation, the resulting z component is ignored. Thus, the respective matrix row is omitted. Additionally, the worldToTexture element uses the gml:SRSReferenceGroup attributes to define its CRS. A location in world space has to be first transformed into this CRS before the transformation matrix can be applied.

The following construction results in a worldToTexture transformation that mimics the process of taking a photograph by projecting a location in world space (in the city model) to a location in texture space:

NOTE insert transformation matrix

In this formula, f denotes the focal length; w and h represent the image sensor's physical dimensions; r □ , u □ , and d □ define the camera's frame of reference as right, up and directional unit vectors expressed in world coordinates; and P stands for the camera's location in world space. Fig. 19 sketches this setting.

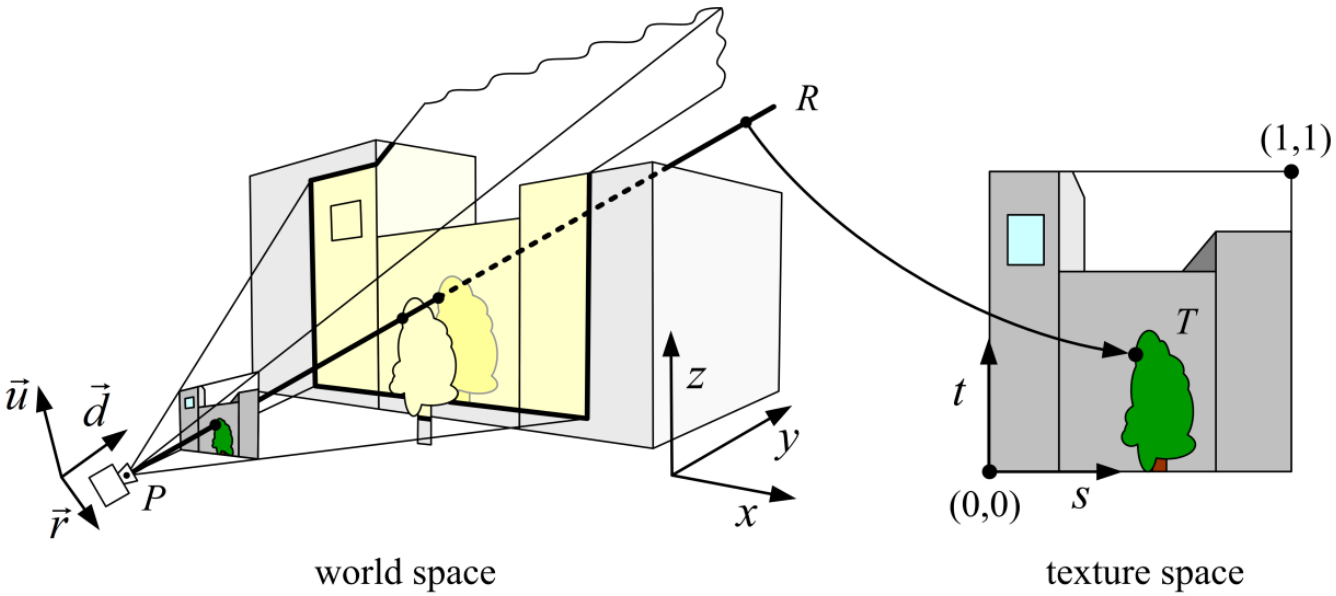


Figure 14. Projective texture mapping. All points on a ray R starting from the projection center P are mapped to the same point T in texture space (image: Hasso-Plattner-Institute, IGG TU Berlin).

Alternatively, if the 3x4 camera matrix MP is known (e.g. through a calibration and registration process), it can easily be adopted for use in worldToTexture. MP is derived from intrinsic and extrinsic camera parameters (interior and exterior orientation) and transforms a location in world space to a pixel location in the image. Assuming the upper left image corner has pixel coordinates (0,0), the complete transformation to texture space coordinates can be written as (widthimage and heightimage denote the image size in pixels):

NOTE insert formula

Please note, that worldToTexture cannot compensate for radial or other non-linear distortions introduced by a real camera lens.

Another use of worldToTexture is texturing a facade with complex geometry without specifying texture coordinates for each gml:LinearRing. Instead, only the facade's aggregated surface becomes the texture target using a TexCoordGen as parameterization. Then, worldToTexture effectively encodes an orthographic projection of world space into texture space. For the special case of a vertical facade this transformation is given by:

NOTE insert formula

```
<math display="block">
  <mrow class="MJX-TeXAtom-ORD">
    <mover>
      <mi mathvariant="normal">&#x2207;</mi>
      <mo stretchy="false">&#x2192;</mo>
    </mover>
  </mrow>
  <mo>&#xD7;</mo>
  <mrow class="MJX-TeXAtom-ORD">
    <mover>
      <mi>F</mi>
      <mo stretchy="false">&#x2192;</mo>
    </mover>
  </mrow>
  <mo>=</mo>
  <mrow>
    <mo>(</mo>
    <mfrac>
      <mrow>
        <mi mathvariant="normal">&#x2202;</mi>
        <msub>
          <mi>F</mi>
          <mi>z</mi>
        </msub>
      </mrow>
      <mrow>
        <mi mathvariant="normal">&#x2202;</mi>
        <mi>y</mi>
      </mrow>
    </mfrac>
    <mo>&#x2212;</mo>
    <mfrac>
      <mrow>
        <mi mathvariant="normal">&#x2202;</mi>
        <msub>
          <mi>F</mi>
          <mi>y</mi>
        </msub>
      </mrow>
```

```
      <mrow>
        <mi mathvariant="normal">&#x2202;</mi>
        <mi>z</mi>
      </mrow>
    </mfrac>
    <mo>)</mo>
  </mrow>
  <mrow class="MJX-TeXAtom-ORD">
    <mi mathvariant="bold">i</mi>
  </mrow>
  <mo>+</mo>
  <mrow>
    <mo>(</mo>
    <mfrac>
      <mrow>
        <mi mathvariant="normal">&#x2202;</mi>
        <msub>
          <mi>F</mi>
          <mi>x</mi>
        </msub>
      </mrow>
      <mrow>
        <mi mathvariant="normal">&#x2202;</mi>
        <mi>z</mi>
      </mrow>
    </mfrac>
    <mo>&#x2212;</mo>
    <mfrac>
      <mrow>
        <mi mathvariant="normal">&#x2202;</mi>
        <msub>
          <mi>F</mi>
          <mi>z</mi>
        </msub>
      </mrow>
      <mrow>
        <mi mathvariant="normal">&#x2202;</mi>
        <mi>x</mi>
      </mrow>
    </mfrac>
    <mo>)</mo>
  </mrow>
  <mrow class="MJX-TeXAtom-ORD">
    <mi mathvariant="bold">j</mi>
  </mrow>
  <mo>+</mo>
  <mrow>
    <mo>(</mo>
    <mfrac>
      <mrow>
        <mi mathvariant="normal">&#x2202;</mi>
```

```
        <msub>
          <mi>F</mi>
          <mi>y</mi>
        </msub>
      </mrow>
      <mrow>
        <mi mathvariant="normal">&#x2202;</mi>
        <mi>x</mi>
      </mrow>
    </mfrac>
    <mo>&#x2212;</mo>
    <mfrac>
      <mrow>
        <mi mathvariant="normal">&#x2202;</mi>
        <msub>
          <mi>F</mi>
          <mi>x</mi>
        </msub>
      </mrow>
      <mrow>
        <mi mathvariant="normal">&#x2202;</mi>
        <mi>y</mi>
      </mrow>
    </mfrac>
    <mo>)</mo>
  </mrow>
  <mrow class="MJX-TeXAtom-ORD">
    <mi mathvariant="bold">k</mi>
  </mrow>
</math>
```

This equation assumes n denoting the facade's overall normal vector (normalized, pointing outward, and being parallel to the ground), F denoting the facade's lower left point, and widthf and heightf specifying the facade's dimensions in world units. For the general case of an arbitrary normal vector the facade orientation matrix assumes a form similar to the camera orientation matrix:

NOTE	insert formula

ParameterizedTextureType, ParameterizedTexture, TextureAssociationType

NOTE	insert UML

AbstractTextureParameterizationType, TexCoordListType, TexCoordGenType

NOTE	insert UML

6.3.5. Related concepts

The notion of appearance clearly relates to the generic coverage approach (cf. ISO 19123 and OGC Abstract specification, Topic 6). Surface data can be described as discrete or continuous coverage over a surface as two-dimensional domain with a specific mapping function. Such an implementation requires the extension of GML coverages (as of version 3.1) by suitable mapping functions and specialisation for valid domain and range sets. For reasons of simplicity and comprehensibility both in implementation and usage, CityGML does not follow this approach, but relies on textures and materials as well-known surface property descriptions from the field of computer graphics (cf. X3D, COLLADA specification, Foley et al.). Textures and materials store data as color using an appropriate mapping. If such a mapping is impractical, data storage can be customised using ADEs. A review of coverages for appearance modelling is considered for CityGML beyond version 2.0.0.

Appearance is also related to portrayal. Portrayal describes the composition and symbolisation of a digital model's image, i.e. presentation, while appearance encodes observations of the real object's surface, i.e. data. Even though being based on graphical terms such as textures and materials, surface data is not limited to being input for portrayal, but similarly serves as input or output for analyses on a feature's surface. Consequently, CityGML does not define mixing or composition of themes for portrayal purposes. Portrayal is left to viewer applications or styling specification languages such as OGC Styled Layer Descriptors (SLD) or OGC Symbolo-gy Encoding (SE).

6.3.6. Code lists

The mimeType attribute of the feature _Texture is specified as gml:CodeType. The values of this property can be enumerated in a code list. A proposal for this code list can be found in annex C.6.

6.4. Bridge Model

Contributors
TBD

6.4.1. Synopsis

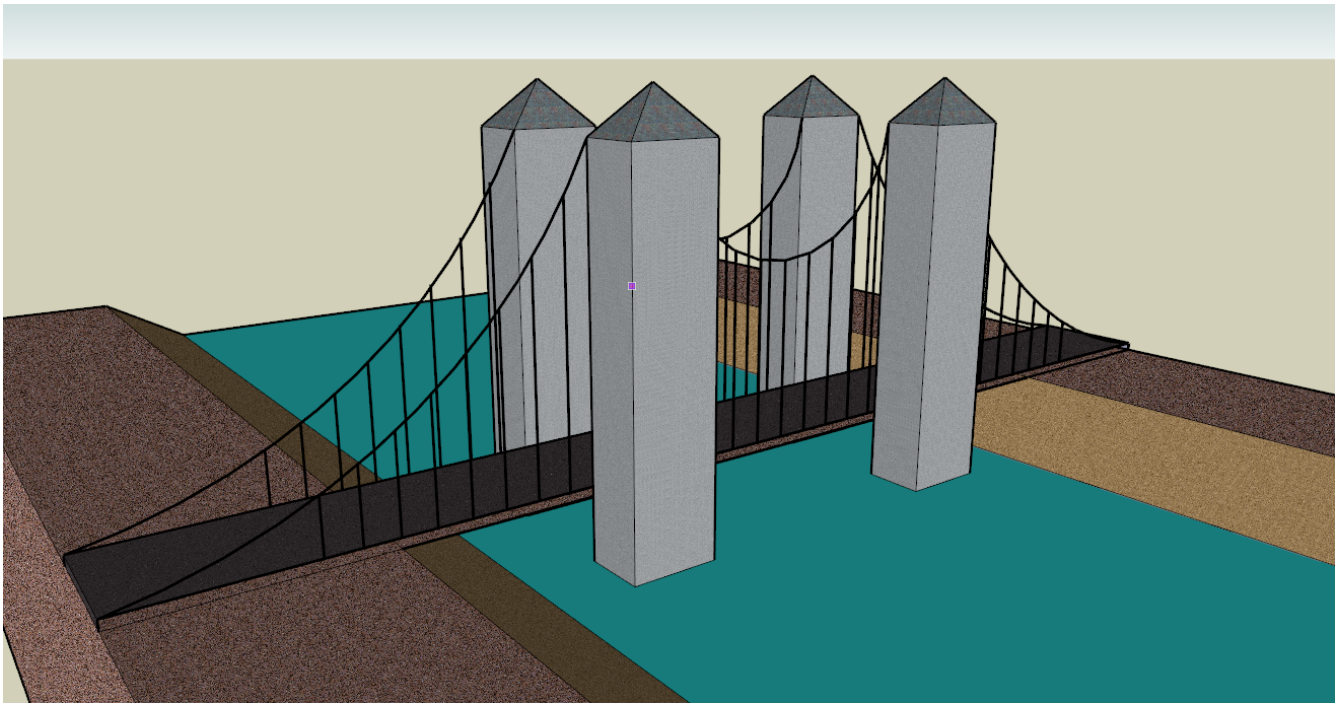
The bridge model allows for the representation of the thematic, spatial and visual aspects of bridges and bridge parts in four levels of detail, LOD 1 – 4.

6.4.2. Key Concepts

6.4.3. Discussion

NOTE	The following text needs to be reviewed and updated.

The bridge model of CityGML is defined by the thematic extension module Bridge (cf. chapter 7). Fig. 44 illustrates examples of bridge models in all LODs.



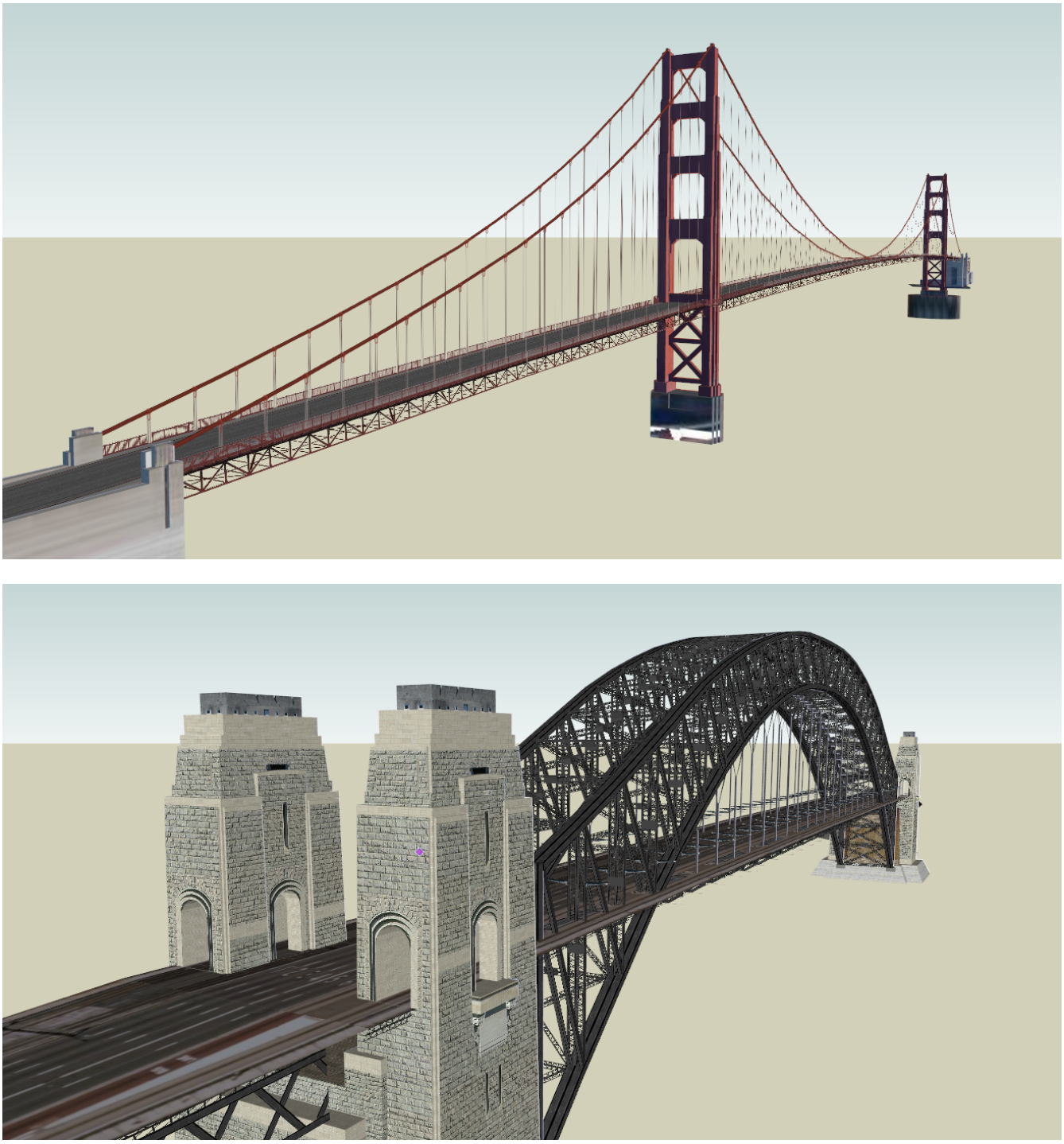


Figure 15. Examples for bridge models in LOD1 (upper left), LOD2 (upper right), LOD3 (lower left) and LOD4 (lower right) (source: Google 3D warehouse)

The bridge model was developed in analogy to the building model (cf. chapter 10.3) with regard to structure and attributes. The UML diagram of the bridge model is depicted in Fig. 45.

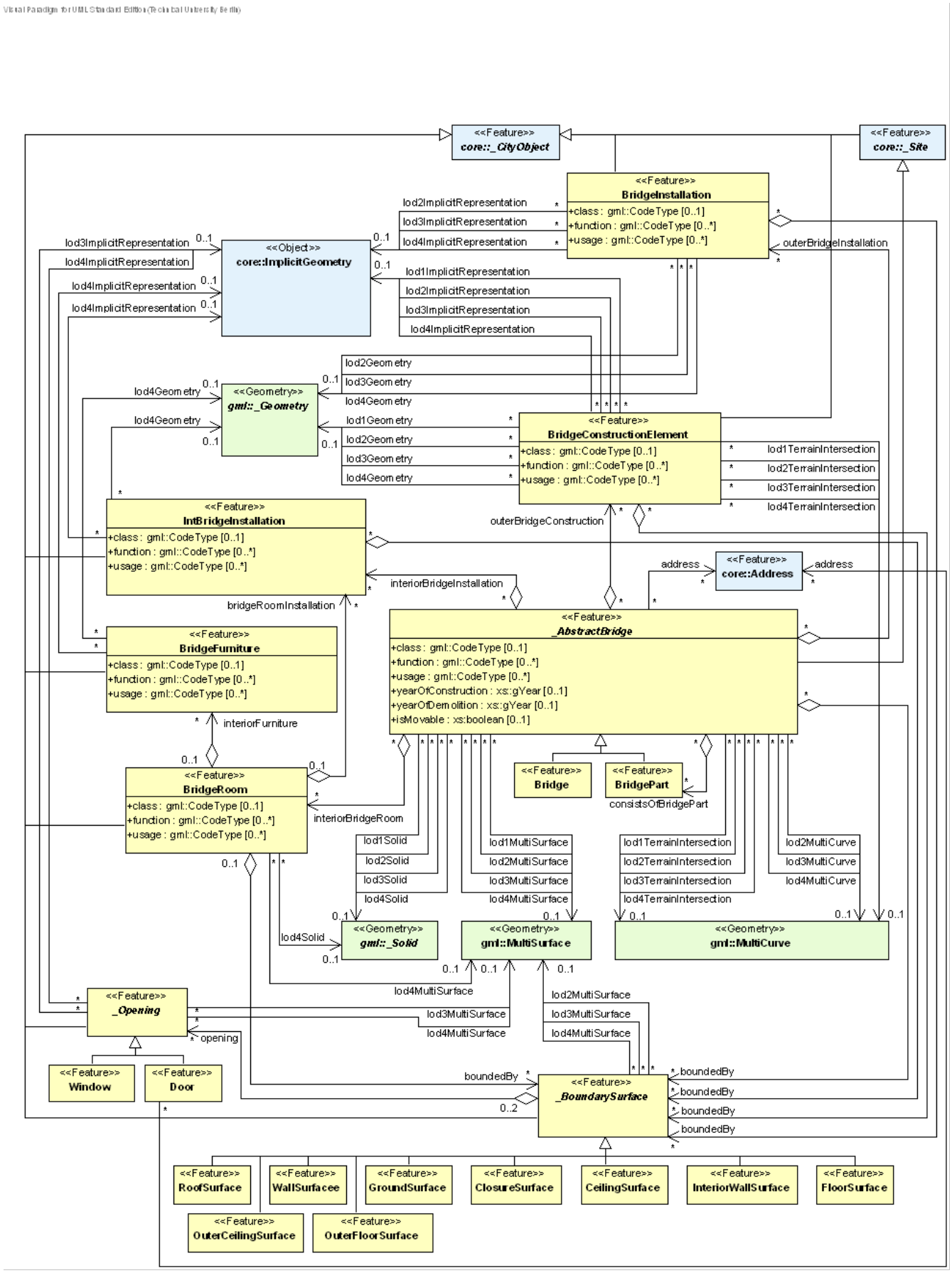
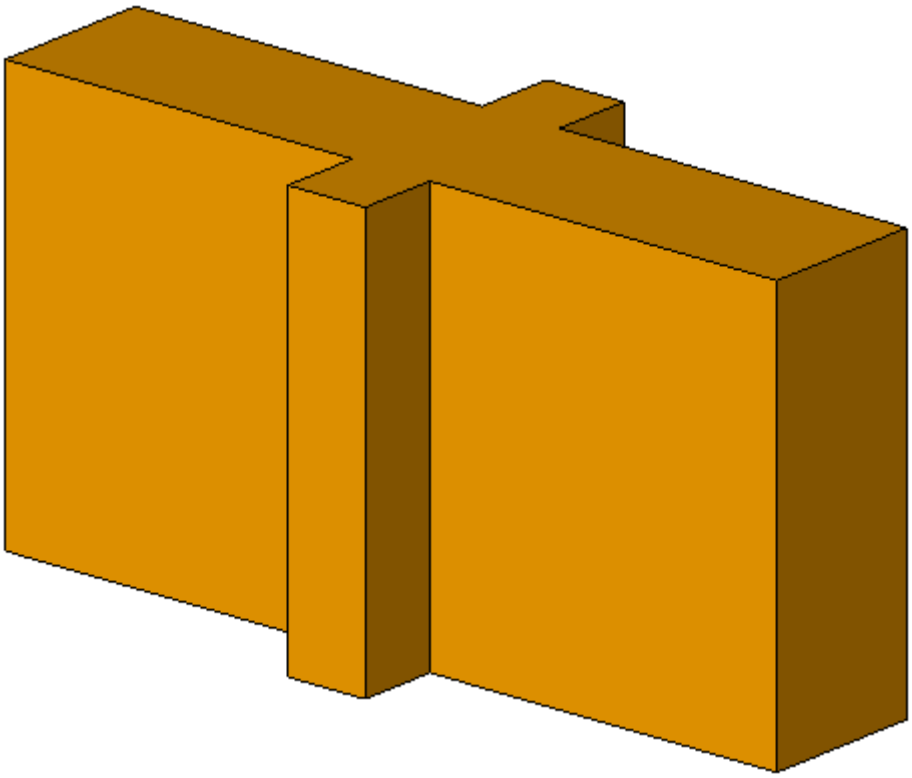
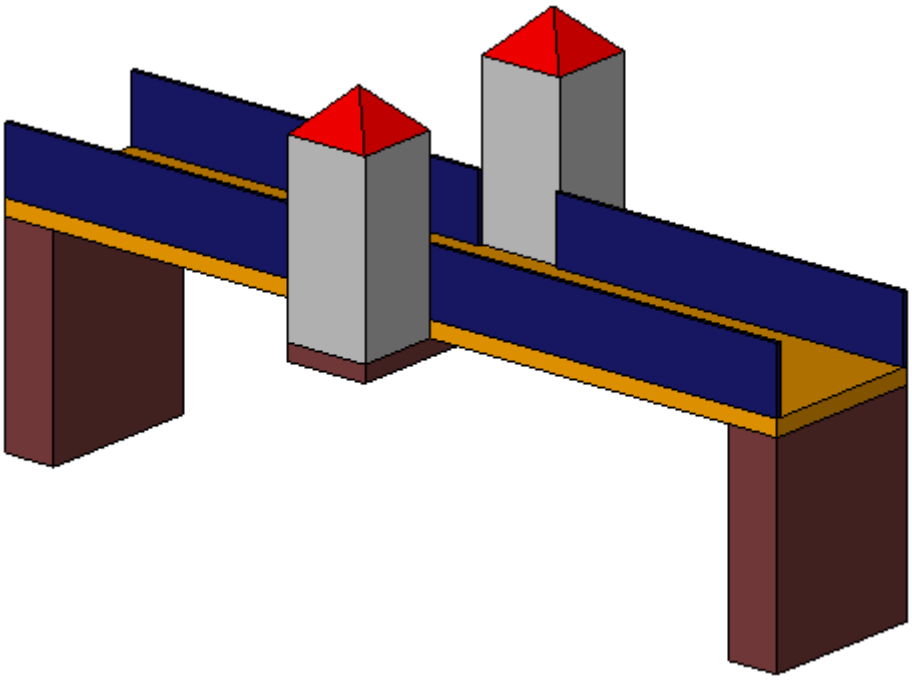


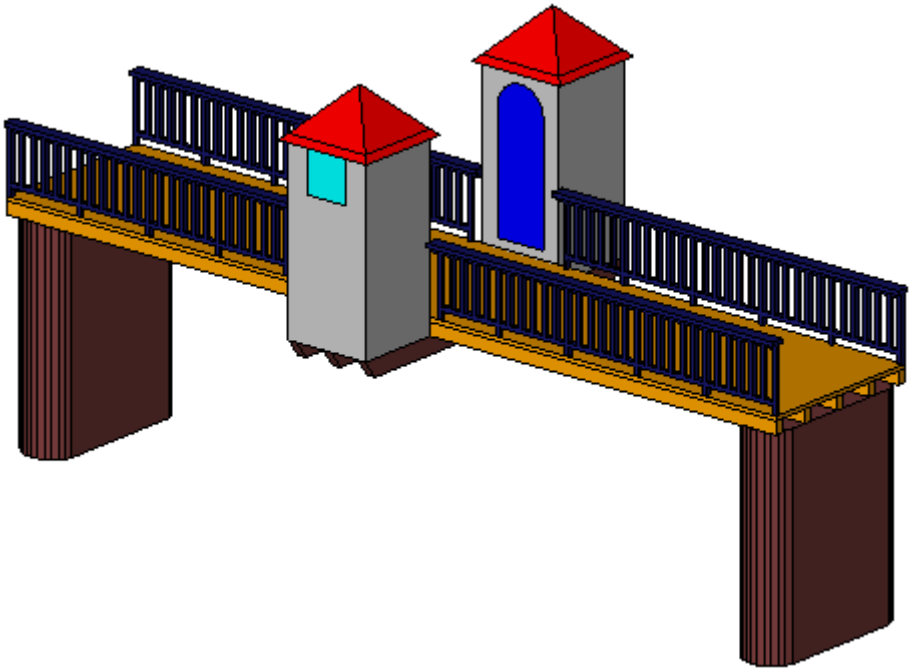
Figure 16. UML diagram of the bridge model, part one

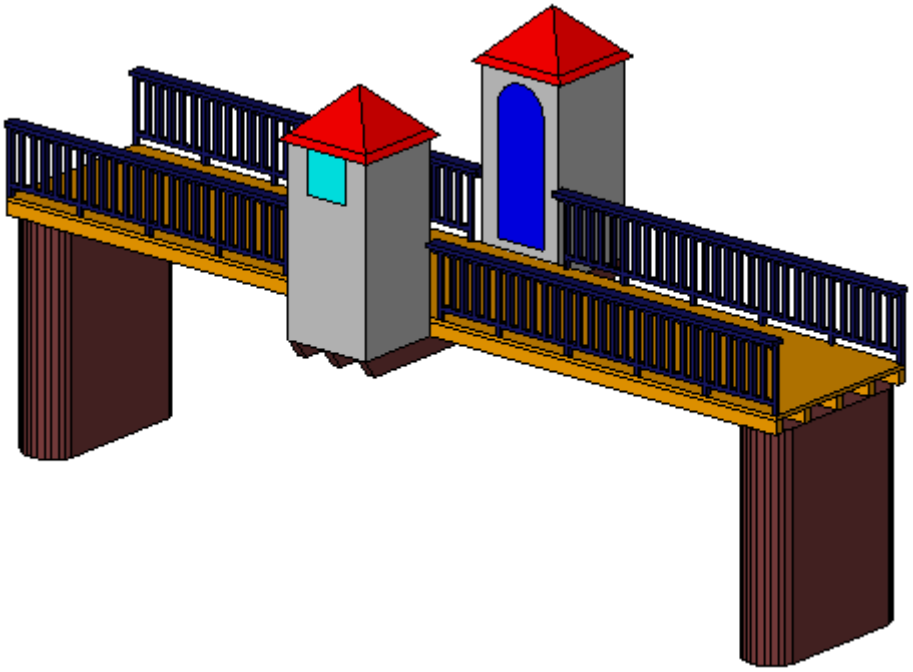
A (movable or unmovable) bridge is represented by an object of the class Bridge. This class inherits its attributes and relations from the abstract base class _AbstractBridge. The spatial properties are defined by a solid for each of the four LODs (relations lod1Solid to lod4Solid). In analogy to the

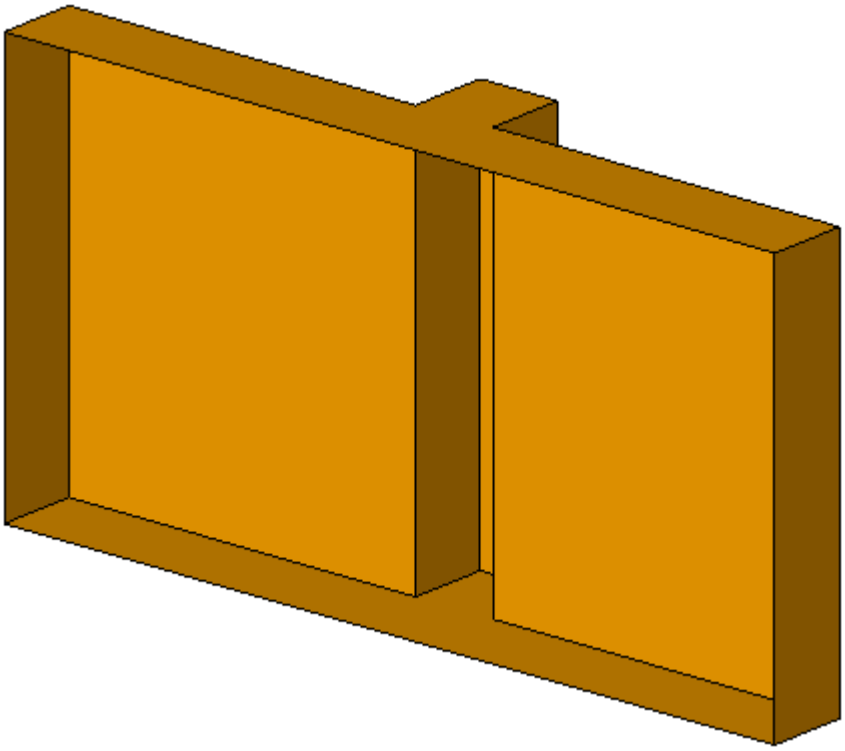
building model, the semantical as well as the geometrical richness increases from LOD1 (blocks model) to LOD3 (architectural model). Simple examples of bridges in each of those LODs are depicted in Fig. 46. Interior structures like rooms are dedicated to LOD4. To cover the case of bridge models where the topology does not satisfy the properties of a solid (essentially water tightness), a multi surface representation is allowed (lod1MultiSurface to lod4MultiSurface). The line where the bridge touches the terrain surface is represented by a terrain intersection curve, which is provided for each LOD (relations lod1TerrainIntersection to lod4TerrainIntersection). In addition to the solid representation of a bridge, linear characteristics like ropes or antennas can be specified geometrically by the lod1MultiCurve to lod4MultiCurve relations. If those characteristics shall be represented semantically, the features BridgeInstalla-tion or BridgeConstructionElement can be used (see section 10.5.2). All relations to semantic objects and geo-metric properties are listed in Tab. 7.

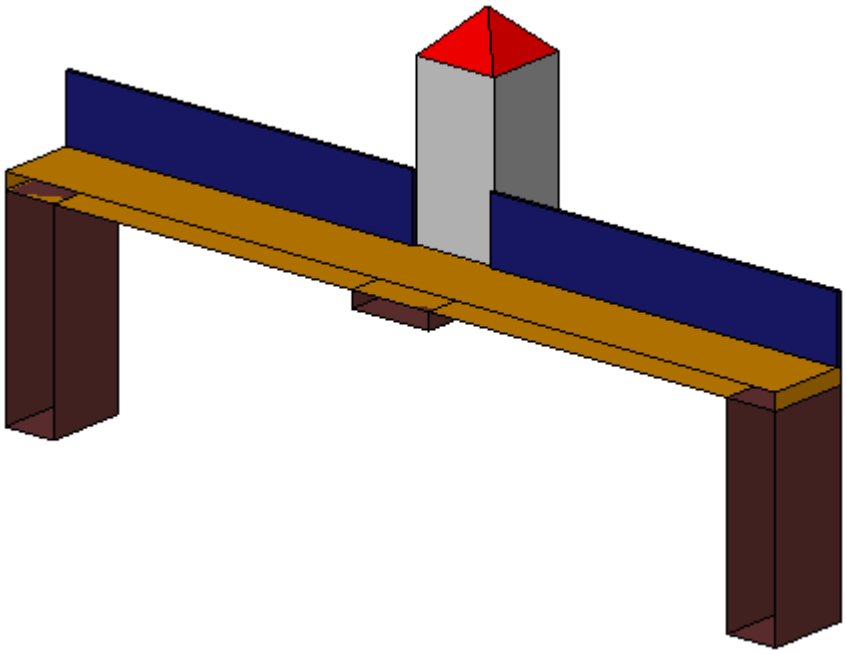


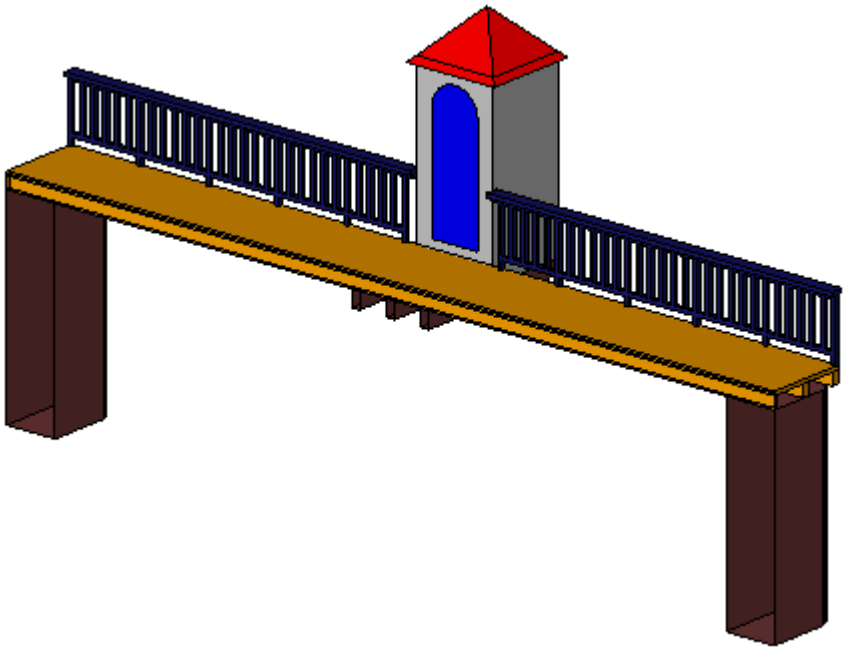


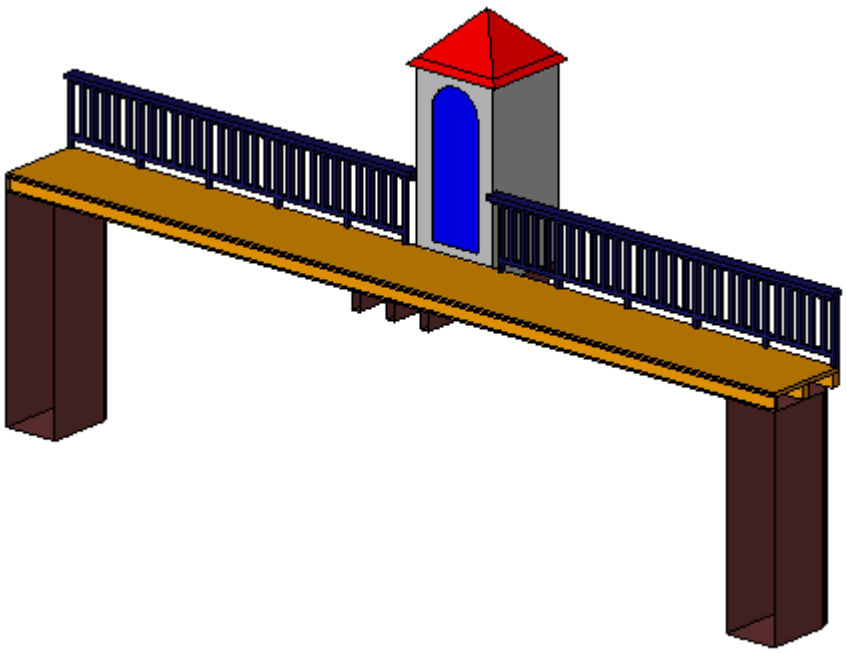


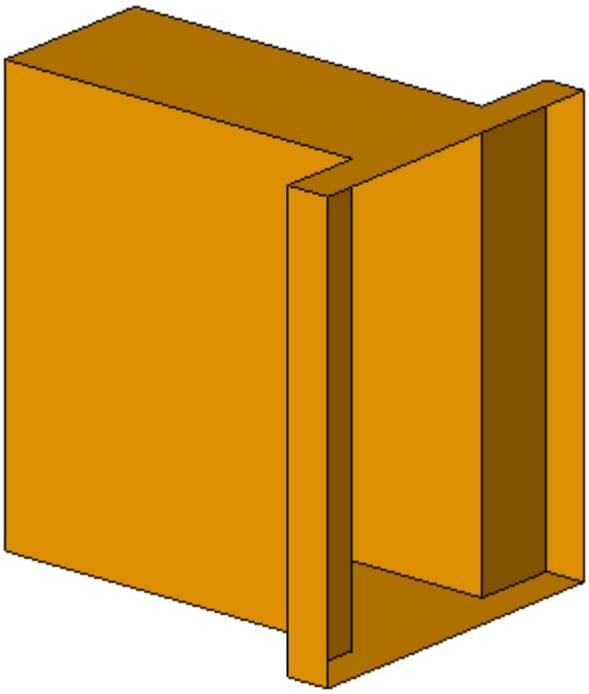


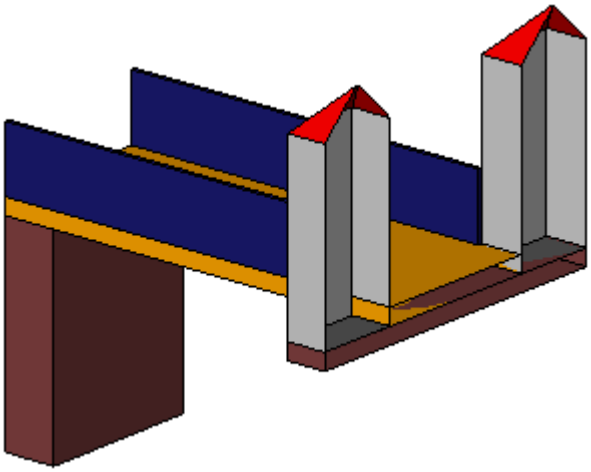


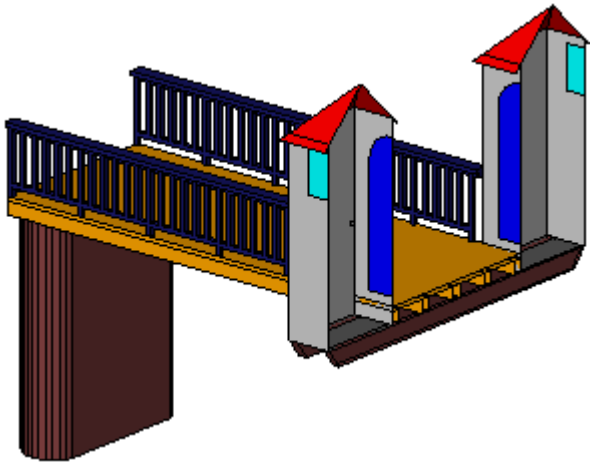












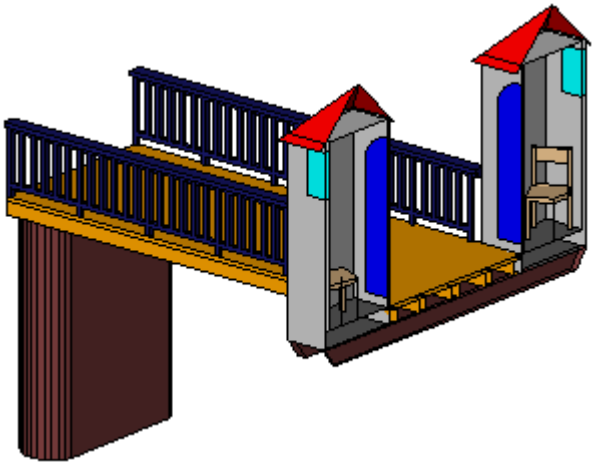


Figure 17. Bridge model in LOD1 – LOD4. (source: Karlsruhe Institute of Technology (KIT))

The semantic attributes of an _AbstractBridge are class, function, usage and is_movable. The attribute class is used to classify bridges, e.g. to distinguish different construction types (cf. Fig. 48). The attribute function allows representing the utilization of the bridge independently of the construction. Possible values may be railway bridge, roadway bridge, pedestrian bridge, aqueduct, etc. The option to denote a usage which is divergent to one of the primary functions of the bridge (function) is given by the attribute usage. The type of these attributes is gml:CodeType, the values of which can be defined in code lists. The name of the bridge can be represented by the gml:name attribute, which is inherited from the base class gml:_GML via the classes gml:_Feature, _CityObject, and _Site. Each Bridge or BridgePart feature may be assigned zero or more addresses using the address property. The corresponding AddressPropertyType is defined within the CityGML core module (cf. chapter 10.1.4).

Table 1. Semantic themes of the class _AbstractBridge

Geometric / semantic theme	Property type	LOD1	LOD2	LOD3	LOD4
Volume part of the bridge shell	gml:SolidType	•	•	•	•
Surface part of the bridge shell	gml:MultiSurfaceType	•	•	•	•
Terrain intersection curve	gml:MultiCurveType	•	•	•	•
Curve part of the bridge shell	gml:MultiCurveType		•	•	•
Bridge parts (chapter 10.5.1)	BridgePartType	•	•	•	•
Boundary surfaces (chapter 10.5.3)	AbstractBoundarySurfaceType		•	•	•
Outer bridge installations (chapter 10.5.2)	BridgeInstallationType		•	•	•

Geometric / semantic theme	Property type	LOD1	LOD2	LOD3	LOD4
Bridge construction elements (chapter 10.5.2)	BridgeConstruction-ElementType	•	•	•	•
Openings (chapter 10.5.4)	AbstractOpeningTyp e			•	•
Bridge rooms (chapter 10.5.5)	BridgeRoomType				•
Interior bridge installations	IntBridgeInstallation Type				•

The boolean attribute is_movable is defined to specify whether a bridge is movable or not. The modeling of the geometric aspects of the movement is delayed to later versions of this standard. Some types of movable bridges are depicted in Fig. 47.

NOTE	the following are annimated GIFs

[Figure 47 1] | *figures/inwork/Figure_47_1.gif*

[Figure 47 2] | *figures/inwork/Figure_47_2.gif*

[Figure 47 3] | *figures/inwork/Figure_47_3.gif*

[Figure 47 4] | *figures/inwork/Figure_47_4.gif*

[Figure 47 5] | *figures/inwork/Figure_47_5.gif*

Figure 18. Examples for movable bridges (source: ISO 6707).

NOTE	*Examples for different types of bridges.* insert Fig 48 - currently these do not render.

6.4.4. Bridge and bridge part

BridgeType, Bridge

NOTE	insert BridgeType, Bridge UML

BridgePartType, BridgePart

NOTE	insert BridgePartType, BridgePart UML

If some parts of a bridge differ from the remaining bridge with regard to attribute values or if parts like ramps can be identified as objects of their own, those parts can be represented as BridgePart. A bridge can consist of multiple BridgeParts. Like Bridge, BridgePart is a subclass of _AbstractBridge and hence, has the same attrib-utes and relations. The relation consistOfBridgePart represents the

aggregation hierarchy between a Bridge (or a BridgePart) and it's BridgeParts. By this means, an aggregation hierarchy of arbitrary depth can be modeled. Each BridgePart belongs to exactly one Bridge (or BridgePart). Similar to the building model, the aggregation structure of a bridge forms a tree. A simple example for a bridge with parts is a twin bridge. Another example is presented in chapter 10.5.6.

AbstractBridgeType, _AbstractBridge

NOTE	insert AbstractBridgeType, _AbstractBridge UML

The abstract class _AbstractBridge is the base class of Bridges and BridgeParts. It contains properties for bridge attributes, purely geometric representations, and geometric/semantic representations of the bridge or bridge part in different levels of detail. The attributes describe:

1. The classification of the bridge or bridge part (class), the different intended usages (function), and the different actual usages (usage). The permitted values for these property types can be specified in code lists.
2. The year of construction (yearOfConstruction) and the year of demolition (yearOfDemolition) of the bridge or bridge part. These attributes can be used to describe the chronology of the bridge development within a city model. The points of time refer to real world time.
3. Whether the bridge is movable is specified by the Boolean attribute isMovable.

6.4.5. Bridge construction elements and bridge installations

BridgeConstructionElementType, BridgeConstructionElement

NOTE	insert BridgeConstructionElementType, BridgeConstructionElement UML

BridgeInstallationType, BridgeInstallation

NOTE	insert BridgeInstallationType, BridgeInstallation UML

Bridge elements which do not have the size, significance or meaning of a BridgePart can be modelled either as BridgeConstructionElement or as BridgeInstallation. Elements which are essential from a structural point of view are modelled as BridgeConstructionElement, for example structural elements like pylons, anchorages etc. (cf. Fig. 49). A general classification as well as the intended and actual function of the construction element are represented by the attributes class, function, and usage. The geometry of a BridgeConstructionElement, which may be present in LOD1 to LOD4, is gml:_Geometry. Alternatively, the geometry may be given as ImplicitGe-ometry object. Following the concept of ImplicitGeometry the geometry of a prototype bridge construction element is stored only once in a local coordinate system and referenced by other bridge construction element features (cf. chapter 8.2). The visible surfaces of a bridge construction element can be semantically classified using the concept of boundary surfaces (cf. chapter 10.5.3).

Whereas a BridgeConstructionElement has structural relevance, a BridgeInstallation represents an element of the bridge which can be eliminated without collapsing of the bridge (e.g. stairway, antenna, railing). BridgeInstalla-tions occur in LOD 2 to 4 only and are geometrically representated

as gml:_Geometry. Again, the concept of ImplicitGeometry can be applied to BridgeInstallations alternatively, and their visible surfaces can be semanti-cally classified using the concept of boundary surfaces (cf. chapter 10.5.3). The class BridgeInstallation contains the semantic attributes class, function and usage. The attribute class gives a classification of installations of a bridge. With the attributes function and usage, nominal and real functions of the bridge installation can be described. The type of all attributes is gml:CodeType and their values can be defined in code lists.

NOTE	*BridgeConstructionElements of a suspension bridge.* insert Fig 49 - currently does not render

6.4.6. Boundary surfaces

AbstractBoundarySurfaceType, _BoundarySurface

NOTE	insert AbstractBoundarySurfaceType, _BoundarySurface UML

The thematic boundary surfaces of a bridge are defined in analogy to the building module. _BoundarySurface is the abstract base class for several thematic classes, structuring the exterior shell of a bridge as well as the visible surfaces of rooms, bridge construction elements and both outer and interior bridge installations. It is a subclass of _CityObject and thus inherits all properties like the GML3 standard feature properties (gml:name etc.) and the CityGML specific properties like ExternalReferences. From _BoundarySurface, the thematic classes RoofSur-face, WallSurface, GroundSurface, OuterCeilingSurface, OuterFloorSurface, ClosureSurface, FloorSurface, InteriorWallSurface, and CeilingSurface are derived.

For each LOD between 2 and 4, the geometry of a _BoundarySurface may be defined by a different gml:MultiSurface geometry.

In LOD3 and LOD4, a _BoundarySurface may contain _Openings (cf. chapter 10.5.4) like doors and windows. If the geometric location of _Openings topologically lies within a surface component (e.g. gml:Polygon) of the gml:MultiSurface geometry, these _Openings must be represented as holes within that surface. A hole is repre-sented by an interior ring within the corresponding surface geometry object. According to GML3, the points have to be specified in reverse order (exterior boundaries counter-clockwise and interior boundaries clockwise when looking in opposite direction of the surface's normal vector). If such an opening is sealed by a Door, a Window, or a ClosureSurface, their outer boundary may consist of the same points as the inner ring (denoting the hole) of the surrounding surface. The embrasure surfaces of an Opening belong to the relevant adjacent _BoundarySurface. If, for example a door seals the Opening, the embrasure surface on the one side of the door belongs to the InteriorWallSurface and on the other side to the WallSurface.

Fig. 50 depicts a bridge with RoofSurfaces, WallSurfaces, OuterFloorSurfaces and OuterCeilingSurfaces. Besides Bridges and BridgeParts, BridgeConstructionElements, BridgeInstallations as well as IntBridgeInstalla-tions can be related to _BoundarySurface. _BoundarySurfaces occur in LOD2 to LOD4. In LOD3 and LOD4, such a surface may contain _Openings (see chapter 10.3.4) like doors and windows.

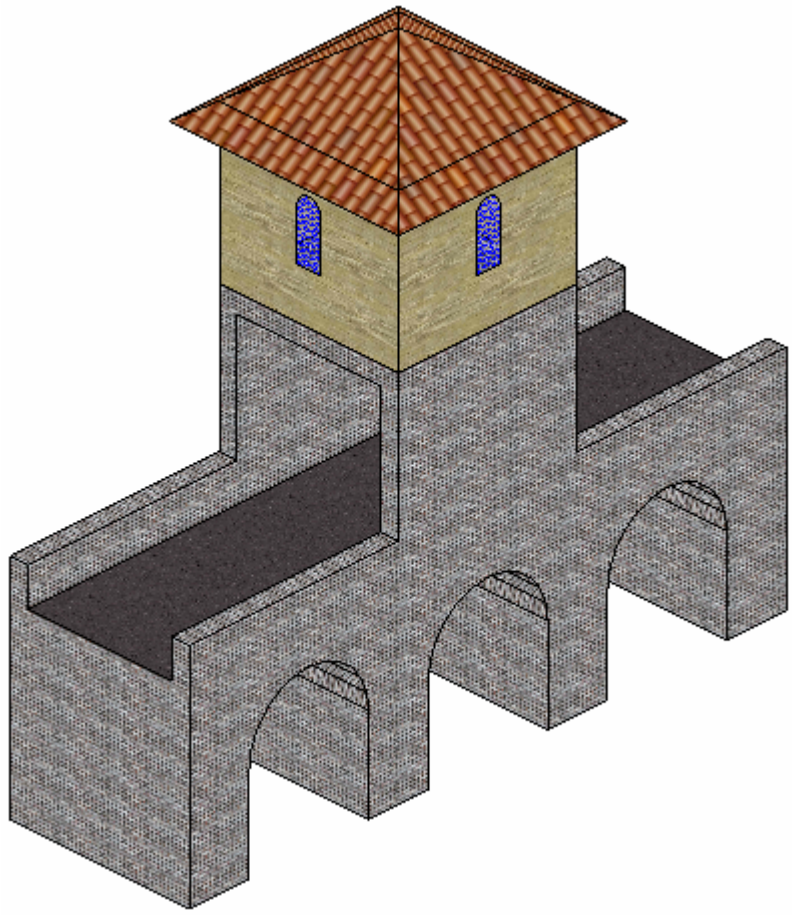


Figure 19. Different BoundarySurfaces of a bridge.

NOTE	need to add annotations to Figure 50

GroundSurfaceType, GroundSurface

NOTE	insert GroundSurfaceType, GroundSurface UML

The ground plate of a bridge or bridge part is modelled by the class GroundSurface. The polygon defining the ground plate is congruent with the bridge's footprint. However, the surface normal of the ground plate is point-ing downwards.

OuterCeilingSurfaceType, OuterCeilingSurface

Note: insert OuterCeilingSurfaceType, OuterCeilingSurface UML

A mostly horizontal surface belonging to the outer bridge shell and having the orientation pointing downwards can be modeled as an OuterCeilingSurface.

WallSurfaceType, WallSurface

NOTE	insert WallSurfaceType, WallSurface UML

All parts of the bridge facade belonging to the outer bridge shell can be modelled by the class WallSurface

OuterFloorSurfaceType, OuterFloorSurface

NOTE	insert OuterFloorSurfaceType, OuterFloorSurface UML

A mostly horizontal surface belonging to the outer bridge shell and with the orientation pointing upwards can be modeled as an OuterFloorSurface

RoofSurfaceType, RoofSurface

NOTE	insert RoofSurfaceType, RoofSurface UML

The major roof parts of a bridge or bridge part are expressed by the class RoofSurface.

ClosureSurfaceType, ClosureSurface

NOTE	insert ClosureSurfaceType, ClosureSurface UML

An opening in a bridge not filled by a door or window can be sealed by a virtual surface called ClosureSurface (cf. chapter 6.4). Hence, bridge with open sides can be virtually closed in order to be able to compute their volume. ClosureSurfaces are also used in the interior bridge model. If two rooms with a different are directly connected without a separating door, a ClosureSurface should be used to separate or connect the volumes of both rooms.

FloorSurfaceType, FloorSurface

NOTE	insert FloorSurfaceType, FloorSurface UML

The class FloorSurface must only be used in the LOD4 interior bridge model for modelling the floor of a bridge room.

InteriorWallSurfaceType, InteriorWallSurface

NOTE	insert InteriorWallSurfaceType, InteriorWallSurface UML

The class InteriorWallSurface must only be used in the LOD4 interior bridge model for modelling the visible surfaces of the bridge room walls.

CeilingSurfaceType, CeilingSurface

NOTE	insert CeilingSurfaceType, CeilingSurface UML

The class CeilingSurface must only be used in the LOD4 interior bridge model for modelling the ceiling of a bridge room.

6.4.7. Openings

AbstractOpeningType, _Opening

NOTE	insert AbstractOpeningType, _Opening UML

The class _Opening is the abstract base class for semantically describing openings like doors or windows in outer or inner boundary surfaces like walls and roofs. Openings only exist in models of LOD3 or LOD4. Each _Opening is associated with a gml:MultiSurface geometry. Alternatively, the geometry may be given as Implic-itGeometry object. Following the concept of ImplicitGeometry the geometry of a prototype opening is stored only once in a local coordinate system and referenced by other opening features (see chapter 8.2).

WindowType, Window

NOTE	insert WindowType, Window UML

The class Window is used for modelling windows in the exterior shell of a bridge, or hatches between adjacent rooms. The formal difference between the classes Window and Door is that – in normal cases – Windows are not specifically intended for the transit of people or vehicles.

DoorType, Door

NOTE	insert DoorType, Door UML

The class Door is used for modelling doors in the exterior shell of a bridge, or between adjacent rooms. Doors can be used by people to enter or leave a bridge or room. In contrast to a ClosureSurface a door may be closed, blocking the transit of people. A Door may be assigned zero or more addresses. The corresponding Address-PropertyType is defined within the CityGML core module (cf. chapter 10.1.4).

6.4.8. Bridge Interior

The classes BridgeRoom, IntBridgeInstallation and BridgeFurniture allow for the representation of the bridge interior. They are designed in analogy to the classes Room, IntBuildingInstallation and BuildingFurniture of the building module and share the same meaning. The bridge interior can only be modeled in LOD4.

BridgeRoomType, BridgeRoom

NOTE	insert BridgeRoomType, BridgeRoom UML

A BridgeRoom is a semantic object for modelling the free space inside a bridge and should be uniquely related to exactly one bridge or bridge part object. It should be closed (if necessary by using ClosureSurfaces) and the geometry normally will be described by a solid (lod4Solid). However, if the topological correctness of the boundary cannot be guaranteed, the geometry can alternatively be given as a MultiSurface (lod4MultiSurface). The surface normals of the outer shell of a GML solid must point outwards. This is important to consider when BridgeRoom surfaces should be assigned Appearances. In this case, textures and colors must be placed on the backside of the corresponding surfaces in order to be visible from the inside of the room.

In addition to the geometrical representation, different parts of the visible surface of a room can be modelled by specialised BoundarySurfaces (FloorSurface, CeilingSurface, InteriorWallSurface, and ClosureSurface; cf. chapter 10.5.3).

BridgeFurnitureType, BridgeFurniture

NOTE	insert BridgeFurnitureType, BridgeFurniture UML

BridgeRooms may have BridgeFurnitures and IntBridgeInstallations. A BridgeFurniture is a movable part of a room, such as a chair or furniture. A BridgeFurniture object should be uniquely related to exactly one room object. Its geometry may be represented by an explicit geometry or an ImplicitGeometry object. Following the concept of ImplicitGeometry the geometry of a prototype bridge furniture is stored only once in a local coordi-nate system and referenced by other bridge furniture features (see chapter 8.2).

IntBridgeInstallationType, IntBridgeInstallation

NOTE	insert IntBridgeInstallationType, IntBridgeInstallation UML

An IntBridgeInstallation is an object inside a bridge with a specialised function or semantic meaning. In contrast to BridgeFurniture, IntBridgeInstallations are permanently attached to the bridge structure and cannot be moved. Examples for IntBridgeInstallations are stairways, railings and heaters. Objects of the class IntBridgeInstallation can either be associated with a room (class BridgeRoom), or with the complete bridge / bridge part (class _AbstractBridge, cf. chapter 10.5.1). However, they should be uniquely related to exactly one room or one bridge / bridge part object. An IntBridgeInstallation optionally has attributes class, function and usage. The attribute class, which can only occur once, represents a general classification of the internal bridge component. With the attributes function and usage, nominal and real functions of a bridge installation can be described. For all three attributes the list of feasible values can be specified in a code list. For the geometrical representation of an IntBridgeInstallation, an arbitrary geometry object from the GML subset shown in Fig. 9 can be used. Alter-natively, the geometry may be given as ImplicitGeometry object. Following the concept of ImplicitGeometry the geometry of a prototype interior bridge installation is stored only once in a local coordinate system and refer-enced by other interior bridge installation features (see chapter 8.2). The visible surfaces of an interior bridge installation can be semantically classified using the concept of boundary surfaces (cf. 10.5.3).

6.4.9. Level of Detail

6.4.10. UML Model

6.4.11. Examples

The bridge of Rees crossing the Rhine in Germany has three bridge parts which are separated by pylons. Fig. 51 (left) depicts the Rees bridge model containing one Bridge feature which consists of three BridgePart features. The pylons, which are structurally essential, are represented by BridgeConstructionElements. On the top of the pylons, four lamps are located which are modeled as BridgeInstallation features (cf. right part of Fig. 51).

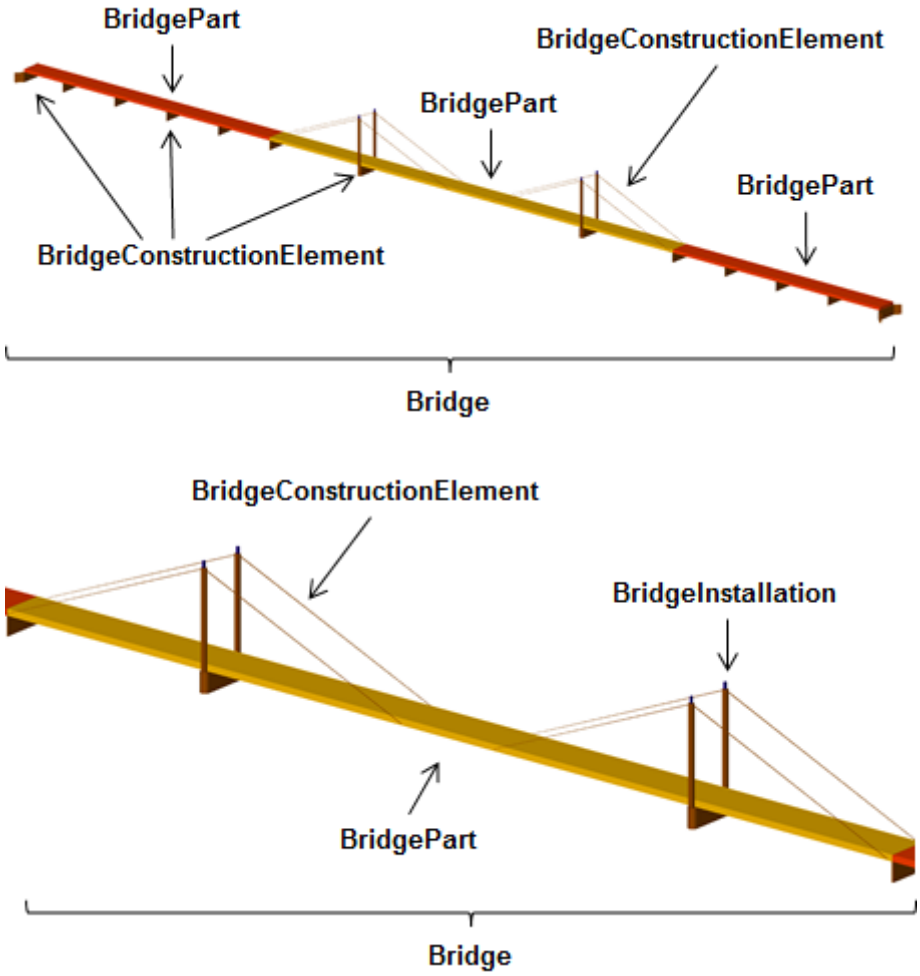


Figure 20. The bridge of Rees, consisting of a Bridge feature and three BridgePart features (left). The bridge contains BridgeConstructionElement and BridgeInstallation features (right).

In the following Fig. 52, the main part of the bridge of Rees is shown as photograph on the left side (source: Harald Halfpapp), and the corresponding part of the LOD2 bridge model is depicted on the the right side (source: District of Recklinghausen / KIT).

NOTE Figures 52, 53 and 54 are from the images folder.

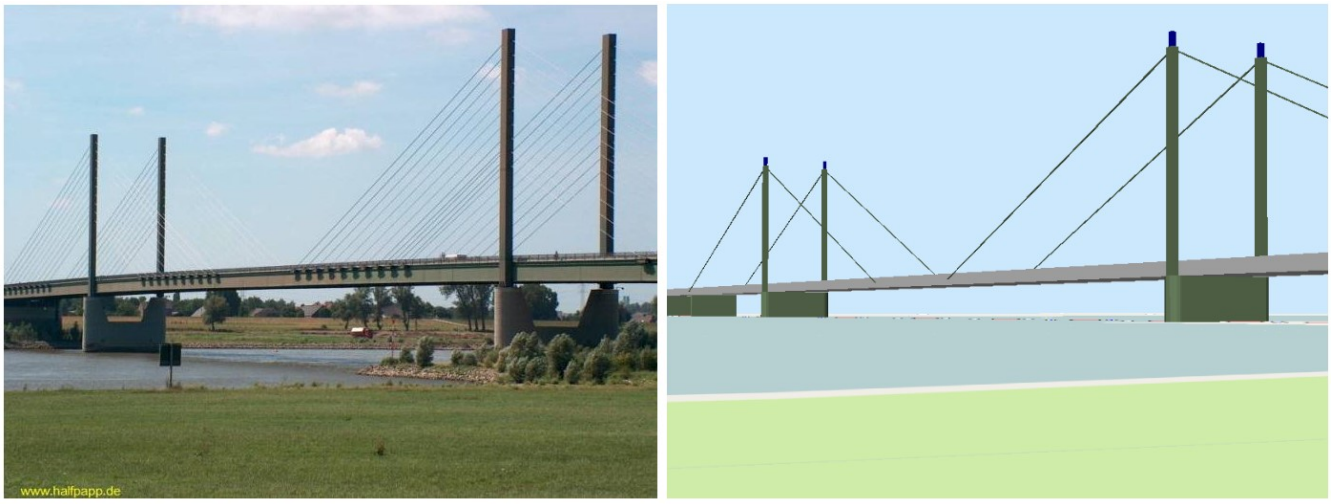


Figure 21. The bridge of Rees (left photo (source: Harald Halfpapp); right LOD2 model (source: District of Recklinghausen / KIT)).

There are two bridges crossing the river Rhine at Karlsruhe, Germany. The first one is a two track railway bridge constructed as a truss bridge (cf. Fig. 53 front). The second one is a four lane highway bridge constructed as a cable-stayed bridge (cf. Fig. 53 background).

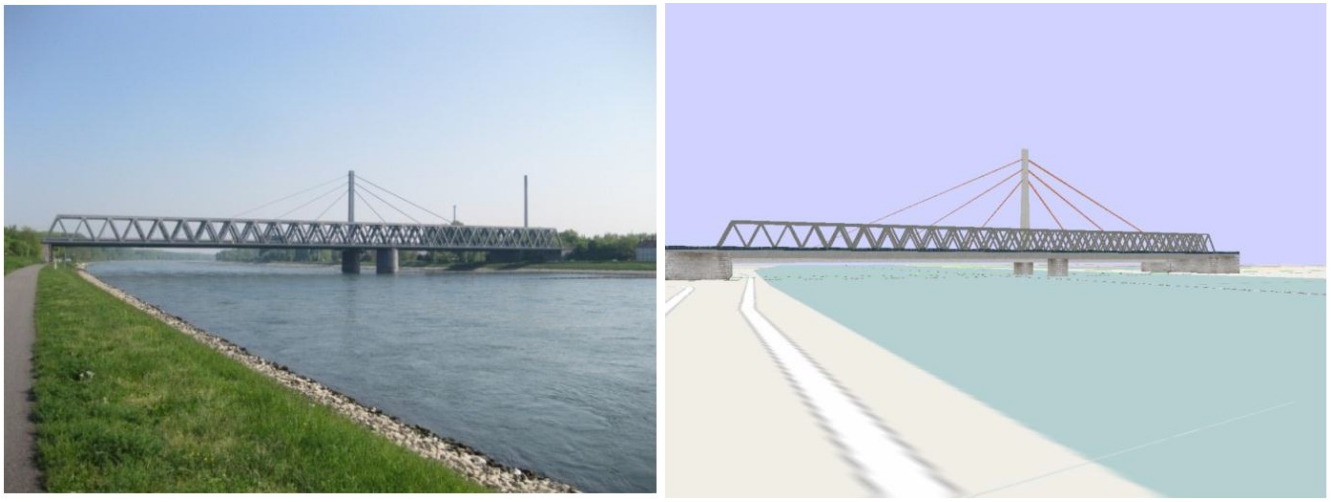


Figure 22. Bridge over the river Rhine at Karlsruhe (left a photo, right the 3D CityGML model) (source: Karlsruhe Institute of Technology (KIT), courtesy of City of Karlsruhe).

In CityGML both bridges are modeled as single Bridge object with BridgeConstructionElements and BridgeIn-stallations. The construction elements of the cable stayed bridge are the footings on both river sides and in the middle of the river, as well as the cables and the pylon. The construction elements of the truss bridge are the footings and the truss itself. Both bridges have several railings which are modeled as BridgeInstallation. The bridge “Oberbaumbrücke” shown in Fig. 54 is located in the centre of Berlin crossing the river Spree and serves as example for bridges having interior rooms. The real-world bridge is depicted in the left part of Fig. 54, whereas the corresponding CityGML model is shown on the right. The outer geometry of the bridge is modeled as gml:MultiSurface element (lod4MultiSurface property) and is assigned photorealistic textures. Additionally, the interior rooms located in both bridge towers are represented as BridgeRoom objects with solid geometries (gml:Solid assigned through the lod4Solid property). Due to its geometric accuracy and the representation of the interior structures of both bridge towers, the model is classified as LOD4.

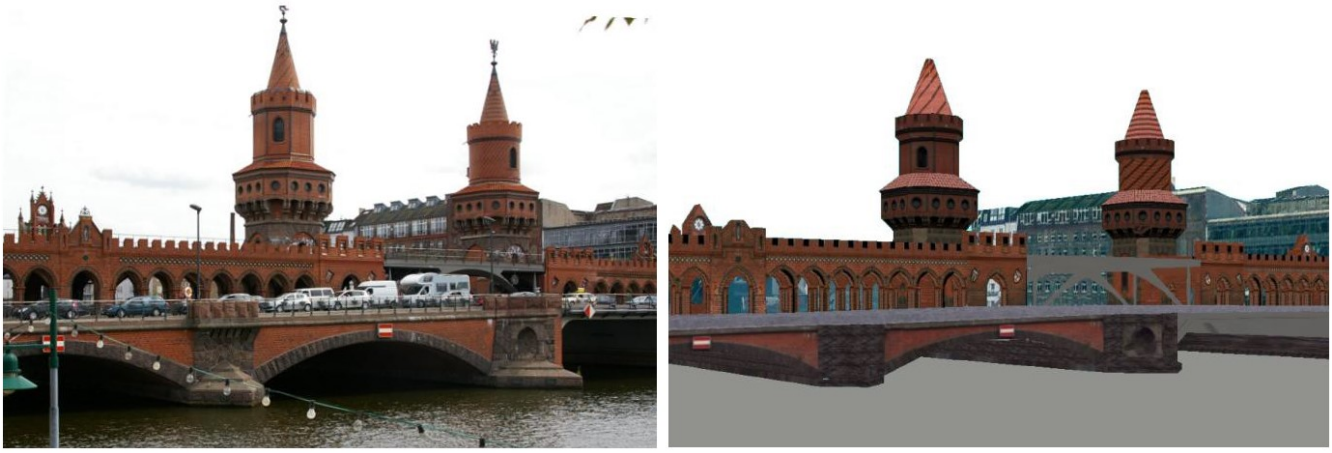


Figure 23. The bridge“Oberbaumbrücke” in Berlin represented as bridge model in LOD4 (left a photo, right the 3D CityGML model) (source: Berlin Senate of Business, Technology and Women; Business Location Center, Berlin; Technische Universität Berlin; Karlsruhe Institute of Technology (KIT)).

6.5. Building Model

Contributors
TBD

NOTE The following text needs to be reviewed and updated.

The building model is one of the most detailed thematic concepts of CityGML. It allows for the representation of thematic and spatial aspects of buildings and building parts in five levels of detail, LOD0 to LOD4. The building model of CityGML is defined by the thematic extension module Building (cf. chapter 7). Fig. 26 provides examples of 3D city and building models in LOD1 – 4.

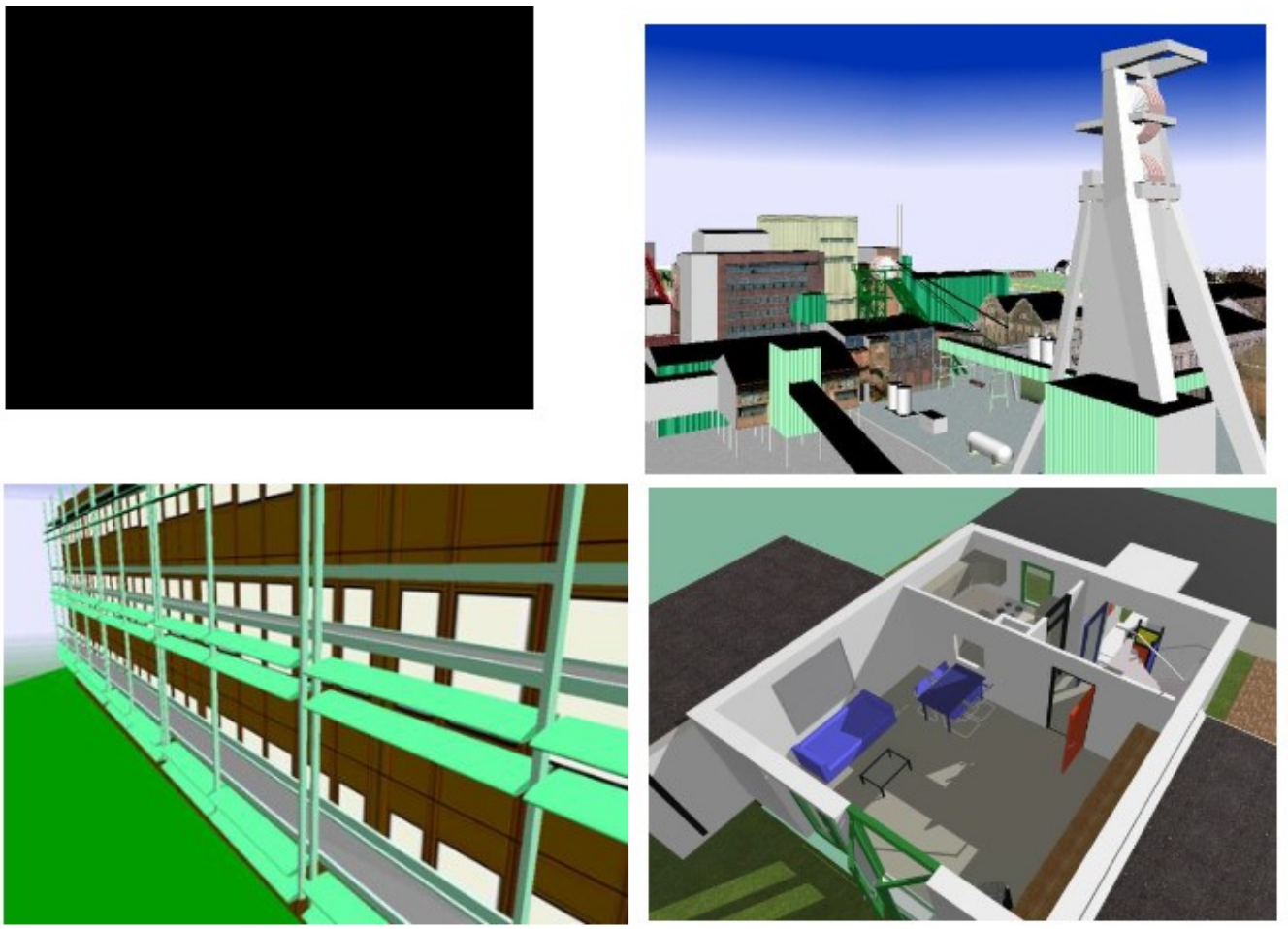


Figure 24. Examples for city or building models in LOD1 (upper left), LOD2 (upper right), LOD3 (lower left), and LOD4 (lower right) (source: District of Recklinghausen, m-g-h ingenieure+architekten GmbH).

NOTE this figure needs to be re-built.

The UML diagram of the building model is depicted in Fig. 27, for the XML schema definition see annex A.4 and below. The pivotal class of the model is _AbstractBuilding, which is a subclass of the thematic class _Site (and transitively of the root class _CityObject). _AbstractBuilding is specialised either to a Building or to a BuildingPart. Since an _AbstractBuilding consists of BuildingParts, which again are _AbstractBuildings, an aggregation hierarchy of arbitrary depth may be realised. As subclass of the root class _CityObject, an _AbstractBuilding inherits all properties from _CityObject like the GML3 standard feature properties (gml:name etc.) and the CityGML specific properties like

ExternalReferences (cf. chapter 6.7). Further properties not explicitly covered by _AbstractBuilding may be modelled as generic attributes provided by the CityGML Generics module (cf. chapter 10.12) or using the CityGML Application Domain Extension mechanism (cf. chapter 10.13).

Building complexes, which consist of a number of distinct buildings like a factory site or hospital complex, should be aggregated using the concept of CityObjectGroups (cf. chapter 6.8). The main building of the complex can be denoted by providing “main building” as the role name of the corresponding group member.

Both classes Building and BuildingPart inherit the attributes of _AbstractBuilding: the class of the building, the function (e.g. residential, public, or industry), the usage, the year of construction, the year of demolition, the roof type, the measured height, and the number and individual heights of the storeys above and below ground. This set of parameters is suited for roughly reconstructing the three-dimensional shape of a building and can be provided by cadastral systems. Furthermore, Address features can be assigned to Buildings or BuildingParts.

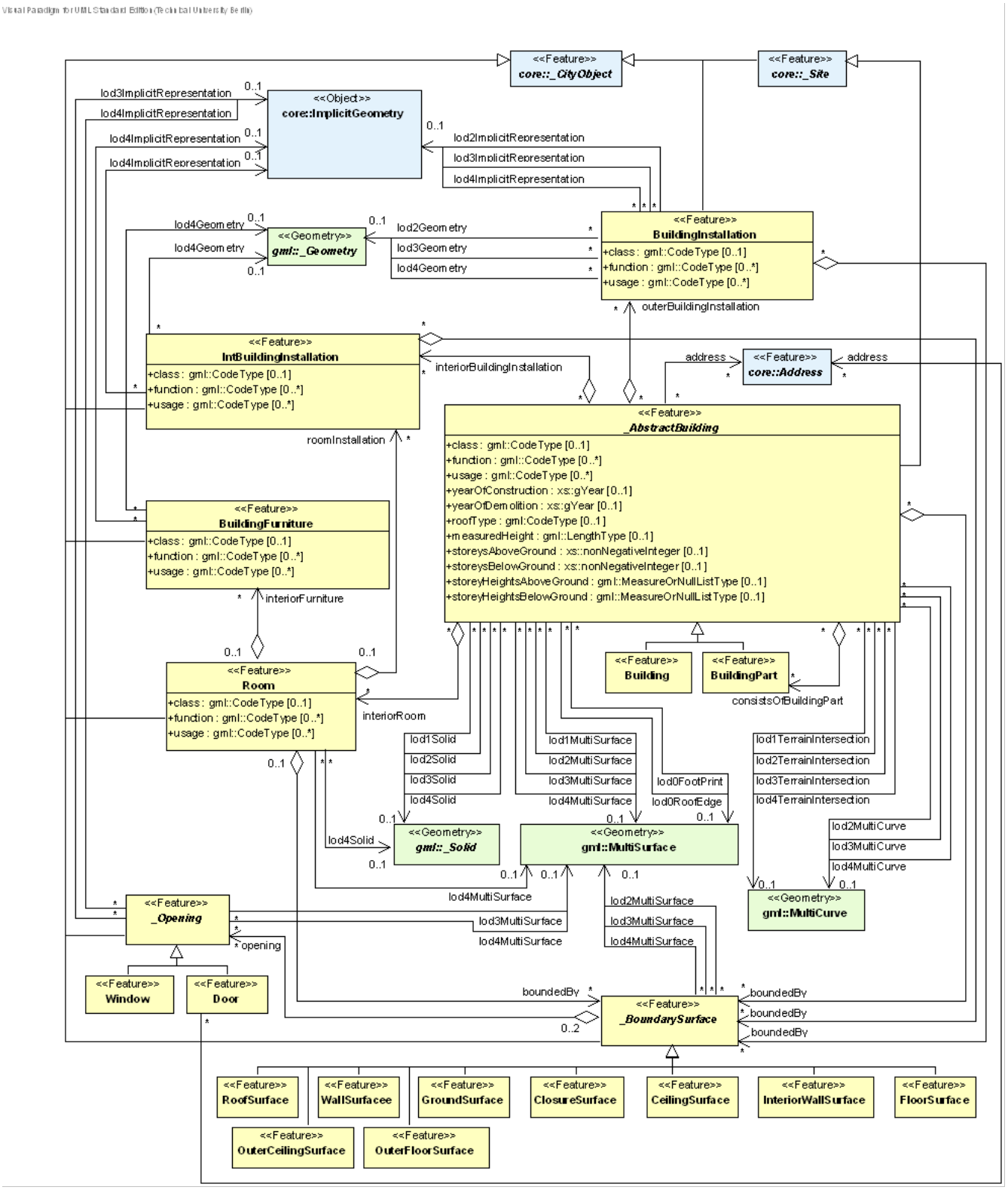


Figure 25. UML diagram of CityGML's building model. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Building module.

The geometric representation and semantic structure of an _AbstractBuilding is shown in Fig. 27. The model is successively refined from LOD0 to LOD4. Therefore, not all components of a building model are represented equally in each LOD and not all aggregation levels are allowed in each LOD. In CityGML, all object classes are associated to the LODs with respect to the proposed minimum acquisition criteria for each LOD (cf. chapter 6.2). An object can be represented simultaneously in different LODs by providing distinct geometries for the corresponding LODs.

In LOD0, the building can be represented by horizontal, 3-dimensional surfaces. These can represent the foot-print of the building and, separately, the roof edge. This allows the easy integration of 2D data into the model. In many countries these 2D geometries readily exist, for example in cadastral or topographic data holdings. Cadas-tre data typically depicts the shape of the building on the ground (footprints) and topographic data is often a mixture between footprints and geometries at roof level (roof edges), which are often photogrametrically extract-ed from area/satellite images or derived from airborne laser data. The building model allows the inclusion of both. In this case large overhanging roofs can be modelled as a preliminary stage to more detailed LOD2 and LOD3 depictions. The surface geometries require 3D coordinates, though it is mandated that the height values of all vertices belonging to the same surface are identical. If 2D geometries are imported into any of these two LOD0 geometries, an appropriate height value for all vertices needs to be chosen. The footprint is typically located at the lowest elevation of the ground surface of the building whereas the roof edge representation should be placed at roof level (e.g., eaves height).

In LOD1, a building model consists of a generalized geometric representation of the outer shell. Optionally, a gml:MultiCurve representing the TerrainIntersectionCurve (cf. chapter 6.5) can be specified. This geometric representation is refined in LOD2 by additional gml:MultiSurface and gml:MultiCurve geometries, used for modelling architectural details like roof overhangs, columns, or antennas. In LOD2 and higher LODs the outer facade of a building can also be differentiated semantically by the classes _BoundarySurface and BuildingInstal-lation. A _BoundarySurface is a part of the building's exterior shell with a special function like wall (WallSur-face), roof (RoofSurface), ground plate (GroundSurface), outer floor (OuterFloorSurface), outer ceiling (Outer-CeilingSurface) or ClosureSurface. The BuildingInstallation class is used for building elements like balconies, chimneys, dormers or outer stairs, strongly affecting the outer appearance of a building. A BuildingInstallation may have the attributes class, function, and usage (cf. Fig. 27).

In LOD3, the openings in _BoundarySurface objects (doors and windows) can be represented as thematic ob-jects. In LOD4, the highest level of resolution, also the interior of a building, composed of several rooms, is represented in the building model by the class Room. This enlargement allows a virtual accessibility of buildings, e.g. for visitor information in a museum ("Location Based Services"), the examination of accommodation standards or the presentation of daylight illumination of a building. The aggregation of rooms according to arbitrary, user defined criteria (e.g. for defining the rooms corresponding to a certain storey) is achieved by employing the general grouping concept provided by CityGML (cf. chapter 10.3.6). Interior installations of a building, i.e. objects within a building which (in contrast to furniture) cannot be moved, are represented by the class IntBuildingInstallation. If an installation is attached to a specific room (e.g. radiators or lamps), they are associated with the Room class, otherwise (e.g. in case of rafters or pipes) with _AbstractBuilding. A Room may have the attributes class, function and usage whose value can be defined in code lists (chapter 10.3.8 and annex C.1). The class attribute allows a classification of rooms with respect to the stated function, e.g. commercial or private rooms, and occurs only once. The function attribute is intended to express the main purpose of the room, e.g. living room, kitchen. The attribute usage can be used if the way the object is actually used differs from the function. Both attributes can occur multiple times.

The visible surface of a room is represented geometrically as a Solid or MultiSurface. Semantically, the surface can be structured into specialised _BoundarySurfaces, representing floor (FloorSurface), ceiling (CeilingSur-face), and interior walls (InteriorWallSurface). Room furniture,

like tables and chairs, can be represented in the CityGML building model with the class BuildingFurniture. A BuildingFurniture may have the attributes class, function and usage. Annexes G.1 to G.6 provide example CityGML documents containing a single building model which is subsequently refined from a coarse LOD0 representation up to a semantically rich and geomet-ric-topologically sound LOD4 model including the building interior.

6.5.1. Building and Building Part

NOTE	Version 2.0 uses XML schema to illustrate this section. Replace those schema with UML.

BuildingType, Building

NOTE	Insert BuildingType, Building UML

The Building class is one of the two subclasses of _AbstractBuilding. If a building only consists of one (homo-geneous) part, this class shall be used. A building composed of structural segments differing in, for example the number of storeys or the roof type has to be separated into one Building having one or more additional Build-ingPart (see Fig. 28). The geometry and non-spatial properties of the central part of the building should be represented in the aggregating Building feature.

BuildingType, Building Part

NOTE	Insert BuildingType, Building Part UML

The class BuildingPart is derived from _AbstractBuilding. It is used to model a structural part of a building (see Fig. 28). A BuildingPart object should be uniquely related to exactly one building or building part object.

Figure 26. Examples of buildings consisting of one and two building parts (source: City of Coburg)

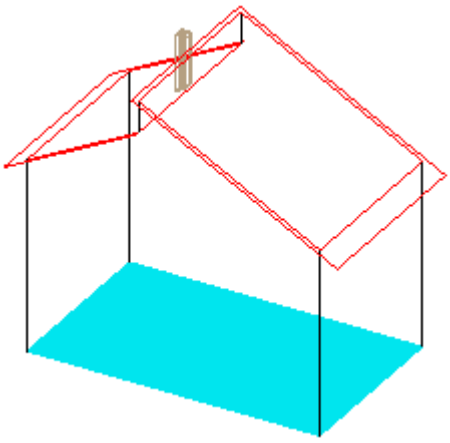
AbstractBuildingType, _AbstractBuilding

NOTE AbstractBuildingType, _AbstractBuilding UML

The abstract class _AbstractBuilding contains properties for building attributes, purely geometric representations, and geometric/semantic representations of the building or building part in different levels of detail. The attributes describe:

1. classification of the building or building part (class), the different intended usages (function), and the different actual usages (usage). The permitted values for these attributes can be specified in code lists.
2. The year of construction (yearOfConstruction) and the year of demolition (yearOfDemolition) of the build-ing or building part. These attributes can be used to describe the chronology of the building development within a city model. The points of time refer to real world time.
3. The roof type of the building or building part (roofType). The permitted values for this attribute can be specified in a code list.
4. The measured relative height (measuredHeight) of the building or building part.
5. The number of storeys above (storeyAboveGround) and below (storeyBelowGround) ground level.
6. The list of storey heights above (storeyHeightsAboveGround) and below (storeyHeightsBelowGround) ground level. The first value in a list denotes the height of the nearest storey wrt. to the ground level and last value the height of the farthest.

Spanning the different levels of detail, the building model differs in the complexity and granularity of the geo-metric representation and the thematic structuring of the model into components with a special semantic mean-ing. This is illustrated in Fig. 29 and Fig. 30, showing the same building in five different LODs. The class _AbstractBuilding has a number of properties which are associated with certain LODs.



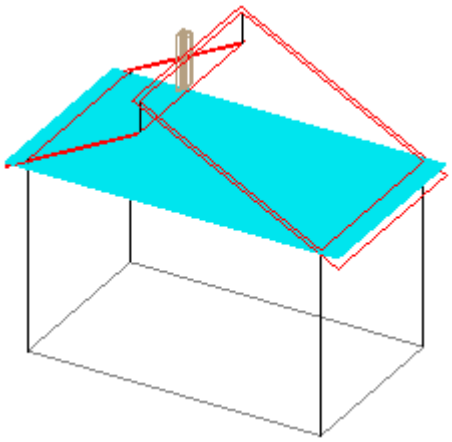


Figure 27. The two possibilities of modeling a building in LOD0 using horizontal 3D surfaces. On the left, the building footprint (lod0FootPrint) is shown (cyan) which denotes the shape of the building on the ground. The corresponding surface representation is located at ground level. On the right, the lod0RoofEgde representation is illustrated (cyan) which results from a horizontal projection of the building's roof and which is located at the eaves height (source: Karlsruhe Institute of Technology (KIT), courtesy of Franz-Josef Kaiser).

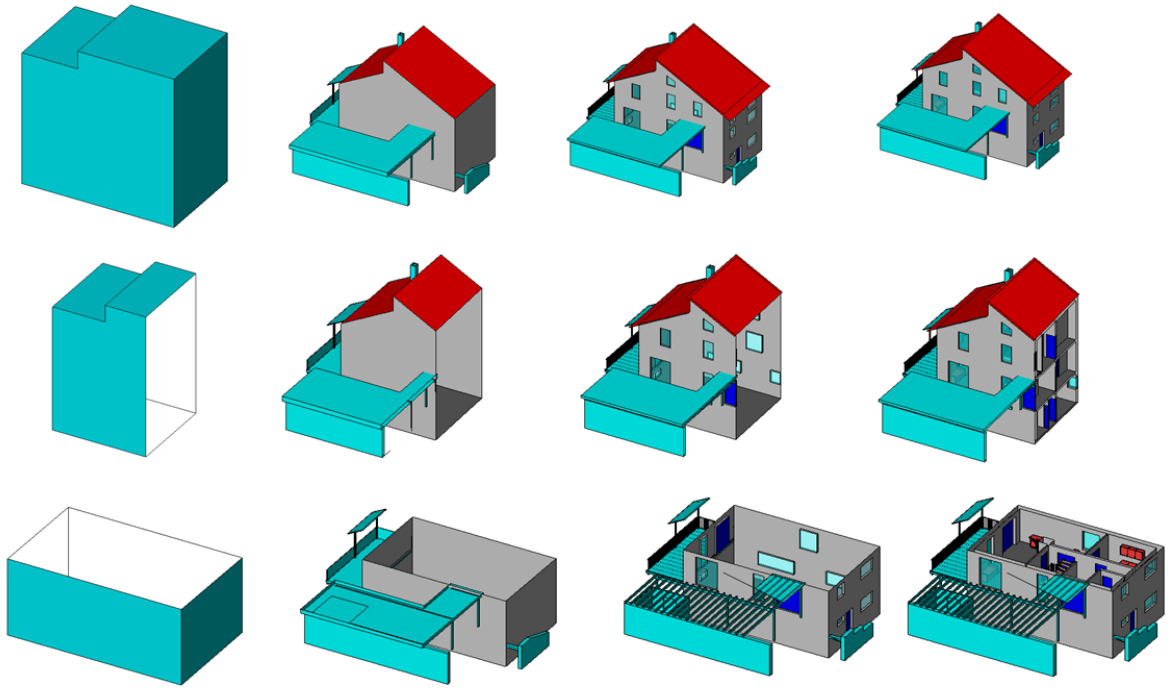


Figure 28. Building model in LOD1 – LOD4 (source: Karlsruhe Institute of Technology (KIT), courtesy of Franz-Josef Kaiser).<o:p></o:p>

Tab. 5 shows the correspondence of the different geometric and semantic themes of the building model to LODs. In LOD1 – 4, the volume of a building can be expressed by a gml:Solid geometry and/or a gml:MultiSurface geometry. The definition of a 3D Terrain Intersection Curve (TIC), used to integrate buildings from different sources with the Digital Terrain Model, is also possible in LOD1 – 4. The TIC can – but does not have to – build closed rings around the building or building parts.

In LOD0 (cf. Fig. 29) the building is represented by horizontal surfaces describing the footprint and the roof edge.

In LOD1 (cf. Fig. 30), the different structural entities of a building are aggregated to a simple block

and not differentiated in detail. The volumetric and surface parts of the exterior building shell are identical and only one of the corresponding properties (lod1Solid or lod1MultiSurface) must be used.

In LOD2 and higher levels of detail, the exterior shell of a building is not only represented geometrically as gml:Solid geometry and/or a gml:MultiSurface geometry, but it can also be composed of semantic objects. The base class for all objects semantically structuring the building shell is _BoundarySurface (cf. chapter 10.3.2), which is associated with a gml:MultiSurface geometry. If in a building model there is both a geometric represen-tation of the exterior shell as volume or surface model and a semantic representation by means of thematic _BoundarySurfaces, the geometric representation must not explicitly define the geometry, but has to reference the corresponding geometry components of the gml:MultiSurface of the _BoundarySurface elements.

Table 2. Semantic themes of the class _AbstractBuilding

Geometric / semantic theme	Property type	LOD0	LOD1	LOD2	LOD3	LOD4
Building footprint and roof edge	gml:MultiSurfaceType	•				
Volume part of the building shell	gml:SolidType		•	•	•	•
Surface part of the building shell	gml:MultiSurfaceType		•	•	•	•
Terrain intersection curve	gml:MultiCurveType		•	•	•	•
Curve part of the building shell	gml:MultiCurveType			•	•	•
Building parts	BuildingPartType		•	•	•	•
Boundary surfaces (chapter 10.3.3)	AbstractBoundarySurfaceType			•	•	•
Outer building installations (chapter 10.3.2)	BuildingInstallationType			•	•	•
Openings (chapter 10.3.4)	AbstractOpeningType				•	•
Rooms (chapter 10.3.5)	RoomType					•
Interior building installations (chapter 10.3.5)	IntBuildingInstallationType					•

Apart from BuildingParts, smaller features of the building (“outer building installations”) can also strongly affect the building characteristic. These features are modelled by the class BuildingInstallation (cf. chapter 10.3.2). Typical candidates for this class are chimneys (see. Fig. 30), dormers (see Fig. 28), balconies, outer stairs, or antennas. BuildingInstallations may only be

included in LOD2 models, if their extents exceed the proposed minimum dimensions as specified in chapter 6.2. For the geometrical representation of the class Build-ingInstallation, an arbitrary geometry object from the GML subset shown in Fig. 9 can be used.

The class _AbstractBuilding has no additional properties for LOD3. Besides the higher requirements on geomet-ric precision and smaller minimum dimensions, the main difference of LOD2 and LOD3 buildings concerns the class _BoundarySurface (cf. chapter 10.3.3). In LOD3, openings in a building corresponding with windows or doors (see Fig. 30) are modelled by the abstract class _Opening and the derived subclasses Window and Door (cf. chapter 10.3.4).

With respect to the exterior building shell, the LOD4 data model is identical to that of LOD3. But LOD4 pro-vides the possibility to model the interior structure of a building with the classes IntBuildingInstallation and Room (cf. chapter 10.3.5).

Each Building or BuildingPart feature may be assigned zero or more addresses using the address property. The corresponding AddressPropertyType is defined within the CityGML core module (cf. chapter 10.1.4).

6.5.2. Outer building installations

BuildingInstallationType, BuildingInstallation

Note: insert BuildingInstallation UML

A BuildingInstallation is an outer component of a building which has not the significance of a BuildingPart, but which strongly affects the outer characteristic of the building. Examples are chimneys, stairs, antennas, balconies or attached roofs above stairs and paths. A BuildingInstallation optionally has attributes class, function and usage. The attribute class - which can only occur once - represents a general classification of the installation. With the attributes function and usage, nominal and real functions of a building installation can be described. For all three attributes the list of feasible values can be specified in a code list. For the geometrical representation of a BuildingInstallation, an arbitrary geometry object from the GML subset shown in Fig. 9 can be used. Alterna-tively, the geometry may be given as ImplicitGeometry object. Following the concept of ImplicitGeometry the geometry of a prototype building installation is stored only once in a local coordinate system and referenced by other building installation features (see chapter 8.2). The visible surfaces of a building installation can be seman-tically classified using the concept of boundary surfaces (cf. 10.3.3). A BuildingInstallation object should be uniquely related to exactly one building or building part object.

6.5.3. Boundary surfaces

AbstractBoundarySurfaceType, _BoundarySurface

NOTE	Insert AbstractBoundarySurfaceType, _BoundarySurface UML

_BoundarySurface is the abstract base class for several thematic classes, structuring the exterior shell of a build-ing as well as the visible surfaces of rooms and both outer and interior building installations. It is a subclass of _CityObject and thus inherits all properties like the GML3 standard feature properties (gml:name etc.) and the CityGML specific properties like ExternalReferences.

From _BoundarySurface, the thematic classes RoofSur-face, WallSurface, GroundSurface, OuterCeilingSurface, OuterFloorSurface, ClosureSurface, FloorSurface, InteriorWallSurface, and CeilingSurface are derived. The thematic classification of building surfaces is illustrat-ed in Fig. 31 (outer building shell) and Fig. 32 (additional interior surfaces) and subsequently specified.

For each LOD between 2 and 4, the geometry of a _BoundarySurface may be defined by a different gml:MultiSurface geometry.

In LOD3 and LOD4, a _BoundarySurface may contain _Openings (cf. chapter 10.3.4) like doors and windows. If the geometric location of _Openings topologically lies within a surface component (e.g. gml:Polygon) of the gml:MultiSurface geometry, these _Openings must be represented as holes within that surface. A hole is repre-sented by an interior ring within the corresponding surface geometry object. According to GML3, the points have to be specified in reverse order (exterior boundaries counter-clockwise and interior boundaries clockwise when looking in opposite direction of the surface's normal vector). If such an opening is sealed by a Door, a Window, or a ClosureSurface, their outer boundary may consist of the same points as the inner ring (denoting the hole) of the surrounding surface. The embrasure surfaces of an Opening belong to the relevant adjacent _BoundarySurface. If, for example a door seals the Opening, the embrasure surface on the one side of the door belongs to the InteriorWallSurface and on the other side to the WallSurface (Fig. 32 on the right).

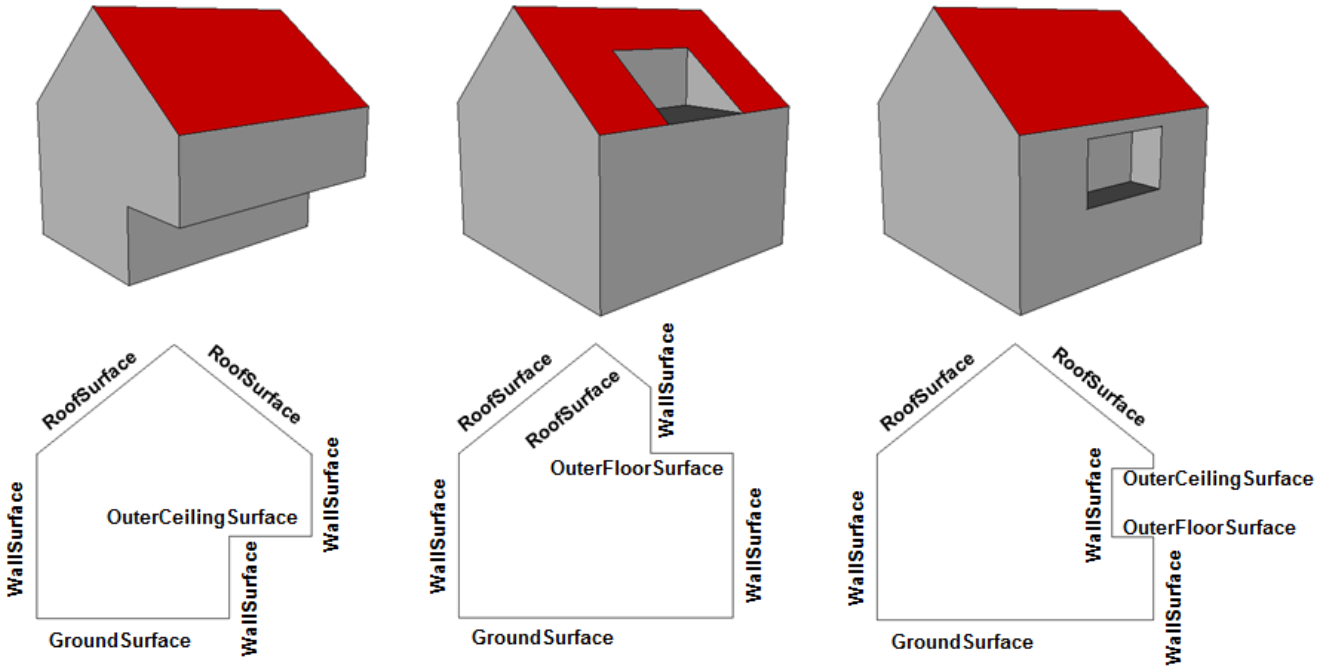
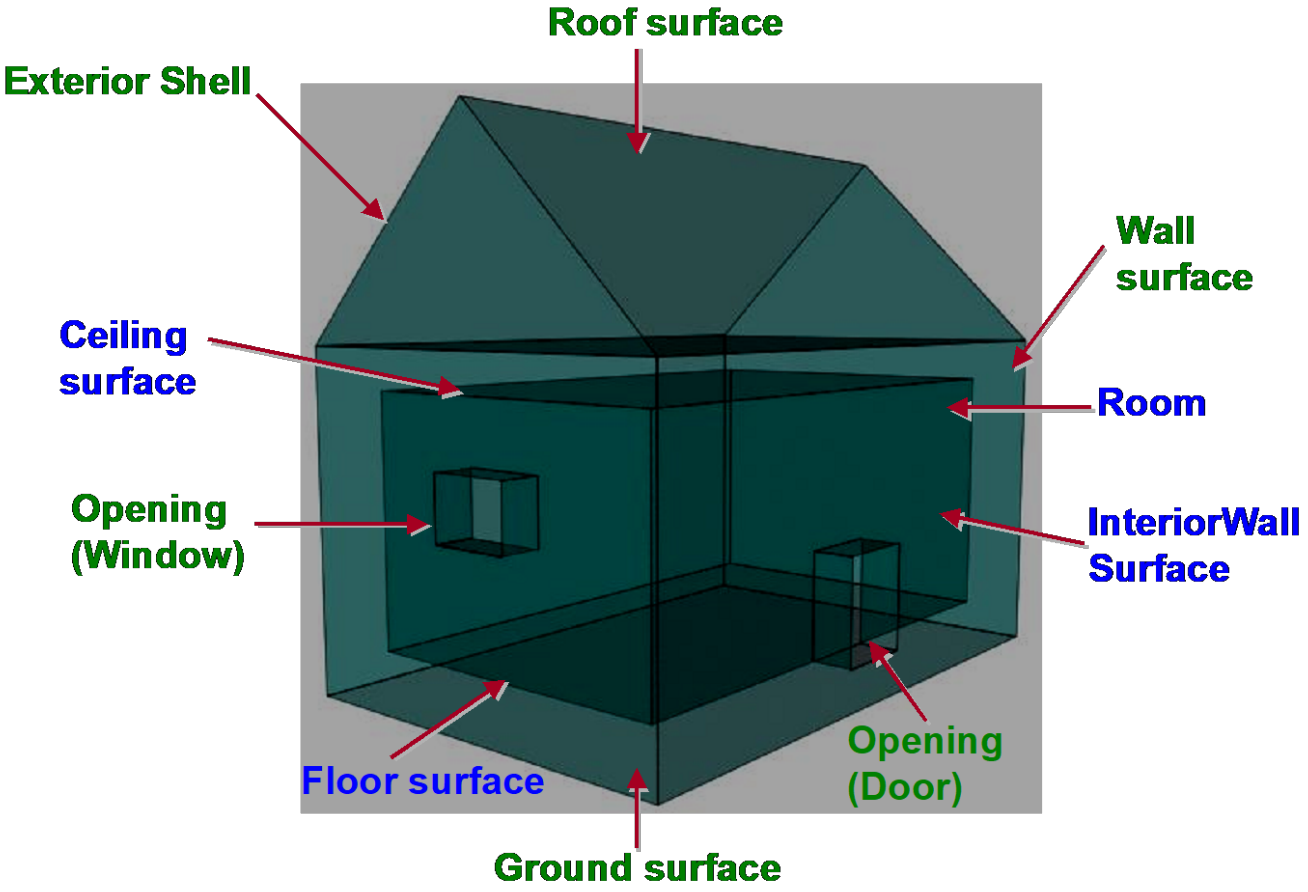


Figure 29. Examples of the classification of _BoundarySurfaces of the outer building shell (source: Karlsruhe Institute of Technology (KIT))


Roof surface
Exterior Shell
Wall surface
Ceiling surface
Room
Opening (Window)
InteriorWall Surface
Opening (Door)
Floor surface
Ground surface

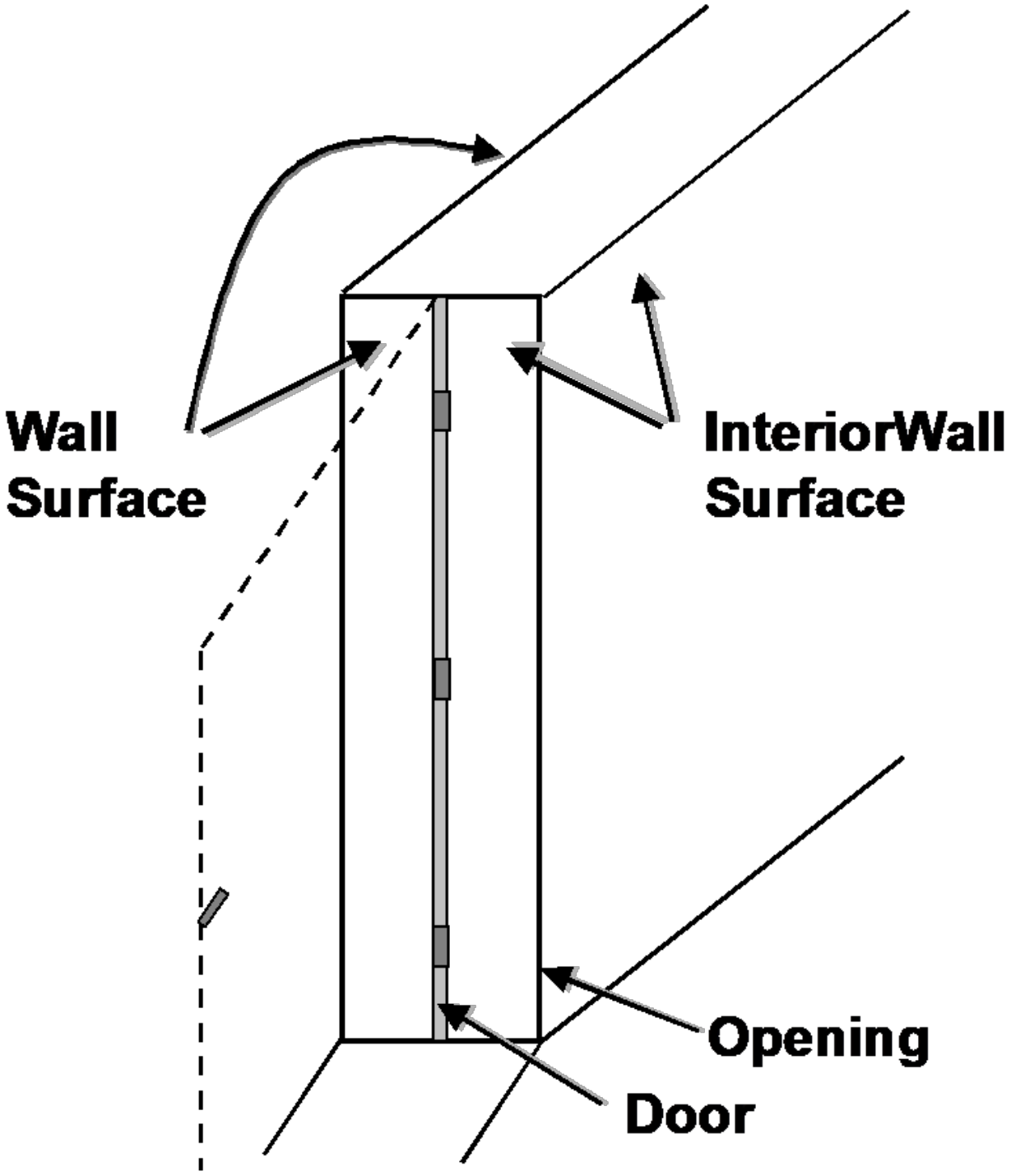


Figure 30. Classification of BoundarySurfaces (left), in particular for Openings (right) (graphic: IGG Uni Bonn).

GroundSurfaceType, GroundSurface

NOTE	insert GroundSurfaceType, GroundSurface uml

The ground plate of a building or building part is modelled by the class GroundSurface. The polygon defining the ground plate is congruent with the building's footprint. However, the surface normal of the ground plate is pointing downwards.

OuterCeilingSurfaceType, OuterCeilingSurface

NOTE insert OuterCeilingSurfaceType, OuterCeilingSurface UML

A mostly horizontal surface belonging to the outer building shell and having the orientation pointing downwards can be modeled as an OuterCeilingSurface. Examples are the visible part of the ceiling of a loggia or the ceiling of a passage.

WallSurfaceType, WallSurface

NOTE insert WallSurfaceType, WallSurface UML

All parts of the building facade belonging to the outer building shell can be modelled by the class WallSurface.

OuterFloorSurfaceType, OuterFloorSurface

NOTE insert OuterFloorSurfaceType, OuterFloorSurface UML

A mostly horizontal surface belonging to the outer building shell and with the orientation pointing upwards can be modeled as an OuterFloorSurface. An example is the floor of a loggia.

RoofSurfaceType, RoofSurface

NOTE insert RoofSurfaceType, RoofSurface UML

The major roof parts of a building or building part are expressed by the class RoofSurface. Secondary parts of a roof with a specific semantic meaning like dormers or chimneys should be modelled as BuildingInstallation.

ClosureSurfaceType, ClosureSurface

NOTE insert ClosureSurfaceType, ClosureSurface UML

An opening in a building not filled by a door or window can be sealed by a virtual surface called ClosureSurface (cf. chapter 6.4). Hence, buildings with open sides like a barn or a hangar, can be virtually closed in order to be able to compute their volume. ClosureSurfaces are also used in the interior building model. If two rooms with a different function (e.g. kitchen and living room) are directly connected without a separating door, a ClosureSur-face should be used to separate or connect the volumes of both rooms.

FloorSurfaceType, FloorSurface

NOTE insert FloorSurfaceType, FloorSurface UML

The class FloorSurface must only be used in the LOD4 interior building model for modelling the floor of a room.

InteriorWallSurfaceType, InteriorWallSurface

NOTE	insert InteriorWallSurfaceType, InteriorWallSurface UML

The class InteriorWallSurface must only be used in the LOD4 interior building model for modelling the visible surfaces of the room walls.

CeilingSurfaceType, CeilingSurface

NOTE	Insert CeilingSurfaceType, CeilingSurface UML

The class CeilingSurface must only be used in the LOD4 interior building model for modelling the ceiling of a room.

6.5.4. Openings

AbstractOpeningType, _Opening

NOTE	insert AbstractOpeningType, _Opening UML

The class _Opening is the abstract base class for semantically describing openings like doors or windows in outer or inner boundary surfaces like walls and roofs. Openings only exist in models of LOD3 or LOD4. Each _Opening is associated with a gml:MultiSurface geometry. Alternatively, the geometry may be given as Implic-itGeometry object. Following the concept of ImplicitGeometry the geometry of a prototype opening is stored only once in a local coordinate system and referenced by other opening features (see chapter 8.2).

WindowType, Window

NOTE	insert WindowType, Window UML

The class Window is used for modelling windows in the exterior shell of a building, or hatches between adjacent rooms. The formal difference between the classes Window and Door is that – in normal cases – Windows are not specifically intended for the transit of people or vehicles.

DoorType, Door

NOTE	insert DoorType, Door UML

The class Door is used for modelling doors in the exterior shell of a building, or between adjacent rooms. Doors can be used by people to enter or leave a building or room. In contrast to a ClosureSurface a door may be closed, blocking the transit of people. A Door may be assigned zero or more addresses. The corresponding Address-PropertyType is defined within the CityGML core module (cf. chapter 10.1.4) .

6.5.5. Building Interior

RoomType, Room

NOTE	insert RoomType, Room UML

A Room is a semantic object for modelling the free space inside a building and should be uniquely related to exactly one building or building part object. It should be closed (if necessary by using ClosureSurfaces) and the geometry normally will be described by a solid (lod4Solid). However, if the topological correctness of the boundary cannot be guaranteed, the geometry can alternatively be given as a MultiSurface (lod4MultiSurface). The surface normals of the outer shell of a GML solid must point outwards. This is important to consider when Room surfaces should be assigned Appearances. In this case, textures and colors must be placed on the backside of the corresponding surfaces in order to be visible from the inside of the room.

In addition to the geometrical representation, different parts of the visible surface of a room can be modelled by specialised BoundarySurfaces (FloorSurface, CeilingSurface, InteriorWallSurface, and ClosureSurface cf. chapter 10.3.3).

A special task is the modelling of passages between adjacent rooms. The room solids are topologically connected by the surfaces representing hatches, doors or closure surfaces that seal open doorways. Rooms are defined as being adjacent, if they have common _Openings or ClosureSurfaces. The surface that represents the opening geometrically is part of the boundaries of the solids of both rooms, or the opening is referenced by both rooms on the semantic level. This adjacency implies an accessibility graph, which can be employed to determine the spread of e.g. smoke or gas, but which can also be used to compute escape routes using classical shortest path algorithms (see Fig. 33).

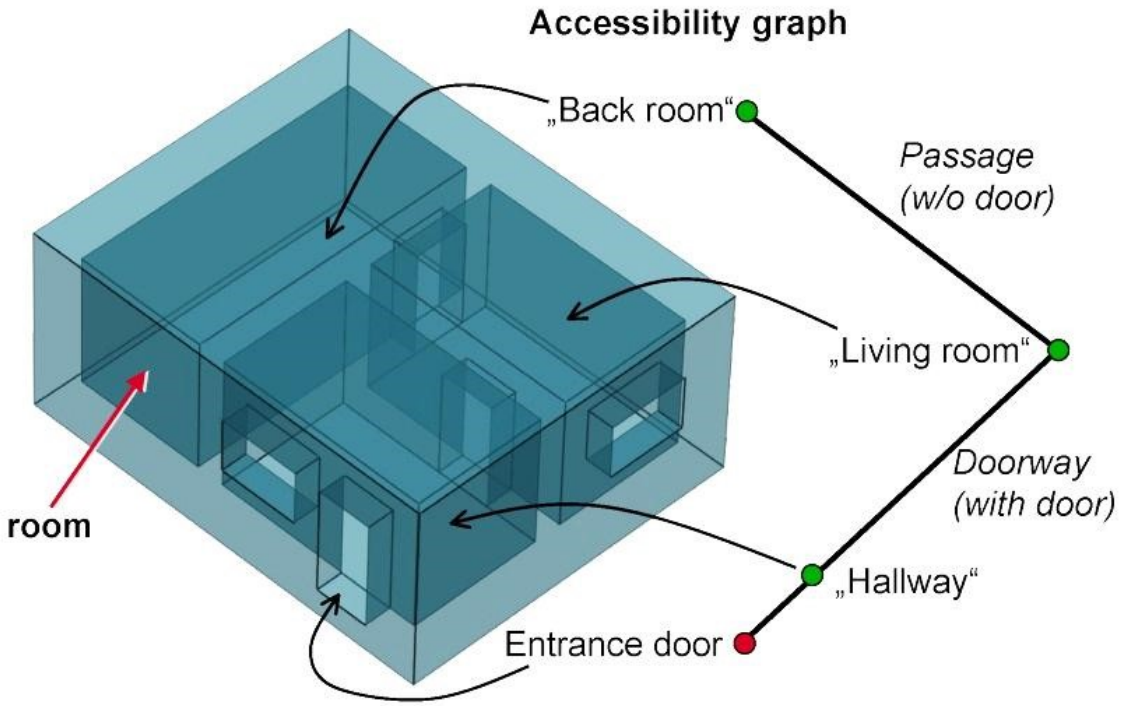


Figure 31. Accessibility graph derived from topological adjacencies of room surfaces (graphic: IGG Uni Bonn).

BuildingFurnitureType, BuildingFurniture

NOTE	insert BuildingFurnitureType, BuildingFurniture UML

Rooms may have BuildingFurnitures and IntBuildingInstallations. A BuildingFurniture is a movable part of a room, such as a chair or furniture. A BuildingFurniture object should be uniquely related to exactly one room object. Its geometry may be represented by an explicit geometry or an ImplicitGeometry object. Following the concept of ImplicitGeometry the geometry of a prototype building furniture is stored only once in a local coordi-nate system and referenced by other building furniture features (see chapter 8.2).

IntBuildingInstallationType, IntBuildingInstallation

NOTE	insert IntBuildingInstallationType, IntBuildingInstallation UML

An IntBuildingInstallation is an object inside a building with a specialised function or semantic meaning. In contrast to BuildingFurniture, IntBuildingInstallations are permanently attached to the building structure and cannot be moved. Typical examples are interior stairs, railings, radiators or pipes. Objects of the class IntBuild-ingInstallation can either be associated with a room (class Room), or with the complete building / building part (class _AbstractBuilding, cf. chapter 10.3.1). However, they should be uniquely related to exactly one room or one building / building part object. An IntBuildingInstallation optionally has attributes class, function and usage. The attribute

class, which can only occur once, represents a general classification of the internal building component. With the attributes function and usage, nominal and real functions of a building installation can be described. For all three attributes the list of feasible values can be specified in a code list. For the geometrical representation of an IntBuildingInstallation, an arbitrary geometry object from the GML subset shown in Fig. 9 can be used. Alternatively, the geometry may be given as ImplicitGeometry object. Following the concept of ImplicitGeometry the geometry of a prototype interior building installation is stored only once in a local coordi-nate system and referenced by other interior building installation features (see chapter 8.2). The visible surfaces of an interior building installation can be semantically classified using the concept of boundary surfaces (cf. 10.3.3).

6.5.6. Modelling building storeys using CityObjectGroups

CityGML does currently not provide a specific concept for the representation of storeys as it is available in the AEC/FM standard IFC (IAI 2006). However, a storey can be represented as an explicit aggregation of all build-ing features on a certain height level using CityGML's notion of CityObjectGroups (cf. chapter 10.11). This would include Rooms, Doors, Windows, IntBuildingInstallations and BuildingFurniture. If thematic surfaces like walls and interior walls should also be associated to a specific storey, this might require the vertical fragmenta-tion of these surfaces (one per storey), as in virtual 3D city models they typically span the whole façade.

In order to model building storeys with CityGML's generic grouping concept, a nested hierarchy of CityObject-Group objects has to be used. In a first step, all semantic objects belonging to a specific storey are grouped. The attributes of the corresponding CityObjectGroup object are set as follows:

- The class attribute shall be assigned the value "building separation".
- The function attribute shall be assigned the value "lodXStorey" with X between 1 and 4 in order to de-note that this group represents a storey wrt. a specific LOD.
- The storey name or number can be stored in the gml:name property. The storey number attribute shall be assigned the value "storeyNo_X" with decimal number X in order to denote that this group repre-sents a storey wrt. a specific number.

In a second step, the CityObjectGroup objects representing different storeys are grouped themselves. By using the generic aggregation concept of CityObjectGroup, the "storeys group" is associated with the corresponding Building or BuildingPart object. The class attribute of the storeys group shall be assigned the value "building storeys".

6.5.7. Examples

The LOD1 model of the Campus North of the Karlruhe Institute of Technology (KIT) shown in Fig. 34 consists of 596 buildings and 187 building parts. The footprint geometries of the buildings are taken from a cadastral information system and extruded by a given height. Buildings with a unique identifier and a single height value are modeled as one building (bldg:Building). Buildings having a unique identifier but different height values are modeled as one building (bldg:Building) with one or more building parts (bldg:BuildingPart). Both buildings and building parts have solid geometries and their height values are additionally represented as thematic attribute (bldg:measuredHeight). Fig. 34 shows an aerial photograph of the KIT Campus North (left) and the CityGML LOD1 model (right).

Figure 32. LOD1 model of the KIT Campus North (source: Karlsruhe Institute of Technology (KIT)).

An example for a fully textured LOD2 building model is given in Fig. 35 which shows the Bernhardus church located in the city of Karlsruhe, Germany. On the left side of Fig. 35, a photograph of the church in real world is shown whereas the CityGML building model of the church with photorealistic textures is illustrated on the right. The model is bounded by a ground surface, several wall and roof surfaces. The railing above the church clock is modeled as a building installation (BuildingInstallation).

Figure 33. Textured LOD2 model of the Bernhardus church in Karlsruhe (source: Karlsruhe Institute of Technology (KIT), courtesy of City of Karlsruhe).

The model shown in Fig. 36 was derived from a 3D CAD model generated during the planning phase of the building. On the left side of Fig. 36, the building is shown whereas on the right side the LOD3 model is present-ed. The building itself is bounded by wall surfaces, roof surfaces and a ground surface. Doors and windows are modeled including reveals. According to the cadaster data, the car port next to the building is not part of the building. Therefore the car port, the balcony and the chimney are modeled as building installations (BuildingIn-stallation). The model also contains the terrain intersection curve (lod3TerrainIntersection) as planned by the architect.

In order to determine the volume of the building, the geometries of all boundary surfaces, including doors and windows, are referenced by the building solid (lod3Solid) using the XLink mechanism. Consequently, the roof surfaces are split into surfaces representing the roof itself and surfaces representing the roof overhangs.

Figure 34. Example of a buildingmodeled in the Level of Detail 3. The chimney, the balcony and the car port are modeled as building installations (source: Karlsruhe Institute of Technology (KIT), courtesy of Franz-Josef Kaiser).

6.6. City Furniture

Contributors
TBD

NOTE The following text needs to be reviewed and updated.

City furniture objects are immovable objects like lanterns, traffic lights, traffic signs, flower buckets, advertising columns, benches, delimitation stakes, or bus stops (Fig. 67, Fig. 68). City furniture objects can be found in traffic areas, residential areas, on squares or in built-up areas. The modelling of city furniture objects is used for visualisation of, for example city traffic, but also for analysing local structural conditions. The recognition of special locations in a city model is improved by the use of these detailed city furniture objects, and the city model itself becomes more alive and animated. The city furniture model of CityGML is defined by the thematic extension module CityFurniture (cf. chapter 7).

City furniture objects can have an important influence on simulations of, for example city traffic situations. Navigation systems can be realised, for example for visually handicapped people using a traffic light as routing target. Likewise, city furniture objects are important to plan a heavy vehicle transportation, where the exact position and further conditions of obstacles must be known.

Figure 35. Real situation showing a bus stop (left). The advertising billboard and the refuge are modelled as CityFurniture objects in the right image (source: 3D city model of Barkenberg).

Figure 36. Real situation showing lanterns and delimitation stakes (left). In the right image they are modelled as CityFurniture ob-jects with ImplicitGeometry representations (source: 3D city model of Barkenberg).

The UML diagram of the city furniture model is depicted in Fig. 69, for the XML schema definition see below and annex A.5.

The class CityFurniture may have the attributes class, function and usage. Their possible values can be specified in corresponding code lists (chapter 10.9.2 and annex C.4). The class attribute allows an object classification like traffic light, traffic sign, delimitation stake, or garbage can, and can occur only once. The function attribute describes, to which thematic area the city furniture object belongs to (e.g. transportation, traffic regulation, architecture), and can occur multiple times. The attribute usage denotes the real purpose of the object.

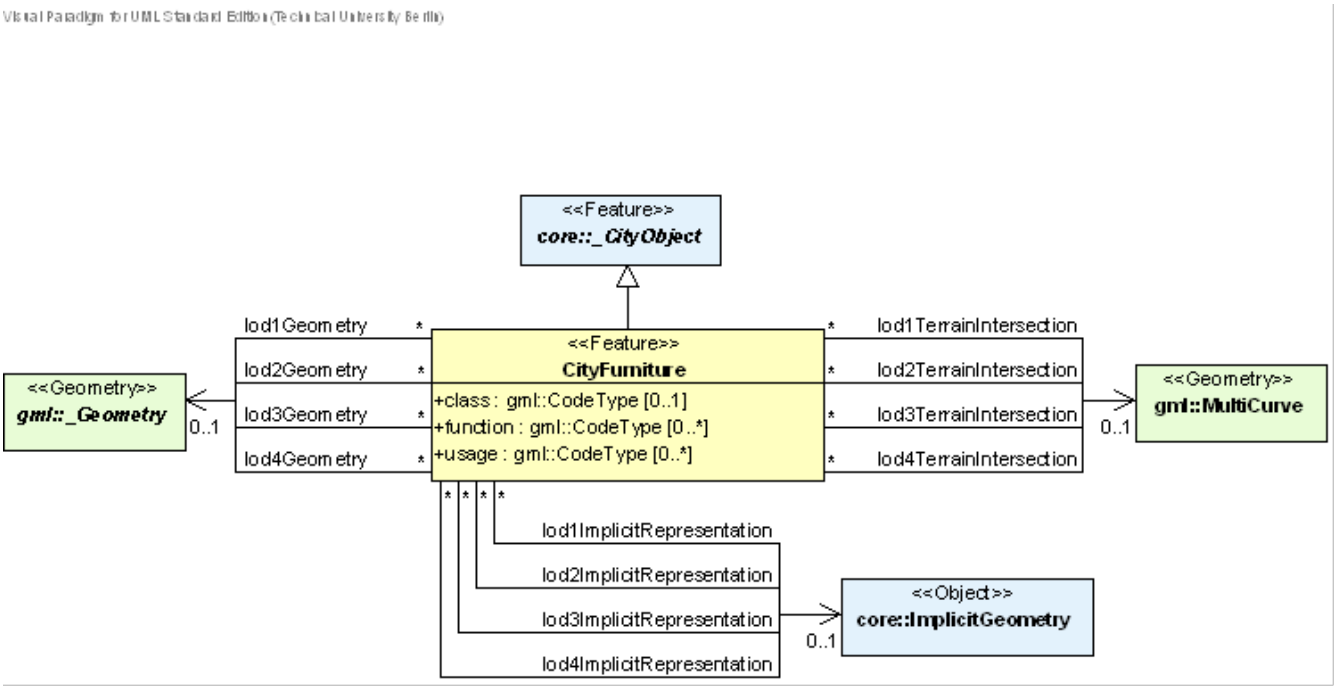


Figure 37. UML diagram of city furniture objects in CityGML. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML CityFurniture module.

Since CityFurniture is a subclass of _CityObject and hence is a feature, it inherits the attribute gml:name. As with any _CityObject, CityFurniture objects may be assigned ExternalReferences (cf. chapter 6.7) and may be augmented by generic attributes using CityGML's Generics module (cf. chapter 10.12). For ExternalReferences city furniture objects can have links to external thematic databases. Thereby, semantic information of the objects, which cannot be modelled in CityGML, can be transmitted and used in the 3D city model for further processing, for example information from systems of powerlines or pipelines, traffic sign cadaster, or water resources for disaster management.

City furniture objects can be represented in city models with their specific geometry, but in most cases the same kind of object has an identical geometry. The geometry of CityFurniture objects in LOD 1-4 may be represented by an explicit geometry (lodXGeometry where X is between 1 and 4) or an ImplicitGeometry object (lodXImplic-itRepresentation with X between 1 and 4). Following the concept of ImplicitGeometry the geometry of a proto-type city furniture is stored only once in a local coordinate system and referenced by other city furniture features (see chapter 8.2). Spatial information of city furniture objects can be taken, for example, from city maps or from public and private external information systems.

In order to specify the exact intersection of the DTM with the 3D geometry of a city furniture object, the latter can have a TerrainIntersectionCurve (TIC) for each LOD (cf. chapter 6.5). This allows for ensuring a smooth transition between the DTM and the city furniture object.

XML namespace

The XML namespace of the CityGML CityFurniture module is identified by the Uniform Resource Identifier (URI) http://www.opengis.net/citygml/cityfurniture/2.0. Within the XML Schema definition of the CityFurniture module, this URI is also used to identify the default namespace.

6.6.1. City furniture object

CityFurnitureType, CityFurniture

NOTE	insert CityFurnitureType, CityFurniture UML

6.6.2. Code lists

The attributes class, function, and usage of the feature CityFurniture are specified as gml:CodeType. The values of these properties can be enumerated in code lists. Proposals for corresponding code lists can be found in annex C.4.

6.6.3. Example CityGML dataset

The following example of a CityGML dataset is an extract of the model of a delimitation stake in LOD3 and contains the attributes class = 1000 and function = 1520 (delimitation stake) whose coded attribute values are taken from a code list proposed by the SIG 3D (cf. annex C.4). The delimitation stake with the object ID stake0815 has an ExternalReference pointing to a cadastre object within the German ALKIS database (www.adv-online.de) which is identified by the URI urn:adv:oid:DEHE123400007001.

The example dataset shows the geometry of the top surface (gml:id “cover”) and of the left surface (gml:id “surfLeft”) of the stake which are both depicted in Fig. 70. The top surface is assigned a constant material (white color) and the left surface is textured using the texture image stake.gif by denoting the relevant texture coordi-nates. Both surface appearances are modelled using the CityGML Appearance module (cf. chapter 9).

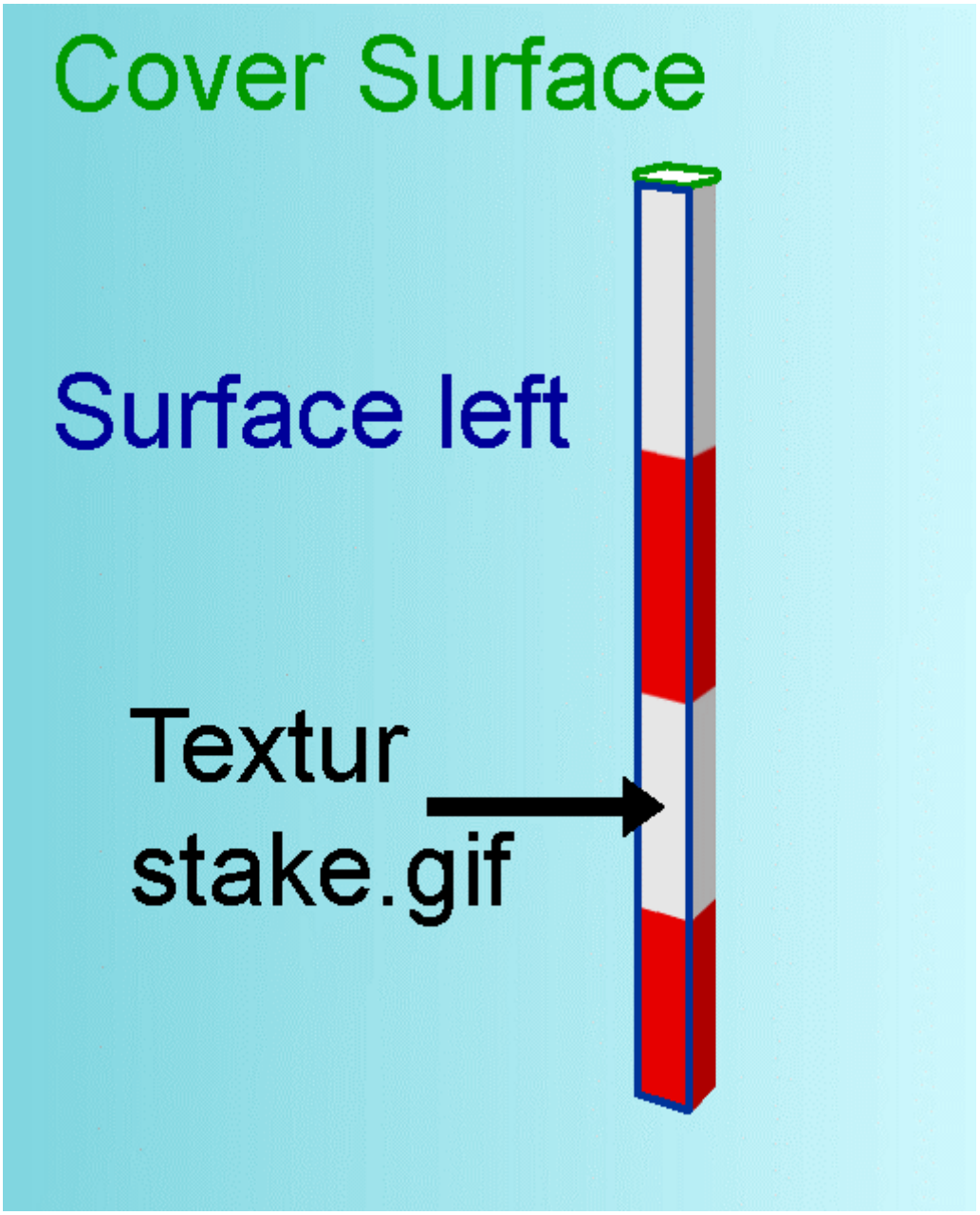


Figure 38. Example of a simple city furniture object (source: District of Recklinghausen).

NOTE	insert example - GML?

6.7. City Object Group

Contributors
TBD

NOTE	The following text needs to be reviewed and updated.

The CityGML grouping concept has been introduced in chapter 6.8. CityObjectGroups are modelled using the Composite Design Pattern from software engineering (cf. Gamma et al. 1995): CityObjectGroups aggregate CityObjects and furthermore are defined as special CityObjects. This implies that a group may become a member of another group realizing a recursive aggregation schema. However, in a CityGML instance document it has to be ensured (by the generating

application) that no cyclic groupings are included. Fig. 73 shows the UML dia-gram for the class CityObjectGroup, for the XML schema see annex A.6. The grouping concept of CityGML is defined by the thematic extension module CityObjectGroup (cf. chapter 7).

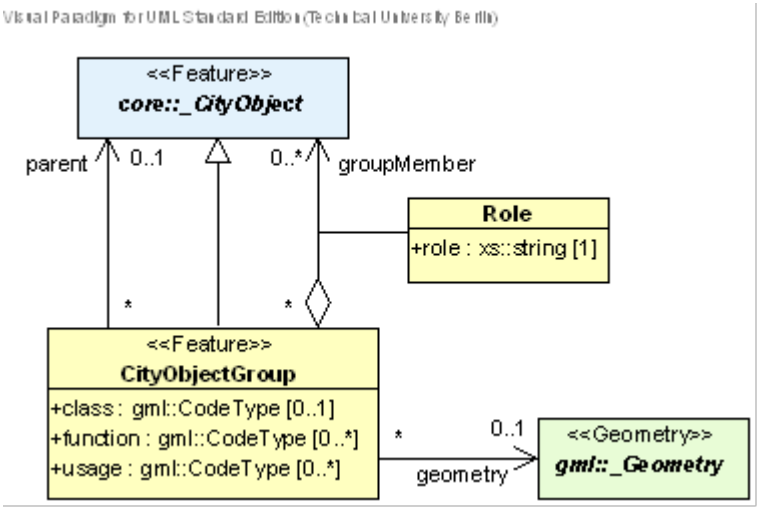


Figure 39. UML diagram of city object groups in CityGML. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML CityObjectGroup module.

The class CityObjectGroup has the optional attributes class, function and usage. The class attribute allows a group classification with respect to the stated function and may occur only once. The function attribute is intend-ed to express the main purpose of a group, possibly to which thematic area it belongs (e.g. site, building, trans-portation, architecture, unknown etc.). The attribute usage can be used, if the way the object is actually used differs from the function. Both attributes can occur multiple times. Each member of a group may be qualified by a role name, reflecting the role each _CityObject plays in the context of the group. Furthermore, a CityObject-Group can optionally be assigned an arbitrary geometry object from the GML3 subset shown in Fig. 9 in chapter 8.1. This may be used to represent a generalised geometry generated from the members geometries.

The parent association linking a CityObjectGroup to a _CityObject allows for the modelling of a generic hierar-chical grouping concept. Named aggregations of components (CityObjects) can be added to specific CityObjects considered as the parent object. The parent association links to the aggregate, while the parts are given by the group members. This concept is used, for example, to represent storeys in buildings (see section 10.3.6: Mo-delling building storeys using CityObjectGroups).

XML namespace

The XML namespace of the CityGML CityObjectGroup module is identified by the Uniform Resource Identifier (URI) http://www.opengis.net/citygml/cityobjectgroup/2.0. Within the XML Schema definition of the CityOb-jectGroup module, this URI is also used to identify the default namespace.

6.7.1. City object group

CityObjectGroupType, CityObjectGroup

NOTE	insert CityObjectGroupType, CityObjectGroup UML

6.7.2. Code lists

The attributes class, function, and usage of the feature CityObjectGroup are specified as gml:CodeType. The values of these properties can be enumerated in code lists. Proposals for corresponding code lists can be found in annex C.10.

6.8. Construction

Contributors
TBD

6.8.1. Synopsis

6.8.2. Key Concepts

6.8.3. Discussion

6.8.4. Level of Detail

6.8.5. UML Model

The UML diagram of the Construction module is depicted in UML diagram of the Construction Model..

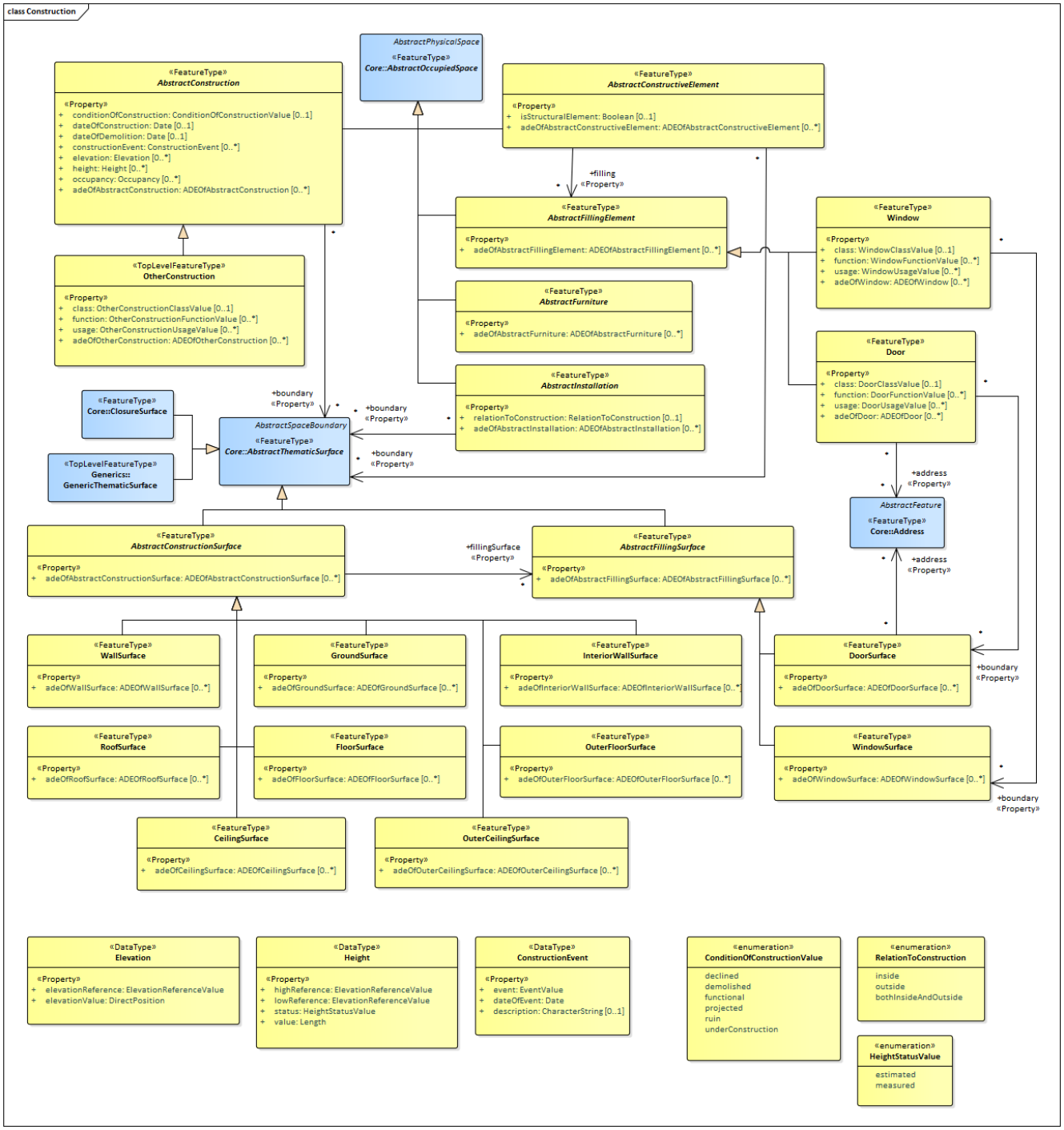


Figure 40. UML diagram of the Construction Model.

The ADE data types provided for the Construction module are illustrated in ADE classes of the CityGML Construction module..

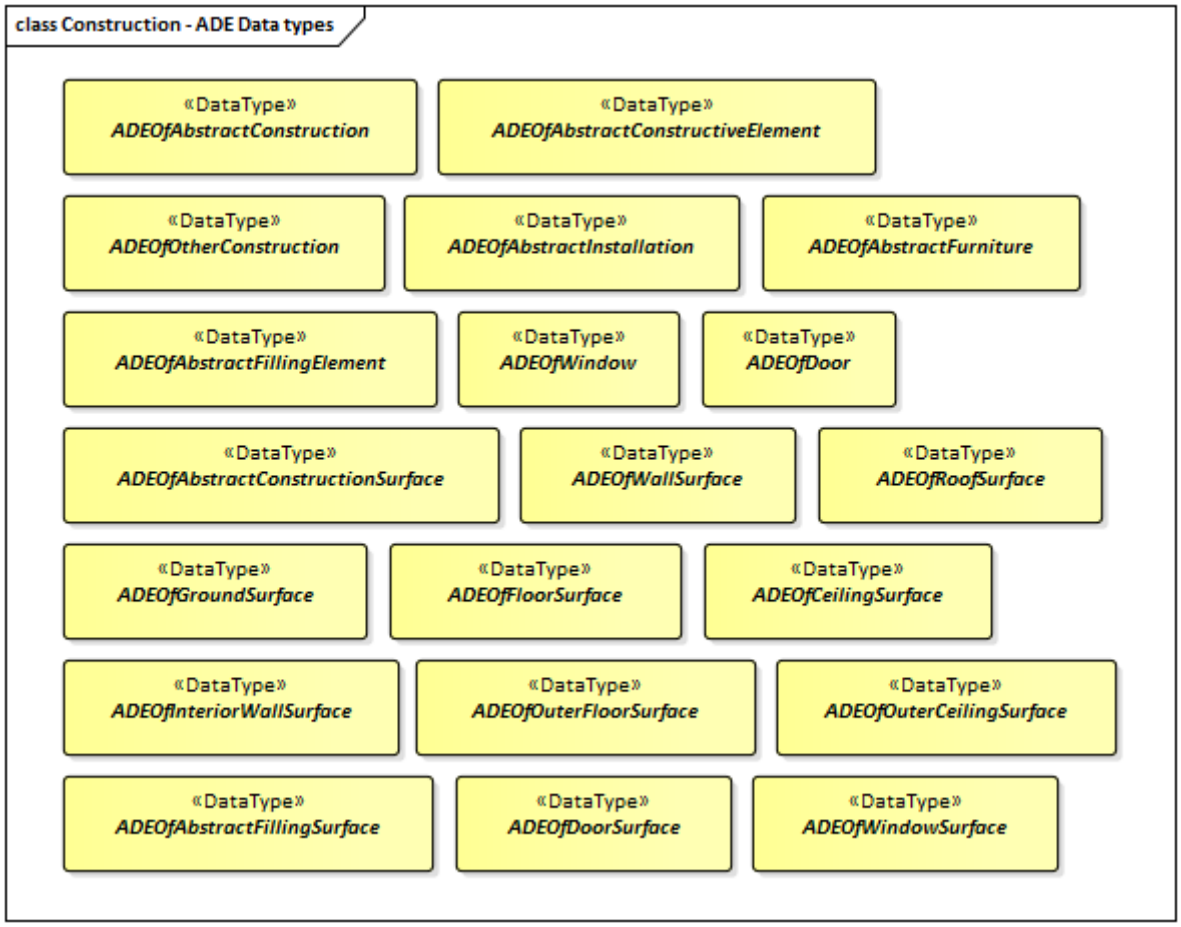


Figure 41. ADE classes of the CityGML Construction module.

The Code Lists provided for the Construction module are illustrated in Codelists from the CityGML Construction module..

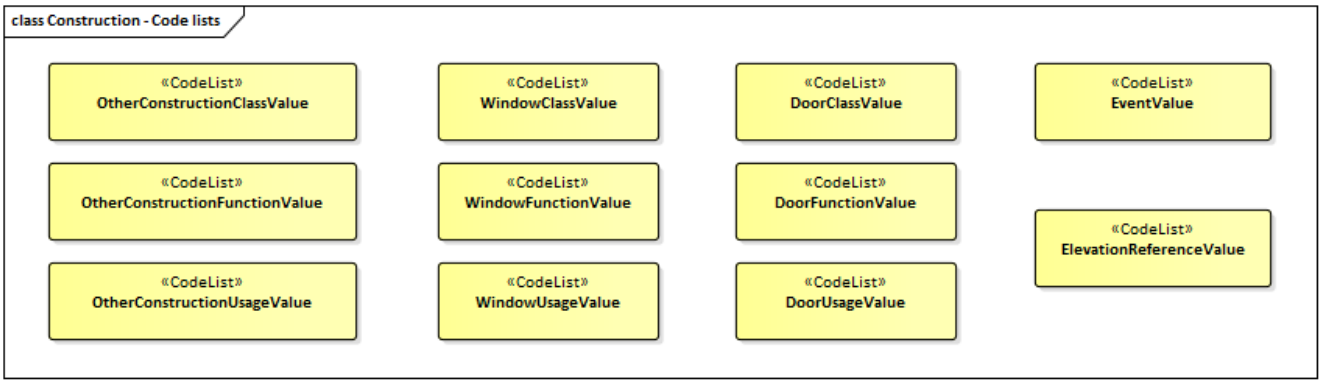


Figure 42. Codelists from the CityGML Construction module.

6.8.6. Examples

6.9. Dynamizer

Contributors
C. Heazel - first draft

6.9.1. Synopsis

The Dynamizer module provides the concepts that enable representation of time-varying data for city object properties. In particular, they allow sensor readings to be integrated into 3D city models.

Dynamizers inject timeseries data for an individual attribute of the city object. In order to represent dynamic (time-dependent) variations of its value, the timeseries data overrides the static value of the attribute.

6.9.2. Key Concepts

Dynamizer:

An object that injects timeseries data for an individual attribute of the city object in which it is included.

Timeseries:

The period of time designated by a first and last time stamp. A time series does not have to be continuous. In that case, each continuous period within the time series is itself a time series. A non-continuous time series is essentially a collection of time series.

6.9.3. Discussion

NOTE	This content is from section 8.6 of the standard

The Dynamizer module provides the concepts that enable representation of time-varying data for city object properties as well as for integrating sensors with 3D city models. Dynamizers are objects that inject timeseries data for an individual attribute of the city object in which the Dynamizer is included. In order to represent dynamic (time-dependent) variations of its value, the timeseries data overrides the static value of the referenced city object attribute.

The dynamic values may be given by retrieving observation results directly from external sensor/IoT services using a sensor connection (e.g. OGC SensorThings API, Sensor Observation Service, or other sensor data platforms including MQTT). Alternatively, the dynamic values may be provided as atomic timeseries that represent time-varying data of a specific data type for a single contiguous time interval. The data can be provided in:

- external tabulated files, such as CSV or Excel sheets,
- external files that format timeseries data according to the OGC TimeseriesML Standard or the OGC Observations & Measurements standards,
- or inline as embedded time-value-pairs.

Furthermore, timeseries data can also be aggregated to form composite timeseries with non-overlapping time intervals.

By using the Dynamizer module, fast changes over a short or longer time period with respect to cities and city models can be represented. This includes:

- variations of spatial properties such as change of a feature's geometry, both in respect to shape and to location (e.g. moving objects),
- variations of thematic attributes such as changes of physical quantities like energy demands, temperatures, solar irradiation, traffic density, pollution concentration, or overpressure on building walls,
- and variations with respect to sensor or real-time data resulting from simulations or measurements.

6.9.4. UML Model

The UML diagram of the Dynamizer module is depicted in UML diagram of the Dynamizer Model..

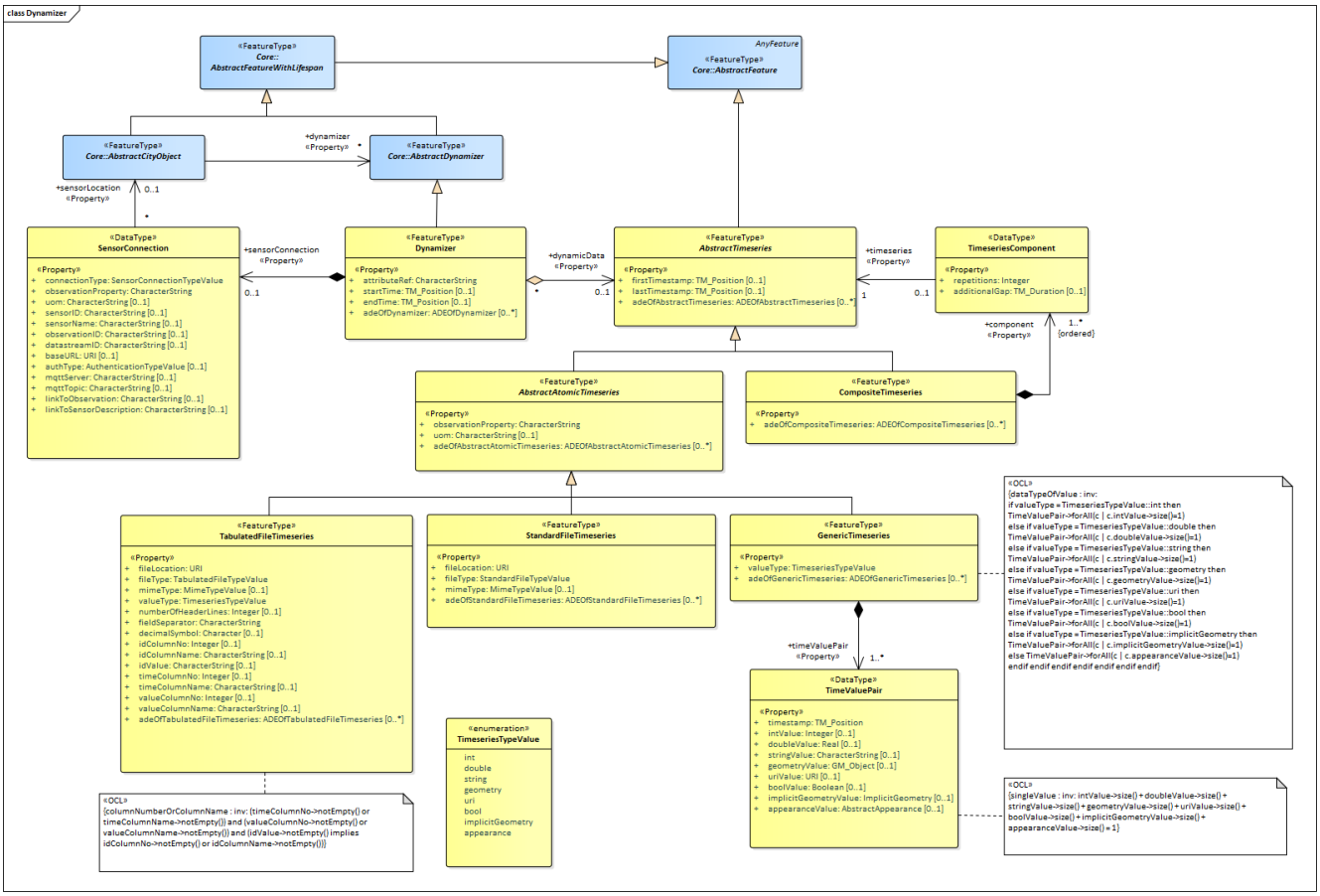


Figure 43. UML diagram of the Dynamizer Model.

The ADE data types provided for the Dynamizer module are illustrated in ADE classes of the CityGML Dynamizer module..

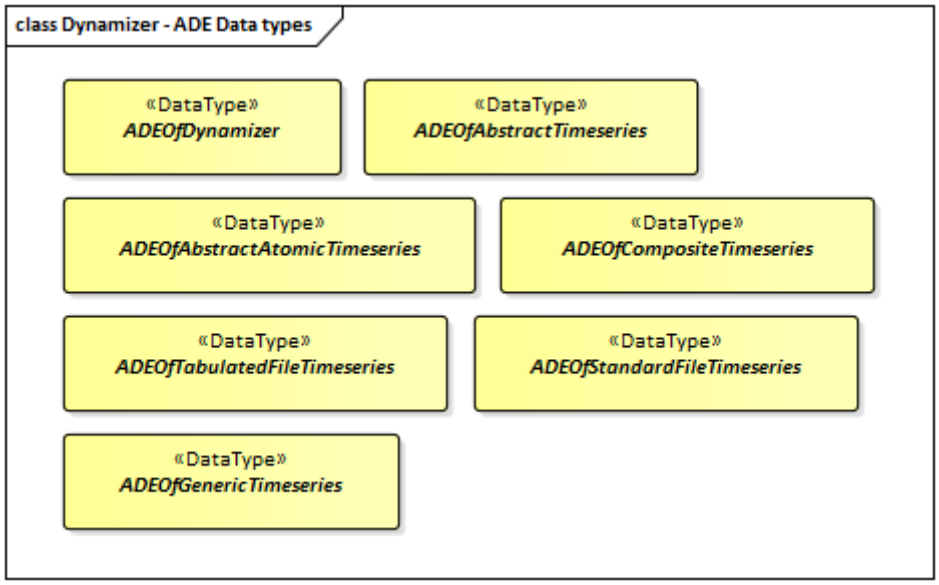


Figure 44. ADE classes of the CityGML Dynamizer module.

The Code Lists provided for the Dynamizer module are illustrated in Codelists from the CityGML Dynamizer module..

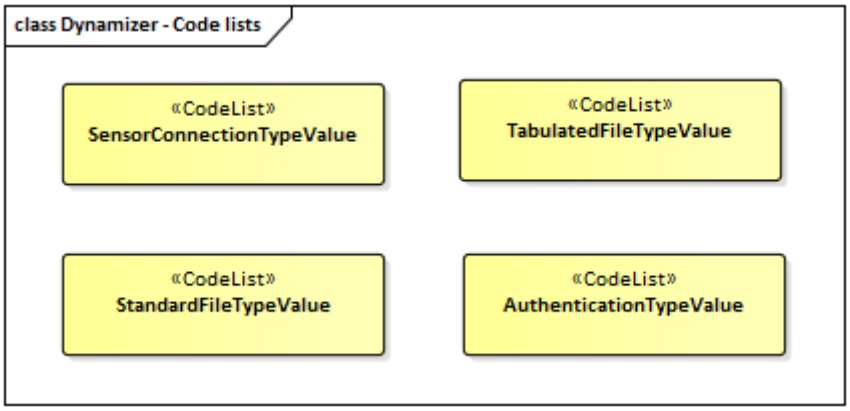


Figure 45. Codelists from the CityGML Dynamizer module.

6.9.5. Examples

6.10. Generics

Contributors
TBD

NOTE The following text needs to be reviewed and updated.

The concept of generic city objects and attributes allows for the storage and exchange of 3D objects which are not covered by any explicitly modelled thematic class within CityGML or which require attributes not represent-ed in CityGML. These generic extensions to the CityGML data model are realised by the classes GenericCityOb-ject and _genericAttribute defined within the thematic extension module Generics (cf. chapter 7). In order to avoid problems concerning semantic interoperability, generic extensions shall only be used if appropriate the-matic classes or attributes

are not provided by any other CityGML module.

Figure 74 shows the UML diagram of generic objects and attributes. For XML schema definition see below and annex A.7.

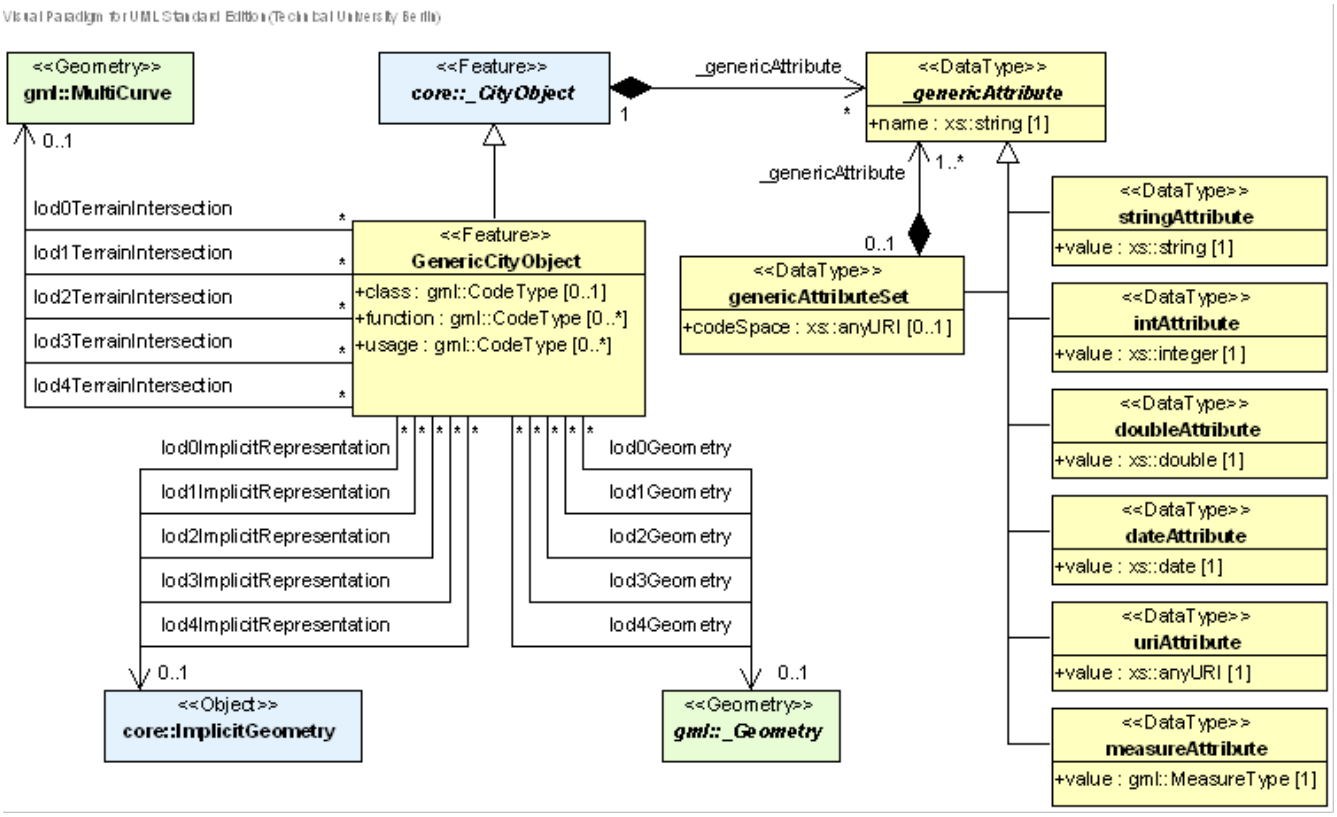


Figure 46. UML diagram of generic objects and attributes in CityGML. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Generics module.

Generic attributes are name-value pairs associated with a city object. Each generic attribute has a mandatory name identifier which can be freely chosen. The data type of the attribute value may be String, Integer, Double (floating point number), URI, Date, and gml:MeasureType. The attribute type is defined by the selection of the particular subclass of _genericAttribute, for example stringAttribute, intAttribute, etc. A measureAttribute facilitates the representation of measured values. Its value is of the structured type gml:MeasureType which provides an optional attribute uom (units of measure) of type xs:anyURI that points to a reference system for the amount.

Generic attributes can be grouped under a common name using a genericAttributeSet. The genericAttributeSet class is derived from _genericAttribute and thus is also realized as generic attribute. Its value is the set of con-tained generic attributes and its name property provides a name identifier for the entire set. Since genericAttrib-uteSet is itself a generic attribute, it may also be contained in a generic attribute set facilitating a recursive nesting of arbitrary depth. The optional codeSpace attribute (of type xs:anyURI) of genericAttributeSet is used to associate the attribute set with an authority, e.g. the organisation or community who defined the attribute set and its contained attributes. By this means, generic attribute sets can be clearly distinguished even if they share the same name.

In order to model generic attributes, the abstract base class _CityObject defined within the CityGML Core module is augmented by the additional property element _genericAttribute using CityGML's

Application Do-main Extension mechanism (cf. chapter 6.12). By this means, each thematic subclass of _CityObject inherits this property and, thus, may be assigned an arbitrary number of generic attributes in order to represent additional properties of features not represented by the explicitly modelled thematic classes of the CityGML data model.

Thus, the Generics module has a deliberate impact on all CityGML extension modules defining thematic sub-classes of _CityObject.

A GenericCityObject may have the attributes class, function and usage defined as gml:CodeType. The class attribute allows an object classification within the thematic area such as pipe, power line, dam, or unknown. The function attribute describes to which thematic area the GenericCityObject belongs (e.g. site, transportation, architecture, energy supply, water supply, unknown etc.). The attribute usage can be used, if the way the object is actually used differs from the function. Both attributes can occur multiple times.

The geometry of a GenericCityObject can either be an explicit GML3 geometry or an ImplicitGeometry (see chapter 8.2). In the case of an explicit geometry the object can have only one geometry for each LOD, which may be an arbitrary 3D GML geometry object (class gml:_Geometry, which is the base class of all GML geome-tries, lodXGeometry, X [0..4]). Absolute coordinates according to the reference system of the city model must be given for the explicit geometry. In the case of an ImplicitGeometry, 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 GML3 geometry with local cartesian coordinates using the property relativeGeometry (further details are given in chapter 8.2).

In order to specify the exact intersection of the DTM with the 3D geometry of a GenericCityObject, the latter can have TerrainIntersectionCurves for every LOD (cf. chapter 6.5). This is important for 3D visualisation but also for certain applications like driving simulators. For example, if a city wall (e.g., the Great Wall of China) should be represented as a GenericCityObject, a smooth transition between the DTM and the city wall would have to be ensured.

XML namespace

The XML namespace of the CityGML Generics module is identified by the Uniform Resource Identifier (URI) http://www.opengis.net/citygml/generics/2.0. Within the XML Schema definition of the Generics module, this URI is also used to identify the default namespace.

6.10.1. Generic city object

GenericCityObjectType, GenericCityObject

NOTE	insert GenericCityObjectType, GenericCityObject UML

6.10.2. Generic attributes

NOTE	insert AbstractGenericAttributeType, _genericAttribute, StringAttributeType, stringAttribute, etc. UML

GenericAttributeSetType, genericAttributeSet

NOTE	insert GenericAttributeSetType, genericAttributeSet UML

6.10.3. Code lists

The attributes class, function, and usage of the feature GenericCityObject are specified as gml:CodeType. The values of these properties can be enumerated in code lists.

6.11. Land Use

Contributors
TBD

NOTE	The following text needs to be reviewed and updated.

LandUse objects can be used to describe areas of the earth's surface dedicated to a specific land use, but also to describe areas of the earth's surface having a specific land cover with or without vegetation, such as sand, rock, mud flats, forest, grasslands, etc (i.e. the physical appearance). Land use and land cover are different concepts; the first describes human activities on the earth's surface, the second describes its physical and biological cover. However, the two are interlinked and often mixed in practice. LandUse objects in CityGML represent both concepts: They can be employed to represent, parcels, spatial planning objects, recreational objects and objects describing the physical characteristics of an area, in 3D (e.g. wetlands). Fig. 71 shows the UML diagram of land use objects, for the XML schema definition see chapter 10.10.1 and annex A.8. The land use model of CityGML is provided by the thematic extension module LandUse (cf. chapter 7).

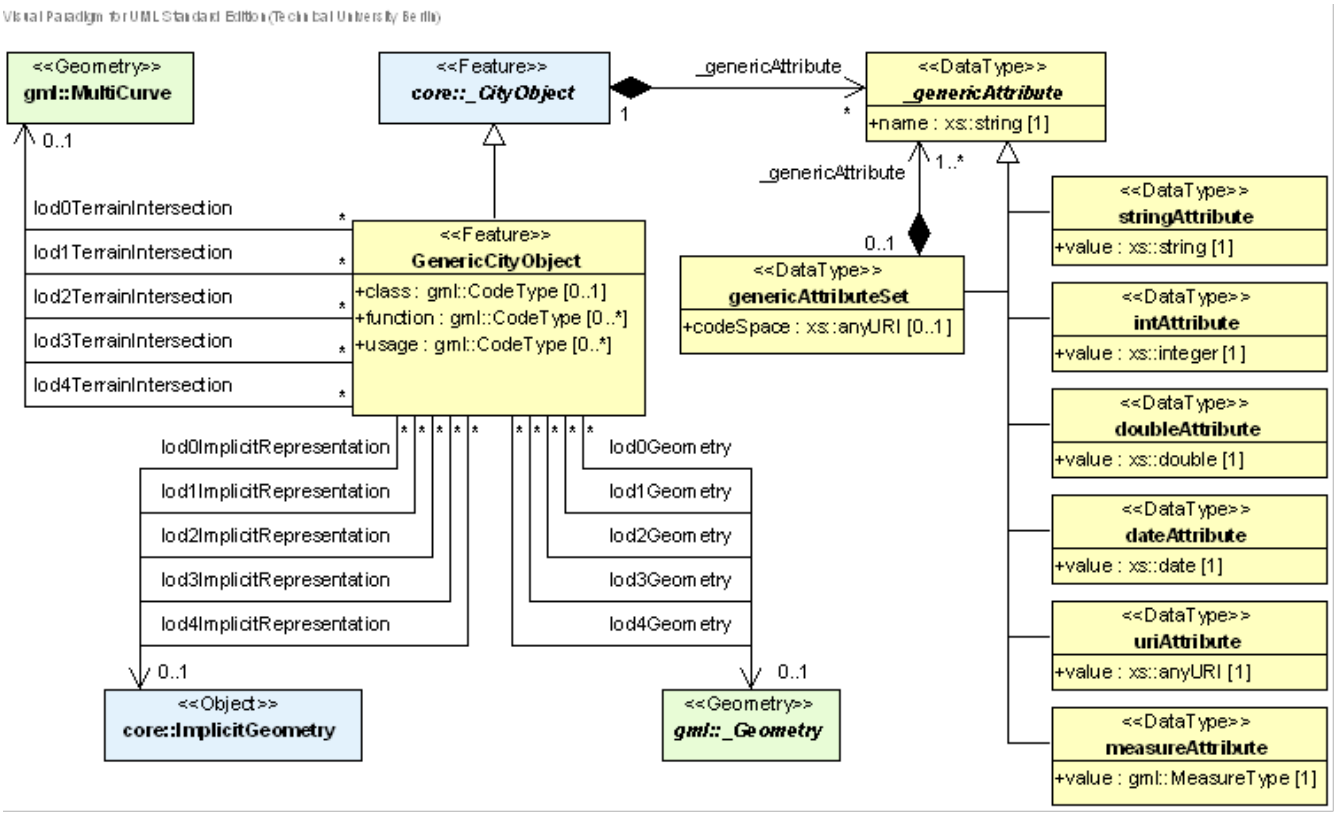


Figure 47. UML diagram of land use objects in CityGML. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML LandUse module.

Every LandUse object may have the attributes class, function, and usage. The class attribute is used to represent the classification of land use objects, like settlement area, industrial area, farmland etc., and can occur only once. The possible values can be specified in a code list (cf. annex C.5). The attribute function defines the purpose of the object or their nature, like e.g. cornfield or heath, while the attribute usage can be used, if the way the object is actually used differs from the function. Both attributes can occur multiple times.

The LandUse object is defined for all LOD 0-4 and may have different geometries in any LOD. The surface geometry of a LandUse object is required to have 3D coordinate values. It must be a GML3 MultiSurface, which might be assigned appearance properties like textures or colors (using CityGML's appearance model, cf. chapter 9).

LandUse objects can be employed to establish a coherent geometric/semantical tesselation of the earth's surface. In this case topological relations between neighbouring LandUse objects should be made explicit by defining the boundary LineStrings only once and by referencing them in the corresponding Polygons using XLinks (cf. chapter 8.1). Fig. 72 shows a land use tesselation, where the geometries of the land use objects are represented as triangulated surfaces. In fact, they are the result of a constrained triangulation of a DTM with consideration of breaklines defined by a 2D vector map of land use classifications.

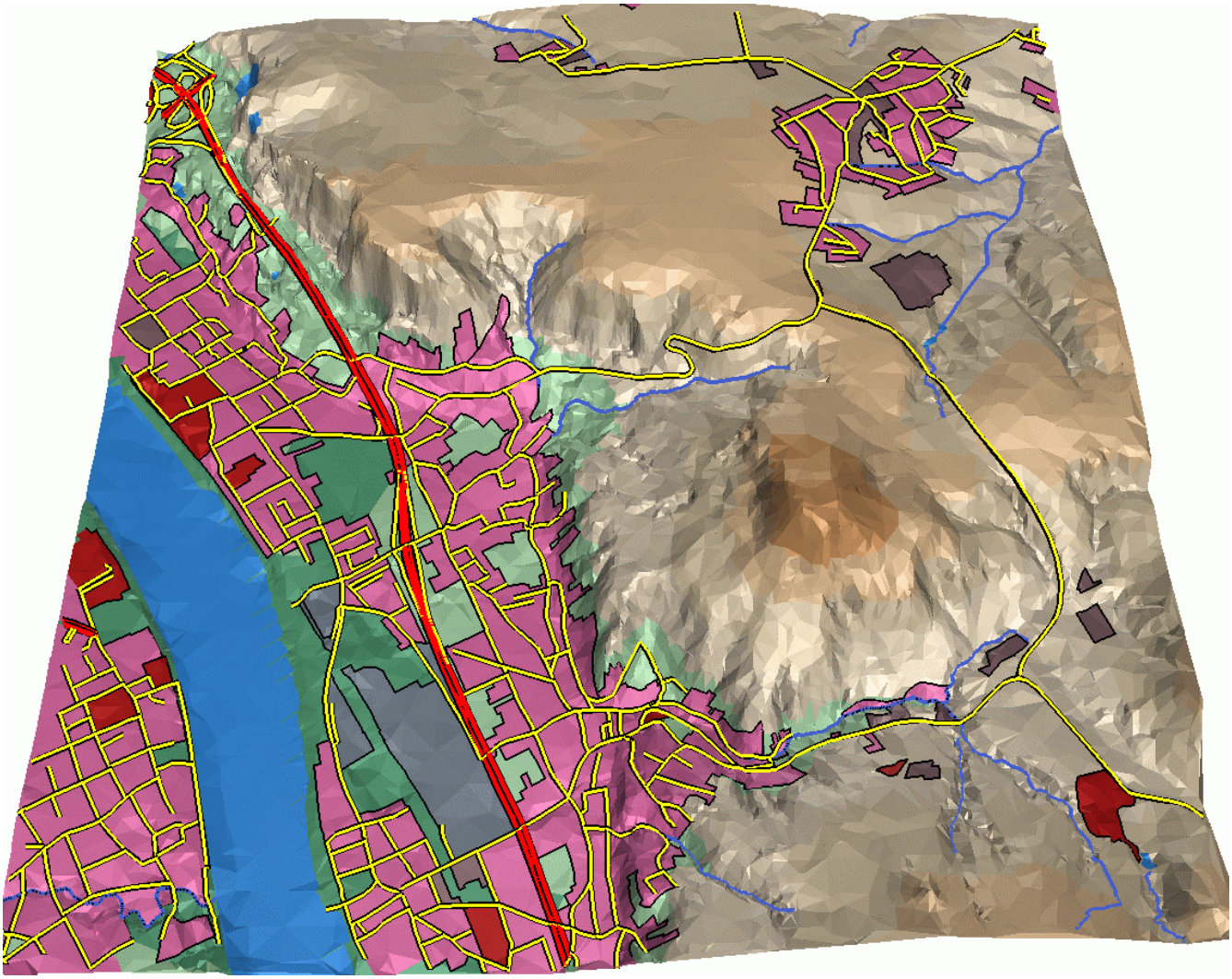


Figure 48. LOD0 regional model consisting of land use objects in CityGML (source: IGG Uni Bonn).

XML namespace

The XML namespace of the CityGML LandUse module is identified by the Uniform Resource Identifier (URI) http://www.opengis.net/citygml/landuse/2.0. Within the XML Schema definition of the LandUse module, this URI is also used to identify the default namespace. 10.10.1

6.11.1. Land use object

LandUseType, LandUse

NOTE	insert LandUseType, LandUse UML

6.11.2. Code lists

The attributes class, function, and usage of the feature LandUse are specified as gml:CodeType. The values of these properties can be enumerated in code lists. Proposals for corresponding code lists can be found in annex C.5.

6.12. Point Cloud

Contributors
TBD

NOTE Content Needed

6.13. Relief

Contributors
TBD

NOTE The following text needs to be reviewed and updated.

An essential part of a city model is the terrain. The Digital Terrain Model (DTM) of CityGML is provided by the thematic extension module Relief (cf. chapter 7). In CityGML, the terrain is represented by the class ReliefFea-ture in LOD 0-4 (Fig. 24 depicts the UML diagram, for the XML schema definition see annex A.9). A Re-liefFeature consists of one or more entities of the class ReliefComponent. Its validity may be restricted to a certain area defined by an optional validity extent polygon. As ReliefFeature and ReliefComponent are deriva-tives of _CityObject, the corresponding attributes and relations are inherited. The class ReliefFeature is associat-ed with different concepts of terrain representations which can coexist. The terrain may be specified as a regular raster or grid (RasterRelief), as a TIN (Triangulated Irregular Network, TINReflief), by break lines (BreaklineRe-lief), or by mass points (MasspointRelief). The four types are implemented by the corresponding GML3 classes: grids by gml:RectifiedGridCoverage, break lines by gml:MultiCurve, mass points by gml:MultiPoint and TINs either by gml:TriangulatedSurface or by gml:Tin. In case of gml:TriangulatedSurfaces, the triangles are given explicitly while in case of gml:Tin only 3D points are represented, where the triangulation can be reconstructed by standard methods (Delaunay triangulation, cf. Okabe et al. 1992). Break lines are represented by 3D curves. Mass points are simply a set of 3D points.

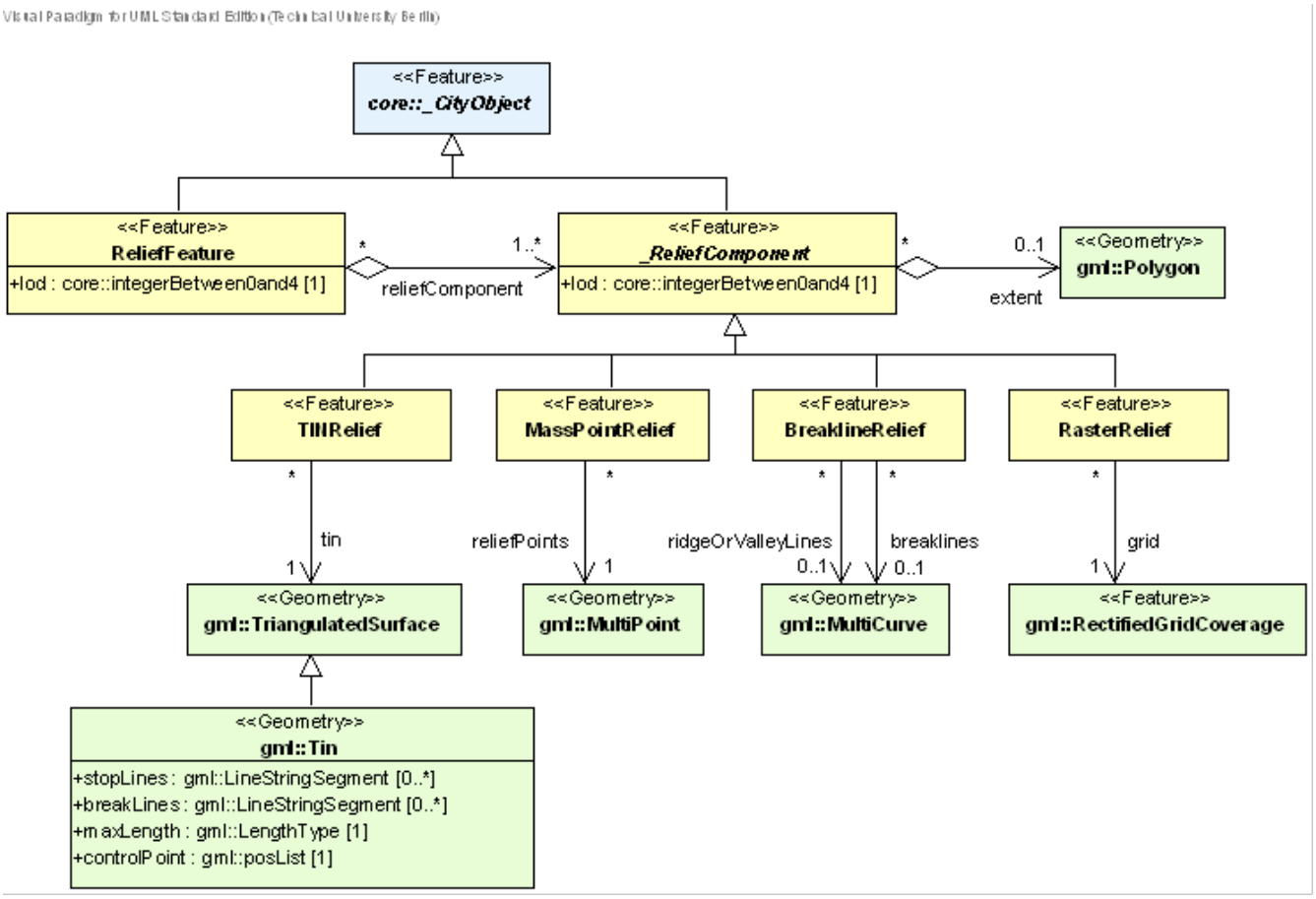


Figure 49. UML diagram of the Digital Terrain Model in CityGML. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Relief module.

In a CityGML dataset the four terrain types may be combined in different ways, yielding a high flexibility. First, each type may be represented in different levels of detail, reflecting different accuracies or resolutions. Second, a part of the terrain can be described by the combination of multiple types, for example by a raster and break lines, or by a TIN and break lines. In this case, the break lines must share the geometry with the triangles. Third, neighboring regions may be represented by different types of terrain models. To facilitate this combination, each terrain object is provided with a spatial attribute denoting its extent of validity (Fig. 25). In most cases, the extent of validity of a regular raster dataset corresponds to its bounding box. This validity extent is represented by a 2D footprint polygon, which may have holes. This concept enables, for example, the modelling of a terrain by a coarse grid, where some distinguished regions are represented by a detailed, high-accuracy TIN. The boundaries between both types are given by the extent attributes of the corresponding terrain objects.

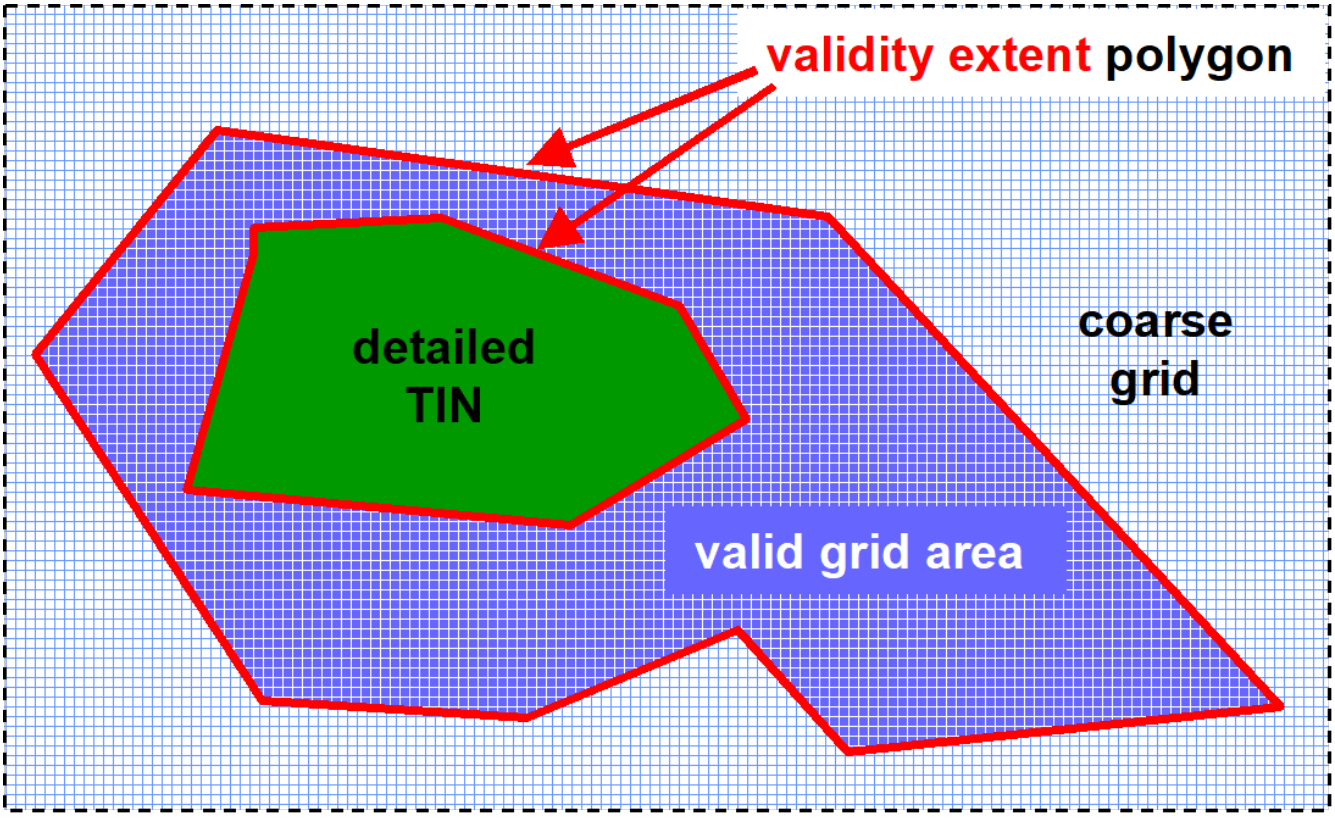


Figure 50. Nested DTMs in CityGML using validity extent polygons (graphic: IGG Uni Bonn).

Accuracy and resolution of the DTM are not necessarily dependent on features of other CityGML extenstion modules such as building models. Hence, there is the possibility to integrate building models with higher LOD to a DTM with lower accuracy or resolution.

This approach interacts with the concept of TerrainIntersectionCurves TIC (cf. chapter 6.5). The TIC can be used like break lines to adjust the DTM to different features such as buildings, bridges, or city furnitures, and hence to ensure a consistent representation of the DTM. If necessary, a retriangulation may have to be processed. A TIC can also be derived by the individual intersection of the DTM and the corresponding feature.

ReliefFeature and its ReliefComponents both have an lod attribute denoting the corresponding level of detail. In most cases, the LOD of a ReliefFeature matches the LOD of its ReliefComponents. However, it is also allowed to specify a ReliefFeature with a high LOD which consists of ReliefComponents where some of them can have a LOD lower than that of the aggregating ReliefFeature. The idea is that, for example, for a LOD3 scene it might be sufficient to use a regular grid in LOD2 with certain higher precision areas defined by ReliefComponents in LOD3. The LOD2 grid and the LOD3 components can easily be integrated using the concept of the validity extent polygon. Therefore, although some of the ReliefComponents would have been classified to a lower LOD, the whole ReliefFeature would be appropriate to use with other LOD3 models which is indicated by setting its lod value to 3.

6.13.1. Relief feature and relief component

ReliefFeatureType, ReliefFeature

NOTE	insert ReliefFeatureType, ReliefFeature UML

AbstractReliefComponentType, _ReliefComponent

NOTE	insert AbstractReliefComponentType, _ReliefComponent UML

6.13.2. TIN relief

TINReliefType, TINRelief

NOTE	insert TINReliefType, TINRelief UML

The geometry of a TINRelief is defined by the GML geometry class gml:TriangulatedSurface. This allows either the explicit provision of a set of triangles (gml:TriangulatedSurface) or specifying of only the control points, break and stop lines using the subclass gml:Tin of gml:TriangulatedSurface. In the latter case, an application that processes an instance document containing a gml:Tin has to reconstruct the triangulated surface by the applica-tion of a constrained Delaunay triangulation algorithm (cf. Okabe et al. 1992).

6.13.3. Raster relief

RasterReliefType, RasterRelief, Elevation

NOTE	insert RasterReliefType, RasterRelief, Elevation UML

6.13.4. Mass point relief

MassPointReliefType, MassPointRelief

NOTE	insert MassPointReliefType, MassPointRelief UML

6.13.5. Breakline relief

BreaklineReliefType, BreaklineRelief

NOTE	insert BreaklineReliefType, BreaklineRelief UML

The geometry of a BreaklineRelief can be composed of break lines and ridge/valley lines. Whereas break lines indicate abrupt changes of terrain slope, ridge/valley lines in addition mark a change of the sign of the terrain slope gradient. A BreaklineRelief must have at least one of the two properties.

6.14. Transportation

Contributors
C. Heazel - first draft

6.14.1. Synopsis

The Transportation module defines central elements of the traffic infrastructure. This includes the transportation classes `road`, `square`, and `track` for the movement of vehicles, bicycles, and pedestrians, the transportation class `railway` for the movement of wheeled vehicles on rails, as well as the transportation class `waterway` for the movement of vessels upon or within water bodies.

6.14.2. Key Concepts

Railway: Railway represents routes that are utilized by rail vehicles like trams or trains.

Road: Road is intended to be used to represent transportation features that are mainly used by vehicles like cars, for example streets, motorways, and country roads.

Square: A Square is an open area commonly found in cities (e.g. a plaza, market square).

Track: A Track is a small path mainly used by pedestrians.

Waterway: A Waterway is a transportation space used for the movement of vessels upon or within a water body.

Abstract Transportation Space: AbstractTransportationSpace defines the properties and associated concepts which are common to Tracks, Roads, Railways, Squares, and Waterways. The associated concepts include:

- **Traffic Space**: A TrafficSpace is a space in which traffic (transportation) takes place. This is more than a geometry. It includes properties which may of of interest to anyone who wishes to traverse this space. This includes properties such as overhead clearance, surface type, and direction of travel.
- **Markings**: A Marking is a visible pattern on a transportation area relevant to the structuring or restriction of traffic. Examples are road markings and markings related to railway or waterway traffic.
- **Holes**: A Hole is an opening in the surface of a Road, Track or Square such as road damages, manholes or drains. Holes can span multiple transportation objects.
- **Auxillary Traffic Space**: a space within the transportation space not intended for traffic purposes.

6.14.3. Discussion

The transportation model of CityGML is a multi-functional, multi-scale model focusing on thematic and functional as well as on geometrical/topological aspects. Transportation features are represented as a linear network in LOD0. Starting from LOD1, all transportation features are geometrically described by 3D surfaces. The areal modelling of transportation features allows for the application of geometric route planning algorithms. This can be useful to determine restrictions and manoeuvres required along a transportation route. This information can also be employed for trajectory planning of mobile robots in the real world or the automatic placement of avatars (virtual people) or vehicle models in 3D visualisations and training simulators.

The main class is AbstractTransportationSpace, which represents, for example, a road, a track, a

railway, a waterway, or a square. Transportation classes (from left to right: examples of road, track, rail, waterway, and square) (source: Rheinmetall Defence Electronics) illustrates the different transportation classes.

NOTE Need to add a waterway to figure 57.

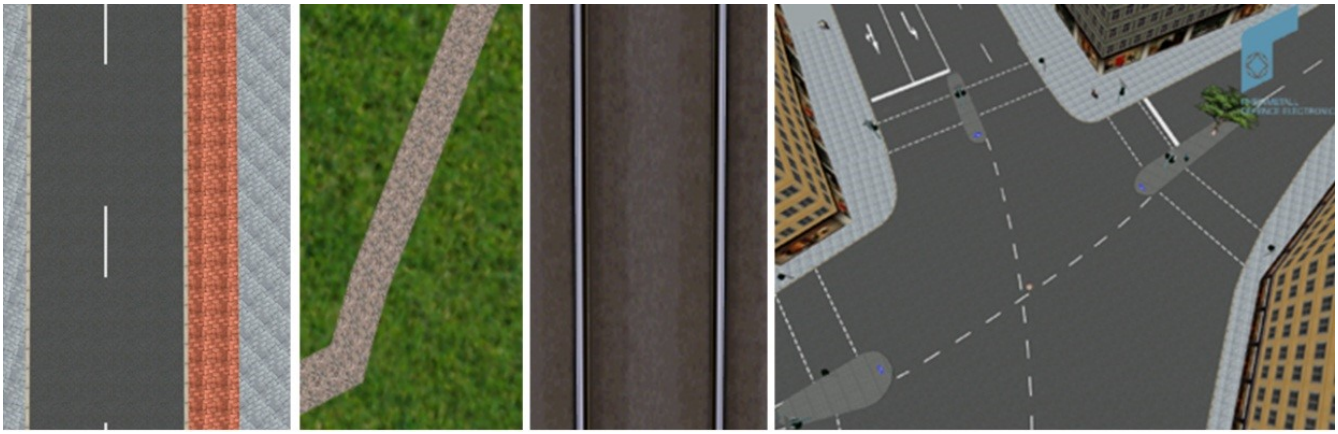


Figure 51. Transportation classes (from left to right: examples of road, track, rail, waterway, and square) (source: Rheinmetall Defence Electronics)

Two major concepts associated with all transportation classes are Traffic Space and Auxillary Traffic Space. Traffic Space describes the space that traffic can traverse. Auxillary Traffic Spaces are associated spaces where traffic does not traverse.

Example LOD2 Traffic Spaces in CityGML: a road, which is the aggregation of TrafficAreas and AuxiliaryTrafficAreas (source: City of Solingen, IGG Uni Bonn) depicts an example for a LOD2 Transportation Space configuration within a virtual 3D city model. The Road consists of several Traffic Spaces for the sidewalks, road lanes, parking lots, and of Auxiliary Traffic Spaces below the raised flower beds.

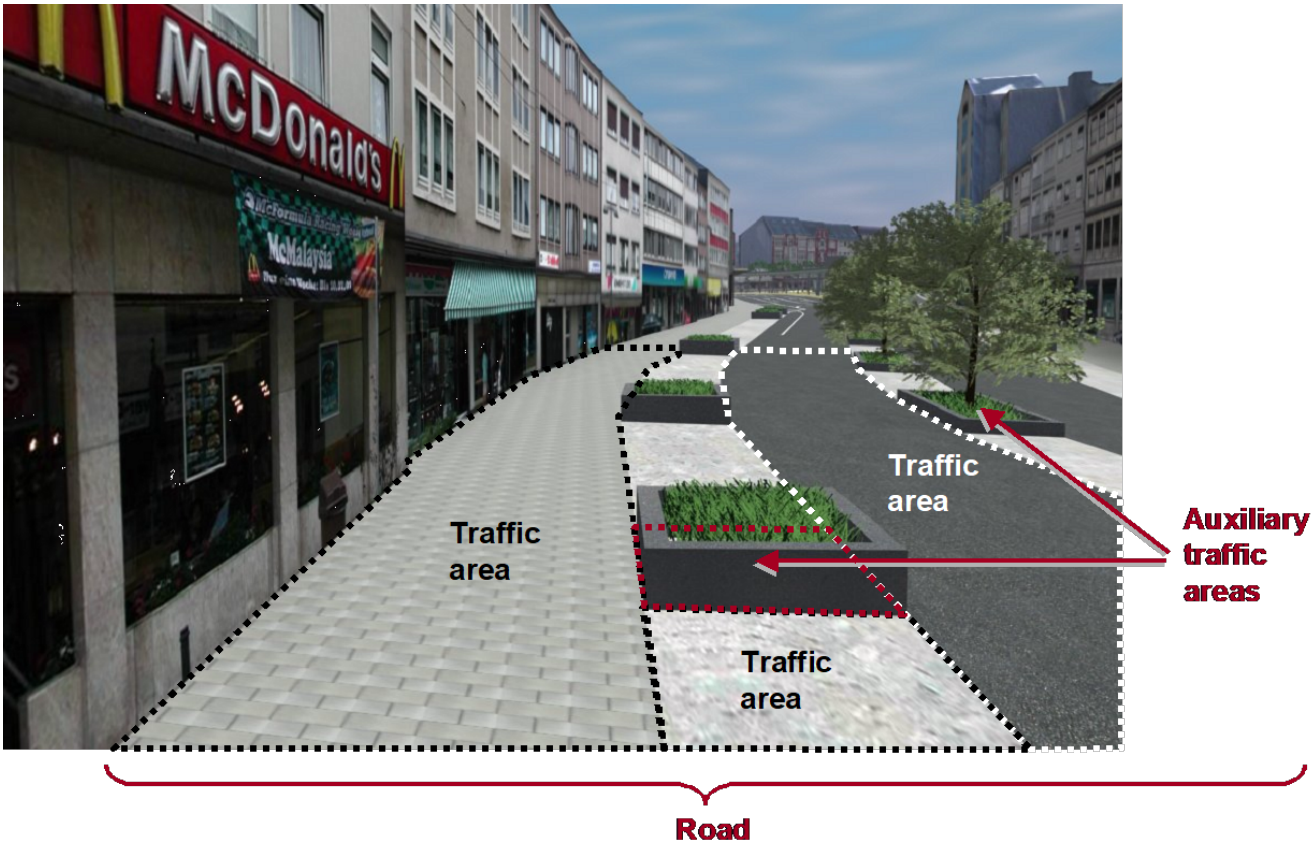


Figure 52. Example LOD2 Traffic Spaces in CityGML: a road, which is the aggregation of TrafficAreas and AuxiliaryTrafficAreas (source: City of Solingen, IGG Uni Bonn)

The road itself is represented as a Road object with associated TrafficSpaces and AuxiliaryTrafficSpaces. The TrafficSpaces are those elements which are important in terms of traffic usage such as car driving lanes, pedestrian zones, and cycle lanes. The AuxiliaryTrafficSpaces describe further elements of the road like kerbstones, middle lanes, and green areas.

6.14.4. Level of Detail

The geometrical representation of the Transportation objects varies through the different levels of detail.

In the coarsest LOD0 the transportation objects are modelled by line objects establishing a linear network. On this abstract level, path finding algorithms or similar analyses can be executed. It also can be used to generate schematic drawings and visualisations of the transport network. Since this abstract definition of transportation network does not contain explicit descriptions of the transportation objects, it may be task of the viewer application to generate the graphical visualisation, for example by using a library with style-definitions (width, color resp. texture) for each transportation object.

Starting from LOD1 a Transportation object provides an explicit surface geometry. This geometry reflects the actual shape of the object, not just its centerline. In LOD2 to LOD4, the object is further subdivided thematically into Traffic Spaces, which are used by transportation, such as cars, trains, public transport, airplanes, bicycles or pedestrians and into Auxiliary Traffic Spaces, which are of minor importance for transportation purposes. The different representations of a Transportation objects for each LOD are illustrated in [figure-60].

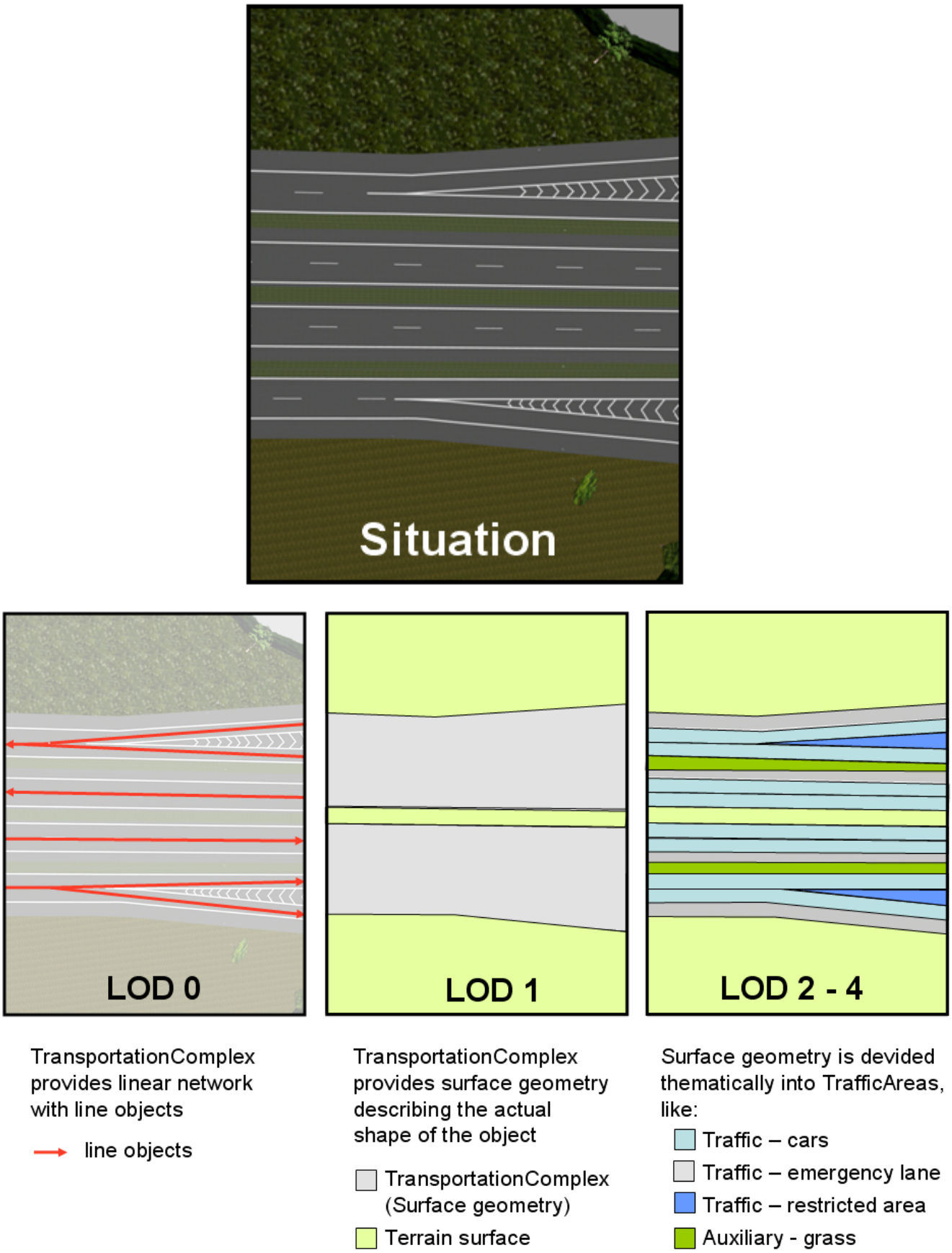


Figure 53. Transportation Concepts in LOD0, 1, and 2-4 (example shows part of a motorway) (source: Rheinmetall Defence Electronics).

In LOD0 areal transportation objects, like squares, should be modeled in the same way as in GDF. GDF is the ISO standard for transportation networks and is used in most car navigation systems. In GDF a square is typically represented as a ring surrounding the place and to which the incident

roads connect. CityGML does not cover further functional aspects of transportation network models (e.g. speed limits) as it is intended to complement and not replace existing standards like GDF. However, if specific functional aspects have to be associated with CityGML transportation objects, generic attributes provided by CityGML's Generics module can be used.

Moreover, further objects of interest can be added from other information systems by the use of ExternalReferences (see Core module). For example, GDF datasets, which provide additional information for car navigation, can be used for simulation and visualisation of traffic flows. The values of the object attributes can be augmented or replaced by the use of CodeLists or ADEs (see Core module). These extensions may be country or user-specific (especially for country-specific road signs and signals).

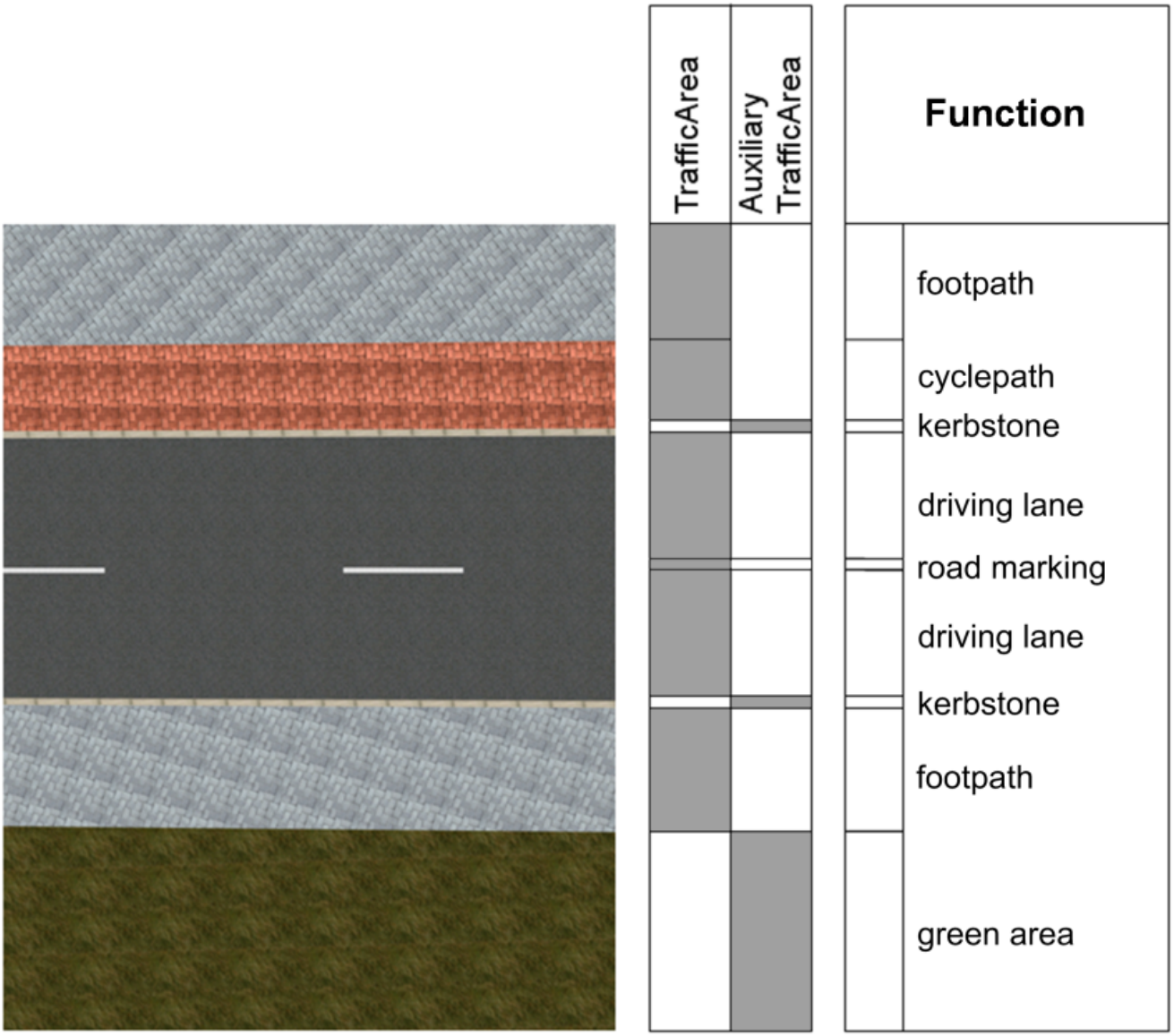


Figure 54. TransportationComplex in LOD 2-4: representation of a road with a complex cross-section profile (example shows urban road) (source: Rheinmetall Defence Electronics).

6.14.5. UML Model

The UML diagram of the Transportation module is depicted in UML diagram of the Transportation Model..

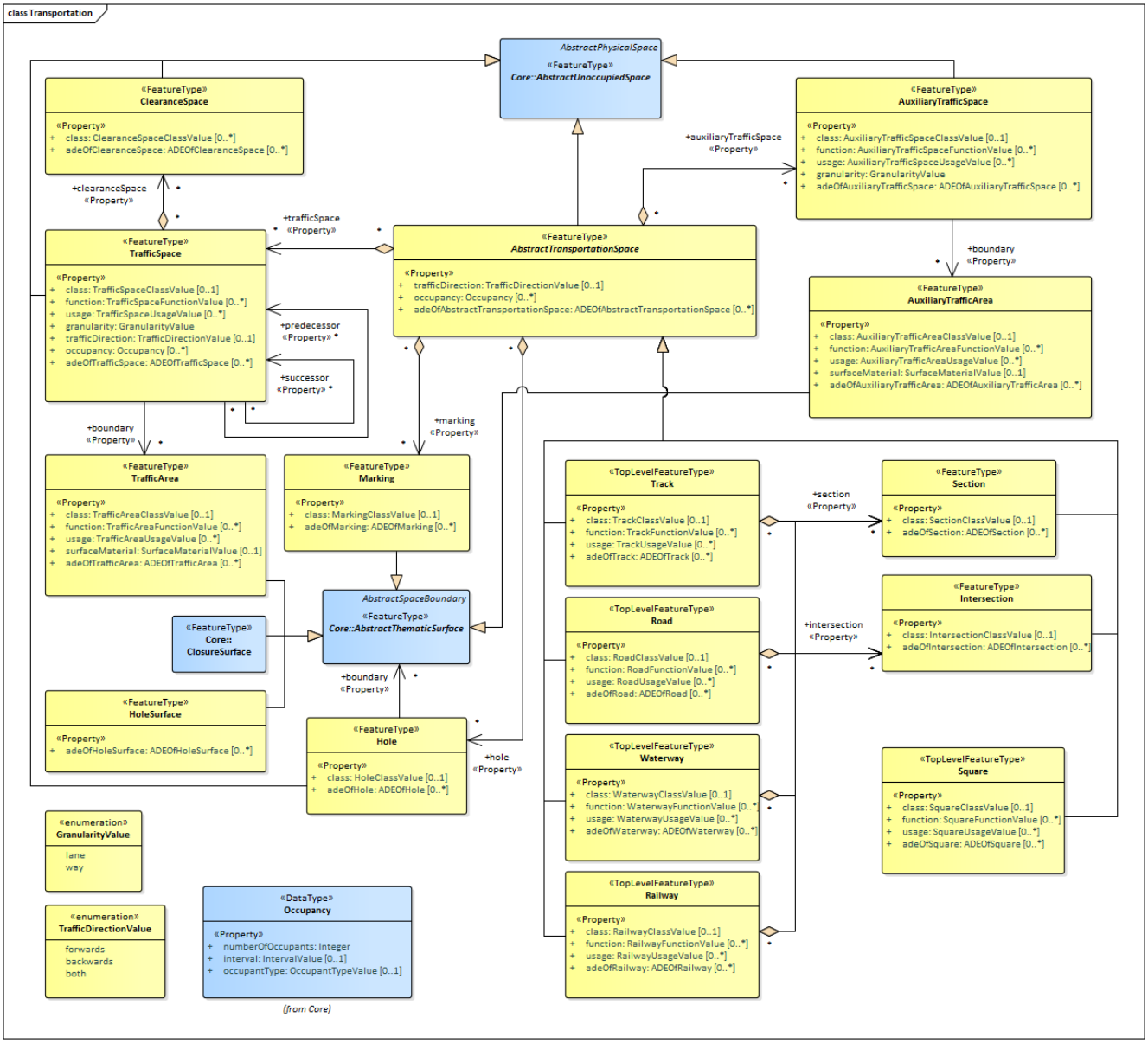


Figure 55. UML diagram of the Transportation Model.

The ADE data types provided for the Transportation module are illustrated in ADE classes of the CityGML Transportation module..

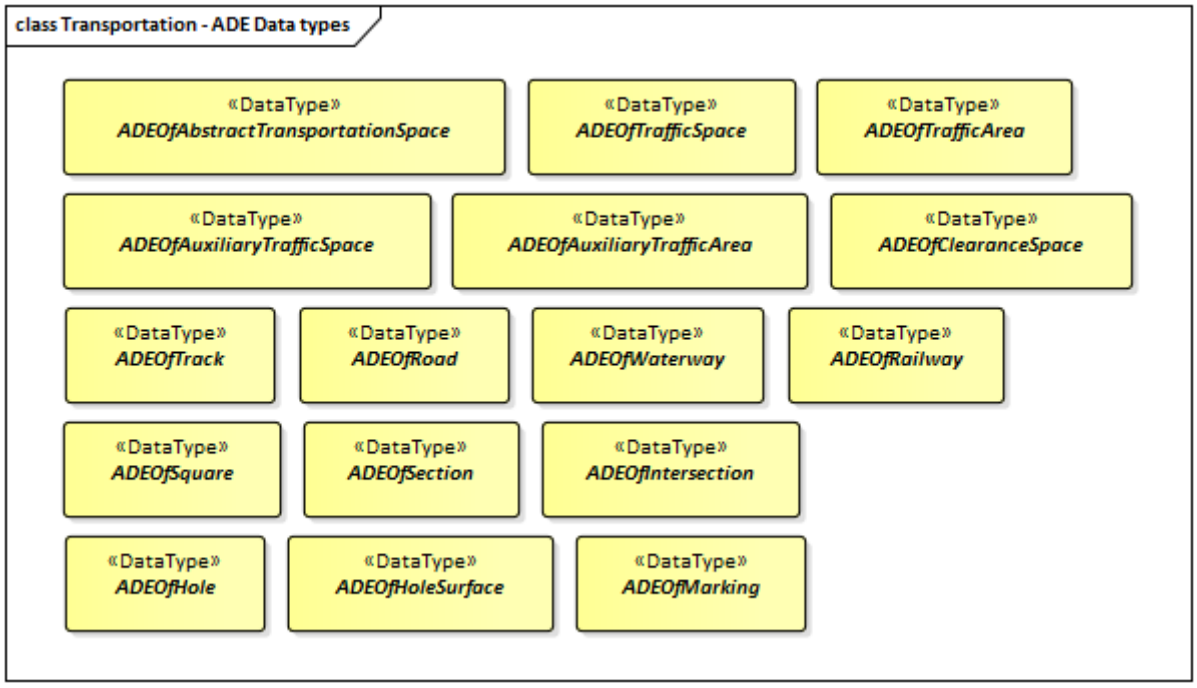


Figure 56. ADE classes of the CityGML Transportation module.

The Code Lists provided for the Transportation module are illustrated in Codelists from the CityGML Transportation module..

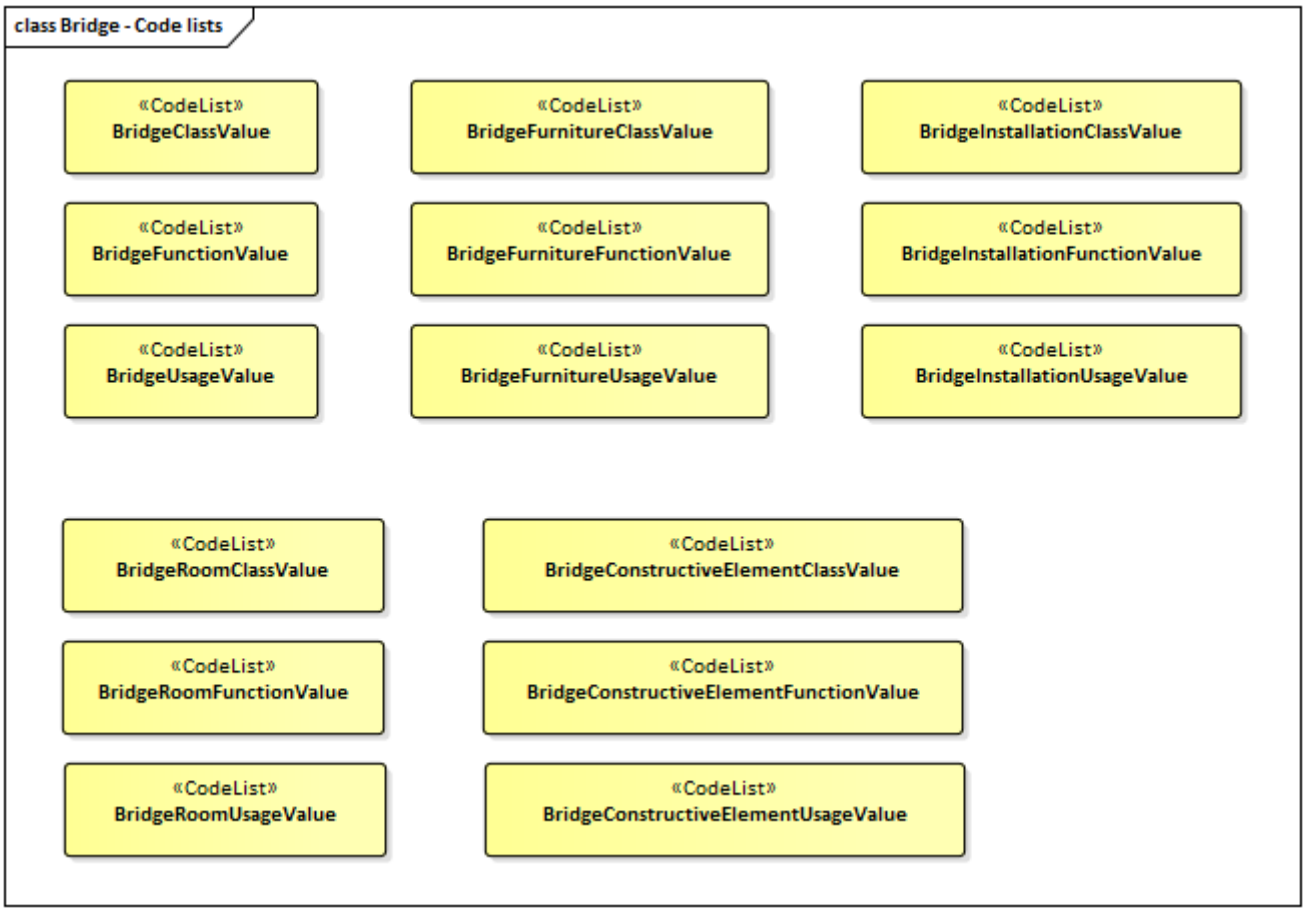


Figure 57. Codelists from the CityGML Transportation module.

6.14.6. Examples

Table 3. Examples of TrafficArea

Example	Country Road	Motorway Entry
TransportationComplex – Function	road	road
TrafficArea – Usage	car, truck, bus, taxi, motorcycle	car, truck, bus, taxi, motorcycle
TrafficArea – Function	driving lane	motorway_entry
TrafficArea – SurfaceMaterial	asphalt	concrete

The following example shows a complex urban crossing. The picture on the left is a screenshot of an editor application for a training simulator, which allows the definition of road networks consisting of transportation objects, external references, buildings and vegetation objects. On the right, the 3D representation of the defined crossing is shown including all referenced static and dynamic models.

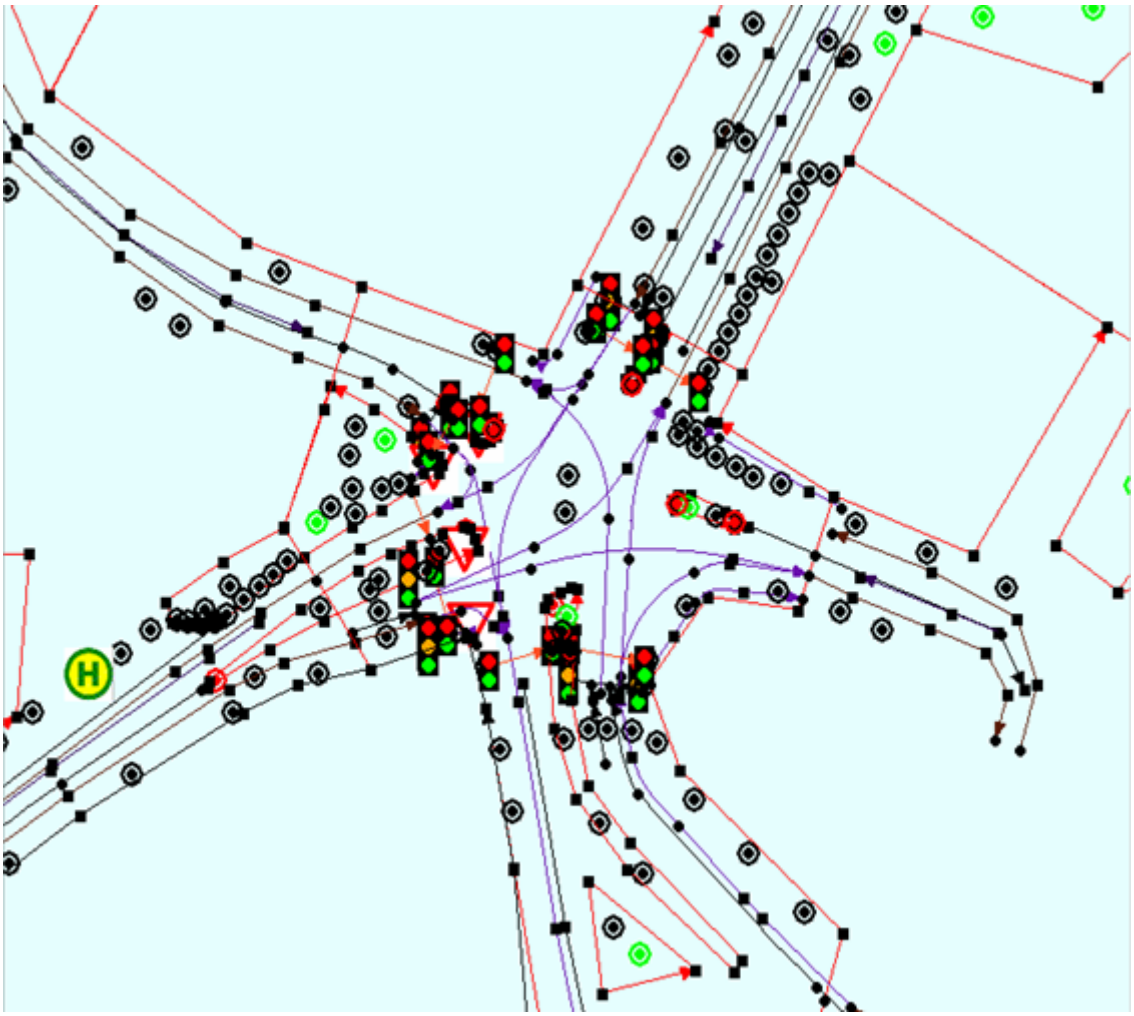


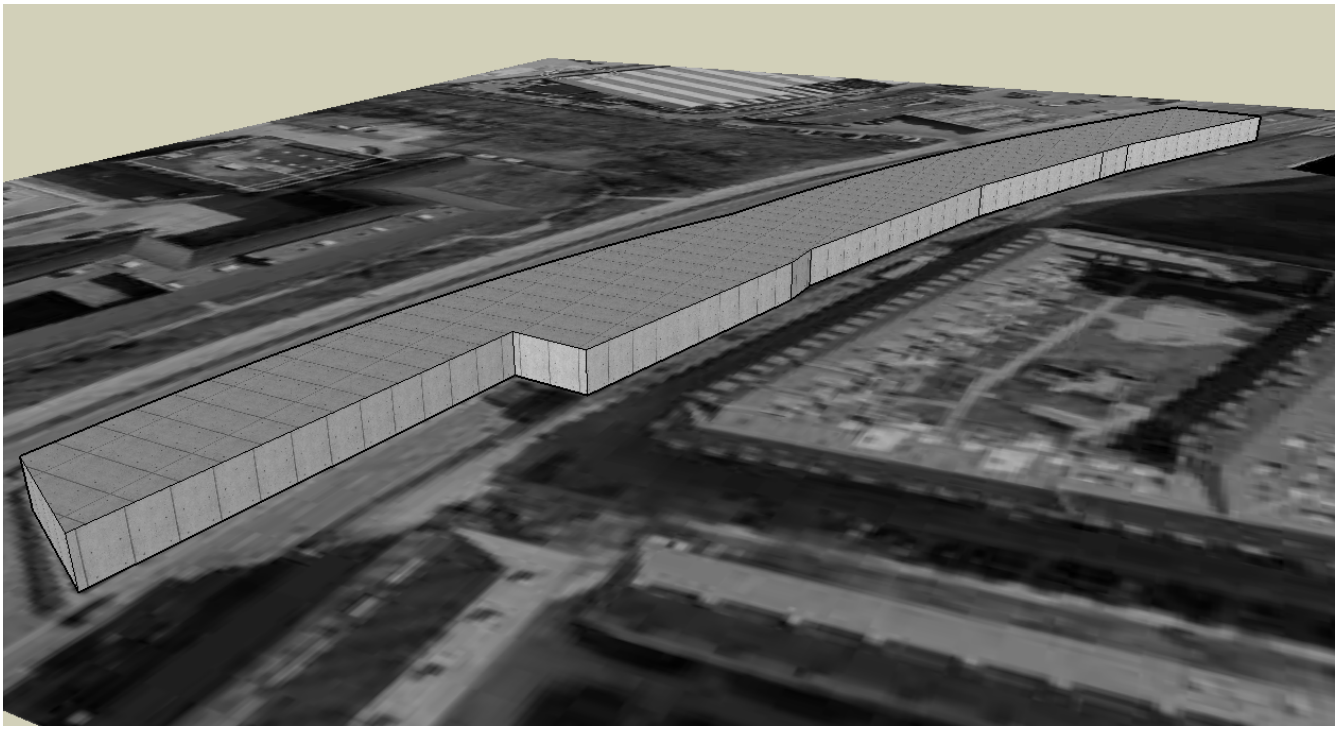
Figure 58. Complex urban intersection (left: linear transportation network with surface descriptions and external references, right: generated scene) (source: Rheinmetall Defence Electronics).

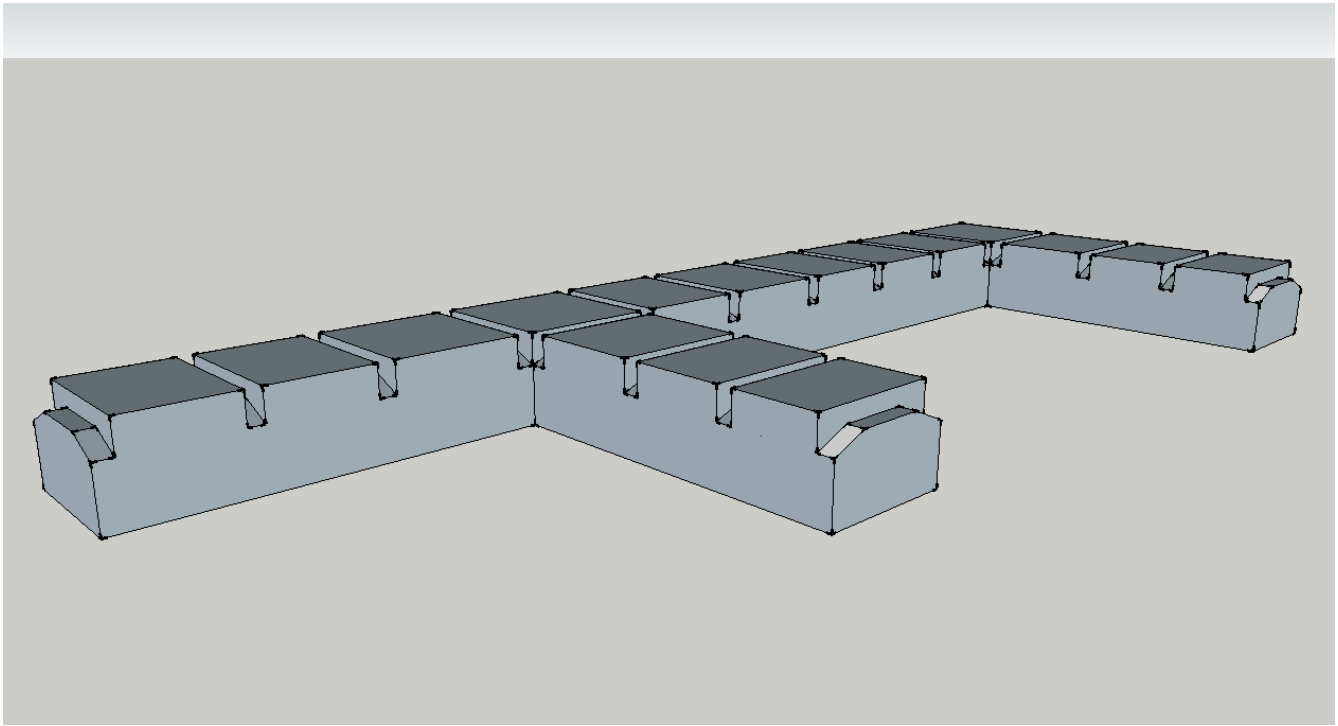
6.15. Tunnel Model

Contributors
TBD

NOTE The following text needs to be reviewed and updated.

The tunnel model is closely related to the building model. It supports the representation of thematic and spatial aspects of tunnels and tunnel parts in four levels of detail, LOD1 to LOD4. The tunnel model of CityGML is defined by the thematic extension module Tunnel (cf. chapter 7). Fig. 37 provides examples of tunnel models for each LOD.





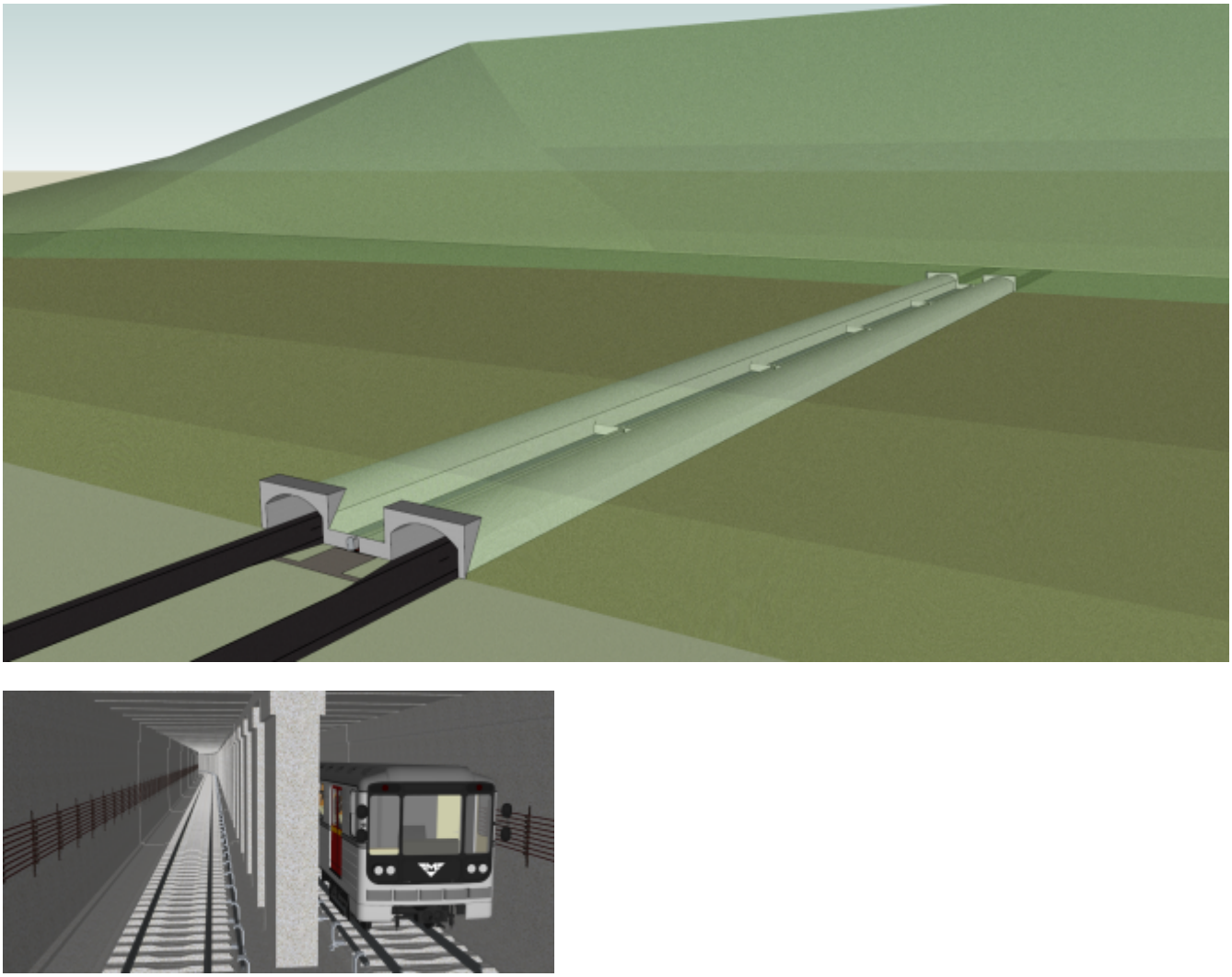


Figure 59. Examples for tunnel models in LOD1 (upper left), LOD2 (upper right), LOD3 (lower left) and LOD4 (lower right) (source: Google 3D warehouse).

The UML diagram of the tunnel model is shown in Fig. 38. The XML schema definition is attached in annex A.11. The pivotal class of the model is _AbstractTunnel, which is a subclass of the thematic class _Site (and transitively of the root class _CityObject). _AbstractTunnel is specialized either to a Tunnel or to a TunnelPart. Since an _AbstractTunnel consists of TunnelParts, which again are _AbstractTunnels, an aggregation hierarchy of arbitrary depth may be realized. As subclass of the root class _CityObject, an _AbstractTunnel inherits all properties from _CityObject like the GML3 standard feature properties (gml:name etc.) and the CityGML specific properties like ExternalReferences (cf. chapter 6.7). Further properties not explicitly covered by _AbstractTunnel may be modelled as generic attributes provided by the CityGML Generics module (cf. chapter 10.12) or using the CityGML Application Domain Extension mechanism (cf. chapter 10.13).

Both classes Tunnel and TunnelPart inherit the attributes of _AbstractTunnel: the class of the tunnel, the func-tion, the usage, the year of construction and the year of demolition. In contrast to _AbstractBuilding, Address features cannot be assigned to _AbstractTunnel.

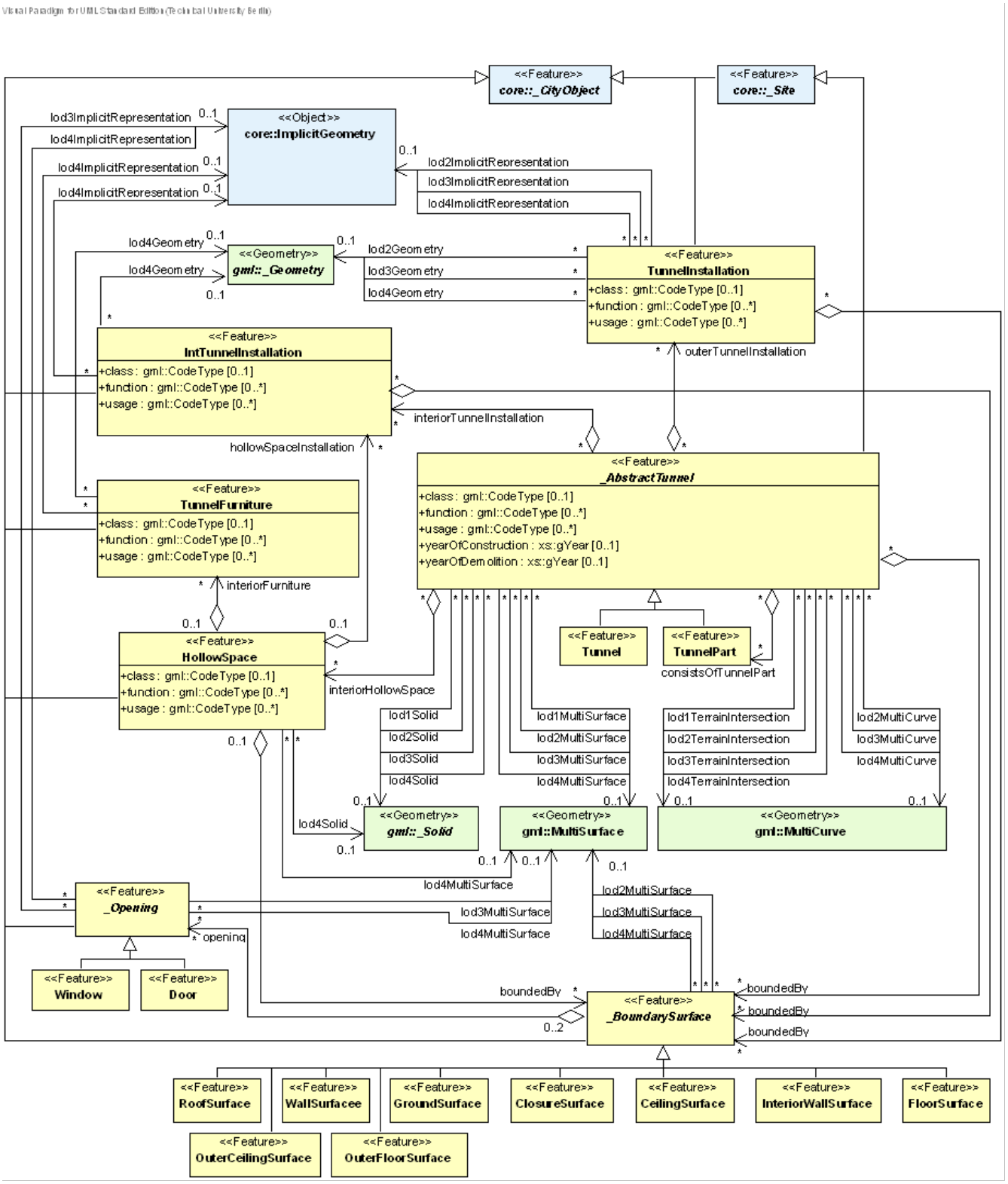


Figure 60. UML diagram of CityGML's tunnel model. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Tunnel module.

The geometric representation and semantic structure of an _AbstractTunnel is shown in Fig. 38. The model is successively refined from LOD1 to LOD4. Therefore, not all components of a tunnel model are represented equally in each LOD and not all aggregation levels are allowed in each LOD. In CityGML, all object classes are associated to the LODs with respect to the proposed minimum acquisition criteria for each LOD (cf. chapter 6.2). An object can be represented simultaneously in different LODs by providing distinct geometries for the corresponding LODs.

Similar to the building and brigde models (cf. chapters 10.3 and 10.5), only the outer shell of a tunnel is repre-sented in LOD1 – 3, which is composed of the tunnel's boundary surfaces to the surrounding earth, water, or outdoor air. The interior of a tunnel may only be modeled in LOD4. Although the interior built environment is especially relevant for subsurface objects like tunnels or underground buildings, CityGML employs a consistent LOD concept for all thematic modules. If, in contrast, the representation of the interior of subsurface objects would be possible in all LODs, the LOD concept for subsurface objects would have to substantially differ from the LOD concept for aboveground objects. This would require the precise definition of a "transition surface" which delimits the scope of both LOD concepts. Furthermore, features being partially above and below ground would have to be split into an above-ground part (modeled according to the aboveground LOD concept) and a subsurface part (modeled according to the subsurface LOD concept). However, such a splitting violates the CityGML concept of unity of features and would not be feasible in many cases where the transition between above and below ground is often not precisely known or depends on (the LOD of) the terrain model. Hence, CityGML applies a single and consistent LOD concept to both aboveground and subsurface objects. As a conse-quence, penetrations between a tunnel and objects inside this tunnel (e.g., roads and railways) may occur in LOD1 – 3.

In LOD1, a tunnel model consists of a geometric representation of the tunnel volume. Optionally, a MultiCurve representing the TerrainIntersectionCurve (cf. chapter 6.5) can be specified. The geometric representation is refined in LOD2 by additional MultiSurface and MultiCurve geometries.

In LOD2 and higher LODs the outer structure of a tunnel can also be differentiated semantically by the classes _BoundarySurface and TunnelInstallation. A boundary surface is a part of the tunnel's exterior shell with a special function like wall (WallSurface), roof (RoofSurface), ground plate (GroundSurface), outer floor (Outer-FloorSurface), outer ceiling (OuterCeilingSurface) or ClosureSurface. The TunnelInstallation class is used for tunnel elements like outer stairs, strongly affecting the outer appearance of a tunnel. A TunnelInstallation may have the attributes class, function and usage (see Fig. 38).

In LOD3, the openings in _BoundarySurface objects (doors and windows) can be represented as thematic ob-jects.

In LOD4, the highest level of resolution, also the interior of a tunnel, composed of several hollow spaces, is represented in the tunnel model by the class HollowSpace. This enlargement allows a virtual accessibility of tunnels, e.g. for driving through a tunnel, for simulating disaster management or for presenting the light illumi-nation within a tunnel. The aggregation of hollow spaces according to arbitrary, user defined criteria (e.g. for defining the hollow spaces corresponding to horizontal or vertical sections) is achieved by employing the general grouping concept provided by CityGML (cf. chapter 10.11). Interior installations of a tunnel, i.e. objects within a tunnel which (in contrast to furniture) cannot be moved, are represented by the class IntTunnelInstallation. If an installation is attached to a specific hollow space (e.g. lamps, ventilator), they are associated with the Hol-lowSpace class, otherwise (e.g. pipes) with _AbstractTunnel. A HollowSpace may have the attributes class, function and usage whose possible values can be enumerated in code lists (chapter 10.4.7, Annex C). The class attribute allows a general classification of hollow spaces, e.g. commercial or private rooms, and occurs only once. The function attribute is intended to express the main purpose of the hollow space, e.g. control area, installation space, storage space. The attribute usage can be used if the way the object is actually used differs from the function. Both attributes can occur multiple times.

The visible surface of a hollow space is represented geometrically as a Solid or MultiSurface. Semantically, the surface can be structured into specialised _BoundarySurfaces, representing floor (FloorSurface), ceiling (Ceil-ingSurface), and interior walls (InteriorWallSurface). Hollow space furniture, like movable equipment in control areas, can be represented in the CityGML tunnel model with the class TunnelFurniture. A TunnelFurniture may have the attributes class, function and usage.

6.15.1. Tunnel and Tunnel Part

TunnelType, Tunnel

NOTE	insert TunnelType, Tunnel UML

The Tunnel class is one of the two subclasses of _AbstractTunnel. If a tunnel only consists of one (homogene-ous) part, this class shall be used. A tunnel composed of structural segments, for example tunnel entrance and subway, has to be separated into one tunnel having one or more additional TunnelPart (see Fig. 39). The geome-try and non-spatial properties of the central part of the tunnel should be represented in the aggregating Tunnel feature.

TunnelPartType, TunnelPart

NOTE	insert TunnelPartType, TunnelPart UML

If sections of a tunnel differ in geometry and / or attributes, the tunnel can be separated into parts (see Fig. 39). Like Tunnel, the class TunnelPart is derived from _AbstractTunnel and inherites all attributes of _AbstractTunnel. A TunnelPart object should be uniquely related to exactly one tunnel or tunnel part object.

Figure 61. Example of a tunnel modeled with two tunnel parts (source: Helmut Stracke).

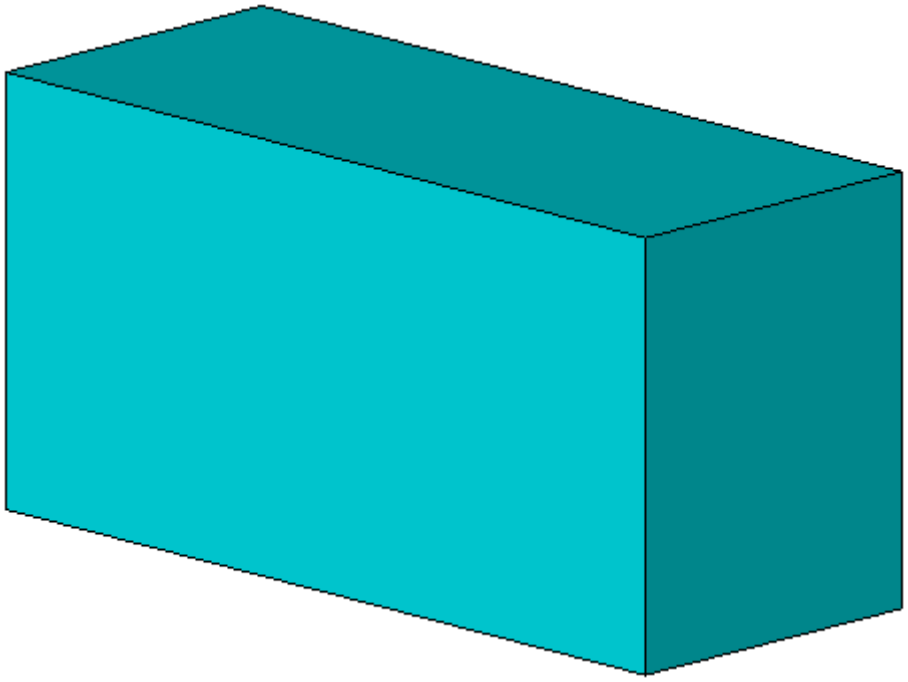
AbstractTunnelType, _AbstractTunnel

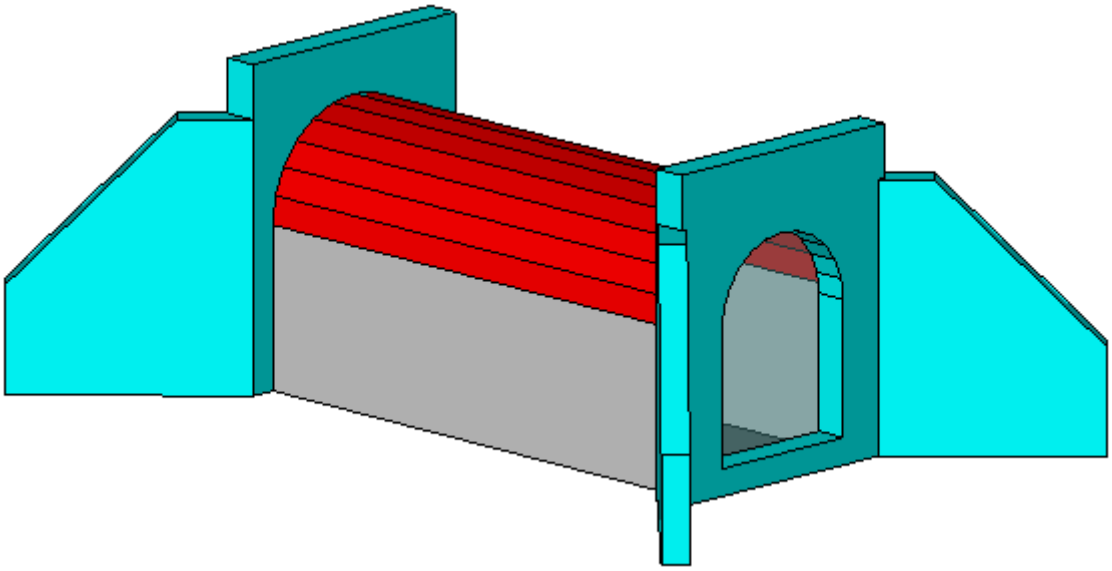
NOTE	insert AbstractTunnelType, _AbstractTunnel UML

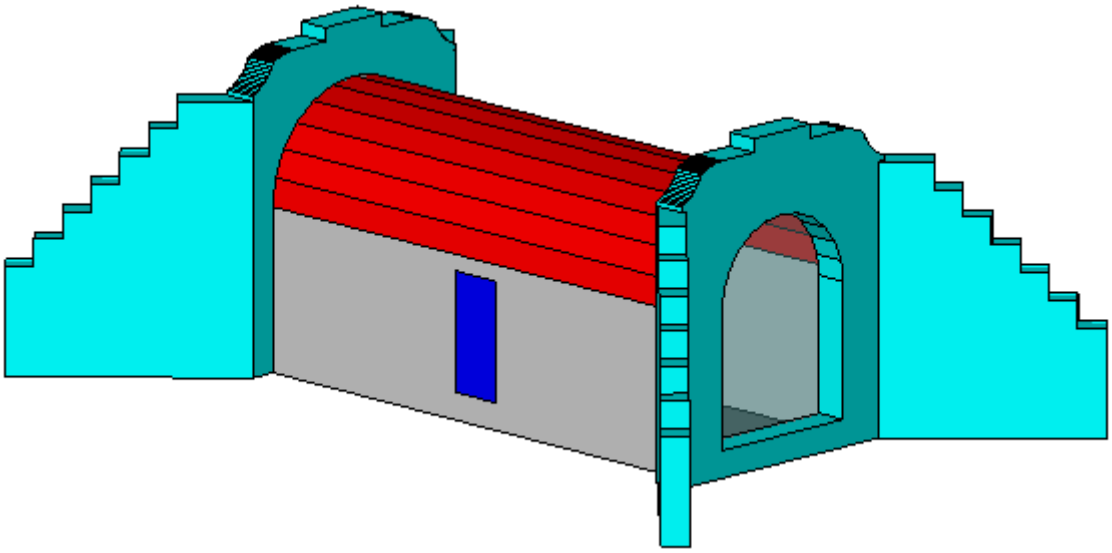
The abstract class _AbstractTunnel contains properties for tunnel attributes, purely geometric representations, and geometric/semantic representations of the tunnel or tunnel part in different levels of detail. The attributes describe:

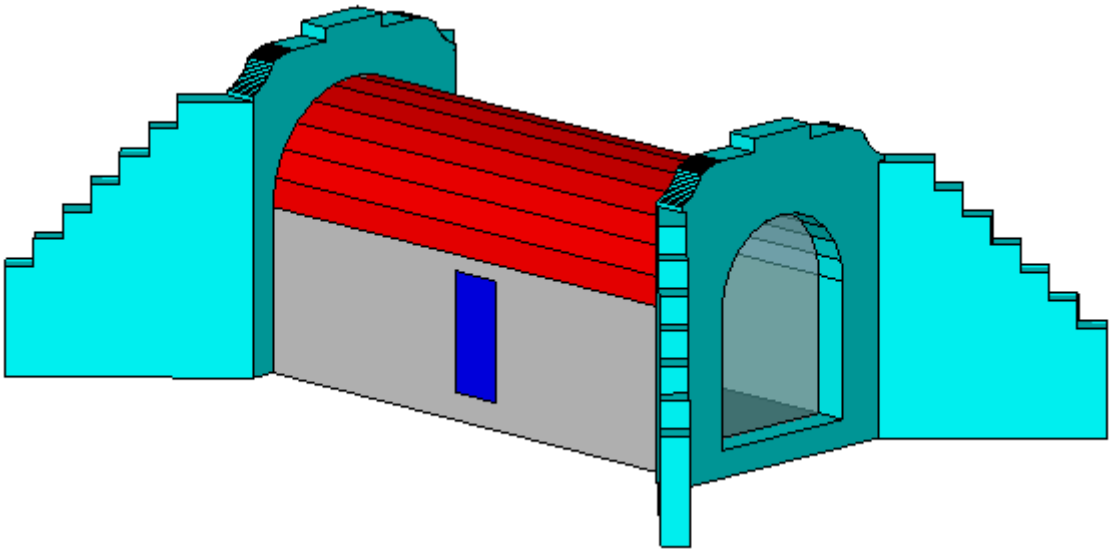
1. The classification of the tunnel or tunnel part (class), the different functions (function), and the usage (us-age). The type of these attributes is gml:CodeType and the values can be specified in separate code lists.
2. The year of construction (yearOfConstruction) and the year of demolition (yearOfDemolition) of the tunnel or tunnel part. The yearOfConstruction is the year of completion of the tunnel. The yearOfDemolition is the year when the demolition of the tunnel was completed. The date (year) refer to real world time (e.g. 2011).

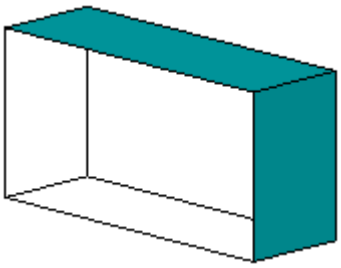
Spanning the different levels of detail, the tunnel model differs in the complexity and granularity of the geomet-ric representation and the thematic structuring of the model into components with a special semantic meaning. This is illustrated in Fig. 40, showing the same tunnel in four different LODs. Some properties of the class _AbstractTunnel are also associated with certain LODs.

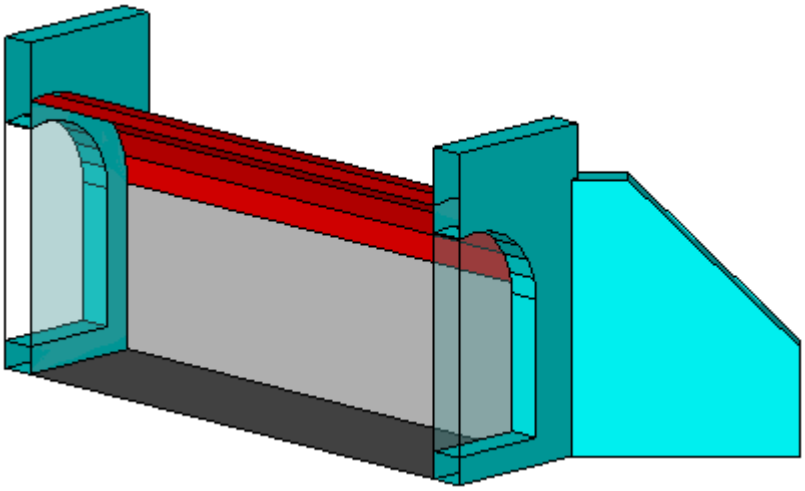


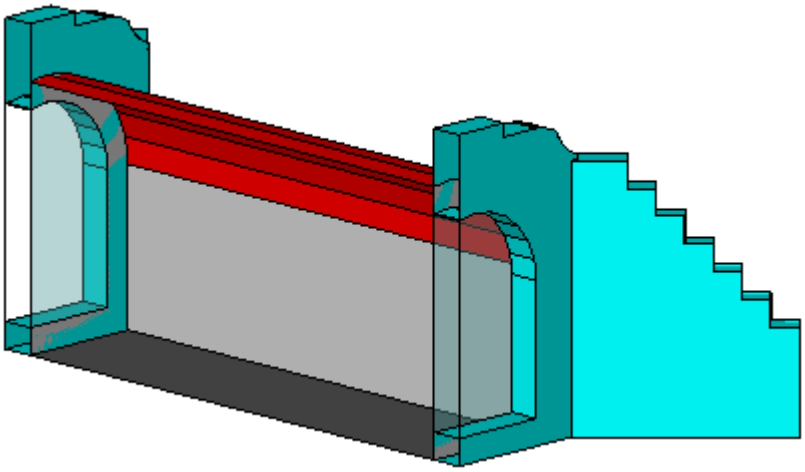


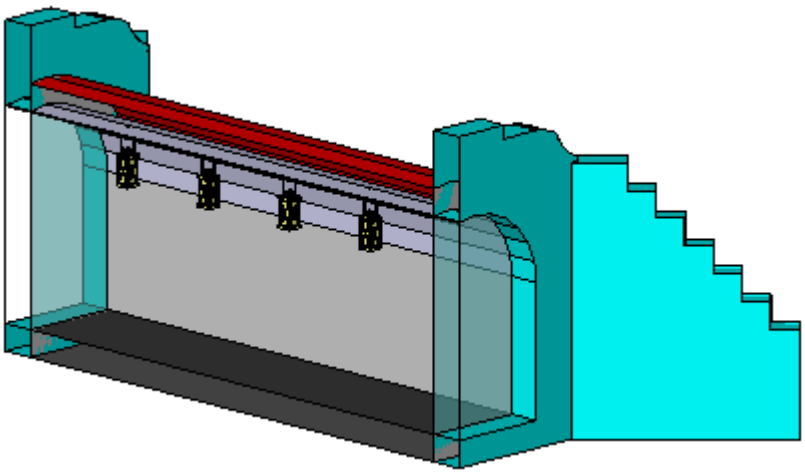












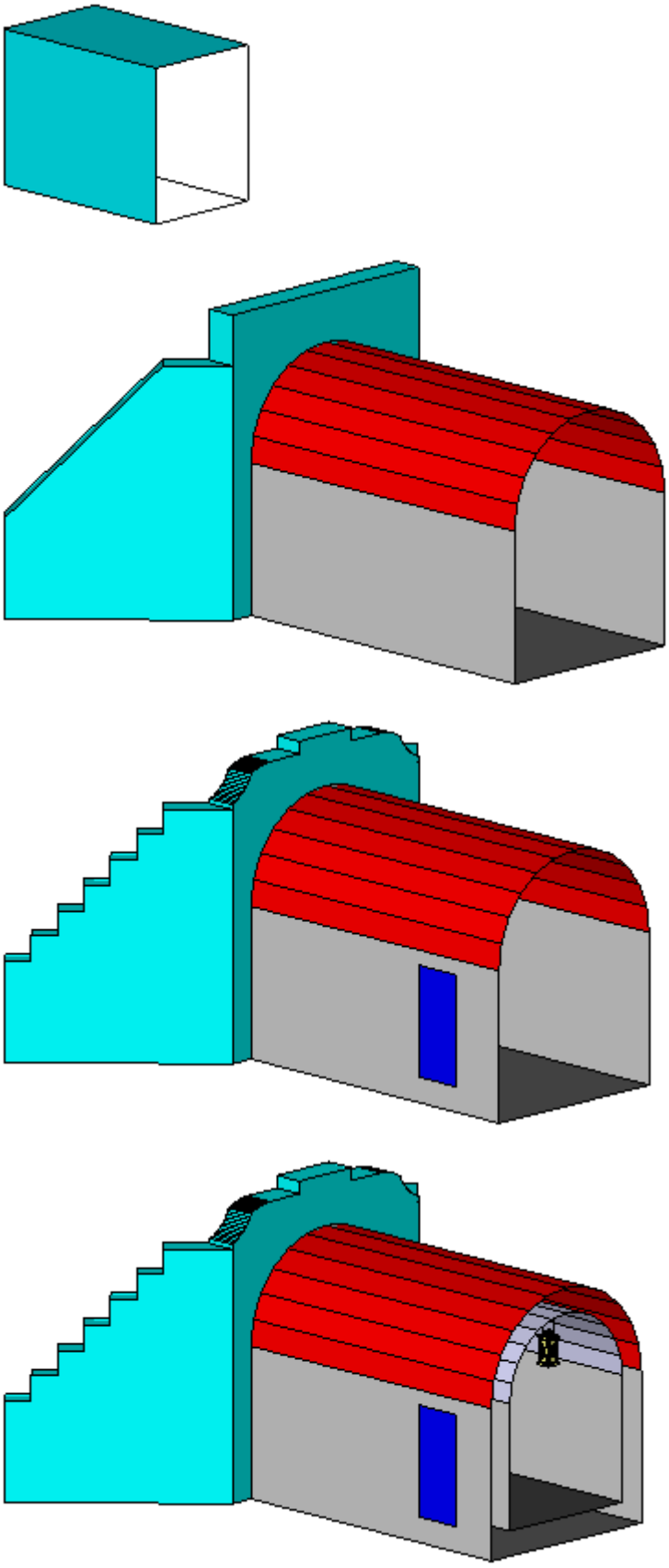


Figure 62. Tunnel model in LOD1 – LOD4 (source: Karlsruhe Institute of Technology (KIT)).

Tab. 6 shows the correspondence of the different geometric and semantic themes of the tunnel model to LODs. In each LOD, the volume of a tunnel can be expressed by a gml:Solid geometry and/or a gml:MultiSurface geometry. The definition of a 3D Terrain Intersection Curve (TIC), used to integrate tunnels from different sources with the Digital Terrain Model, is also possible in all LODs. The TIC can – but does not have to – build closed rings around the tunnel or tunnel parts.

Table 4. Semantic themes of the class _AbstractTunnel

Geometric / semantic theme	Property type	LOD1	LOD2	LOD3	LOD4
Building footprint and roof edge	gml:MultiSurfaceType	•			
	Volume part of the tunnel shell	gml:SolidType	•	•	•
•	Surface part of the tunnel shell	gml:MultiSurfaceType	•	•	•
•	Terrain intersection curve	gml:MultiCurveType	•	•	•
•	Curve part of the tunnel shell	gml:MultiCurveType		•	•
•	Tunnel parts	TunnelPartType	•	•	•
•	Boundary surfaces (chapter 10.4.3)	AbstractBoundarySurfaceType		•	•
•	Outer tunnel installations (chapter 10.4.2)	TunnelInstallationType		•	•
•	Openings	AbstractOpeningType			•
•	Hollow spaces (chapter 10.4.5)	HollowSpaceType			
•	Interior tunnel installations	IntTunnelInstallationType			

6.15.2. Outer Tunnel Installations

TunnelInstallationType, TunnelInstallation

A TunnelInstallation is an outer component of a tunnel which has not the significance of a TunnelPart, but which strongly affects the outer characteristic of the tunnel, for examples stairs. A TunnelInstallation optionally has attributes class, function and usage. The attribute class - which can only occur once - represents a general classification of the installation. With the attributes function and usage, nominal and real functions of a tunnel installation can be described. For all

three attributes the list of feasible values can specified in a code list. For the geometrical representation of a TunnelInstallation, an arbitrary geometry object from the GML subset shown in Fig. 9 can be used. Alternatively, the geometry may be given as ImplicitGeometry object. Following the concept of ImplicitGeometry the geometry of a prototype tunnel installation is stored only once in a local coordinate system and referenced by other tunnel installation features (see chapter 8.2). The visible surfaces of a tunnel installation can be semantically classified using the concept of boundary surfaces (cf. 10.3.3). A TunnelInstalla-tion object should be uniquely related to exactly one tunnel or tunnel part object.

6.15.3. Boundary surfaces

NOTE	insert Boundary surfaces UML

_BoundarySurface is the abstract base class for several thematic classes, structuring the exterior shell of a tunnel as well as the visible surface of hollow spaces and both outer and interior tunnel installations. It is a subclass of _CityObject and thus inherits all properties like the GML3 standard feature properties (gml:name etc.) and the CityGML specific properties like ExternalReferences. From _BoundarySurface, the thematic classes RoofSur-face, WallSurface, GroundSurface, OuterCeilingSurface, OuterFloorSurface, ClosureSurface, FloorSurface, InteriorWallSurface, and CeilingSurface are derived. The thematic classification of tunnel surfaces is illustrated in Fig. 41 for different types of tunnel cross sections and are specified below.

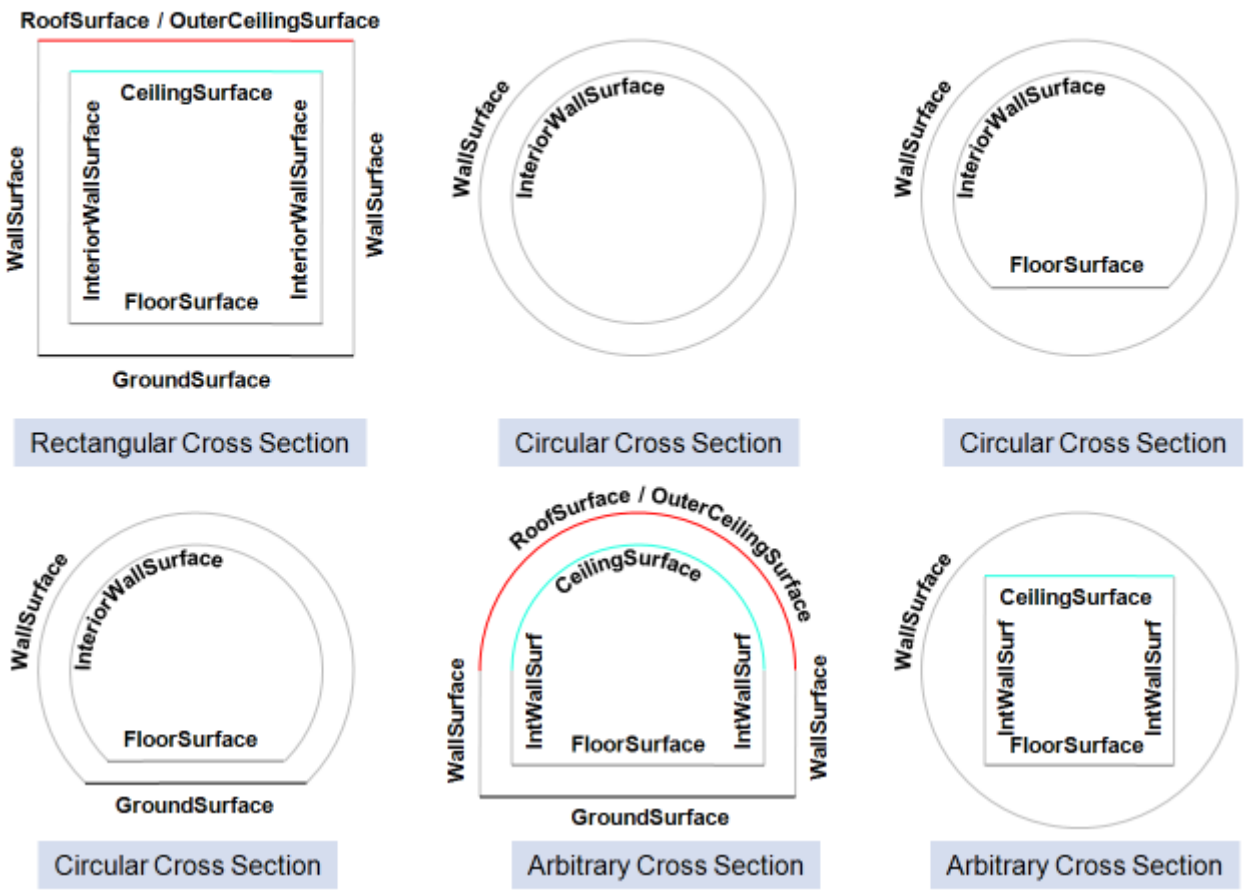


Figure 63. Examples for the use of boundary surfaces for tunnels with different cross sections. WallSurface, RoofSurface, GroundSurface, OuterCeilingSurface and OuterFloorSurface are available in LOD2– 4, whereas InteriorWallSurface, FloorSurface and CeilingSurface may only be used in LOD4 to model the interior boundary surfaces of a hollow space.

For each LOD between 2 and 4, the geometry of a _BoundarySurface may be defined by a different gml:MultiSurface geometry. Starting from LOD3, a _BoundarySurface may contain _Openings (cf. chapter 10.4.4) like doors and windows. If the geometric location of openings topologically lies within a surface compo-nent (e.g. gml:Polygon) of the gml:MultiSurface geometry, these openings must be represented as holes within that surface. A hole is represented by an interior ring within the corresponding surface geometry object. Accord-ing to GML3, the points have to be specified in reverse order (exterior boundaries counter-clockwise and interior boundaries clockwise when looking in opposite direction of the surface's normal vector). If such an opening is sealed by a Door or a Window, their outer boundary may consist of the same points as the inner ring (denoting the hole) of the surrounding surface. The embrasure surfaces of an opening belong to the relevant adjacent _BoundarySurface. If, for example a door seals the opening, the embrasure surface on the one side of the door belongs to the InteriorWallSurface and on the other side to the WallSurface (cf. right part of Fig. 32 for the same situation in a building model).

GroundSurfaceType, GroundSurface

NOTE	insert GroundSurfaceType, GroundSurface UML

The ground plate of a tunnel or tunnel part is modelled by the class GroundSurface. Usually a GroundSurface is a boundary surface between the tunnel and the surrounding earth (soil, rock etc.) or water.

OuterCeilingSurfaceType, OuterCeilingSurface

NOTE	insert OuterCeilingSurfaceType, OuterCeilingSurface UML

A mostly horizontal surface belonging to the outer tunnel shell and with the orientation pointing downwards can be modeled as an OuterCeilingSurface. Examples are the visible part of an avalanche protector or the boundary surface between the tunnel and the surrounding earth or water.

WallSurfaceType, WallSurface

NOTE	insert WallSurfaceType, WallSurface UML

All parts of the tunnel facade belonging to the outer tunnel shell can be modelled by the class WallSurface. Usually a WallSurface is a boundary surface between the tunnel and the surrounding earth (soil, rock etc.) or water.

OuterFloorSurfaceType, OuterFloorSurface

NOTE	insert OuterFloorSurfaceType, OuterFloorSurface UML

A mostly horizontal surface belonging to the outer tunnel shell and with the orientation pointing upwards can be modeled as an OuterFloorSurface.

RoofSurfaceType, RoofSurface

NOTE	insert RoofSurfaceType, RoofSurface UML

Boundary surfaces belonging to the outer tunnel shell and with the main purpose to protect the tunnel from above are expressed by the class RoofSurface. The orientation of these boundaries is mainly pointing upwards.

ClosureSurfaceType, ClosureSurface

NOTE	insert ClosureSurfaceType, ClosureSurface UML

Openings in tunnels or hollow spaces not filled by a door or a window can be sealed by a virtual surface called ClosureSurface (cf. chapter 6.4). For example, the doorways of tunnels can be modelled as ClosureSurface.

FloorSurfaceType, FloorSurface

NOTE	insert FloorSurfaceType, FloorSurface UML

The class FloorSurface must only be used in the LOD4 interior tunnel model for modelling the floor of hollow spaces.

InteriorWallSurfaceType, InteriorWallSurface

NOTE	insert InteriorWallSurfaceType, InteriorWallSurface UML

The class InteriorWallSurface is only allowed to be used in the LOD4 interior tunnel model for modelling the visible wall surfaces of hollow spaces.

CeilingSurfaceType, CeilingSurface

NOTE	insert CeilingSurfaceType, CeilingSurface UML

The class CeilingSurface is only allowed to be used in the LOD4 interior tunnel model for modelling the ceiling of hollow spaces.

6.15.4. Openings

AbstractOpeningType, _Opening

NOTE	insert AbstractOpeningType, _Opening UML

The class _Opening is the abstract base class for semantically describing openings like doors or windows in outer and inner boundary surfaces. Openings only exist in models of LOD3 or LOD4. Each _Opening is associ-ated with a gml:MultiSurface geometry. Alternatively, the geometry may be given as ImplicitGeometry object. Following the concept of ImplicitGeometry the geometry of a prototype opening is stored only once in a local coordinate system and referenced by other opening features (see chapter 8.2).

WindowType, Window

NOTE	insert WindowType, Window UML

The class Window is used for modelling windows in the in the exterior shell of a tunnel and in hollow spaces, or hatches between adjacent hollow spaces. The formal difference between the classes Window and Door is that – in normal cases – Windows are not specifically intended for the transit of people or vehicles.

DoorType, Door

NOTE	insert DoorType, Door UML

The class Door is used for modelling doors in the exterior shell of a tunnel, or between adjacent hollow spaces. Doors can be used by people to enter or leave a tunnel or ahollow space. In contrast to a ClosureSurface a door may be closed, blocking the transit of people or vehicles.

6.15.5. Tunnel Interior

HollowSpaceType, HollowSpace

NOTE	insert HollowSpaceType, HollowSpace UML

A HollowSpace is a semantic object for modelling the free space inside a tunnel and should be uniquely related to exactly one tunnel or tunnel part object. It should be closed (if necessary by using ClosureSurface) and the geometry normally will be described by a solid (lod4Solid). However, if the topological correctness of the boundary cannot be guaranteed, the geometry can alternatively be given as a MultiSurface (lod4MultiSurface). The surface normals of the outer shell of a GML solid must point outwards. This is important if appearences should be assigned to HollowSpace surfaces. In this case, textures and colors must be placed on the backside of the corresponding surfaces in order to be visible from the inside of the hollow space.

In addition to the geometrical representation, different parts of the visible surface of a hollow space can be modelled by specialised boundary surfaces (FloorSurface, CeilingSurface, InteriorWallSurface, and ClosureSur-face, cf. chapter 10.4.3).

TunnelFurnitureType, TunnelFurniture

NOTE	insert TunnelFurnitureType, TunnelFurniture UML

Hollow spaces may have TunnelFurniture. A TunnelFurniture is a movable part of a hollow space. A Tunnel-Furniture object should be uniquely related to exactly one hollow space. Its geometry may be represented by an explicit geometry or an ImplicitGeometry object. Following the concept of ImplicitGeometry the geometry of a prototype tunnel furniture is stored only once in a local coordinate system and referenced by other tunnel furni-ture features (see chapter 8.2).

IntTunnelInstallationType, IntTunnelInstallation

NOTE	insert IntTunnelInstallationType, IntTunnelInstallation UML

An IntTunnelInstallation is an object inside a tunnel with a specialized function or semantic meaning. In contrast to TunnelFurniture, objects of the class IntTunnelInstallation are permanently attached to the tunnel structure and cannot be moved. Typical examples are interior stairs, railings, radiators or pipes. Objects of the class IntTunnelInstallation can either be associated with a hollow space (class HollowSpace), or with the complete tunnel or tunnel part (class _AbstractTunnel, see chapter 10.4.1). However, they should be uniquely related to exactly one hollow space or one tunnel / tunnel part object. An IntTunnelInstallation optionally has the attributes class, function and usage. The attribute class, which can only occur once, represents a general classification of the internal tunnel component. With the attributes function and usage, nominal and real functions of a tunnel installation can be described. For all three attributes the list of feasible values can be specified in a code list. For the geometrical representation of an IntTunnelInstallation, an arbitrary geometry object from the GML subset shown in Fig. 9 can be used. Alternatively, the geometry may be given as ImplicitGeometry object. Following the concept of ImplicitGeometry the geometry of a prototype interior tunnel installation is stored only once in a local coordinate system and referenced by other interior tunnel installation features (see chapter 8.2). The visible surfaces of an interior tunnel

installation can be semantically classified using the concept of boundary surfaces (cf. 10.4.3).

6.15.6. Examples

The example in Fig. 42 shows a pedestrian underpass in the city centre of Karlsruhe, Germany. On the left side of Fig. 42, a photo illustrates the real world situation. Both entrances of the underpass are marked in the photo by dashed rectangles. On the right side of the figure, the CityGML tunnel model is shown. The terrain surrounding the tunnel has been virtually cut out of model in order to visualize the entire tunnel with its subsurface body. The same underpass is illustrated in Fig. 43 from a different perspective. The camera is positioned in front of the left entrance (black dashed rectangle in Fig. 42) and pointing in the direction of the right entrance (white dashed rectangle in Fig. 42). On the right side of Fig. 43, the tunnel model is shown from the same perspective. Again holes are cut in the terrain surface in order to make the subsurface part of the tunnel visible. An LOD1 representation of the nearby buildings is shown in the background of the model.

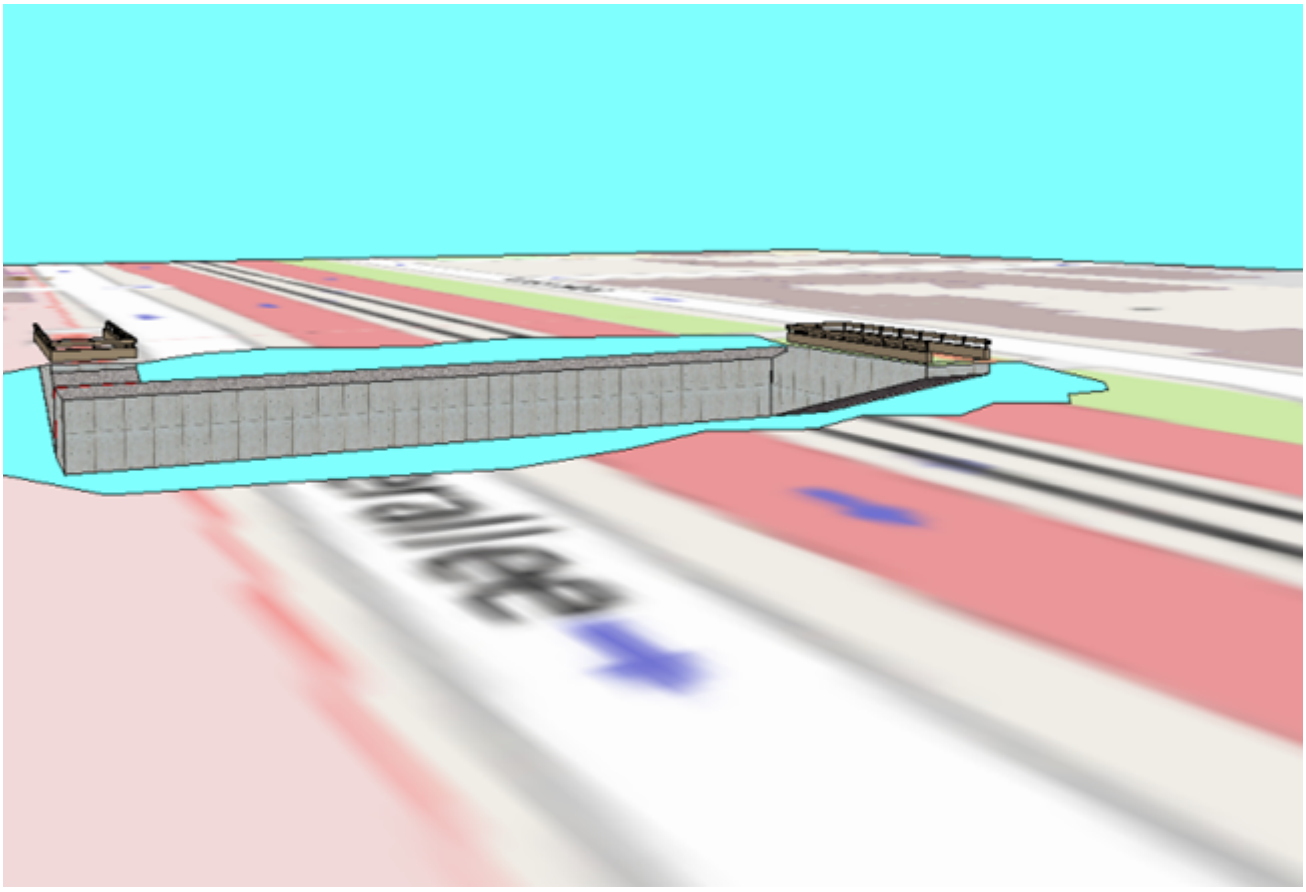


Figure 64. Example of a tunnel modeled in LOD3 (real situation on the left side; CityGML model on the right side) (source: Karlsruhe Institute of Technology (KIT), courtesy of City of Karlsruhe).

NOTE	insert Fig 43

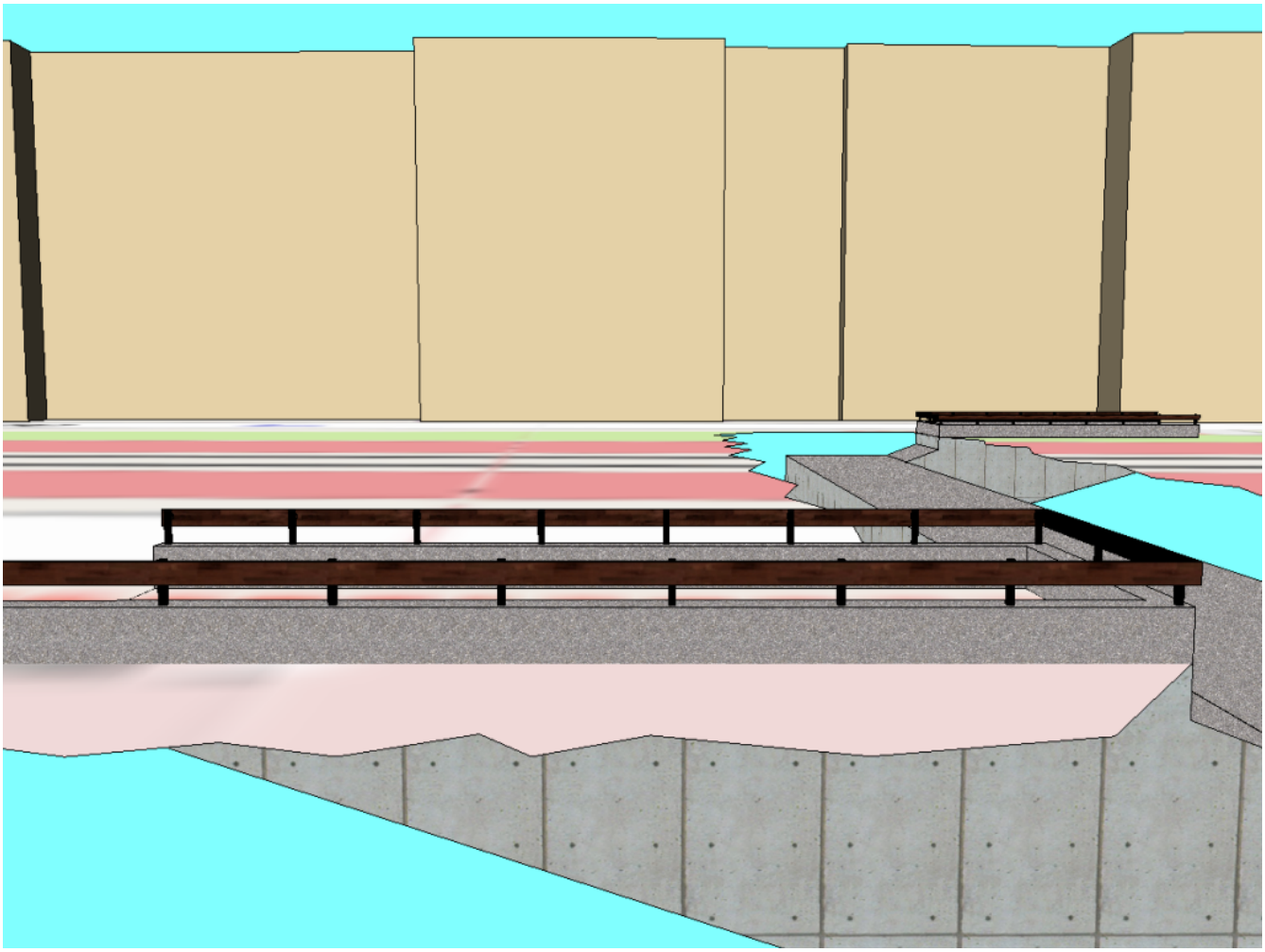


Figure 65. The same LOD3 tunnel shown from a different perspective. The camera is positioned in front of the left entrance and pointing in the direction of the right entrance. (real situation on the left side; CityGML model on the right side). The model on the right also includes an LOD1 representation of the nearby buildings in the background (painted in light brown) (source: Karlsruhe Institute of Technology (KIT), courtesy of City of Karlsruhe).

The model is subdivided into one Tunnel (the actual underpass) and two TunnelParts (both entrances). The tunnel and tunnel parts are bounded by GroundSurface, WallSurface, RoofSurface. ClosureSurface objects are used to virtually seal the tunnel entrances. For safety reasons each of the two entrances has railings which are modeled as TunnelInstallation. Due to the high geometrical accuracy and the semantic richness, the model is classified as LOD3.

6.16. Vegetation

Contributors
C. Heazel - first draft

6.16.1. Synopsis

The Vegetation module defines the concepts to represent vegetation within city models. Vegetation can be represented either as solitary vegetation objects, such as trees, bushes and ferns, or as vegetation areas that are covered by plants of a given species or a typical mixture of plant species, such as forests, steppes and wet meadows.

6.16.2. Key Concepts

Solitary Vegetation Object: A SolitaryVegetationObject represents individual vegetation objects, e.g. trees or bushes.

Plant Cover: A PlantCover represents a space covered by vegetation.

6.16.3. Discussion

Vegetation features are important components of a 3D city model, since they support the recognition of the surrounding environment. By the analysis and visualisation of vegetation objects, statements on their distribution, structure and diversification can be made. Habitats can be analysed and impacts on the fauna can be derived. The vegetation model may be used as a basis for simulations of, for example forest fire, urban aeration or micro climate. The model could be used, for example to examine forest damage, to detect obstacles (e.g. con-cerning air traffic) or to perform analysis tasks in the field of environmental protection.

The vegetation model of CityGML distinguishes between solitary vegetation objects like trees and vegetation areas, which represent biotopes like forests or other plant communities (Example for vegetation objects of the classes SolitaryVegetationObject and PlantCover (graphic: District of Recklinghausen).). Single vegetation objects are modelled by the class `SolitaryVegetationObject`, whereas for areas filled with a specific vegetation the class `PlantCover` is used. The geometry representation of a PlantCover feature may be a MultiSurface or a MultiSolid, depending on the vertical extent of the vegetation. For example regarding forests, a MultiSolid representation might be more appropriate.

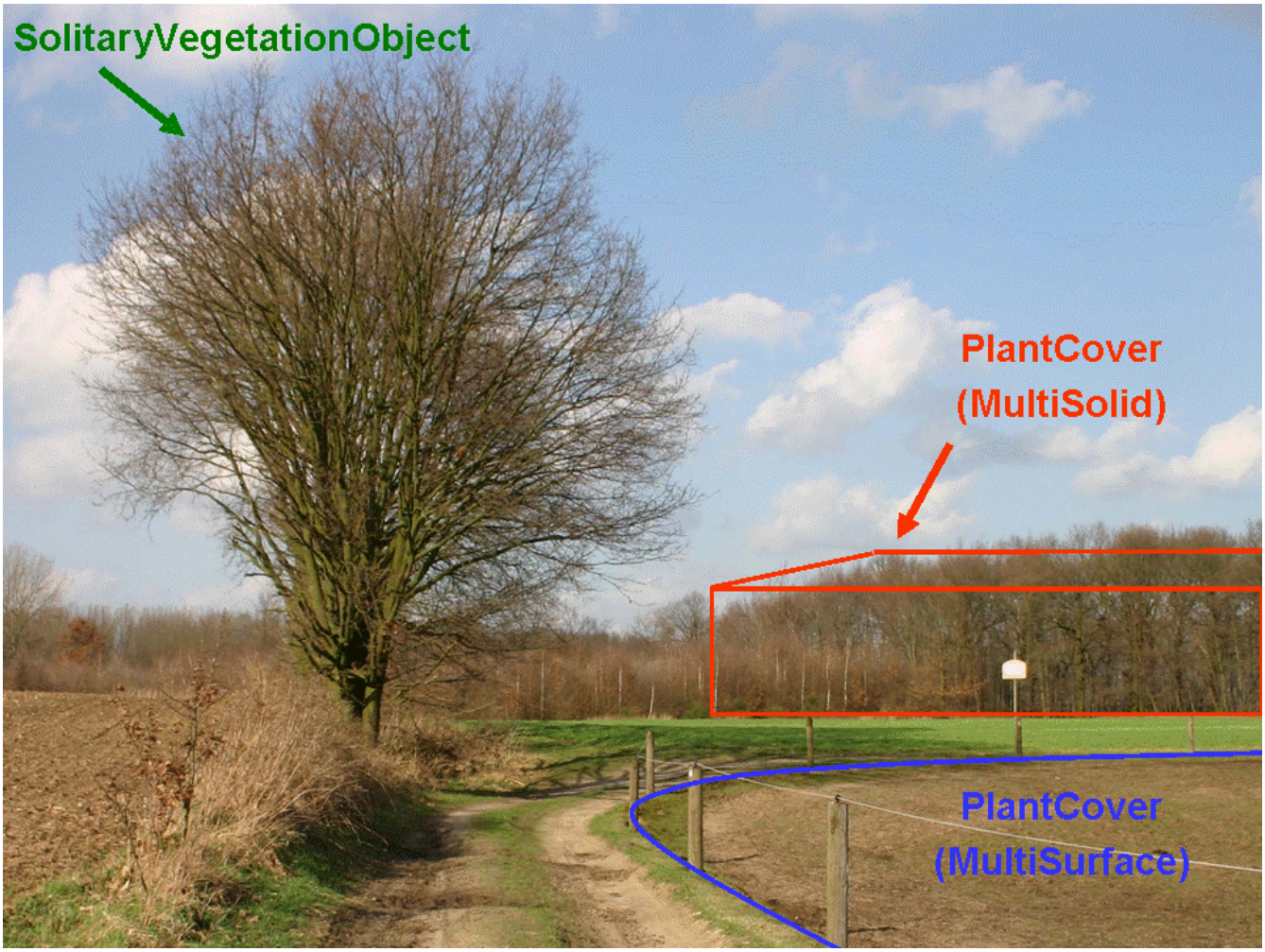
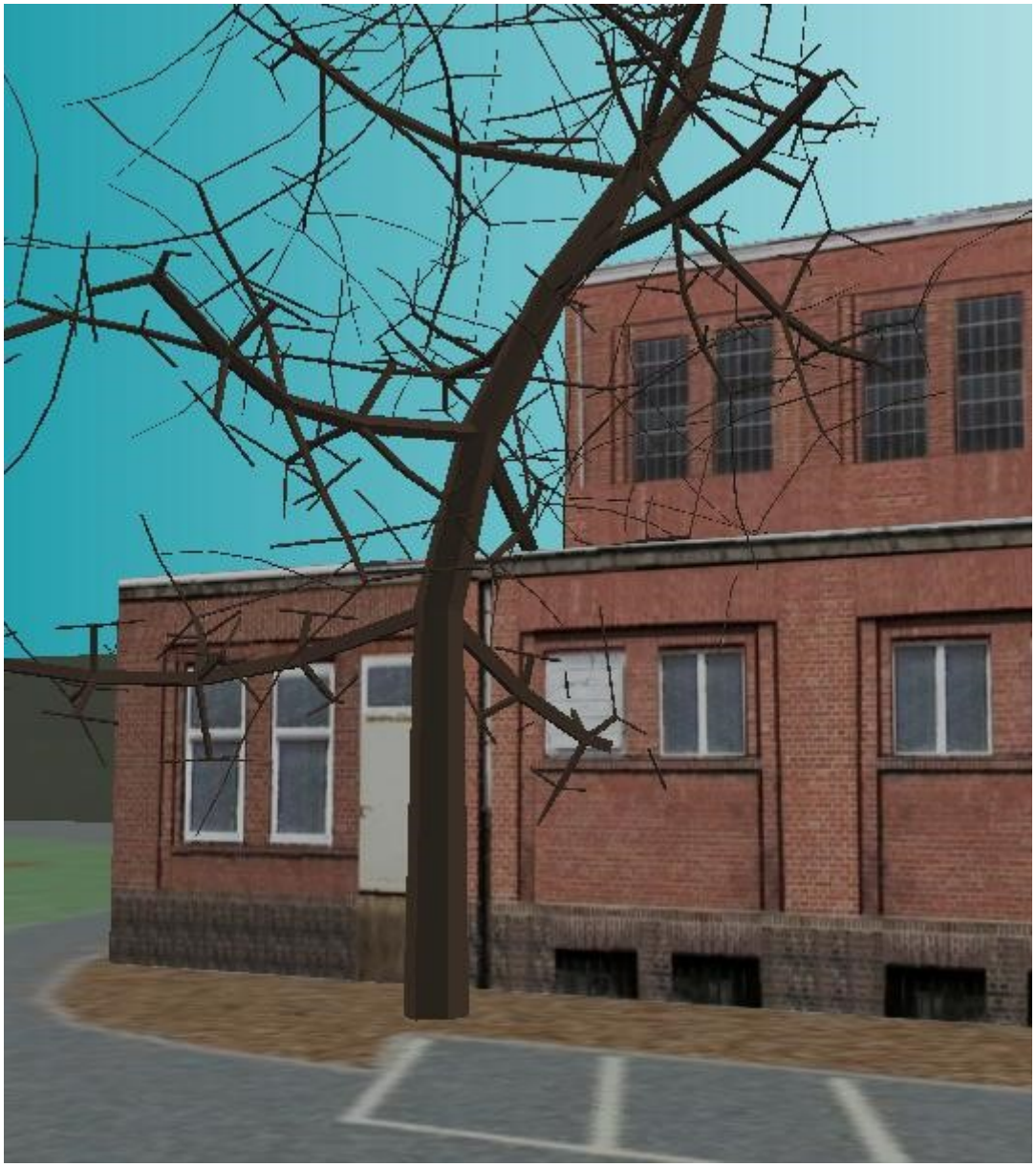


Figure 66. Example for vegetation objects of the classes SolitaryVegetationObject and PlantCover (graphic: District of Recklinghausen).

6.16.4. Level of Detail

The geometry of a SolitaryVegetationObject may be defined in LOD 1-4 explicitly by a GML geometry having absolute coordinates, or prototypically by an ImplicitGeometry. Solitary vegetation objects probably are one of the most important features where implicit geometries are appropriate, since the shape of most types of vegetation objects, such as trees of the same species, can be treated as identical in most cases. Furthermore, season dependent appearances may by mapped using ImplicitGeometry. For visualisation purposes, only the content of the library object defining the object's shape and appearance has to be swapped (see [figure-65]).



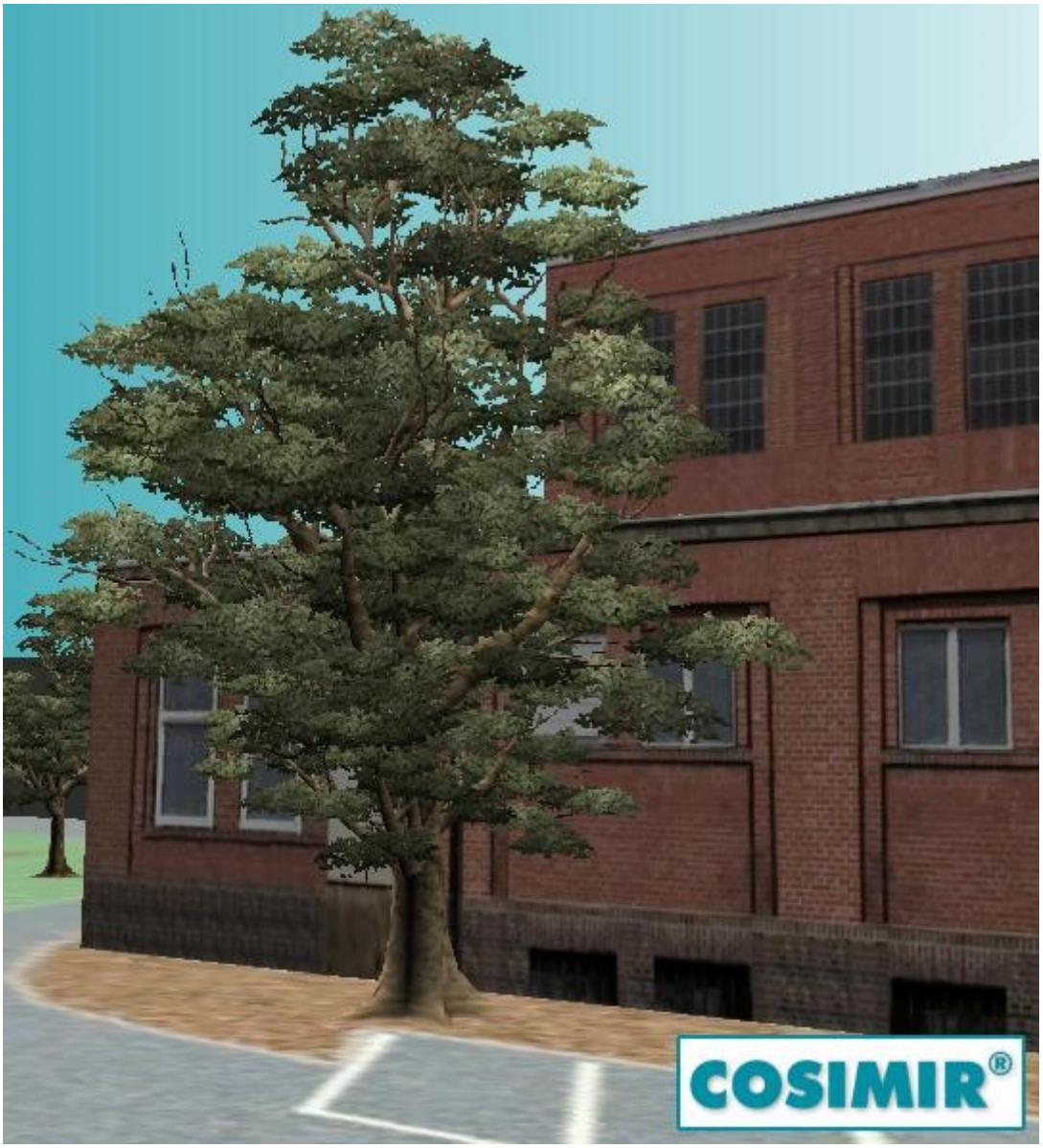


Figure 67. Visualisation of a vegetation object in different seasons (source: District of Recklinghausen).

A SolitaryVegetationObject or a PlantCover may have a different geometry in each LOD. Whereas a SolitaryVegetationObject is associated with the `Geometry` class representing an arbitrary geometry, a PlantCover is restricted to be either a `MultiSolid` or a `MultiSurface`. An example of a PlantCover modeled as MultiSolid is a 'solid forest model' (see Example for the visualisation/modelling of a solid forest (source: District of Recklinghausen).).

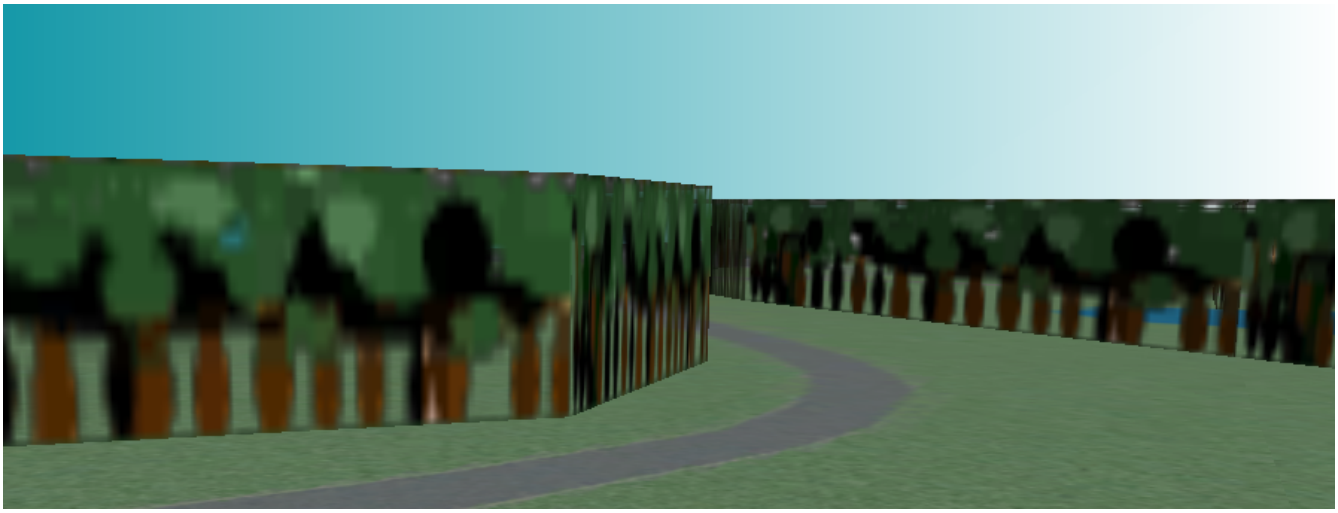


Figure 68. Example for the visualisation/modelling of a solid forest (source: District of Recklinghausen).

6.16.5. UML Model

The UML diagram of the Vegetation module is depicted in UML diagram of the Vegetation Model..

A SolitaryVegetationObject may have the attributes `class`, `function`, `usage`, `species`, `height`, `trunkDiameter`, `crownDiameter`, `rootBallDiameter`, and `maxRootBallDepth`. The attribute `class` contains the classification of the object or plant habit, e.g. tree, bush, grass, and can occur only once. The attribute `species` defines the species' name, for example "Abies alba", and can occur at most once. The optional attributes `function` and `usage` denotes the intended respectively real purpose of the object, for example botanical monument, and can occur multiple times. The possible attribute values for `class`, `species`, `function`, and `usage` can be provided in a code list. The attribute `height` contains the relative height of the object. The attributes `crownDiameter` and `trunkDiameter` represent the plant crown and trunk diameter respectively. The trunk diameter is often used in regulations of municipal cadastre (e.g. tree management rules). The attributes `rootBallDiameter` and `maxRootBallDepth` represent the diameter of the root ball and its' maximum depth respectively.

A PlantCover feature may have the attributes `class`, `function`, `usage`, `averageHeight`, `minHeight`, and `maxHeight`. The plant community of a PlantCover is represented by the attribute `class`. The values of this attribute can be specified in a code list whose values should not only describe one plant type or species, but denote a typical mixture of plant types in a plant community. This information can be used in particular to generate realistic 3D visualisations, where the PlantCover region is automatically, perhaps randomly, filled with a corresponding mixture of 3D plant objects. The attributes `function` and `usage` indicate the intended respectively real purpose of the object, for example national forest, and can occur multiple times. The attributes `averageHeight`, `minHeight`, and `maxHeight` denotes the minimum, maximum, and average relative vegetation heights.

Since SolitaryVegetationObject and PlantCover are subclasses of AbstractOccupiedSpace, they are Features. As such they inherit the attribute `name` from the AbstractFeature class and an ExternalReference to a corresponding object. That external object may be in an external information system which may contains botanical information from public environmental agencies.

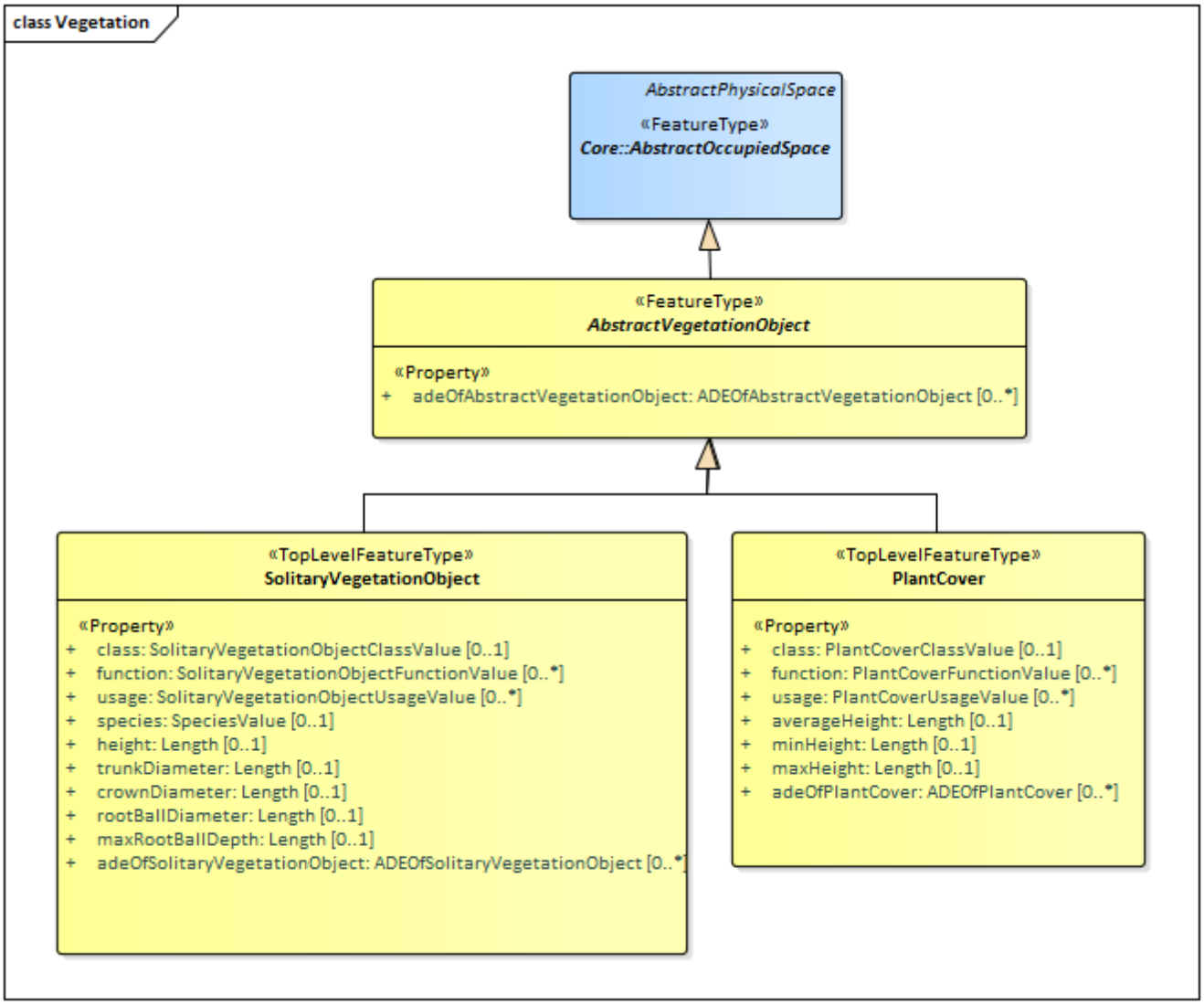


Figure 69. UML diagram of the Vegetation Model.

The ADE data types provided for the Vegetation module are illustrated in ADE classes of the CityGML Vegetation module..

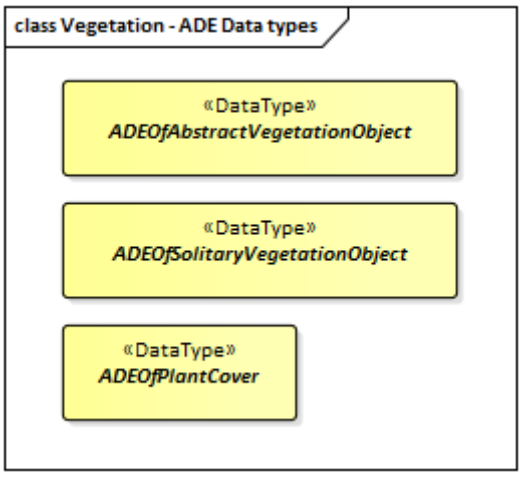


Figure 70. ADE classes of the CityGML Vegetation module.

The Code Lists provided for the Vegetation module are illustrated in Codelists from the CityGML Vegetation module..

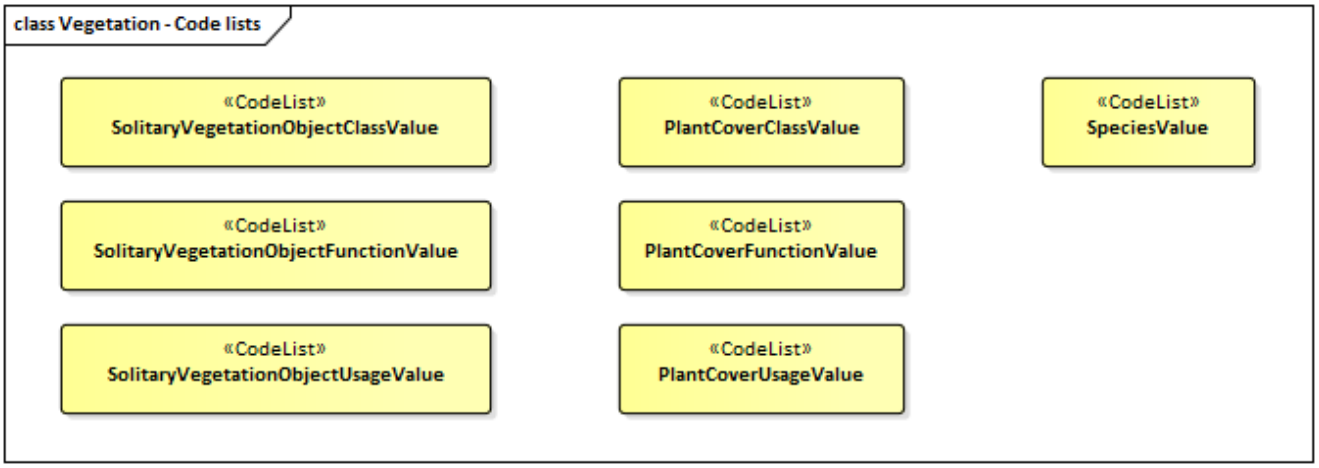


Figure 71. Codelists from the CityGML Vegetation module.

6.16.6. Examples

The following two excerpts of a CityGML dataset contain a solitary tree (SolitaryVegetationObject) and a plant community (PlantCover). The solitary tree has the attributes: class = 1070 (deciduous tree), species = 1040 (Fagus/beech), height = 8 m, trunkDiameter = 0.7 m, crownDiameter = 8.0 m. The plant community has the attributes: class =1180 (isoeto-nanojuncetea), averageHeight = 0.5 m.

NOTE	include examples, GML or other?

6.17. Versioning

Contributors
TBD

NOTE	text needed.

6.18. Waterbodies

Contributors
C. Heazel - first draft

6.18.1. Synopsis

The WaterBody module provides the representation of significant and permanent or semi-permanent accumulations of surface water, usually covering a part of the Earth. Examples of such water bodies that can be modelled with CityGML are rivers, canals, lakes, and basins.

6.18.2. Key Concepts

Waterbody: A WaterBody represents significant and permanent or semi-permanent accumulations of surface water, usually covering a part of the Earth.

Water Surface: A WaterSurface represents the upper exterior interface between a water body and the atmosphere. Water Surface is a boundary property of a Water Body.

Water Ground Surface: A WaterGroundSurface represents the exterior boundary surface of the submerged bottom of a water body. Water Ground Surface is a boundary property of a Water Body.

6.18.3. Discussion

NOTE	The following text needs to be reviewed and updated.

Waters have always played an important role in urbanisation processes and cities were built preferably at rivers and places where landfall seemed to be easy. Obviously, water is essential for human alimentation and sanitation. Water bodies present the most economical way of transportation and are barriers at the same time, that avoid instant access to other locations. Bridging waterways caused the first efforts of construction and resulted in high-tech bridges of today. The landscapes of many cities are dominated by water, which directly relates to 3D city models. Furthermore, water bodies are important for urban life as subject of recreation and possible hazards as e.g. floods.

The distinct character of water bodies compared with the permanence of buildings, roadways, and terrain is considered in this thematic model. Water bodies are dynamic surfaces. Tides occur regularly, but irregular events predominate with respect to natural forces, for example flood events. The visible water surface changes in height and its covered area with the necessity to model its semantics and geometry distinct from adjacent objects like terrain or buildings.

This first modelling approach of water bodies fulfils the requirements of 3D city models. It does not inherit any hydrological or other dynamic aspects. In these terms it does not claim to be complete. However, the semantic and geometric description given here allows further enhancements of dynamics and conceptually different descriptions. The water bodies model of CityGML is embraced by the extension module WaterBody.

The water bodies model represents the thematic aspects and three-dimensional geometry of rivers, canals, lakes, and basins. In the LOD 2-4 water bodies are bounded by distinct thematic surfaces (see Illustration of a water body defined in CityGML (graphic: IGG Uni Bonn).). These surfaces are the obligatory WaterSurface, defined as the boundary between water and air, and the optional WaterGroundSurface, defined as the boundary between water and underground (e.g. DTM or floor of a 3D basin object).

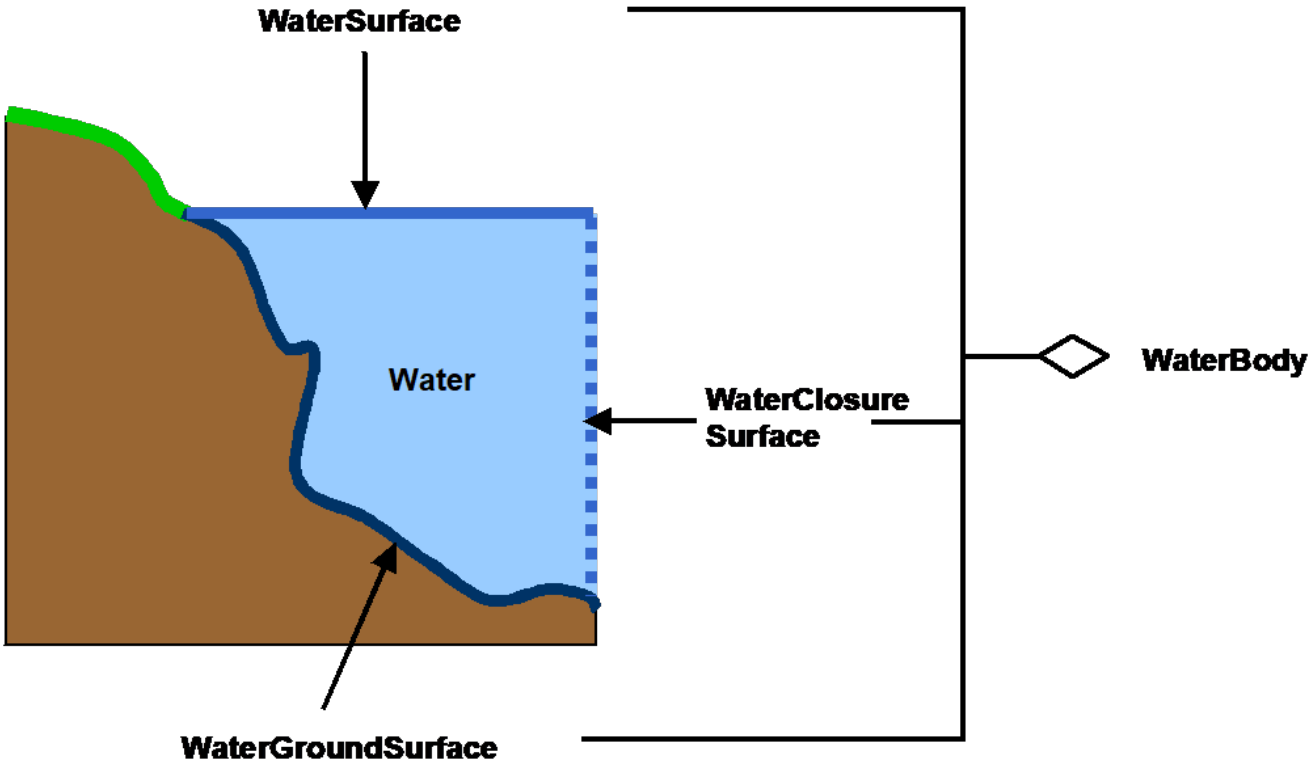


Figure 72. Illustration of a water body defined in CityGML (graphic: IGG Uni Bonn).

1. Boundary class “Air to Water”. The WaterSurface is mandatory to the model and usually is registered using photogrammetric analysis or mapping exploration. The representation may vary due to tidal flats or changing water levels, which can be reflected by including different static water surfaces having different waterLevels, as for example highest flooding event, mean sea level, or minimum water level. This offers the opportunity to describe significant water surfaces due to levels that are important for certain representations e.g. in tidal zones.
2. Boundary class “Water to Ground”. The WaterGroundSurface may be known by sonar exploration or other depth measurements. Also part of the ground surface is the boundary “Water to Construction”. The ground surface might be identical to the underwater terrain model, but also describes the contour to other underwater objects. The usefulness of this concept arises from the existence of water defence constructions like sluices, sills, flood barrage or tidal power stations. The use of WaterGroundSurface as boundary layer to man-made constructions is relevant in urban situations, where such objects may enclose the modeled water body completely, for example fountains and swimming pools. The WaterSurface objects together with the WaterGroundSurface objects enclose the WaterBody as a volume.

6.18.4. Level of Detail

Both LOD0 and LOD1 represent a low level of illustration and high grade of generalisation. Here the rivers are modelled as MultiCurve geometry and brooks are omitted. Seas, oceans and lakes with significant extent are represented as a MultiSurface (Fig. 56). Every WaterBody may be assigned a combination of geometries of different types. Linear water bodies are represented as a network of 3D curves. Each curve is composed of straight line segments, where the line orientation denotes the flow direction (water flows from the first point of a curve, e.g. a gml:LineString, to the last). Areal objects like lakes or seas are represented by 3D surface geome-tries of the water surface.

Starting from LOD1 water bodies may also be modelled as water filled volumes represented by

Solids. If a water body is represented by a gml:Solid in LOD2 or higher, the surface geometries of the corresponding thematic WaterGroundSurface, and WaterSurface objects must coincide with the exterior shell of the gml:Solid. This can be ensured, if for each LOD X the respective lodXSolid representation (where X is between 2 and 4) does not redundantly define the geometry, but instead references the corresponding polygons of the lodXSurface elements (where X is between 2 and 4) of WaterGroundSurface, and WaterSurface.

LOD2 to LOD4 demand a higher grade of detail and therefore any WaterBody can be outlined by thematic surfaces or a solid composed of the surrounding thematic surfaces.

Every object of the class WaterSurface and WaterGroundSurface must have at least one associated surface geometry. This means, that every WaterSurface and WaterGroundSurface feature within a CityGML instance document must contain at least one of the following properties: lod2Surface, lod3Surface, lod4Surface.

The water body model implicitly includes the concept of TerrainIntersectionCurves (TIC), e.g. to specify the exact intersection of the DTM with the 3D geometry of a WaterBody or to adjust a WaterBody or WaterSurface to the surrounding DTM. The rings defining the WaterSurface polygons implicitly delineate the intersection of the water body with the terrain or basin.

6.18.5. UML Model

The UML diagram of the water body model is depicted in UML diagram of the Water Body Model.. Each WaterBody object may have the attributes class, function and usage whose possible values can be enumerated in code lists. The attribute `class` defines the classification of the object, e.g. lake, river, or fountain and can occur only once. The attribute `function` contains the purpose of the object like, for example national waterway or public swimming, while the attribute `usage` defines the actual usages, e.g. whether the water body is navigable. The latter two attributes can occur multiple times.

Since WaterBody is a subclass of AbstractOccupiedSpace and hence a feature, it inherits the attribute `name` from the AbstractFeature class. The WaterBody can be differentiated semantically by the class _WaterBoundarySurface. A _WaterBoundarySurface is a part of the water body's exterior shell with a special function like WaterSurface, WaterGroundSurface or WaterClosureSurface. As with any _CityObject, WaterBody objects as well as WaterSurface and WaterGroundSurface may be assigned ExternalReferences and may be augmented by generic attributes using CityGML's Generics module (see Generics).

The optional attribute `waterLevel` of a WaterSurface can be used to describe the water level, for which the given 3D surface geometry was acquired. This is especially important when the water body is influenced by the tide. The allowed values can be defined in a corresponding code list.

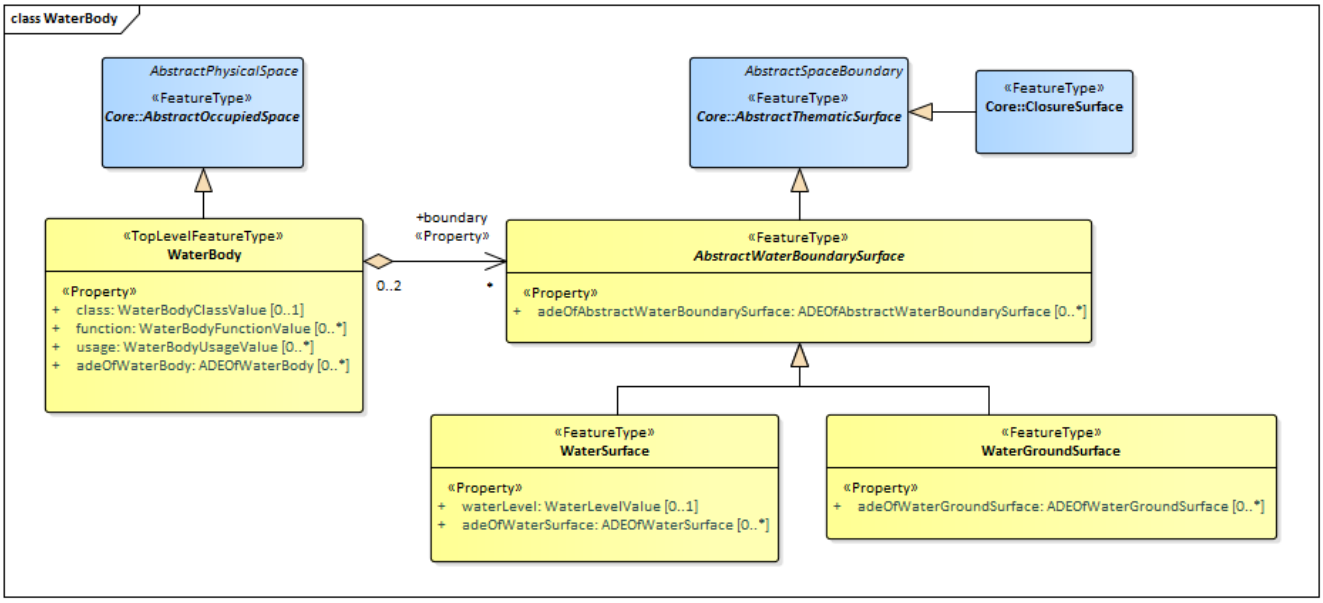


Figure 73. UML diagram of the Water Body Model.

The ADE data types provided for the Water Body module are illustrated in ADE classes of the CityGML Water Body module..

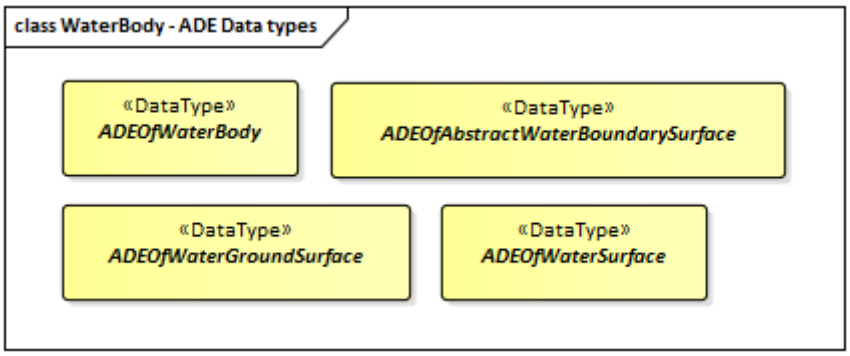


Figure 74. ADE classes of the CityGML Water Body module.

The Code Lists provided for the Water Body module are illustrated in Codelists from the CityGML Water Body module..

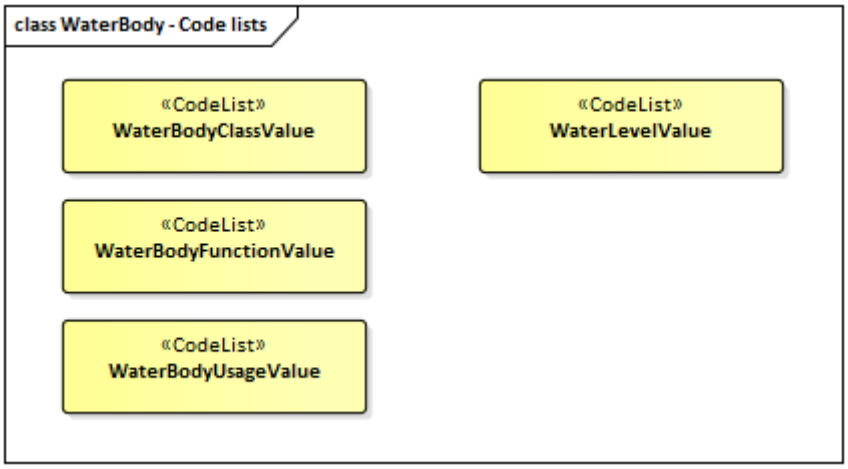


Figure 75. Codelists from the CityGML Water Body module.

6.18.6. Examples

6.19. Extensions

Contributors
TBD

CityGML has been designed as an application independent information model and exchange format for 3D city and landscape models. However, specific applications typically have additional information needs to be modeled and exchanged. In general, there are two different approaches to combine city model data and application data:

1. Embed the CityGML objects into a (larger) application framework and establish the connection between application data and CityGML data within the application framework. For example, CityGML data fragments may be embedded into the application's XML data files or stored as attributes of application objects according to the application's data model.
2. Incorporate application specific information into the CityGML instance documents. This approach is especially feasible if the application specific information follows essentially the same structure as defined by the CityGML schema. This is the case, if the application data could be represented by additional attributes of CityGML objects and only some new feature types would have to be defined.

In the following, we will focus on the second option, as only this approach lies within the scope of the CityGML 3.0 Standard. Generic attributes and objects have already been introduced as a first possibility to support the exchange of application specific data (see Generics). Whereas they allow to extend CityGML without changing its schema definitions, this flexibility has some disadvantages:

- Generic attributes and objects may occur arbitrarily in the CityGML instance documents, but there is no formal specification of the names, datatypes, and multiplicities. Thus, there is no guarantee for an application that a specific instance of a generic attribute is included a minimum or maximum number of times per CityGML feature. Unlike the predefined CityGML objects, the concrete layout and occurrence of generic objects and attributes cannot be validated by a parser. This may reduce semantic interoperability.
- Naming conflicts of generic attributes or objects can occur, if the CityGML instance documents should be augmented by specific information from different applications simultaneously.
- There is only a limited number of predefined data types that can be used for generic attributes. Also the structure of generic objects might not be appropriate to represent more complex objects.

If application specific information are well-structured, it is desirable to represent them in a systematic way, i. e. by the definition of an extra formal schema based on the CityGML schema definitions. Such an schema is called a CityGML Application Domain Extension (ADE). It allows to validate instance documents both against the extended CityGML and the ADE schema and therefore helps to maintain semantic and syntactic interoperability between different systems working in the same application field. In order to prevent naming conflicts, every ADE has to be defined within its own namespace which must differ from the namespaces associated with the CityGML modules. An ADE schema may extend one or more CityGML module schemas. The relevant CityGML module

schemas have to be imported by the ADE schema.

The ADE concept defines a special way of extending existing CityGML feature types which allows to use different ADEs within the same instance document simultaneously. For example, the specification of ADEs can be useful in the following application fields: cultural heritage (extension of abstract class _CityObject e.g. by time period information and monument protection status); representation of subsurface objects (tunnel, underpass) or city lighting (light sources like street lamps and house lights); real estate management (economic parameters of the CityGML features; inclusion of attributes defined for real estate assets as defined by OSCRE); utility networks (as topographic features); additional building properties as defined by the U.S. national building information model standard (NBIMS).

6.19.1. Technical principle of ADEs

NOTE	This section is no longer current but does look like it would be useful. Can we update it?

Each ADE is specified by its own XML schema file. The target namespace is provided by the information community who specifies the CityGML ADE. This is typically not the OGC or the SIG 3D. The namespace should be in the control of this information community and must be given as a previously unused and globally unique URI. This URI will be used in CityGML ADE instance documents to distinguish extensions from CityGML base elements. As the URI refers to the information community it also denotes the originator of the employed ADE.

The ADE's XML schema file must be available (or accessible on the Internet) to everybody creating and parsing CityGML instance documents including these ADE specific augmentations.

An ADE XML schema can define various extensions to CityGML. However, all extensions shall belong to one of the two following categories:

1. New feature types are defined within the ADE namespace and are based on CityGML abstract or con-crete classes. In general, this mechanism follows the same principles as the definition of application schemas for GML. This means, that new feature types have to be derived from existing (here: CityGML) feature types. For example, new feature types could be defined by classes derived from the abstract classes like _CityObject or _AbstractBuilding or the concrete class CityFurniture. The new fea-ture types then automatically inherit all properties (i.e. attributes) and associations (i.e. relations) from the respective CityGML superclasses.
2. Existing CityGML feature types are extended by application specific properties (in the ADE namespace). These properties may have simple or complex data types. Also geometries or embedded features (feature properties) are possible. The latter can also be used to model relations to other features.

   In this case, extension of the CityGML feature type is not being realised by the inheritance mechanism of XML schema. Instead, every CityGML feature type provides a "hook" in its XML schema definition, that allows to attach additional properties to it by ADEs. This "hook" is implemented as a GML proper-ty of the form "_GenericApplicationPropertyOf<Featuretypename>" where <Featuretypename> is equal to the name of the feature type definition in which it is included. The datatype for these kinds of

properties is always “xsd:anyType” from the XSD namespace. The minimum occurrence of the “_GenericApplicationPropertyOf<Featuretypename>” is 0 and the maximum occurrence unbounded. This means, that the CityGML schema allows that every CityGML feature may have an arbitrary num-ber of additional properties with arbitrary XML content with the name “_GenericApplication-PropertyOf<Featuretypename>”. For example, the last property in the definition of the CityGML fea-ture type LandUse is the element _GenericApplicationPropertyOfLandUse (cf. chapter 10.10.1).

Such properties are called “hooks” to attach application specific properties, because they are used as the head of a substitution group by ADEs. Whenever an ADE wants to add an extra property to an existing CityGML feature type, it should declare the respective element with the appropriate dataype within the ADE namespace. In the element declaration this element shall be explicitly assigned to the substitution group defined by the corresponding “_GenericApplicationPropertyOf<Featuretypename>” in the cor-responding CityGML module namespace. An example is given in the following subsection.

By following this concept, it is possible to specify different ADEs for different information communi-ties. Every ADE may add their specific properties to the same CityGML feature type as they all can be-long to the same substitution group. This allows to have CityGML instance documents where CityGML features contain additional information from different ADEs simultaneously.

Please note that usage of ADEs introduces an extra level of complexity as data files may contain mixed infor-mation (features, properties) from different namespaces, not only from the GML and CityGML module namespaces. However, extended CityGML instance documents are quite easy to handle by applications that are not “schema-aware”, i.e. applications that do not parse and interpret GML application schemas in a generic way. These applications can simply skip anything from a CityGML instance document that is not from a CityGML module or GML namespace. Thus, a building is still represented by the <bldg:Building> element with the standard CityGML properties, but with possibly some extra properties from different namespaces. Also features from a different namespace than those declared by CityGML modules or GML could be skipped (e.g. by a viewer application).

6.19.2. Example ADE

In this section, the ADE mechanism is illustrated by a short example, which deals with the application of virtual 3D city models to generate noise pollution maps. In our example, two extensions of CityGML are required for this task: buildings have to be extended to represent a “noise reflection correction” value and the number of inhabitants. As a new feature type noise barriers have to be defined which also have a “noise reflection correc-tion” value.

The XSD schema which has to be defined to implement this model declares a new namespace for the noise extension (http://www.citygml.org/ade/noise_de/2.0). Additionally, the namespaces of the extended CityGML modules are declared (for corresponding prefixes see chapter 4.3 and chapter 7), and the respective schema definition files are imported. The XML schema adds the elements buildingReflectionCorrection and buildingHabitants, both being members of the substitution group bldg:_GenericApplicationPropertyOf-AbstractBuilding which is defined by the CityGML Building module. Thus, both elements may be used as child elements of CityGML building features. Noise

barriers are represented as NoiseCityFurnitureSegment elements. The corresponding type NoiseCityFurnitureSegmentType is defined as subtype of the CityGML abstract type core:AbstractCityObjectType provided by the CityGML Core module, applying the usual subtyping mechanism of XML and XSD. A further element noiseCityFurnitureSegmentProperty is added as a member of the substitu-tion group frn:_GenericApplicationPropertyOfCityFurniture. By this means, noise barriers may be modelled as child elements of CityGML city furniture objects.

The XSD file for this example CityGML Noise ADE is given by the following excerpt (the complete CityGML Noise ADE is given in Annex H):

NOTE insert example here (GML?)

An example for a feature collection in a corresponding instance document is depicted below. Two CityGML buildings contain application specific properties distinguished from CityGML properties by the namespace prefix noise. The other properties, function and geometry, are defined by corresponding CityGML modules. In addition to the buildings, a noise barrier as child of a city furniture element is included in the feature collection. Please note, that the order of the child elements in the sequence is not arbitrary: the child elements defined by an ADE subschema have to occur after the child elements defined by CityGML modules. There is, however, no specific order of the ADE properties.

NOTE insert example here (GML?)

Annex A: Revision History

Date	Release	Editor	Primary clauses modified	Description
2016-04-28	0.1	G. Editor	all	initial version

Annex B: Bibliography

NOTE

Example Bibliography (Delete this note).

The TC has approved Springer LNCS as the official document citation type.

Springer LNCS is widely used in technical and computer science journals and other publications

- For citations in the text please use square brackets and consecutive numbers: [1], [2], [3]

– Actual References:

[n] Journal: Author Surname, A.: Title. Publication Title. Volume number, Issue number, Pages Used (Year Published)

[n] Web: Author Surname, A.: Title, http://Website-Url

[1] OGC: OGC Testbed 12 Annex B: Architecture. (2015).