

Open Geospatial Consortium

Submission Date: <yyyy-mm-dd>

Approval Date: <yyyy-mm-dd>

Publication Date: <2021-01-04>

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

Copyright notice

Copyright © 2020 Open Geospatial Consortium

To obtain additional rights of use, visit <http://www.opengeospatial.org/legal/>

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

1. Introduction	12
2. How To Use This Resource	13
3. Scope	14
4. References	15
5. Terms and Definitions	16
6. Conventions	19
6.1. Identifiers	19
6.2. UML Notation	19
7. CityGML Foundations	23
7.1. Modularization	23
7.2. General Modelling Principles	25
7.2.1. Semantic Modelling of Real-World Objects	25
7.2.2. Class Hierarchy and Inheritance of Properties and Relations	26
7.2.3. Relationships between CityGML objects	26
7.2.4. Definition of the Semantics for all Classes, Properties, and Relations	27
7.3. Representation of Spatial Properties	27
7.3.1. Geometry and Topology	27
7.3.2. Prototypic Objects / Scene Graph Concepts	29
7.3.3. Point Cloud Representation	29
7.3.4. Coordinate Reference Systems (CRS)	30
7.4. CityGML Core Model: Space Concept, Levels of Detail, Special Spatial Types	30
7.4.1. Spaces and Space Boundaries	30
7.4.2. Modelling City Objects by the Composition of Spaces	32
7.4.3. Rules for Surface Orientations of OccupiedSpaces and UnoccupiedSpaces	32
7.4.4. Levels of Detail (LOD)	33
7.4.5. Closure Surfaces	34
7.4.6. Terrain Intersection Curves	35
7.4.7. Coherent Semantical-Geometrical Modelling	36
7.5. Appearances	36
7.6. Modelling Dynamic Data	36
7.6.1. Versioning and Histories	37
7.6.2. Dynamizers: Using Time-Series Data for Object Attributes	37
7.7. Extending CityGML	39
8. CityGML Model	40
8.1. Core	40
8.1.1. Key Concepts	40
8.1.2. ISO Dependencies	42
8.1.3. City Models and City Objects	49

8.1.4. Space Concept	50
8.1.5. Geometry and LOD	53
8.1.6. CityGML Core Model	55
8.1.7. Data types, Enumerations, and Code lists	56
8.2. Appearance	58
8.2.1. Synopsis	58
8.2.2. Key Concepts	58
8.2.3. Discussion	59
8.2.4. Material	59
8.2.5. Texture and texture mapping	60
8.2.6. Related concepts	68
8.2.7. UML Model	68
8.2.8. Examples	70
8.3. Bridge Model	70
8.3.1. Synopsis	70
8.3.2. Key Concepts	70
8.3.3. Discussion	70
8.3.4. Bridge and bridge part	75
8.3.5. Bridge construction elements and bridge installations	75
8.3.6. Boundary surfaces	76
8.3.7. Openings	78
8.3.8. Bridge Interior	79
8.3.9. UML Model	80
8.3.10. Examples	81
8.4. Building Model	84
8.4.1. Synopsis	84
8.4.2. Key Concepts	84
8.4.3. Discussion	85
8.4.4. Building Part	89
8.4.5. Outer building installations	93
8.4.6. Boundary surfaces	93
8.4.7. Openings	97
8.4.8. Building Interior	98
8.4.9. Modelling building storeys using CityObjectGroups	100
8.4.10. UML Model	100
8.4.11. Examples	102
8.5. City Furniture	107
8.5.1. Synopsis	107
8.5.2. Key Concepts	107
8.5.3. Discussion	107
8.5.4. UML Model	111

8.5.5. Examples	112
8.6. City Object Group	113
8.6.1. Synopsis	113
8.6.2. Key Concepts	113
8.6.3. Discussion	114
8.6.4. UML Model	114
8.6.5. Examples	116
8.7. Construction	116
8.7.1. Synopsis	116
8.7.2. Key Concepts	116
8.7.3. Discussion	116
8.7.4. Construction Spaces	117
8.7.5. Construction Surfaces	117
8.7.6. Levels of Detail	117
8.7.7. Spaces and Surfaces	121
8.7.8. UML Model	125
8.7.9. Examples	126
8.8. Dynamizer	126
8.8.1. Synopsis	127
8.8.2. Key Concepts	127
8.8.3. Discussion	127
8.8.4. UML Model	128
8.8.5. Examples	129
8.9. Generics	129
8.9.1. Synopsis	129
8.9.2. Key Concepts	130
8.9.3. Discussion	130
8.9.4. UML Model	130
8.9.5. Examples	132
8.10. Land Use	132
8.10.1. Synopsis	132
8.10.2. Key Concepts	133
8.10.3. Discussion	133
8.10.4. UML Model	134
8.10.5. Examples	135
8.11. Point Cloud	135
8.11.1. Synopsis	135
8.11.2. Key Concepts	135
8.11.3. Discussion	136
8.11.4. UML Model	136
8.11.5. Examples	137

8.12. Relief	137
8.12.1. Synopsis	137
8.12.2. Key Concepts	137
8.12.3. Discussion	137
8.12.4. UML Model	139
8.12.5. Examples	140
8.13. Transportation	140
8.13.1. Synopsis	141
8.13.2. Key Concepts	141
8.13.3. Discussion	141
8.13.4. Level of Detail	142
8.13.5. UML Model	145
8.13.6. Examples	148
8.14. Tunnel Model	149
8.14.1. Synopsis	149
8.14.2. Key Concepts	149
8.14.3. Discussion	150
8.14.4. Level of Detail	154
8.14.5. UML Model	155
8.14.6. Examples	157
8.15. Vegetation	160
8.15.1. Synopsis	160
8.15.2. Key Concepts	161
8.15.3. Discussion	161
8.15.4. Level of Detail	162
8.15.5. UML Model	165
8.15.6. Examples	167
8.16. Versioning	167
8.16.1. Synopsis	167
8.16.2. Key Concepts	167
8.16.3. Discussion	168
8.16.4. UML Model	168
8.16.5. Examples	169
8.17. Waterbodies	169
8.17.1. Synopsis	169
8.17.2. Key Concepts	169
8.17.3. Discussion	169
8.17.4. Level of Detail	170
8.17.5. UML Model	171
8.17.6. Examples	173
8.18. Extensions	173

8.18.1. Technical principle of ADEs	174
8.18.2. Example ADE	175
9. CityGML Data Dictionary	177
9.1. ISO Classes	177
9.1.1. Class AnyFeature (ISO 19109:2015)	177
9.1.2. Class CV_DiscreteGridPointCoverage (ISO 19123:2005)	178
9.1.3. Class DirectPosition (ISO 19107: 2003)	178
9.1.4. Class GM_Object (ISO 19107: 2003)	179
9.1.5. Class GM_MultiCurve (ISO 19107: 2003)	180
9.1.6. Class GM_MultiPoint (ISO 19107:2003)	181
9.1.7. Class GM_MultiSurface (ISO 19107:2003)	181
9.1.8. Class GM_Point (ISO 19107:2003)	182
9.1.9. Class GM_Solid (ISO 19107:2003)	182
9.1.10. Class GM_Surface (ISO 19107:2003)	183
9.1.11. Class GM_Tin (ISO 19107:2003)	184
9.1.12. Class GM_TriangulatedSurface (ISO 19107:2003)	184
9.1.13. Class SC_CRS (ISO 19111:2019)	184
9.1.14. Class TM_Position (ISO 19108:2006)	185
9.2. Core	186
9.2.1. Classes	186
9.2.2. Data Types	198
9.2.3. Basic Types	204
9.2.4. Unions	207
9.2.5. Code Lists	208
9.2.6. Enumerations	210
9.3. Appearance	211
9.3.1. Classes	211
9.3.2. Data Types	216
9.3.3. Basic Types	218
9.3.4. Unions	218
9.3.5. Code Lists	218
9.3.6. Enumerations	218
9.4. CityFurniture	219
9.4.1. Classes	220
9.4.2. Data Types	220
9.4.3. Basic Types	220
9.4.4. Unions	220
9.4.5. Code Lists	221
9.4.6. Enumerations	221
9.5. CityObjectGroup	221
9.5.1. Classes	221

9.5.2. Data Types	223
9.5.3. Basic Types	223
9.5.4. Unions	223
9.5.5. Code Lists	223
9.5.6. Enumerations	224
9.6. Dynamizer	224
9.6.1. Classes	224
9.6.2. Data Types	229
9.6.3. Basic Types	234
9.6.4. Unions	234
9.6.5. Code Lists	234
9.6.6. Enumerations	235
9.7. Generics	236
9.7.1. Classes	236
9.7.2. Data Types	238
9.7.3. Basic Types	242
9.7.4. Unions	242
9.7.5. Code Lists	242
9.7.6. Enumerations	245
9.8. LandUse	245
9.8.1. Classes	245
9.8.2. Data Types	245
9.8.3. Basic Types	246
9.8.4. Unions	246
9.8.5. Code Lists	246
9.8.6. Enumerations	246
9.9. PointCloud	247
9.9.1. Classes	247
9.9.2. Data Types	247
9.9.3. Basic Types	248
9.9.4. Unions	248
9.9.5. Code Lists	248
9.9.6. Enumerations	248
9.10. Relief	248
9.10.1. Classes	248
9.10.2. Data Types	252
9.10.3. Basic Types	253
9.10.4. Unions	253
9.10.5. Code Lists	253
9.10.6. Enumerations	253
9.11. Transportation	253

9.11.1. Classes	253
9.11.2. Data Types	264
9.11.3. Basic Types	267
9.11.4. Unions	267
9.11.5. Code Lists	267
9.11.6. Enumerations	273
9.12. Vegetation	273
9.12.1. Classes	274
9.12.2. Data Types	276
9.12.3. Basic Types	276
9.12.4. Unions	276
9.12.5. Code Lists	276
9.12.6. Enumerations	277
9.13. Versioning	278
9.13.1. Classes	278
9.13.2. Data Types	279
9.13.3. Basic Types	280
9.13.4. Unions	280
9.13.5. Code Lists	281
9.13.6. Enumerations	281
9.14. WaterBody	282
9.14.1. Classes	282
9.14.2. Data Types	284
9.14.3. Basic Types	285
9.14.4. Unions	285
9.14.5. Code Lists	285
9.14.6. Enumerations	286
9.15. Construction	286
9.15.1. Classes	286
9.15.2. Data Types	296
9.15.3. Basic Types	301
9.15.4. Unions	301
9.15.5. Code Lists	301
9.15.6. Enumerations	303
9.16. Bridge	305
9.16.1. Classes	305
9.16.2. Data Types	309
9.16.3. Basic Types	311
9.16.4. Unions	311
9.16.5. Code Lists	311
9.16.6. Enumerations	313

9.17. Building	314
9.17.1. Classes	314
9.17.2. Data Types	321
9.17.3. Basic Types	324
9.17.4. Unions	324
9.17.5. Code Lists	324
9.17.6. Enumerations	327
9.18. Tunnel	327
9.18.1. Classes	327
9.18.2. Data Types	332
9.18.3. Basic Types	333
9.18.4. Unions	333
9.18.5. Code Lists	333
9.18.6. Enumerations	336
Annex A: Revision History	337
Annex B: Glossary	338
B.1. ISO Concepts	339
B.2. Abbreviated Terms	343
Annex C: Bibliography	345

i. Abstract

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

The aim of the development of CityGML is to reach a common definition of the basic entities, attributes, and relations of a 3D city model. This is especially important with respect to the cost-effective sustainable maintenance of 3D city models, allowing the reuse of the same data in different application fields.

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 sustainable maintenance of 3D city models, allowing the possibility of selling the same data to customers from different application fields. The targeted application areas explicitly include city planning, architectural design, tourist and leisure activities, environmental simulation, mobile telecommunication, disaster management, homeland security, real estate management, vehicle and pedestrian navigation, and training simulators.

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

CityGML defines the classes and relations for the most relevant topographic objects in cities and regional models with respect to their geometrical, topological, semantical, and appearance properties. “City” is broadly defined to comprise not just built structures, but also elevation, vegetation, water bodies, “city furniture”, and more. Included are 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 differentiations can be represented, CityGML enables lossless information exchange between different GI systems and users.

Chapter 2. How To Use This Resource

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

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

Chapter 3. Scope

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

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

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

Chapter 4. References

The following 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 5. Terms and Definitions

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

2D data

geometry of features is represented in a two-dimensional space

NOTE In other words, the geometry of 2D data is given using (X,Y) coordinates.

[INSPIRE D2.8.III.2, definition 1]

2.5D data

geometry of features is represented in a three-dimensional space with the constraint that, for each (X,Y) position, there is only one Z

[INSPIRE D2.8.III.2, definition 2]

3D data

Geometry of features is represented in a three-dimensional space.

NOTE In other words, the geometry of 2D data is given using (X,Y,Z) coordinates without any constraints.

[INSPIRE D2.8.III.2, definition 3]

application schema

A set of [conceptual schema](#) for data required by one or more applications. An application schema contains selected parts of the base schemas presented in the ORM Information Viewpoint. Designers of application schemas may extend or restrict the types defined in the base schemas to define appropriate types for an application domain. Application schemas are information models for a specific information community.

OGC Definitions Register at <http://www.opengis.net/def/glossary/term/ApplicationSchema>

codelist

A value domain including a code for each permissible value.

conceptual model

model that defines concepts of a universe of discourse

[ISO 19101-1:2014, 4.1.5]

conceptual schema

1. formal description of a [conceptual model](#)

[ISO 19101-1:2014, 4.1.6]

2. base schema. Formal description of the model of any geospatial information. [Application schemas](#) are built from conceptual schemas.

OGC Definitions Register at <http://www.opengis.net/def/glossary/term/ConceptualSchema>

Implementation Specification

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

levels of detail

quantity of information that portrays the real world

NOTE The concept comprises data capturing rules of spatial object types, the accuracy and the types

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

[INSPIRE Glossary]

life-cycle information

set of properties of a spatial object that describe the temporal characteristics of a version of a spatial object or the changes between versions

[INSPIRE Glossary]

Platform (Model Driven Architecture)

the set of resources on which a system is realized.

[Object Management Group, Model Driven Architecture Guide rev. 2.0]

Platform Independent Model

a model that is independent of a specific 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 6. Conventions

6.1. Identifiers

The normative provisions in this document are denoted by the URI

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

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

6.2. UML Notation

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

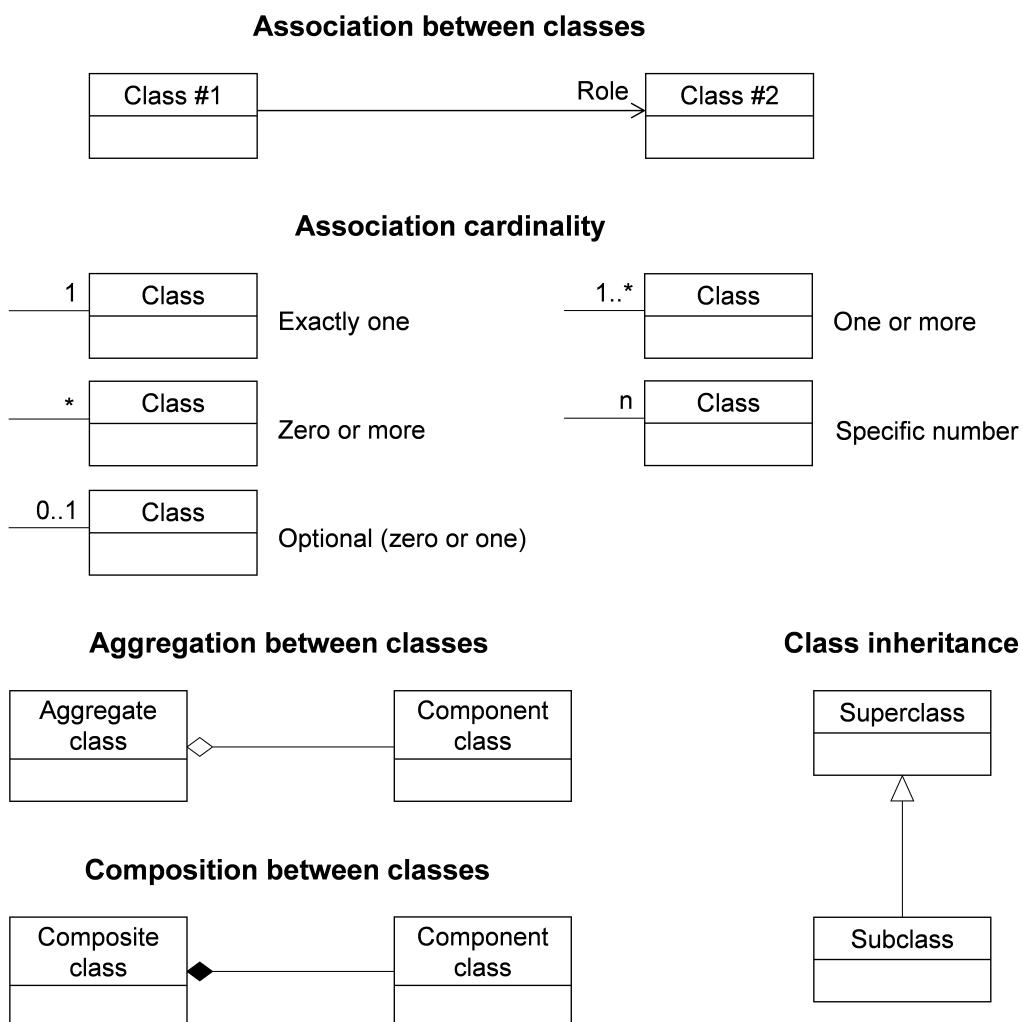


Figure 1. UML notation (see ISO TS 19103, Geographic information - Conceptual schema language).

All associations between model elements in the CityGML Conceptual Model are uni-directional. Thus, associations in the model are navigable in only one direction. The direction of navigation is depicted by an arrowhead. In general, the context an element takes within the association is

indicated by its role. The role is displayed near the target of the association. If the graphical representation is ambiguous though, the position of the role has to be drawn to the element the association points to.

The following stereotypes are used in this model:

- «ApplicationSchema» denotes a conceptual schema for data required by one or more applications. In the CityGML Conceptual Model, every module is defined as a separate application schema to allow for modularization.
- «FeatureType» represents features that are similar and exhibit common characteristics. Features are abstractions of real-world phenomena and have an identity.
- «TopLevelFeatureType» denotes features that represent the main components of the conceptual model. Top-level features may be further semantically and spatially decomposed and substructured into parts.
- «Type» denotes classes that are not directly instantiable, but are used as an abstract collection of operation, attribute and relation signatures. The stereotype is used in the CityGML Conceptual Model only for classes that are imported from the ISO standards 19107, 19109, 19111, and 19123.
- «ObjectType» represents objects that have an identity, but are not features.
- «DataType» defines a set of properties that lack identity. A data type is a classifier with no operations, whose primary purpose is to hold information.
- «Enumeration» enumerates the valid attribute values in a fixed list of named literal values. Enumerations are specified in the CityGML Conceptual Model.
- «BasicType» defines a basic data type.
- «CodeList» enumerates the valid attribute values. In contrast to Enumeration, the list of values is open and, thus, not given inline in the CityGML UML Model. The allowed values can be provided within an external code list.
- «Union» is a list of attributes. The semantics are that only one of the attributes can be present at any time.
- «Property» denotes attributes and association roles. This stereotype does not add further semantics to the conceptual model, but is required to be able to add tagged values to the attributes and association roles that are relevant for the encoding.
- «Version» denotes that the value of an association role that ends at a feature type is a specific version of the feature, not the feature in general.

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

Class defined in this
Requirements Class

Classes painted in yellow belong to the Requirements Class which is subject of discussion in that clause of the standard in which the UML diagram is given. For example, in the context of [rc_core_section], which introduces the *CityGML Core* module, the yellow color is used to denote

classes that are defined in the *CityGML Core* Requirements Class. Likewise, the yellow classes shown in the UML diagram in [rc_building-model_section] are associated with the *Building* Requirements Class that is subject of discussion in that chapter.

Class defined in another Requirements Class

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

Class defined in ISO 19107, ISO 19111 or ISO 19123

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

Class defined in ISO 19109

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

Notes and OCL constraints

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

The example UML diagram in Figure 2 demonstrates the UML notation and coloring scheme used throughout this standard. In this example, the yellow classes are associated with the *CityGML Building* module, the blue classes are from the *CityGML Core* and *Construction* modules, and the green class depicts a geometry element defined by ISO 19107.

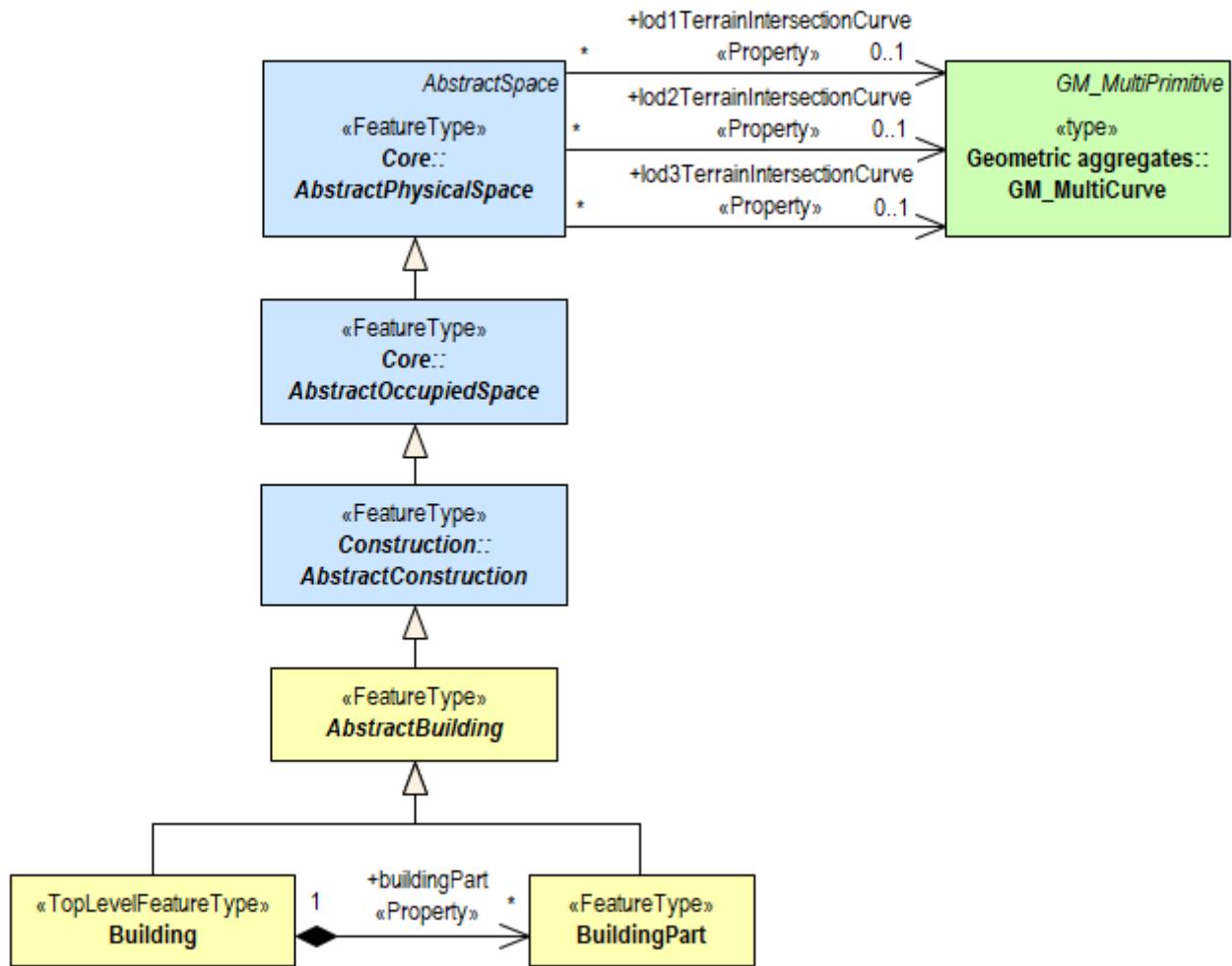


Figure 2. Example UML diagram demonstrating the UML notation and coloring scheme used throughout the CityGML Standard.

Chapter 7. CityGML Foundations

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

While the CityGML Conceptual Model can be used for 3D visualization purposes, its special merits lie in applications that go beyond visualization such as decision support, urban and landscape planning, urban facility management, Smart Cities, navigation (both indoor and outdoor), Building Information Modelling (especially for as-built documentation), integration of city and BIM models, assisted and autonomous driving, and simulations in general (cf. [Kolbe 2009](#)). A comprehensive overview on the many different applications of virtual 3D city models is given in [\[Biljecki et al. 2015\]](#). Many of the applications already use and some even require using CityGML.

In the CityGML CM, all 3D city objects can easily be enriched with thematic data. For example, street objects can be enriched with information about traffic density, speed limit, number of lanes etc., or buildings can be enriched by information on the heating and electrical energy demand, numbers of households and inhabitants, the appraised building value etc. Even building parts such as individual roof or wall surfaces can be enriched with information e.g. about solar irradiation and thermal insulation parameters. For many application domains specific extensions of the CityGML CM have already been created (cf. [Biljecki et al. 2018](#)).

7.1. Modularization

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

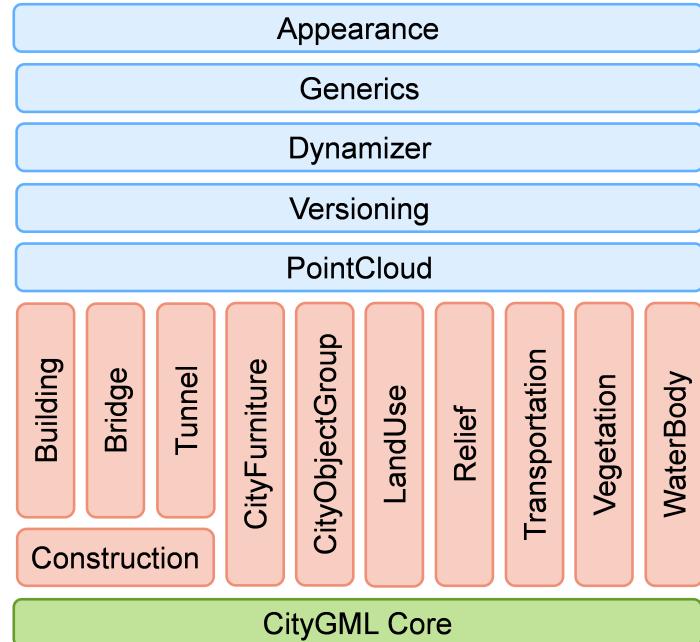


Figure 3. CityGML 3.0 module overview. The vertical boxes show the different thematic modules. Horizontal modules specify concepts that are applicable to all thematic modules.

The CityGML conceptual model is thematically decomposed into a *Core module* and different kinds of *extension modules* as shown in Figure 3. The Core module (shown in green) comprises the basic concepts and components of the CityGML CM and, thus, must be implemented by any conformant system. Each red colored module covers a specific thematic field of virtual 3D city models.

The CityGML CM introduces the following eleven thematic extension modules: *Building*, *Bridge*, *Tunnel*, *Construction*, *CityFurniture*, *CityObjectGroup*, *LandUse*, *Relief*, *Transportation*, *Vegetation*, and *WaterBody*. All three modules *Building*, *Bridge*, and *Tunnel* model civil structures and share common concepts that are grouped within the *Construction* module. The five blue colored extension modules add specific modelling aspects that can be used in conjunction with all thematic modules:

- The *Appearance* module contains the concepts to represent appearances (like textures and colours) of city objects.
- The *PointCloud* module provides concepts to represent the geometry of city objects by 3D point clouds.
- The *Generics* module defines the concepts for generic objects, attributes, and relationships.
- *Versioning* adds concepts for the representation of concurrent versions, real world object histories and feature histories.
- The *Dynamizer* module contains the concepts to represent city object properties by time series data and to link them with sensors, sensor data services or external files.

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

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

7.2. General Modelling Principles

7.2.1. Semantic Modelling 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.

7.2.2. Class Hierarchy and Inheritance of Properties and Relations

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

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

7.2.3. Relationships between CityGML objects

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

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

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

The CityGML CM contains many relationships that are specifically defined between certain feature types. For example, there is the *boundary* relationship from 3D volumetric objects to its

thematically differentiated 3D boundary surfaces. Another example is the *generalizesTo* relation between feature instances that represent objects on different 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.

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

The meanings of all elements defined in the CityGML conceptual model are normatively specified in the data dictionary in [CityGML Data Dictionary](#).

7.3. Representation of Spatial Properties

7.3.1. Geometry and Topology

Spatial properties of all CityGML feature types are represented using the geometry classes defined in ISO 19107. Spatial representations can have 0-, 1-, 2-, or 3-dimensional extents depending on the respective feature type and Levels of Detail (LOD). The LOD concept is discussed in [Levels of Detail \(LOD\)](#) and [Geometry and LOD](#). With only a few exceptions, all geometries must use 3D coordinate values. Besides primitive geometries like single points, curves, surfaces, and solids, CityGML makes use of different kinds of aggregations of geometries like spatial aggregates (*MultiPoint*, *MultiCurve*, *MultiSurface*, *MultiSolid*) and composites (*CompositeCurve*, *CompositeSurface*, *CompositeSolid*). Volumetric shapes are represented in ISO 19107 according to the so-called *Boundary Representation* (B-Rep). For further explanation see [Foley et al. 2002](#).

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

Note that the conceptual schema of ISO 19107 allows composite geometries to be defined by a recursive aggregation for every primitive type of the corresponding dimension. This aggregation schema allows the definition of nested aggregations (hierarchy of components). For example, a building geometry (*CompositeSolid*) can be composed of the house geometry (*CompositeSolid*) and the garage geometry (*Solid*), while the house's geometry is further decomposed into the roof geometry (*Solid*) and the geometry of the house body (*Solid*). This is illustrated in [Figure 4](#).

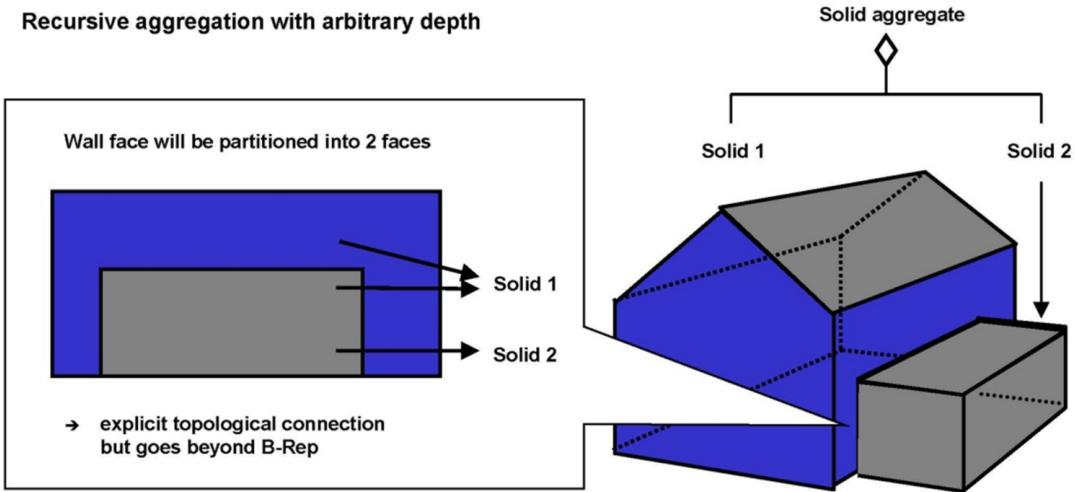


Figure 4. Recursive aggregation of objects and geometries in CityGML (graphic: IGG Uni Bonn).

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

Basically, there are three cases for sharing geometries:

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

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

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

Note that the use of topology in CityGML datasets by sharing geometries is optional. Furthermore, an encoding of the CityGML conceptual model might restrict the usage of shared geometries. For example, it might only be allowed to share identical (support) points from different 3D polygons or only entire polygons can be shared between touching solids (like shown in [Figure 4](#)).

7.3.2. Prototypic Objects / Scene Graph Concepts

In CityGML, objects of equal shape like trees and other vegetation objects, traffic lights and traffic signs can be represented as prototypes which are instantiated multiple times at different locations (see [Figure 5](#)). The geometry of prototypes is defined in local coordinate systems. Every instance is represented by a reference to the prototype, a base point in the world coordinate reference system (CRS) and a transformation matrix that facilitates scaling, rotation, and translation of the prototype. The principle is adopted from the concept of scene graphs used in computer graphics standards. Since the ISO 19107 geometry model does not provide support for scene graph concepts, the CityGML class `ImplicitGeometry` has been introduced (for further description see [Geometry and LOD](#)). The prototype geometry can be represented using ISO 19107 geometry objects or by referencing an external file containing the geometry in another data format.

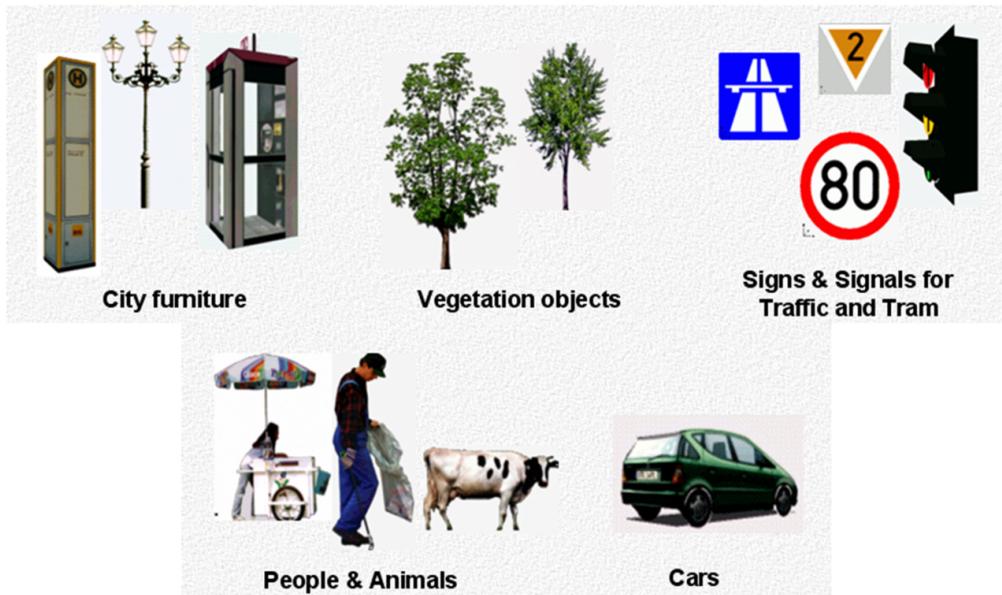


Figure 5. Examples of prototypic shapes (source: Rheinmetall Defence Electronics).

7.3.3. Point Cloud Representation

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

7.3.4. Coordinate Reference Systems (CRS)

CityGML is about 3D city and landscape models. This means that nearly all geometries use 3D coordinates, where each single point and also the points defining the boundaries of surfaces and solids have three coordinate values (x,y,z) each. Coordinates always have to be given with respect to a coordinate reference system (CRS) that relates them unambiguously with a specific position on the Earth. In contrast to CAD or BIM, each 3D point is absolutely georeferenced, which makes CityGML especially suitable to represent geographically large extended structures like airports, railways, bridges, dams, where the Earth curvature has a significant effect on the object's geometry (for further explanations see [Kaden & Clemen 2017](#)).

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

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

7.4.1. Spaces and Space Boundaries

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

To obtain a more precise definition of spaces, they are further subdivided into physical spaces and logical spaces. Physical spaces are spaces that are fully or partially bounded by physical objects. Buildings and rooms, for instance, are physical spaces as they are bounded by walls and slabs. Traffic spaces of roads are physical spaces as they are bounded by road surfaces against the ground. Logical spaces, in contrast, are spaces that are not necessarily bounded by physical objects, but are defined according to thematic considerations. Depending on the application, logical spaces can also be bounded by non-physical, i.e. virtual boundaries, and they can represent aggregations of physical spaces. A building unit, for instance, is a logical space as it aggregates specific rooms to flats, the rooms being the physical spaces that are bounded by wall surfaces, whereas the aggregation as a whole is being delimited by a virtual boundary. Other examples are city districts which are bounded by virtual vertically extruded administrative boundaries, public spaces vs. Security zones in airports, or city zones with specific regulations stemming from urban planning. The definition of physical and logical spaces and of corresponding physical and virtual boundaries is in line with the discussion in [\[Smith & Varzi 2000\]](#) on the difference between bona fide and fiat boundaries to bound objects. Bona fide boundaries are physical boundaries; they correspond to the

physical boundaries of physical spaces in the CityGML 3.0 CM. In contrast, flat boundaries are man-made boundaries. They are equivalent to the virtual boundaries of logical spaces.

Physical spaces, in turn, are further classified into occupied spaces and unoccupied spaces. Occupied spaces represent physical volumetric objects that occupy space in the urban environment. Examples for occupied spaces are buildings, bridges, trees, city furniture, and water bodies. Occupying space means that some space is blocked by these volumetric objects. For instance, the space blocked by the building in [Figure 6](#) cannot be used any more for driving through this space or placing a tree on that space. In contrast, unoccupied spaces represent physical volumetric entities that do not occupy space in the urban environment, i.e. no space is blocked by these volumetric objects. Examples for unoccupied spaces are building rooms and traffic spaces. There is a risk of misunderstanding the term *OccupiedSpace*. However, we decided to use the term anyway, as it is established in the field of robotics for over three decades [[Elfes 1989](#)]. The navigation of mobile robots makes use of a so-called occupancy map that marks areas that are occupied by matter and, thus, are not navigable for robots.

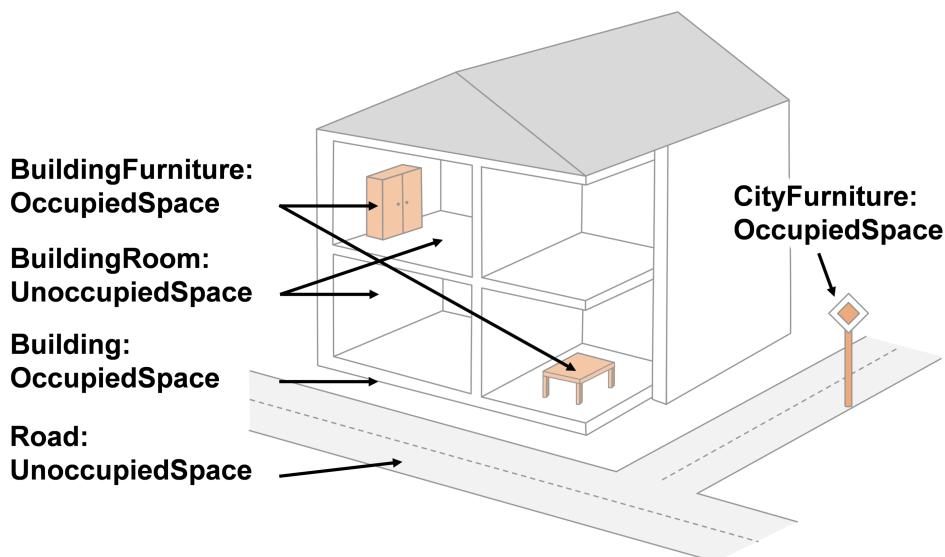


Figure 6. Occupied and unoccupied spaces

The new space concept offers several advantages:

- In the CityGML 3.0 Conceptual Model, all geometric representations are only defined in the *Core* module. This makes (a) models of the thematic modules simpler as they no longer need to be associated directly with the geometry classes, and (b) implementation easier as all spatial concepts have only to be implemented once in the *Core* module. All thematic modules like *Building*, *Relief*, *WaterBody*, etc. inherit their geometric representations from the *Core* module.
- The space concept supports the expression of explicit topological, geometrical, and thematic relations between spaces and spaces, spaces and space boundaries, and space boundaries and space boundaries. Thus, implementing the checking of geometric-topological consistency will become easier. That is because most checks can be expressed and performed on the CityGML *Core* module and then automatically applied to all thematic modules
- For the analysis of navigable spaces (e.g. to generate IndoorGML data from CityGML) algorithms can be defined on the level of the *Core* module. These algorithms will then work with all CityGML feature classes and also ADEs as they are derived from the *Core*. The same is true for other applications of 3D city models listed in [[Biljecki et al. 2015](#)] such as visibility analyses

including shadow casting or solar irradiation analyses.

- Practitioners and developers do not see much of the space concept. That is because the space and space boundary classes are just abstract classes. Only elements representing objects from concrete subclasses such as Building, BuildingRoom, or TrafficSpace will appear in CityGML data sets.

7.4.2. Modelling City Objects by the Composition of Spaces

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

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

7.4.3. Rules for Surface Orientations of OccupiedSpaces and UnoccupiedSpaces

The classification of feature types into OccupiedSpace and UnoccupiedSpace also defines the semantics of the geometries attached to the respective features. For OccupiedSpaces, the attached geometries describe volumes that are (mostly) physically occupied. For UnoccupiedSpaces, the attached geometries describe (or bound) volumes that are (mostly) physically unoccupied. This also has an impact on the required orientation of the surface normal (at point P this is a vector perpendicular to the tangent plane of the surface at P) for attached thematic surfaces. For OccupiedSpaces, the normal vectors of thematic surfaces must point in the same direction as the

surfaces of the outer shell of the volume. For UnoccupiedSpaces, the normal vectors of thematic surfaces must point in the opposite direction as the surfaces of the outer shell of the volume. This means that from the perspective of an observer of a city scene, the surface normals must always be directed towards the observer. In the case of OccupiedSpaces (e.g. Buildings, Furniture), the observer must be located outside the OccupiedSpace for the surface normals being directed towards the observer; whereas in the case of UnoccupiedSpaces (e.g. Rooms, Roads), the observer is typically inside the UnoccupiedSpace.

7.4.4. Levels of Detail (LOD)

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

The classification of real-world objects into spaces and space boundaries is solely based on the semantics of these objects and not on their used geometry type, as the CityGML 3.0 CM allows various geometrical representations for objects. A building, for instance, can be spatially represented by a 3D solid (e.g. in LOD1), but at the same time, the real-world geometry can also be abstracted by a single point, footprint or roof print (LOD0), or by a 3D mesh (LOD3). The outer shell of the building may also be semantically decomposed into wall, roof, and ground surfaces. [Figure 7](#) shows different representations of the same real-world building object in different geometric LODs (and appearances).

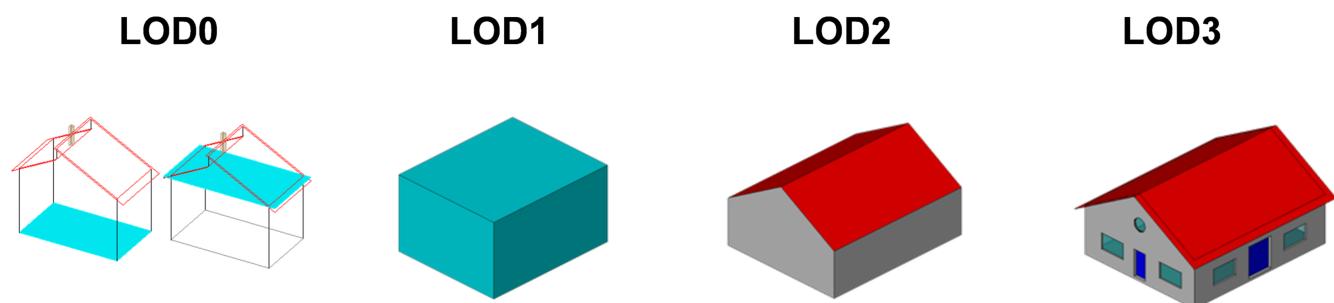


Figure 7. Representation of the same real-world building in the Levels of Detail 0-3.

The biggest changes between CityGML 3.0 and earlier versions are that:

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

and doors can be represented in all LODs 0-3. In CityGML 2.0 or earlier thematic surfaces were only allowed starting from LOD2, openings like doors and windows starting from LOD3, and interior rooms and furniture only in LOD4.

3. In the CityGML 3.0 Conceptual Model the geometry representations were moved from the thematic modules to the *Core* module and are now associated with the semantic concepts of *Spaces* and *Space Boundaries*. This led to a significant simplification of the models of the thematic modules. Since all feature types in the thematic modules are defined as subclasses of the space and space boundary classes, they automatically inherit the geometry classes and, thus, no longer require direct associations with them. This also led to a harmonized LOD representation over all CityGML feature types.
4. If new feature types are defined in Application Domain Extensions (ADEs) based on the abstract Space and Space Boundary classes from the Core module, they automatically inherit the spatial representations and the LOD concept.

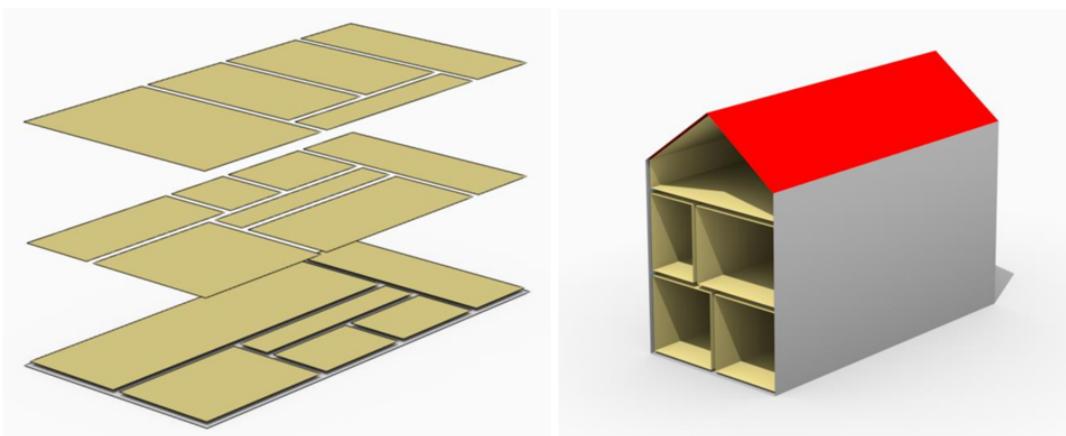


Figure 8. Floor plan representation (LOD0) of a building (left), combined LOD2 indoor and outdoor representation (right). Image adopted from Löwner et al. 2016.

Spaces and all its subclasses like *Building*, *Room*, and *TrafficSpace* can now be spatially represented by single points in LOD0, multi-surfaces in LOD0/2/3, solids in LOD1/2/3, and multi-curves in LOD2/3. *Space Boundaries* and all its subclasses such as *WallSurface*, *LandUse*, or *Relief* can now be represented by multi-surfaces in LOD0/2/3 and as multi-curves in LOD2/3. See [Geometry and LOD](#) for further details on the different Levels of Detail.

7.4.5. Closure Surfaces

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

The concept of *ClosureSurface* can also be employed to model the entrances of subsurface objects. Those objects like tunnels or pedestrian underpasses have to be modelled as closed solids in order to compute their volume. An example would be for use in flood simulations. The entrances to subsurface objects also have to be sealed to avoid holes in the digital terrain model (see [Figure 9](#)). However, in close-range visualizations the entrance should be treated as open. Thus, closure

surfaces are an adequate way to model those entrances.



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

7.4.6. Terrain Intersection Curves

An important issue in city modelling is the integration of 3D objects and the terrain. Problems arise if 3D objects float over or sink into the terrain. This is particularly the case when terrains and 3D objects in different LODs are combined, when the terrain and 3D models are updated independently from each other, or when they come from different data providers [Kolbe & Gröger 2003]. To overcome this problem, the TerrainIntersectionCurve (TIC) of a 3D object is introduced. These curves denote the exact position where the terrain touches the 3D object (see Figure 10). TICs can be applied to all CityGML feature types that are derived from AbstractPhysicalSpace such as buildings, bridges, tunnels, but also city furniture, vegetation, and generic city objects.

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

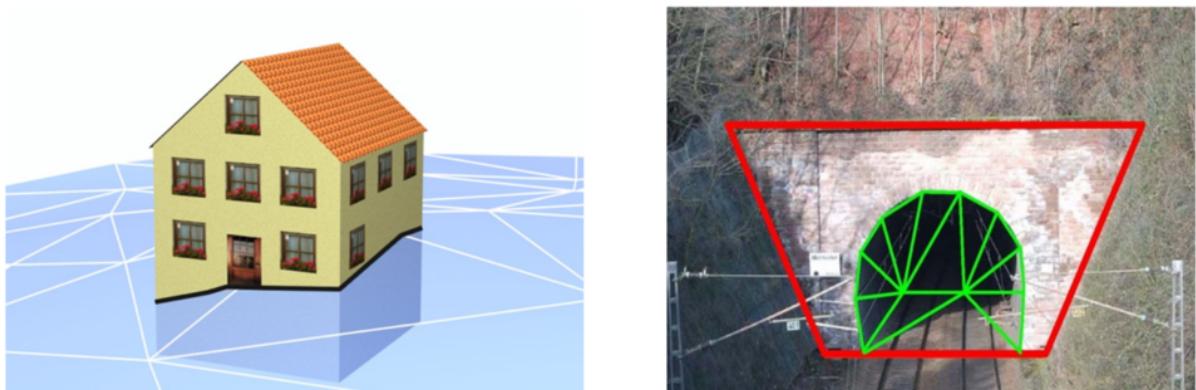


Figure 10. TerrainIntersectionCurve for a building (left, black) and a tunnel object (right, red). The tunnel’s hollow space is sealed by a triangulated ClosureSurface (graphic: IGG Uni Bonn).

7.4.7. Coherent Semantical-Geometrical Modelling

An important design principle for CityGML is the coherent modelling of semantic objects and their spatial representations. At the semantic level, real-world entities are represented by features, such as buildings, walls, windows, or rooms. The description also includes attributes, relations and aggregation hierarchies (part-whole-relations) between features. Thus the part-of-relationship between features can be derived at the semantic level only, without considering geometry. However, at the spatial level, geometry objects are assigned to features representing their spatial location, shape, and extent. So the model consists of two hierarchies: The semantic and the geometrical in which the corresponding objects are linked by relationships (cf. [Stadler & Kolbe 2007](#)). The advantage of this approach is that it can be navigated in both hierarchies and between both hierarchies arbitrarily, for answering thematic and/or geometrical queries or performing analyses.

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

7.5. Appearances

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

The CityGML Conceptual Model supports feature appearances for an arbitrary number of themes per city model. Each LOD of a feature can have an individual appearance. Appearances can represent – among others – textures and georeferenced textures. CityGML's appearance model is packaged within the Appearance module (cf. [\[rc_appearance_section\]](#)).

7.6. Modelling Dynamic Data

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

[Chaturvedi & Kolbe 2019].

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

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

7.6.1. Versioning and Histories

As described in [Semantic Modelling of Real-World Objects](#), the bitemporal timestamps of all CityGML feature types allow representing the evolution of the real city and its model over time. The new *Versioning* module extends this concept by the possibility of representing multiple, concurrent versions of the city model. For that purpose, the module defines two new feature types: 1) *Version*, which can be used to explicitly define named states of the 3D city model and denote all the specific versions of objects belonging to such states. 2) *VersionTransition*, which allows to explicitly link different versions of the 3D city model by describing the reason of change and the modifications applied. Details on the versioning concept are given in [\[Chaturvedi et al. 2015\]](#).

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

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

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



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

As shown in Figure 11, Dynamizers serve three main purposes:

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

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

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

Dynamizers are given in [Chaturvedi & Kolbe 2017].

7.7. Extending CityGML

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

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

The concept of generic objects and attributes enables the runtime extensions of CityGML applications. This means that any city object may be augmented by additional attributes and relations, whose names, data types, and values can be provided by a running application without requiring extensions to the CityGML conceptual schema and the respective encodings. Similarly, features not represented by the predefined thematic classes of the CityGML conceptual model may be modelled and exchanged using generic objects. The generic extensions of CityGML are provided by the *Generics* module (cf. [\[ug-generics-section\]](#)).

Application Domain Extensions (ADE) specify additions to the CityGML conceptual model. Such additions comprise the introduction of new properties to existing CityGML feature types such as the energy demand of a building or the definition of additional feature types. The difference between ADEs and generic objects and attributes is, that an ADE has to be defined in an extra conceptual schema (provided in UML) with its own namespace. Encodings have to be extended accordingly. The advantage of this approach is that the extension is formally specified. Extended CityGML datasets can be validated against the CityGML CM and the respective ADE schema. ADEs can be defined (and even standardized) by information communities which are interested in specific application fields. More than one ADE can be used simultaneously in the same dataset. Examples for popular ADEs are the Utility Network ADE [[Becker et al. 2011](#); [Kutzner et al. 2018](#)] and the Energy ADE [[Nouvel et al. 2015](#); [Agugiaro et al. 2018](#)]. A comprehensive overview of CityGML ADEs is given in [[Biljecki et al. 2018](#)]. Further details on ADEs are given in [\[ug_ade_section\]](#).

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

Chapter 8. CityGML Model

8.1. Core

Contributors
C. Heazel - first draft

The CityGML Core module defines the basic concepts and components of city models. This rather large body of work is divided into seven sections. These sections build on each other from the fundamental principles specified by the relevant ISO standards up to the full CityGML model. These sections are summarized in [Table 1](#).

Table 1. CityGML Core Sections

Key Concepts	Summarizes the key concepts described in the Core module.
The Use of ISO Standards	Describes the use of the ISO 19100 series of International Standards to provide a foundation to the CityGML model.
City Models and City Objects	Defines the basic building blocks of the CityGML model.
Space Concept	Defines the concepts of space as used in the CityGML model.
Geometry and LOD	Defines the geometry and Levels Of Detail concepts.
CityGML Core Model	Presents the complete Core model.
Types, Enumerations, and Codelist	Defines the little things which make this model work.

8.1.1. Key Concepts

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

CityModel: The CityModel class is the root class of every CityGML conceptual model. Its primary purpose is to aggregate CityModelMembers.

CityModelMember: CityModelMember is a Union of all possible classes which can be included in a city model. Those classes are summarized in [Table 2](#).

Table 2. City Model Members

AbstractAppearance	Rules for rendering the associated objects
AbstractCityObject	Physical objects with a location and geometry

AbstractFeature	Others
AbstractVersion	Version information
AbstractVersionTransition	Version information

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

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

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

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

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

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

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

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

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

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

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

AbstractSpaceBoundary: Classes which bound a space.

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

ImplicitGeometry: A geometry representation where the shape is stored only once as a prototypical geometry, for example a tree or other vegetation object, a traffic light or a traffic sign. This prototypic geometry object can be re-used or referenced many times, wherever the corresponding feature occurs in the 3D city model. A type of [Abstract Feature](#).

8.1.2. ISO Dependencies

CityGML builds on the ISO 19100 family of standards. The applicable standards are identified in [Figure 12](#). A [Data dictionaries](#) is also included for all of the ISO-defined classes explicitly referenced in the CityGML UML model. These data dictionaries are provided for the convenience of the user. The ISO standards are the normative source.

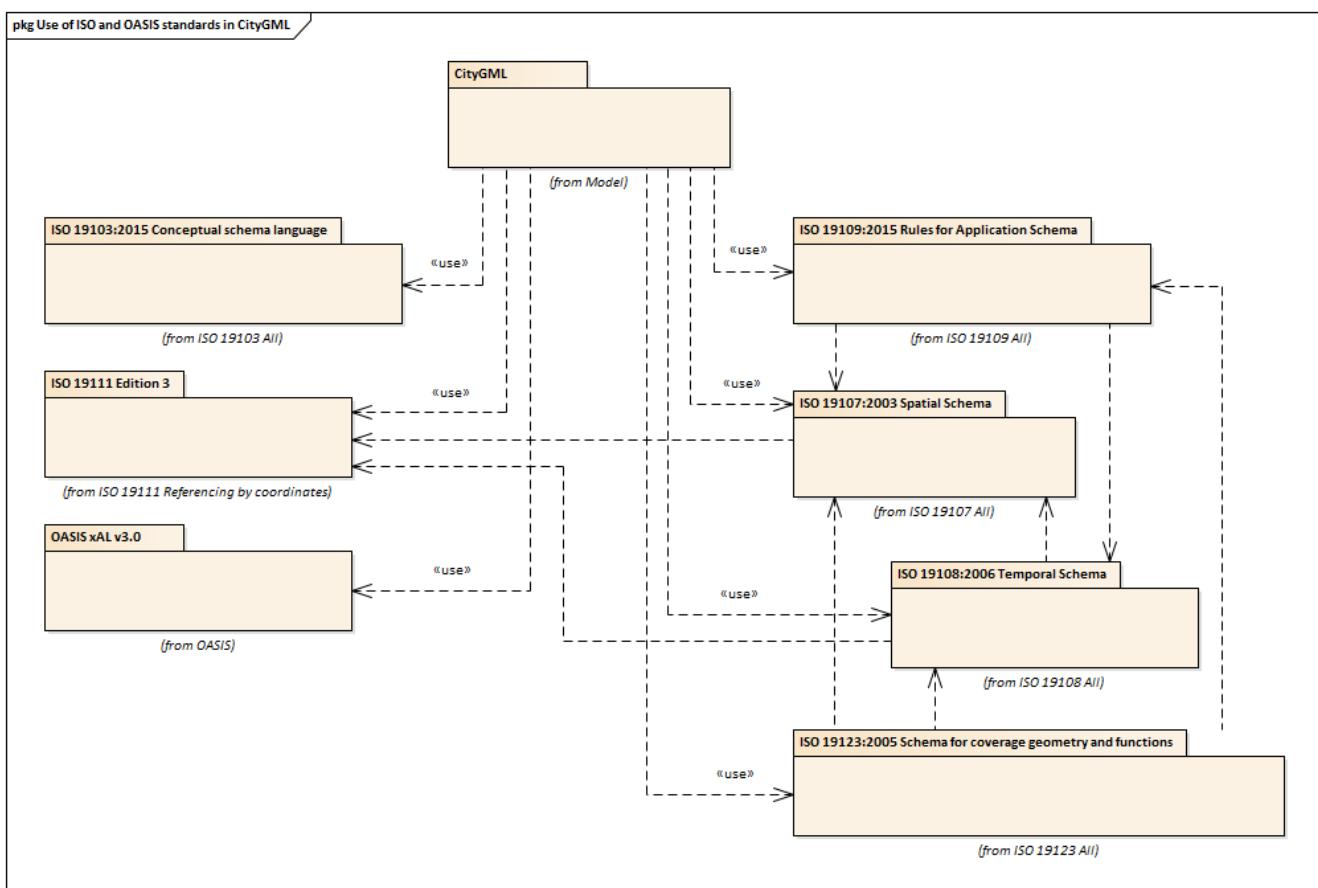


Figure 12. Use of ISO Standards in CityGML

ISO Classes

The ISO classes explicitly used in the CityGML UML model are introduced in [Table 3](#). Detailed descriptions are provided in the [Data Dictionary](#).

Table 3. ISO Classes used in CityGML

Class Name	Description
AnyFeature	A generalization of all feature types

CV_DiscreteGridPointCoverage	A coverage that returns the same feature attribute values for every direct position within any object in its domain.
Direct Position	The coordinates for a position within some coordinate reference system.
GM_Object	root class of the geometric object taxonomy.
GM_MultiCurve	An aggregate class containing only instances of GM_OrientableCurve.
GM_MultiPoint	An aggregate class containing only points.
GM_MultiSurface	An aggregate class containing only instances of GM_OrientableSurface.
GM_Point	The basic data type for a geometric object consisting of one and only one point.
GM_Solid	The basis for 3-dimensional geometry. The extent of a solid is defined by the boundary surfaces.
GM_Surface	The basis for 2-dimensional geometry.
GM_Tin	A GM_TriangulatedSurface which uses the Delaunay or similar algorithm.
GM_TriangulatedSurface	A GM_PolyhedralSurface that is composed only of triangles
SC_CRS	Coordinate reference system which is usually single but may be compound.
TM_Position	A union class that consists of one of the data types listed as its attributes.

ISO Geometry

The most common geometry concept found in the CityGML 3.0 Standard is the concept of [multi-primitives](#). These are homogeneous collections of [GM_Primitives](#) which are aggregated to form a more complex geometry.

[GM_Composites](#) are another form of [GM_Primitive](#) collection. These differ from [GM_MultiPrimitive](#) in that the collection can be heterogeneous. It should be noted that none of the classes in the CityGML 3.0 Standard are descended from [GM_Composites](#). However, the terms "CompositeCurve", "CompositeSurface", and "CompositeSolid" do appear in the text. The [composit](#) concept can also be seen in the association between spaces and surfaces. Therefore, an explanation of [composites](#) has been included for completeness.

GM_Primitive

[GM_Primitive](#) is the abstract root class of the geometric primitives. Its main purpose is to define the basic "boundary" operation that ties the primitives in each dimension together. A geometric primitive ([GM_Primitive](#)) is a geometric object that is not decomposed further into other primitives in the system. This includes curves and surfaces, even though they are composed of curve segments and surface patches, respectively. This composition is a strong aggregation: curve segments and surface patches cannot exist outside the context of a primitive.

NOTE Most geometric primitives are decomposable infinitely many times. Adding a centre point to a line may split that line into two separate lines. A new curve drawn across a surface may divide that surface into two parts, each of which is a surface. This is the reason that the normal definition

of primitive as "non-decomposable" is not plausible in a geometry model - the only non-decomposable object in geometry is a point.

GM_MultiPrimitive

Any geometric object that is used to describe a feature is a collection of [geometric primitives](#). A homogeneous collection of geometric primitives may be a multi-primitive (GM_MultiPrimitive). Geometric complexes have additional properties specific to the type of [geometric primitive](#) they aggregate.

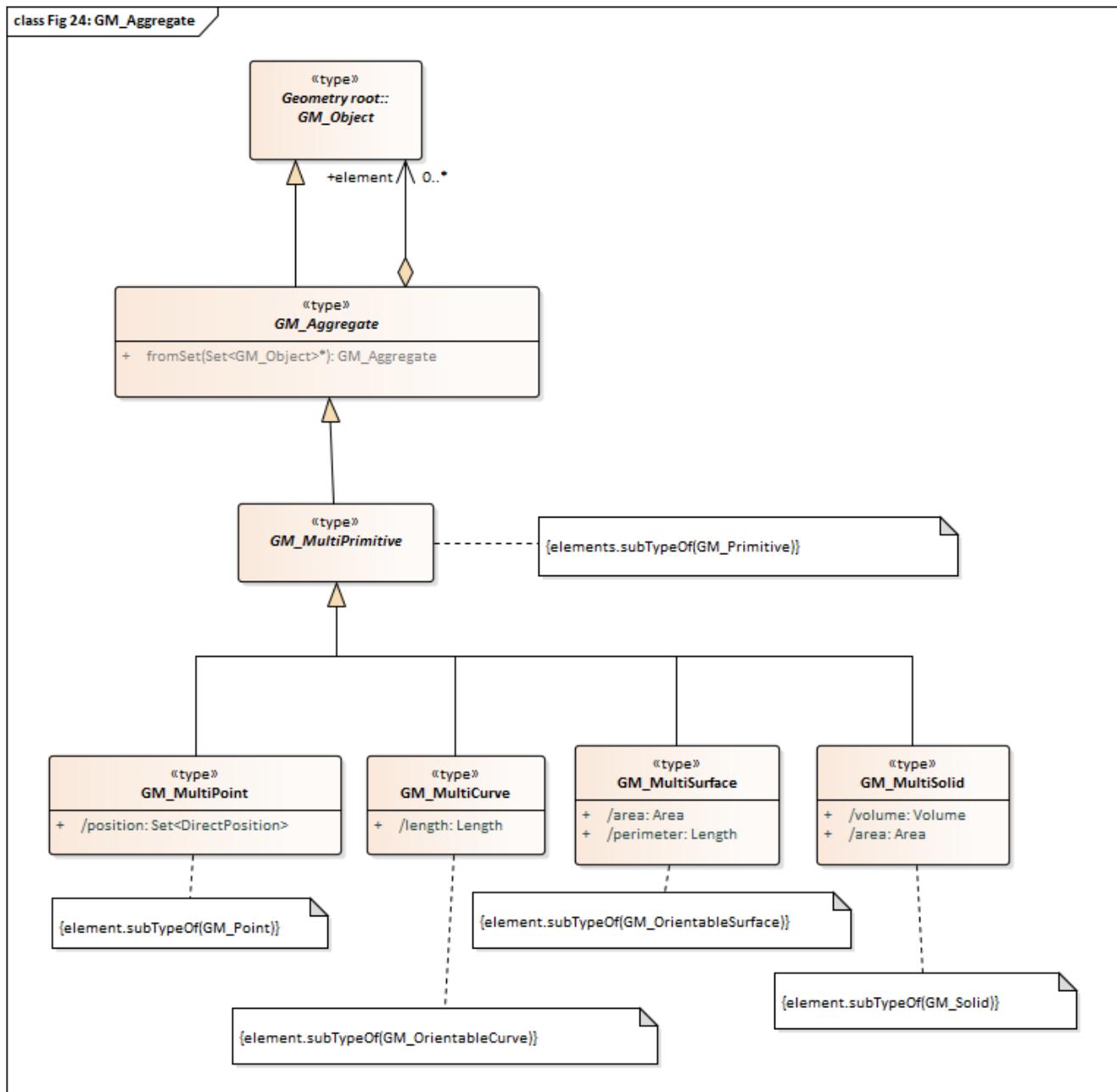


Figure 13. GM_MultiPrimitive Context Diagram

GM_Complex

A GM_Complex is a set of disjoint geometric primitives ([GM_Primitive](#)) such that the boundary of each primitive can be represented as the union of other geometric primitives within the complex.

Any geometric object that is used to describe a feature is a collection of [geometric primitives](#). A

collection of geometric primitives may be a geometric complex (GM_Complex). Geometric complexes have additional properties such as closure by boundary operations and mutually exclusive component parts.

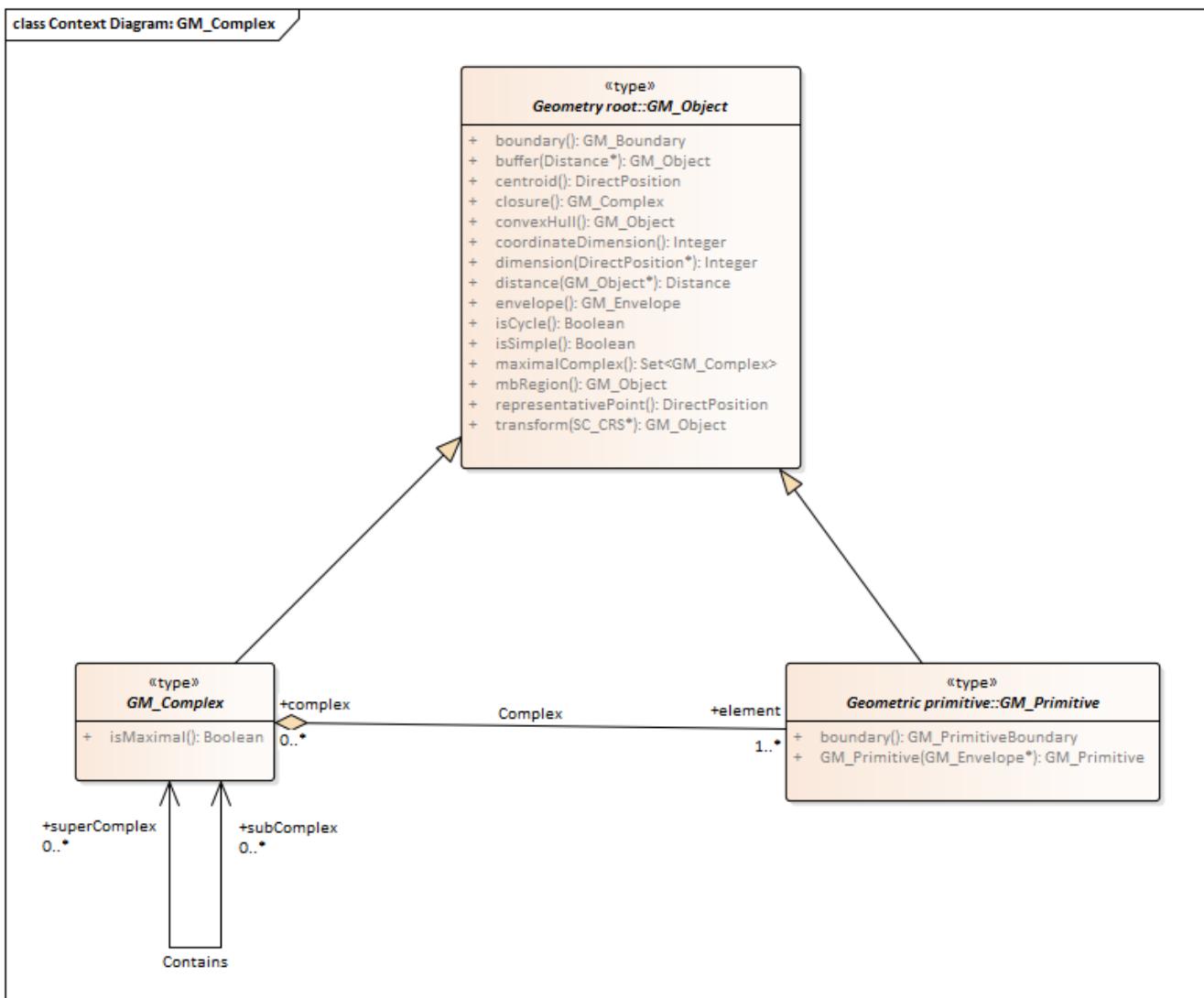


Figure 14. GM_Complex Context Diagram

GM_Primitive and **GM_Complex** share most semantics, in the meaning of operations, attributes and associations. There is an exception in that a **GM_Primitive** shall not contain its boundary (except in the trivial case of **GM_Point** where the boundary is empty), while a **GM_Complex** shall contain its boundary in all cases. This means that if an instantiated object implements **GM_Object** operations both as **GM_Primitive** and as a **GM_Complex**, the semantics of each set theoretic operation is determined by the its name resolution. Specifically, for a particular object such as **GM_CompositeCurve**, **GM_Primitive::contains** (returns FALSE for end points) is different from **GM_Complex::contains** (returns TRUE for end points). Further, if that object is cast as a **GM_Primitive** value and as a **GM_Complex** value, then the two values need not be equal as **GM_Objects**.

GM_Complex aggregates **GM_Primitives** through the `element` property. Since this is an aggregation, the target **GM_Primitive** may be associated with more than one **GM_Complex**.

A **GM_Complex** object can also have a whole/part relationship with other **GM_Complex** objects. The `contains` association is used to associate the `superComplex` instance with the `subComplex` instance.

Note that the geometric primitives in the set are mutually exclusive in the sense that no point is interior to more than one primitive. The set is closed under boundary operations, meaning that for each element in the complex, there is a collection (also a complex) of geometric primitives that represents the boundary of that element.

GM_Composite

GM_Composite is a subclass of [GM_Complex](#). Like [GM_Complex](#), it has an association with [GM_Primitives](#). In this case this is an **composition** association with a **composite** role (GM_Composite) and a **generator** role (GM_Primitive). As with the GM_Complex, the GM_Primitive may be associated with more than one GM_Composite.

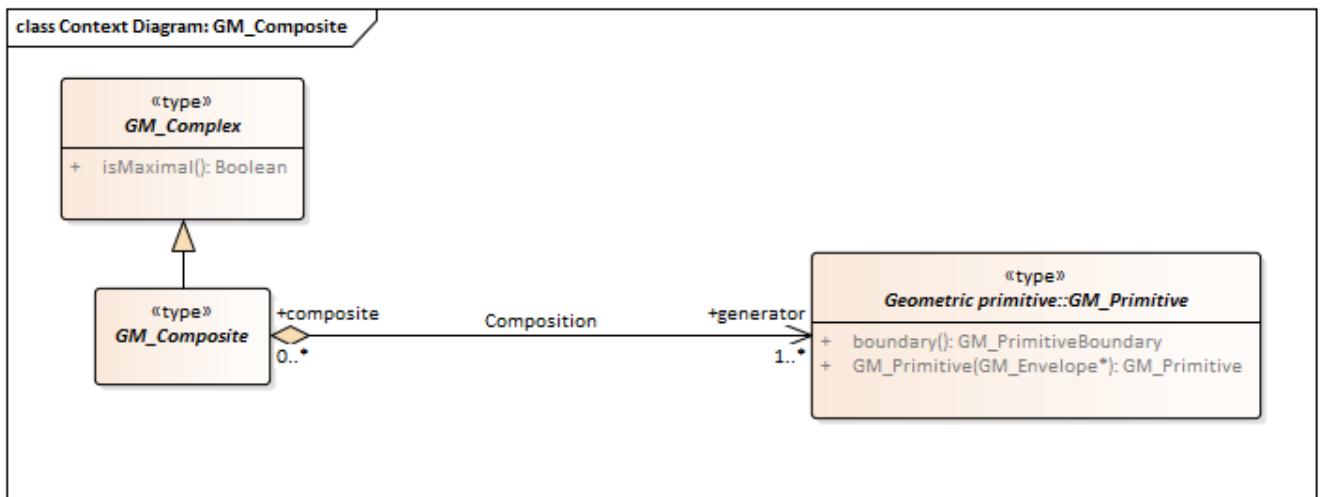


Figure 15. GM_Composite Context Diagram

GM_CompositeSurface

A GM_Composite where the [GM_Primitives](#) is a [GM_OrientatableSurface](#).

A GM_CompositeSurface is also a subclass of [GM_Primitives](#) is a [GM_OrientatableSurface](#). One of the few examples of multiple inheritance.

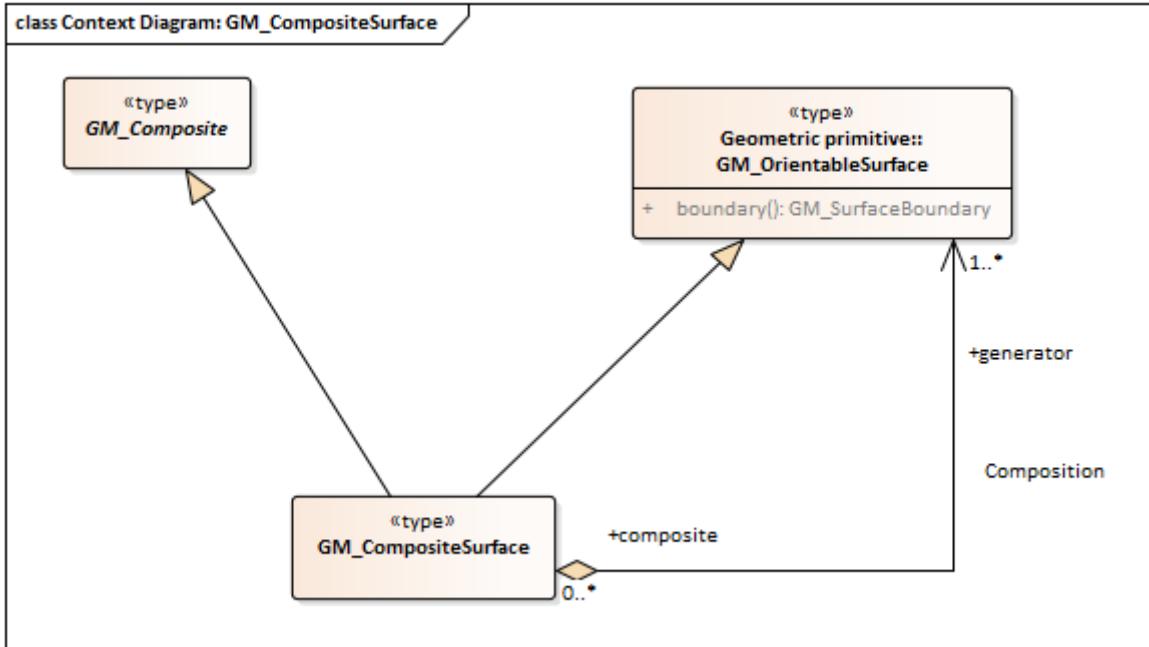


Figure 16. GM_CompositeSurface Context Diagram

GM_OrientatableSurface

GM_OrientatableSurface consists of a surface and an orientation inherited from GM_OrientablePrimitive. If the orientation is "+", then the GM_OrientableSurface is a GM_Surface. If the orientation is "-", then the GM_OrientableSurface is a reference to a GM_Surface with an upNormal that reverses the direction for this GM_OrientableSurface, the sense of "the top of the surface" (see 6.4.33.2).

```

GM_OrientableSurface:
{Orientation = "+" implies primitive = self};
{((Orientation = "-" and TransfiniteSet::contains(p : DirectPosition)) implies
(primitive.upNormal(p) = - self.upNormal(p)));
  
```

GM_CompositeCurve

A GM_CompositeCurve is a list of geometric curves such that the each geometric curve in the set terminates at the start point of the subsequent curve in the list

The **generator** is a GM_OrientableCurve.

A GM_CompositeCurve is also a subclass of GM_OrientatableCurve. One of the few examples of multiple inheritance.

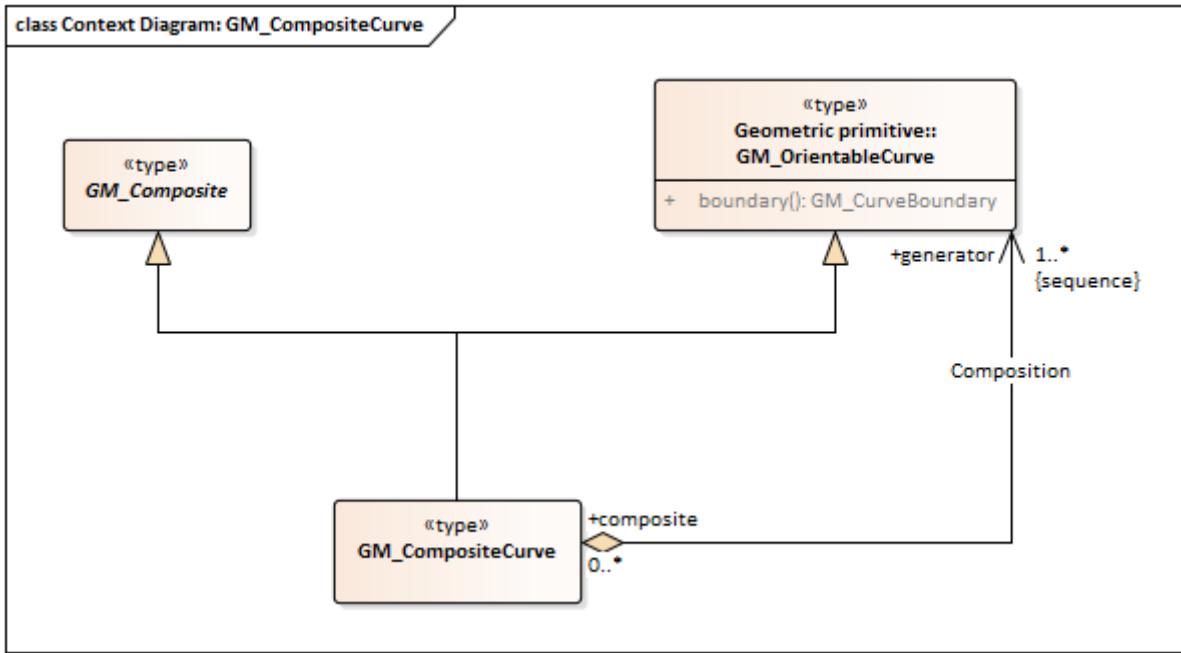


Figure 17. GM_CompositeCurve Context Diagram

GM_OrientatableCurve

GM_OrientatableCurve consists of a curve and an orientation inherited from GM_OrientablePrimitive. If the orientation is "+", then the GM_OrientableCurve is a [GM_Curve](#). If the orientation is "-", then the GM_OrientableCurve is related to another [GM_Curve](#) with a parameterization that reverses the sense of the curve traversal.

```

GM_OrientableCurve:
{Orientation = "+" implies primitive = self}; +
{Orientation = "-" implies primitive.parameterization(length()-s) =
parameterization(s)};
  
```

GM_CompositeSolid

A GM_CompositeSolid is a set of geometric solids adjoining one another along common boundary geometric surfaces.

The **generator** is a [GM_Solid](#).

A GM_CompositeSolid is also a subclass of [GM_Solid](#). One of the few examples of multiple inheritance.

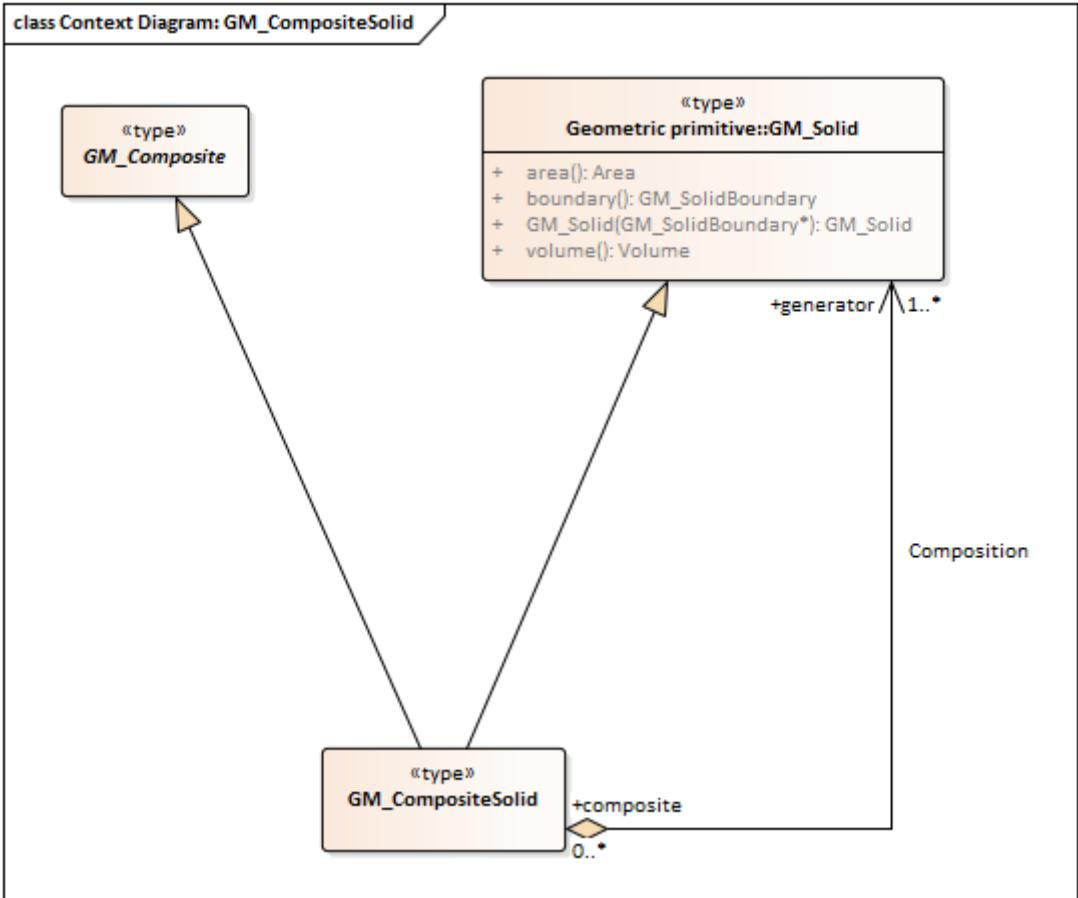


Figure 18. GM_CompositeSolid Context Diagram

8.1.3. City Models and City Objects

City models are virtual representations of real-world cities and landscapes. A city model aggregates different types of objects, which can be city objects, appearances, different versions of the city model, transitions between different versions of the city model, and feature objects. All objects defined in the CityGML CM are **features with lifespan**. This allows the optional specification of the real-world and database times for the existence of each feature, as is required by the Versioning module (cf. [\[ug_versioning_section\]](#)). Features that define thematic concepts related to cities and landscapes, such as building, bridge, water body, or land use, are referred to as city objects. All city objects define properties that describe the objects in more detail. These static properties can be overridden with time-varying data through Dynamizers (cf. [\[ug_dynamizer_section\]](#)).

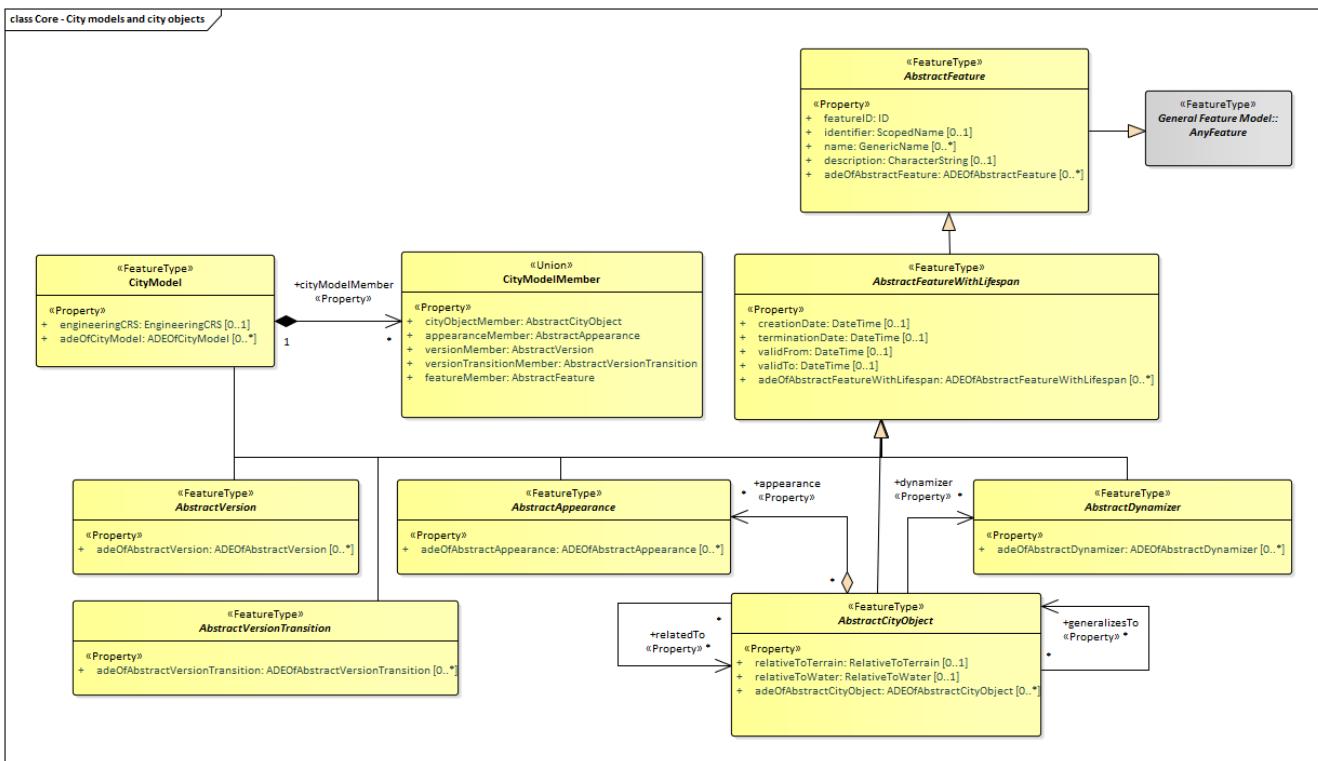


Figure 19. UML City Models and City Objects

The City Model and City Object classes in the in the [Data Dictionary](#).

NOTE Significant concepts

- **AbstractFeature** (provides identifying properties)
- **AbstractFeatureWithLifespan** (adds temporal properties)
- **AbstractThematicSurface** (adds geometry and city context)
- **AbstractOccupiedSpace**
- **AbstractUnoccupiedSpace**
- **AbstractLogicalSpace**
- **AbstractSpaceBoundary** (relief)

8.1.4. Space Concept

All city objects are differentiated into [spaces](#) and [space boundaries](#).

[Spaces](#) are entities of volumetric extent in the real world. Buildings, water bodies, trees, rooms, and traffic spaces, for instance, have a volumetric extent. Spaces can be classified into physical spaces and logical spaces. Physical spaces, in turn, can be further classified into occupied spaces and unoccupied spaces.

[Space boundaries](#), in contrast, are entities with areal extent in the real world. Space boundaries can be differentiated into different types of thematic surfaces, such as wall surfaces and roof surfaces.

The term **Boundary Surface** is used in the CityGML 3.0 Standard but it is not defined. It is best understood as an artifact of the B_Rep model ([Foley et al. 2002](#)).

The [AbstractSpace](#) class has a [boundary](#) association (property) with the [AbstractSpaceBoundary](#) class. [AbstractSpaceBoundary](#) is specialized into [thematic surfaces](#). It is the subclassing of the [thematic surfaces](#) which gives semantics to the CityGML surfaces. Note that a [Closure Surface](#) is also a thematic surface. so a solid can be fully B_Rep enclosed even if part of the surface does not physically exist.

Consider that the [AbstractSpaceBoundary](#) is a Multi-Surface. So it stands to reason that a [space boundary](#) is an [AbstractSpaceBoundary](#) which is composed of one or more [Boundary Surfaces](#). Per the B_Rep concept, those [Boundary Surfaces](#) must completely enclose the Abstract Space. This may include the addition of surfaces both internal (but visible) as well as external. Hence, the distinction between internal and external space is moot. Internal surfaces are just a continuation of the space boundary around the solid.

[Table 4](#) lists the surfaces that are allowed as thematic surface boundaries of the space classes defined in the Core module:

Table 4. Core space classes and their allowed thematic surface boundaries

Space class	Allowed space boundaries
AbstractLogicalSpace	<ul style="list-style-type: none"> Core::AbstractSpaceBoundary and the subclasses: Core::AbstractThematicSurface, Core::ClosureSurface Generics::GenericThematicSurface possible classes from ADEs
AbstractOccupiedSpace	<ul style="list-style-type: none"> Core::AbstractSpaceBoundary and the subclasses: Core::AbstractThematicSurface, Core::ClosureSurface Generics::GenericThematicSurface possible classes from ADEs
AbstractPhysicalSpace	<ul style="list-style-type: none"> Core::AbstractSpaceBoundary and the subclasses: Core::AbstractThematicSurface, Core::ClosureSurface Generics::GenericThematicSurface possible classes from ADEs
AbstractSpace	<ul style="list-style-type: none"> Core::AbstractSpaceBoundary and the subclasses: Core::AbstractThematicSurface, Core::ClosureSurface Generics::GenericThematicSurface possible classes from ADEs

AbstractUnoccupiedSpace	<ul style="list-style-type: none"> • Core::AbstractSpaceBoundary and the subclasses: Core::AbstractThematicSurface, Core::ClosureSurface • Generics::GenericThematicSurface • possible classes from ADEs
-------------------------	---

A detailed introduction to the Space concept can be found in [Spaces and Space Boundaries](#).

In particular, the classification into OccupiedSpace and UnoccupiedSpace might not always be apparent at first sight. Carports, for instance, represent an OccupiedSpace, although they are not closed and most of the space is free of matter, see [Figure 14](#). Since a carport is a roofed, immovable structure with the purpose of providing shelter to objects (i.e. cars), carports are frequently represented as buildings in cadastres. Thus, also in CityGML, a carport should be modelled as an instance of the class Building. Since Building is transitively a subclass of OccupiedSpace, a carport is an OccupiedSpace as well. However, only in LOD1, the entire volumetric region covered by the carport would be considered as physically occupied. In LOD1, the occupied space is defined by the entire carport solid (unless a room would be defined in LOD1 that would model the unoccupied part below the roof); whereas in LOD2 and LOD3, the solids represent more realistically the really physically occupied space of the carport. In addition, for all OccupiedSpaces, the normal vectors of the thematic surfaces like the RoofSurface need to point away from the solids, i.e. consistent with the solid geometry.

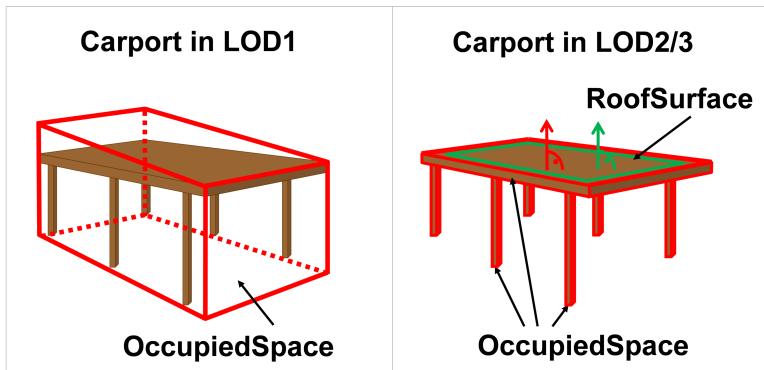


Figure 20. Representation of a carport as OccupiedSpace in different LODs. The red boxes represent solids, the green area represents a surface. In addition, the normal vectors of the roof solid (in red) and the roof surface (in green) are shown.

In contrast, a room is a physically unoccupied space. In CityGML, a room is represented by the class BuildingRoom that is a subclass of UnoccupiedSpace. In LOD1, the entire room solid would be considered as unoccupied space, which can contain furniture and installations, though, as is shown in [Figure 15](#). In LOD2 and 3, the solid represents more realistically the really physically unoccupied space of the room (possibly somewhat generalized as indicated in the figure). For all UnoccupiedSpaces, the normal vectors of the bounding thematic surfaces like the InteriorWallSurface need to point inside the object, i.e. opposite to the solid geometry.

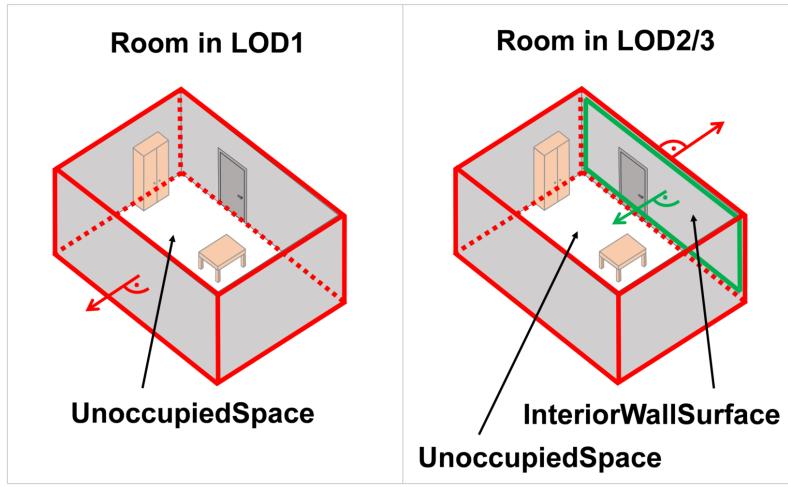


Figure 21. Representation of a room as *UnoccupiedSpace* in different LODs. The red boxes represent solids, the green area represents a surface. In addition, the normal vectors of the room solid (in red) and the wall surface (in green) are shown.

The UML diagram of the Space concept classes is depicted in Figure 16].

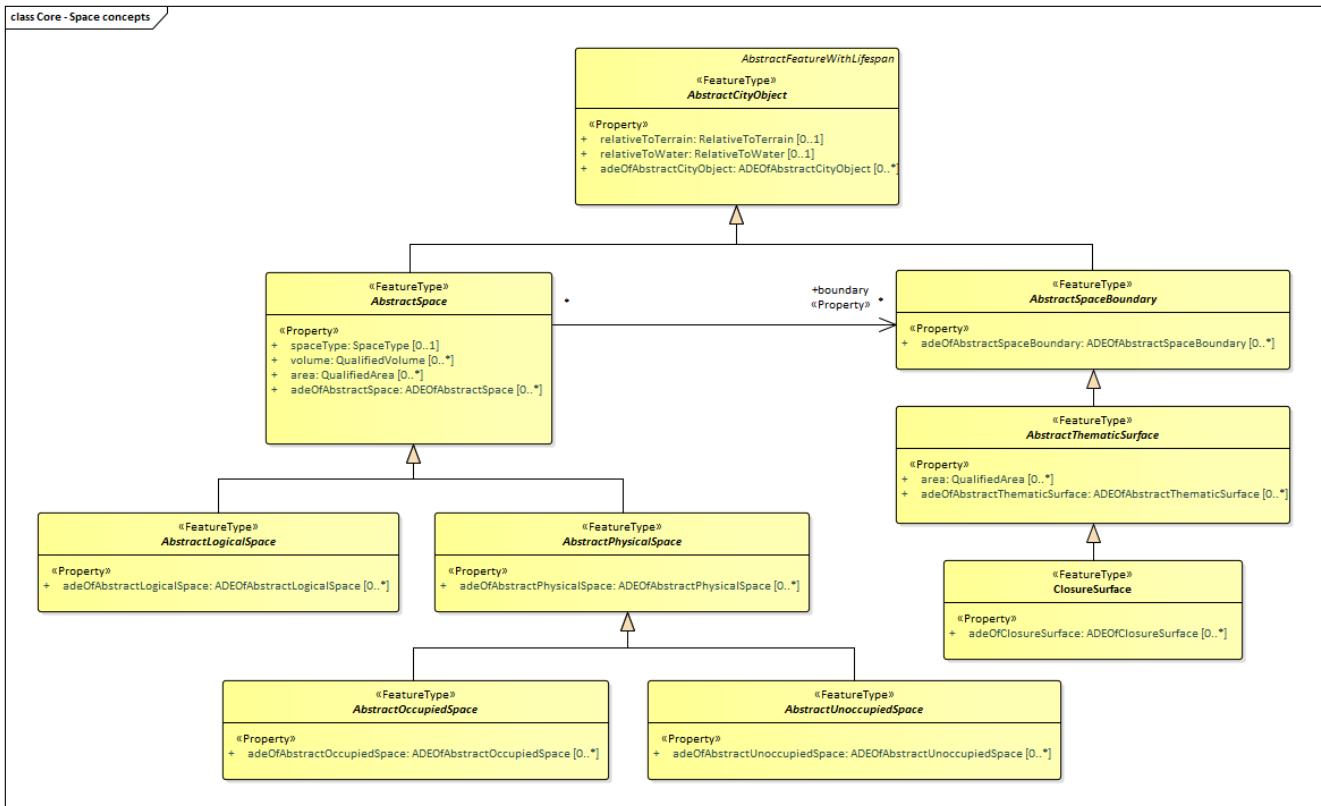


Figure 22. UML Space Concepts

The Space Concept classes defined in the CityGML UML model are documented in the [Data Dictionary](#).

8.1.5. Geometry and LOD

Spaces and space boundaries can have various geometry representations depending on the Levels of Detail (LOD). Spaces can be spatially represented as single points in LOD0, multi-surfaces in LOD0/2/3, solids in LOD1/2/3, and multi-curves in LOD2/3. Space boundaries can be represented as multi-surfaces in LOD0/2/3 and as multi-curves in LOD2/3. All Levels of Detail allow for the

representation of the interior of city objects.

The different Levels of Detail are defined in the following way:

- LOD 0: Volumetric real-world objects (Spaces) can be spatially represented by a single point, by a set of curves, or by a set of surfaces. Areal real-world objects (Space Boundaries) can be spatially represented in LOD0 by a set of curves or a set of surfaces. LOD0 surface representations are typically the result of a projection of the shape of a volumetric object onto a plane parallel to the ground, hence, representing a footprint (e.g. a building footprint or a floor plan of the rooms inside a building). LOD0 curve representations are either the result of a projection of the shape of a vertical surface (e.g. a wall surface) onto a grounding plane or the skeleton of a volumetric shape of longitudinal extent such as a road or river segment.
- LOD 1: Volumetric real-world objects (Spaces) are spatially represented by a vertical extrusion solid, i.e. a solid created from a horizontal footprint by vertical extrusion. Areal real-world objects (Space Boundaries) can be spatially represented in LOD1 by a set of horizontal or vertical surfaces.
- LOD 2: Volumetric real-world objects (Spaces) can be spatially represented by a set of curves, a set of surfaces, or a single solid geometry. Areal real-world objects (Space Boundaries) can be spatially represented in LOD2 by a set of surfaces. The shape of the real-world object is generalized in LOD2 and smaller details (e.g. bulges, dents, sills, but also structures like e.g. balconies or dormers of buildings) are typically neglected. LOD2 curve representations are skeletons of volumetric shapes of longitudinal extent like an antenna or a chimney.
- LOD 3: Volumetric real-world objects (Spaces) can be spatially represented by a set of curves, a set of surfaces, or a single solid geometry. Areal real-world objects (Space Boundaries) can be spatially represented in LOD3 by a set of surfaces. LOD3 is the highest level of detail and respective geometries include all available shape details.

In addition, the geometry can also be represented implicitly. The shape is stored only once as a prototypical geometry, which then is re-used or referenced, wherever the corresponding feature occurs in the 3D city model.

The thematic classes, such as building, tunnel, road, land use, water body, or city furniture are defined as subclasses of the space and space boundary classes within the thematic modules. Since all city objects in the thematic modules represent subclasses of the space and space boundary classes, they automatically inherit the geometries defined in the Core module.

The UML diagram of the Geometry and LoD concept classes is depicted in [Figure 17](#).

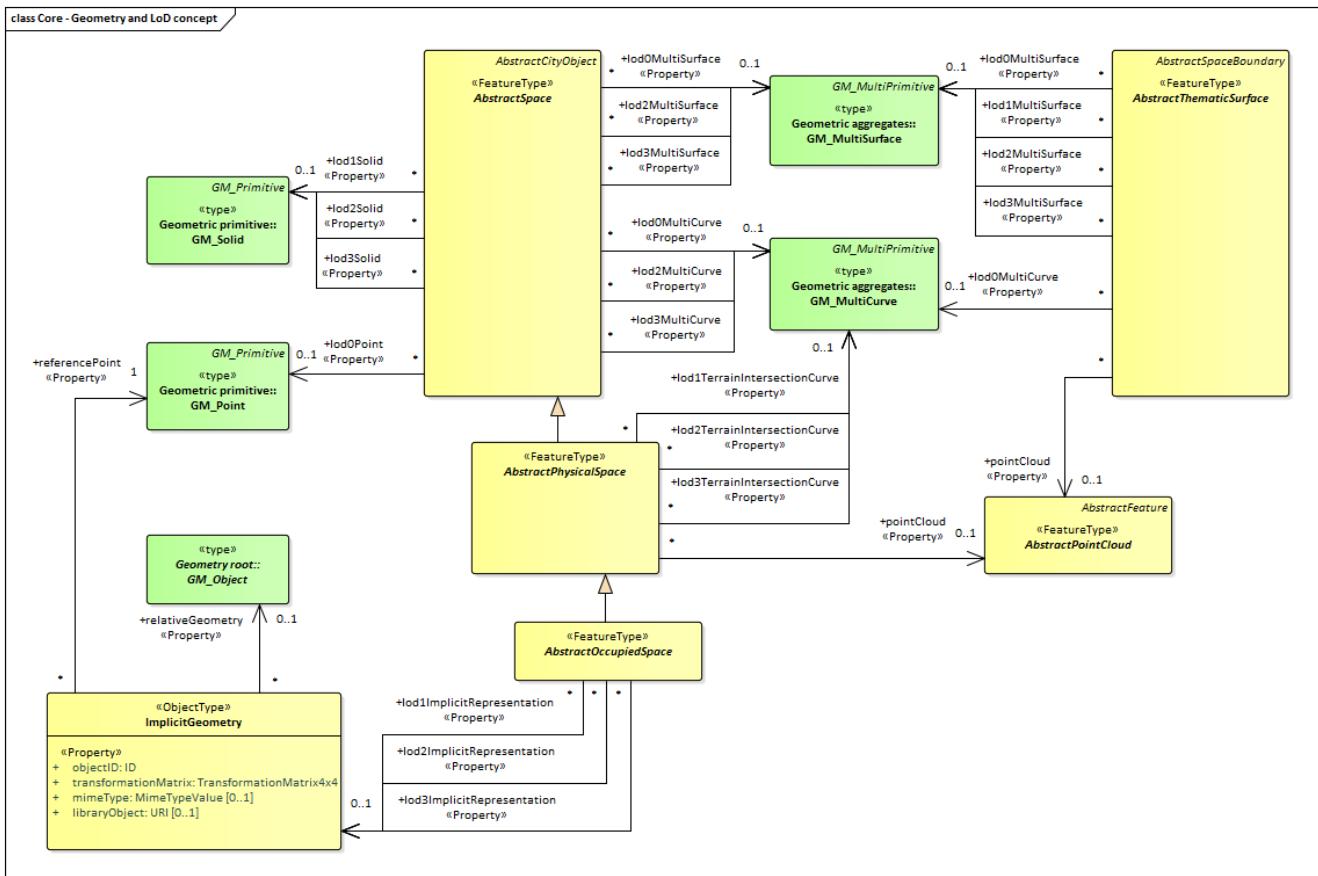


Figure 23. UML Geometry and LOD Concepts

The Geometry and LOD Concept classes defined in the CityGML UML model and documented in the [Data Dictionary](#).

Of particular note is the Implicit Geometry concept. Many of the objects encountered in a city landscape have the same geometry. How many types of street lamps can there be? An Implicit Geometry captures that geometry once, and re-uses that one geometry for all similar street lamp objects.

8.1.6. CityGML Core Model

The [City Model and City Object](#) classes, the [Space Concept](#) classes, and the [Geometry and LOD](#) classes define the majority of the CityGML Core module. In addition to these concepts, the Core module also specifies that city objects can have relations to other city objects and that they can have address information. All other modules defined in the CityGML model refer to the Core module.

The UML diagram of the complete Core module is depicted in [Figure 18](#).

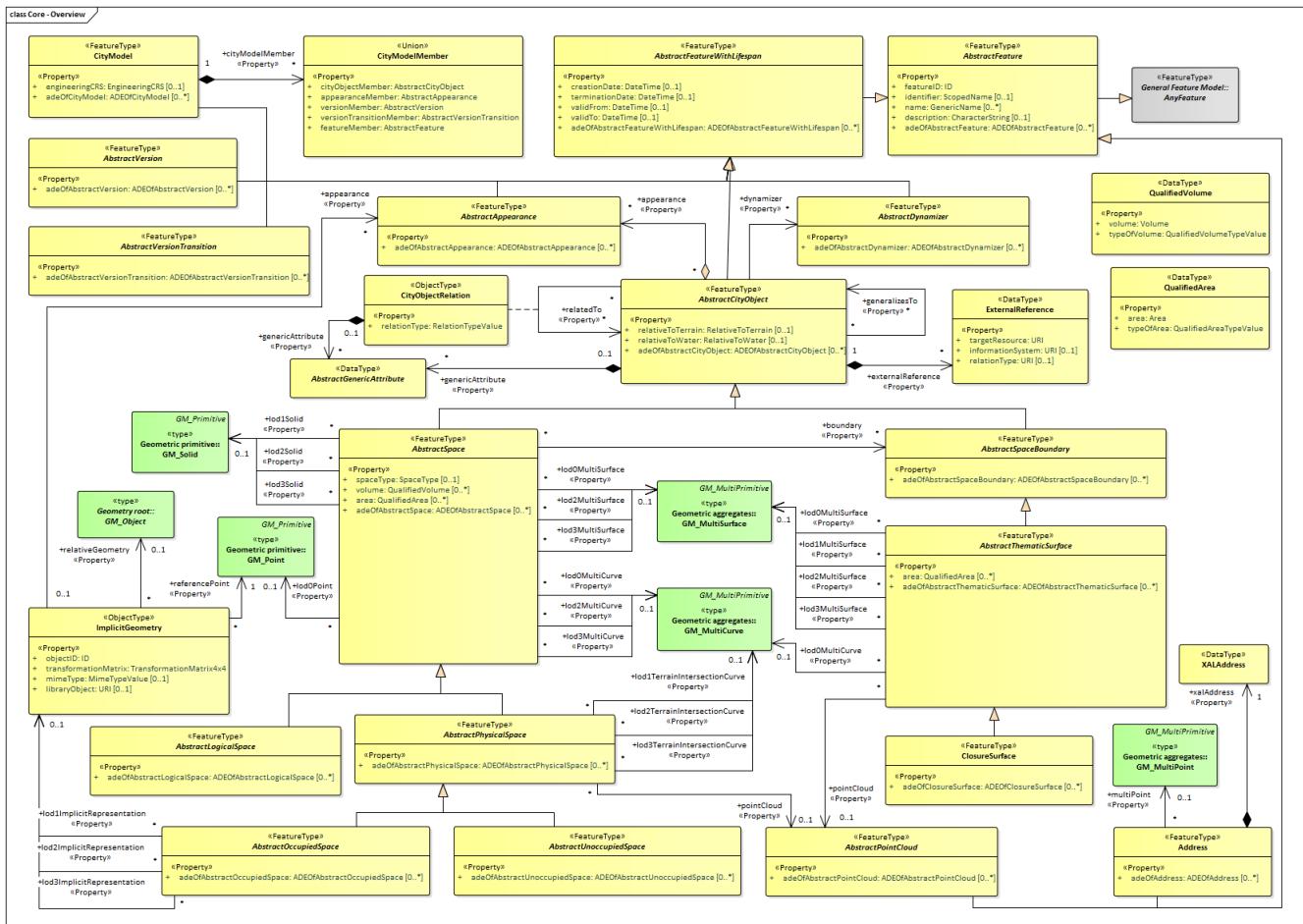


Figure 24. UML diagram of CityGML's core module.

The most important of the Core classes have been introduced already in the [Key Concepts](#) Section. More details about these classes can be found in the [Data Dictionary](#).

8.1.7. Data types, Enumerations, and Code lists

The ADE data types provided for in the Core module are illustrated in [Figure 19](#).

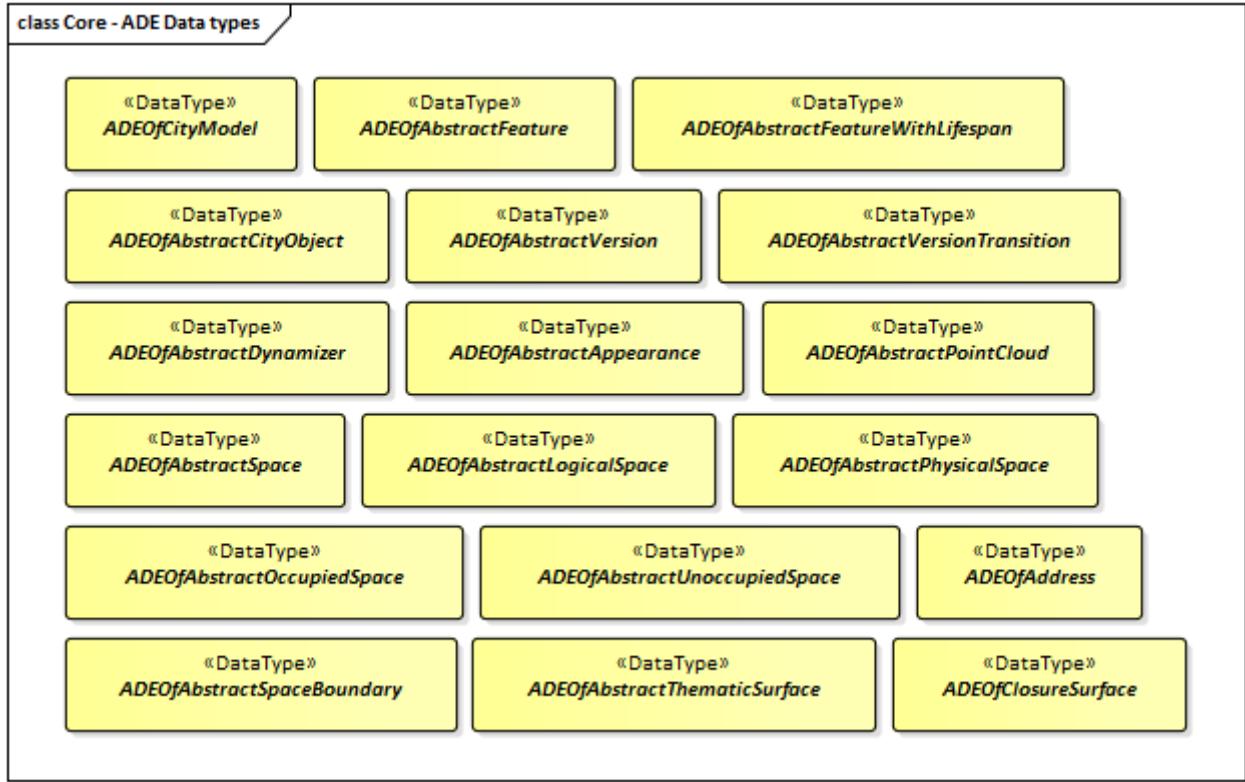


Figure 25. ADE classes of the CityGML Core module.

The Data Types, Basic Types, Enumerations, Unions, and Code Lists provided for the Core module are illustrated in Figure 20.

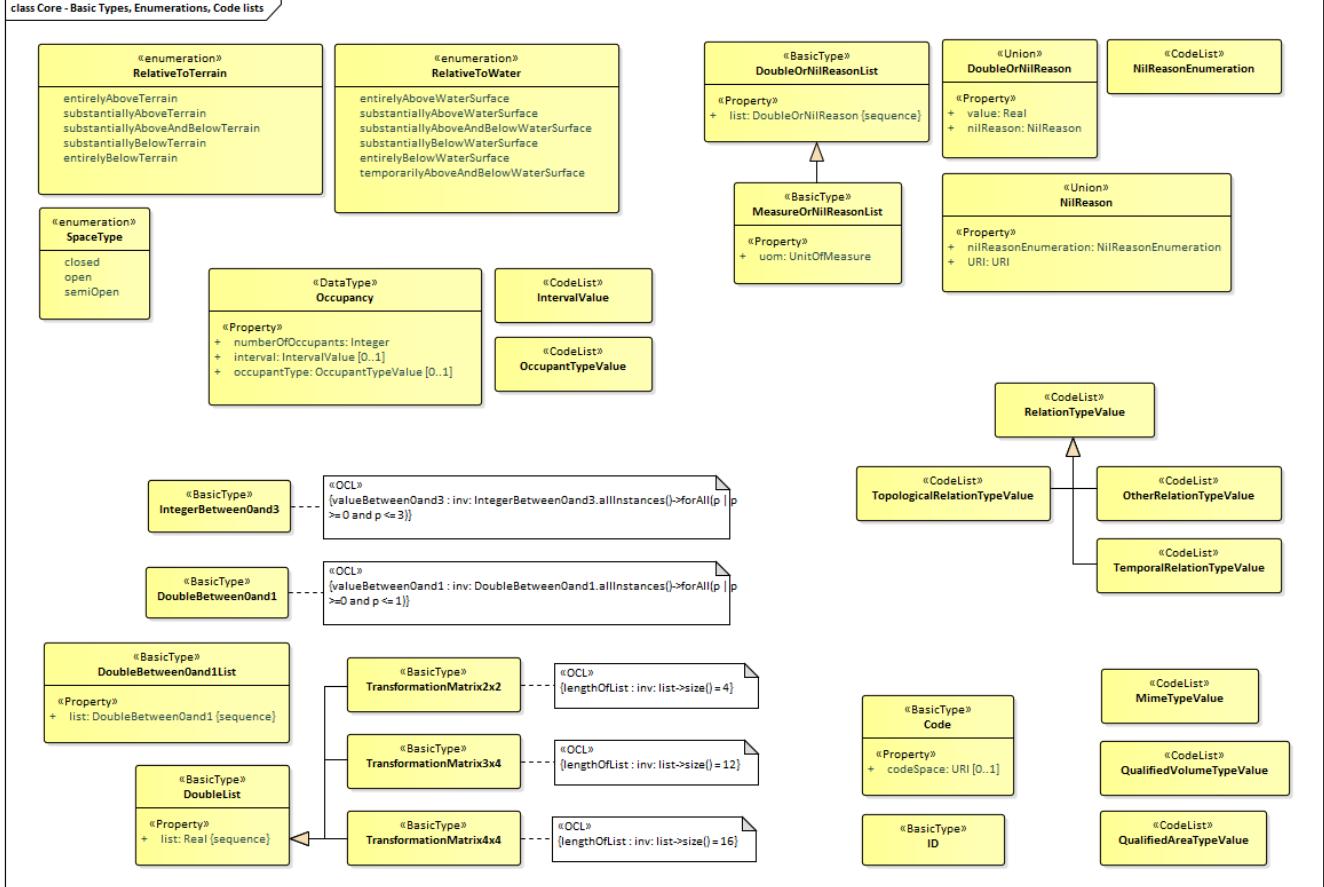


Figure 26. Basic Types, Enumerations, and Codelists from the CityGML Core module.

8.2. Appearance

8.2.1. Synopsis

The Appearance module provides the representation of surface data such as observable properties for surface geometry objects in the form of textures and material.

Appearances are not limited to visual data but represent arbitrary categories called themes such as infrared radiation, noise pollution, or earthquake-induced structural stress. 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).

Surface data that is constant across a surface is modelled as material based on the material definitions from the X3D and COLLADA standards. Surface data that depends on the exact location within the surface is modelled as a texture. This can either be a parameterized texture (a texture that uses texture coordinates) or a transformation matrix for parameterization, or a georeferenced texture (a texture that uses a planimetric projection).

Each surface geometry object can have both, a material and a texture per theme and side. This allows for providing a constant approximation and a complex measurement of a surface's property simultaneously.

8.2.2. Key Concepts

Appearance: An Appearance is a collection of surface data, i.e. observable properties for surface geometry objects in the form of textures and material.

A type of [AbstractAppearance](#).

AbstractSurfaceData: AbstractSurfaceData is the abstract superclass for different kinds of textures and material. Textures and materials are the primitives used to render 3D objects.

A type of [AbstractFeature](#).

X3DMaterial: X3DMaterial defines properties for surface geometry objects based on the material definitions from the standards X3D and COLLADA.

A type of [AbstractSurfaceData](#).

AbstractTexture: AbstractTexture is the abstract superclass to represent the common attributes of CityGML textures.

A type of [AbstractSurfaceData](#).

ParameterizedTexture: A ParameterizedTexture is a texture that uses texture coordinates or a transformation matrix for parameterization.

A type of [AbstractTexture](#).

GeoreferencedTexture: A GeoreferencedTexture is a texture that uses a planimetric projection. It contains an implicit parameterization that is either stored within the image file, an accompanying world file or specified using the orientation and referencePoint elements.

A type of [AbstractTexture](#).

8.2.3. Discussion

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 Surface Data ([AbstractSurfaceData](#)) objects. A single [Surface Data](#) object may have surface data for multiple themes. Similarly, surface data can be shared by multiple [City Objects](#). Finally, surface data values can either be constant across a surface or depend on the exact location within the surface.

A constant surface property is modelled as [material](#). A surface property, which depends on the location within the surface, is modelled as [texture](#). Each [City 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 representation for its task (e.g. analysis or rendering). A specific mixing is not defined since this is beyond the scope of CityGML. If a [City 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.

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 [AbstractSurfaceData](#) and its descendants. The relation between a [City Object](#) and [Surface Data](#) objects is expressed by the [appearance](#) relation from a [City Object](#) to an object of type [Surface Data](#).

8.2.4. 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.

The colors of a material can have three different forms:

- `diffuseColor`: the color of diffusely reflected light.
- `specularColor`: the color of a directed reflection.
- `emissiveColor`: 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](#) property where 0 stands for fully opaque and 1 for fully transparent.

The [ambientIntensity](#) property defines the minimum percentage of [diffuseColor](#) that is visible regardless of light sources.

The [shininess](#) property controls the sharpness of the specular highlight. 0 produces a soft glow

while 1 results in a sharp highlight.

The `isSmooth` property 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).

8.2.5. Texture and texture mapping

The abstract base class for textures is `AbstractTexture`. Textures in CityGML are always raster-based 2D textures. The raster image is specified by the `imageURI` property. This property is an URI which references an arbitrary image data resource, or even a preformatted 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. The `wrapMode` property can have one of five values (Figure 21 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.

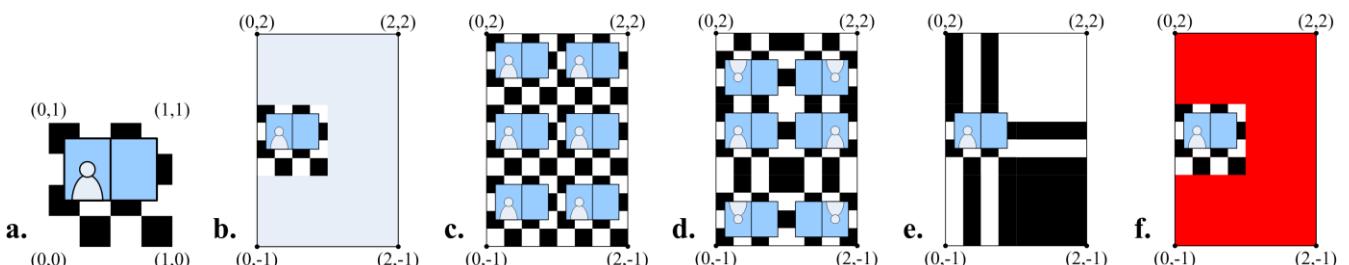


Figure 27. A texture (a) applied to a facade using different wrap modes: (b) none, (c) wrap, (d) mirror, (e) clamp and (f) border. The border color is red. The numbers denote texture coordinates (image: Hasso-Plattner-Institute).

Georeferenced Texture

`AbstractTexture` is further specialised according to the texture parameterisation, i.e. the mapping function from a location 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 coordinate conversion). The mapping function must be known for each surface geometry object to receive texture.

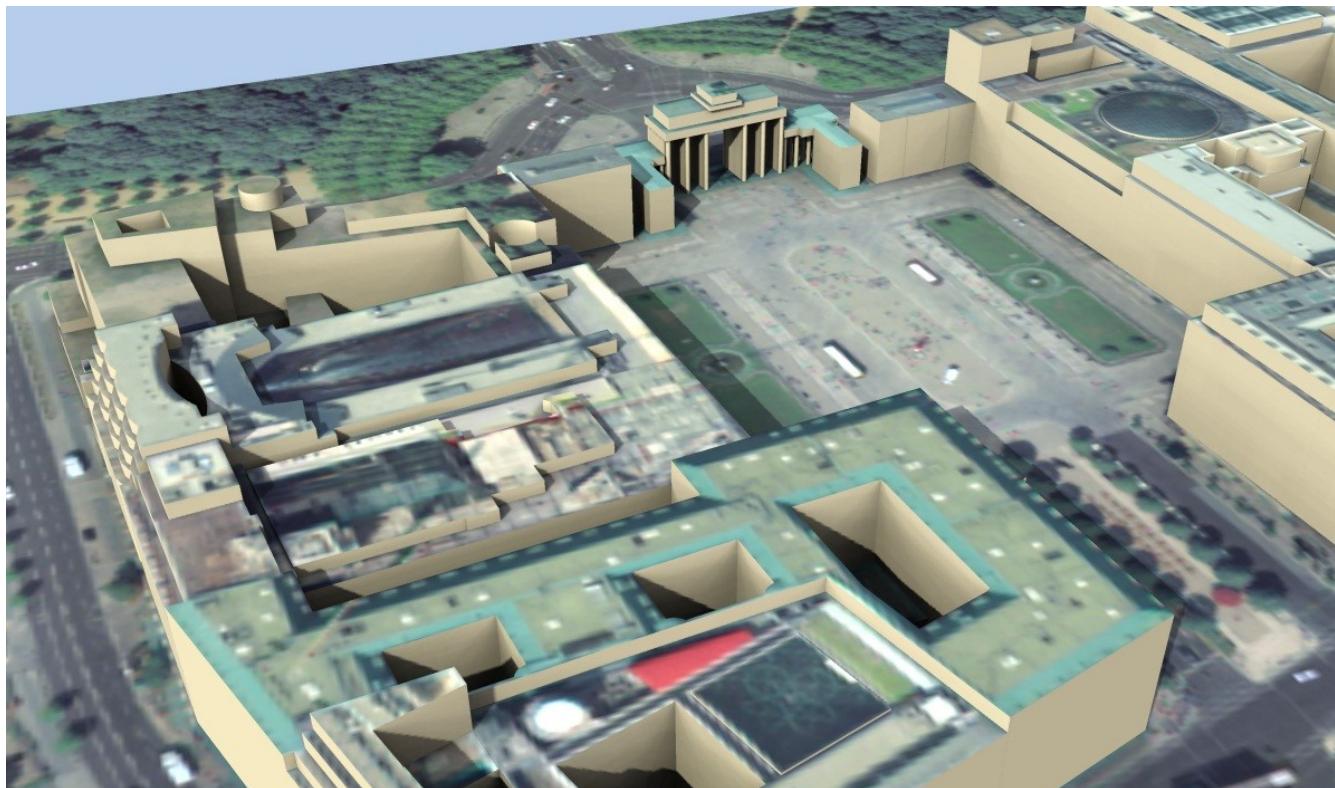


Figure 28. A georeferenced texture applied to ground and roof surfaces (source: Senate of Berlin, Hasso-Plattner-Institute).

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) of an ESRI World file. 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. The property [referencePoint](#) identifies the [GM_Point](#) which represents the location of the center of the upper left image pixel in world space.

If neither an internal nor an external georeference is given the [GeoreferencedTexture](#) is invalid.

Parameterized Texture

The class [ParameterizedTexture](#) describes a texture with target-dependent mapping function.

The target of each mapping function is a subclass of class [AbstractTextureParameterization](#). The [textureParameterization](#) property associates an [AbstractTextureParameterization](#) object with the the [ParameterizedTexture](#) object within which it is contained.

The mapping function is performed through a [TextureAssociation](#) object. This object is a UML

Association Class object which is associated with each `textureParameterization` property. The `TextureAssociation` class contains one property which is a URI identifying a `surface geometry`. The `AbstractTextureParameterization` object is selected based on the URI value in the corresponding `TextureAssociation` object.

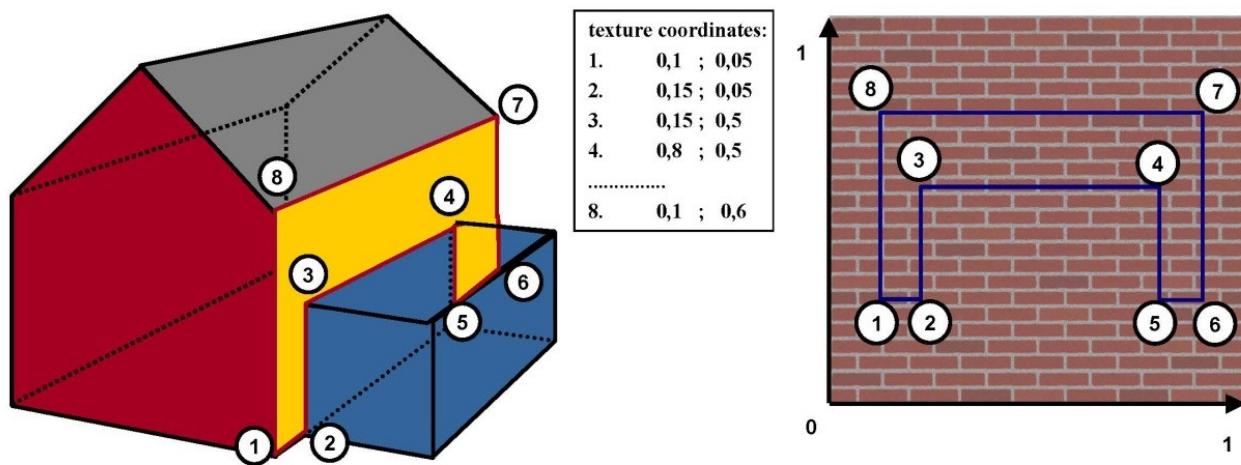


Figure 29. Positioning of textures using texture coordinates (image: IGG Uni Bonn).

Texture coordinates are applicable only to polygonal surfaces, whose boundaries are described by `GM_MultiSurface` or `GM_MultiCurve`. 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.

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 `GM_MultiSurface` or `GM_MultiCurve` composing the boundary of the target `surface geometry` object 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`.

NOTE Linear Rings are a GML construct. What do we replace it with?

`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 \square \square 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 Update the text between this and the previous note.





Figure 30. 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).

NOTE the rest of this section still needs updating from v 2.0

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. [Figure 24](#)). Texture coordinates $\mathbf{q}^T s, t$ are calculated from a space location $\mathbf{q}^T x, y, z$ as $\mathbf{q}^T \mathbf{q}^T T s, t \mathbf{q}^T \mathbf{q}^T$ with $\mathbf{q}^T \mathbf{q}^T T s, t, q^T M x, y, z, 1$. 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 **TexCoordGen** object uses the **crs** association to identify a **SC_CRS** object which defines 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^T , u^T , and d^T 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.

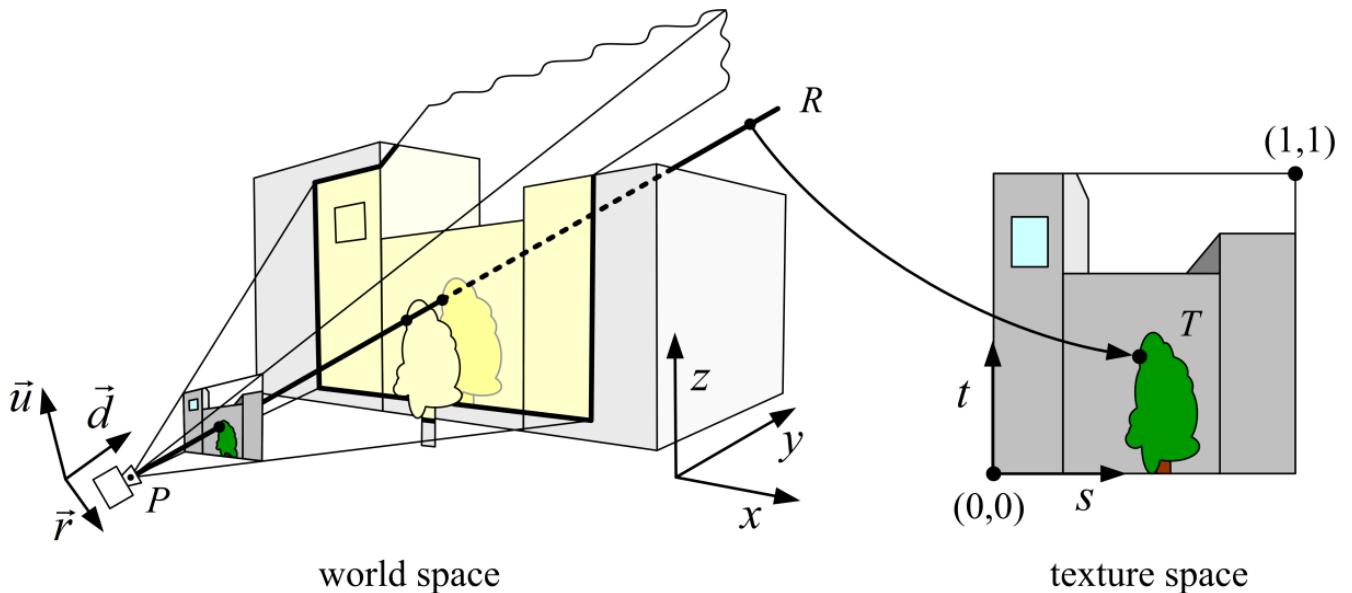


Figure 31. Projective texture mapping. All points on a ray R starting from the projection center P are mapped to the same point T in texture space (image: Hasso-Plattner-Institute, IGG TU Berlin).

Alternatively, if the 3×4 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">R</mi>
    <mo stretchy="false">T</mo>
  </mover>
</mrow>
<mo>*</mo>
<mrow class="MJX-TeXAtom-ORD">
  <mover>
    <mi>F</mi>
  </mover>
</mrow>
```

```

<mo stretchy="false">#x2192;</mo>
</mover>
<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 $n \perp$ 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 width_f and height_f 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

8.2.6. 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).

8.2.7. UML Model

The UML diagram of the Appearance module is illustrated in [UML diagram of CityGML's Appearance model..](#)

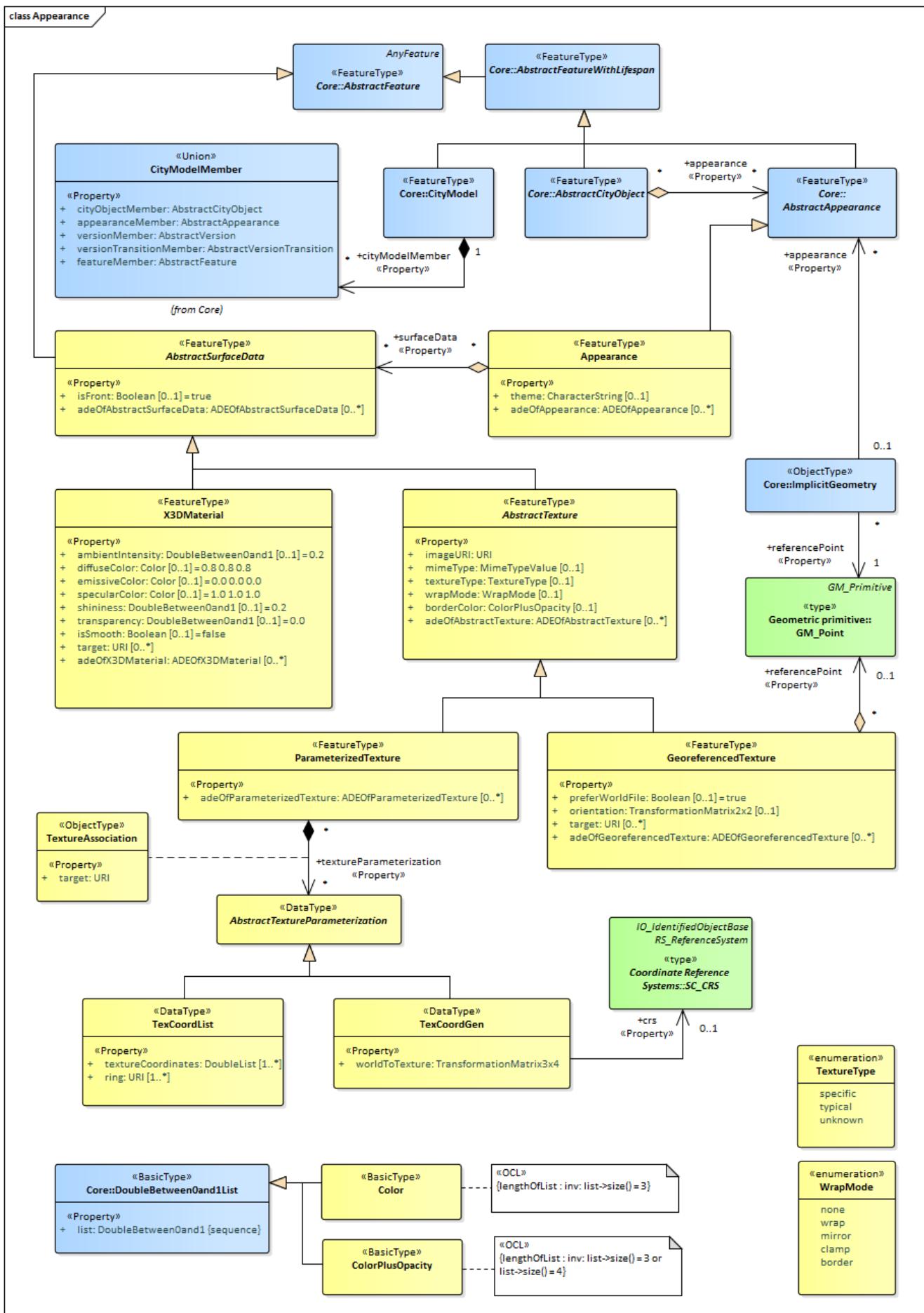


Figure 32. UML diagram of CityGML's Appearance model.

The ADE data types provided for the Appearance module are illustrated in ADE classes of the

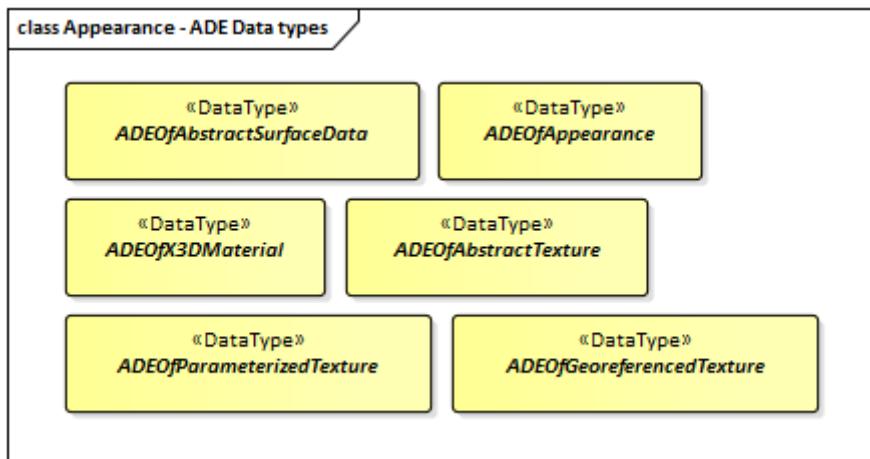


Figure 33. ADE classes of the CityGML Appearance Module.

8.2.8. Examples

8.3. Bridge Model

Contributors
Chuck Heazel - first draft

8.3.1. 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 – 3.

8.3.2. Key Concepts

8.3.3. Discussion

NOTE From CityGML 3.0

The Bridge module provides the representation of thematic and spatial aspects of bridges. Bridges are movable or unmovable structures that span intervening natural or built elements. In this way, bridges allow the passage of pedestrians, animals, vehicles, and service(s) above obstacles or between two points at a height above ground. Bridges are represented in the UML model by the top-level feature type *Bridge*, which is the main class of the Bridge module. Bridges can physically or functionally be subdivided into bridge parts. In addition, bridges can be decomposed into structural elements, such as pylons, anchorages, cables, slabs, and beams.

The free space inside bridges is represented by rooms, which allows a virtual accessibility of bridges. Bridges can contain installations and furniture. Installations are permanent parts of a bridge that strongly affect the outer or inner appearance of the bridge and that cannot be moved. Examples are stairways, signals, railings, and lamps. Furniture, in contrast, represent moveable objects of a bridge, like signs, art works, and benches. Bridges can be bounded by different types of

surfaces. In this way, the outer structure of bridges can be differentiated semantically into wall surfaces, roof surfaces, ground surfaces, outer floor surfaces, and outer ceiling surfaces, whereas the visible surface of rooms can be structured into interior wall surfaces, floor surfaces, and ceiling surfaces. Furthermore, the openings of bridges, i.e. windows and doors, can be represented including their corresponding surfaces.

NOTE The following text is from CityGML 2.0. It needs to be reviewed and updated.

NOTE Many of these concepts have been moved into the Construction module. They should be captured once there. The following text should reference those sections then provide any details specific to bridges.

The bridge model of CityGML is defined by the thematic extension module Bridge (cf. chapter 7). [Figure 26](#) illustrates examples of bridge models in all LODs.

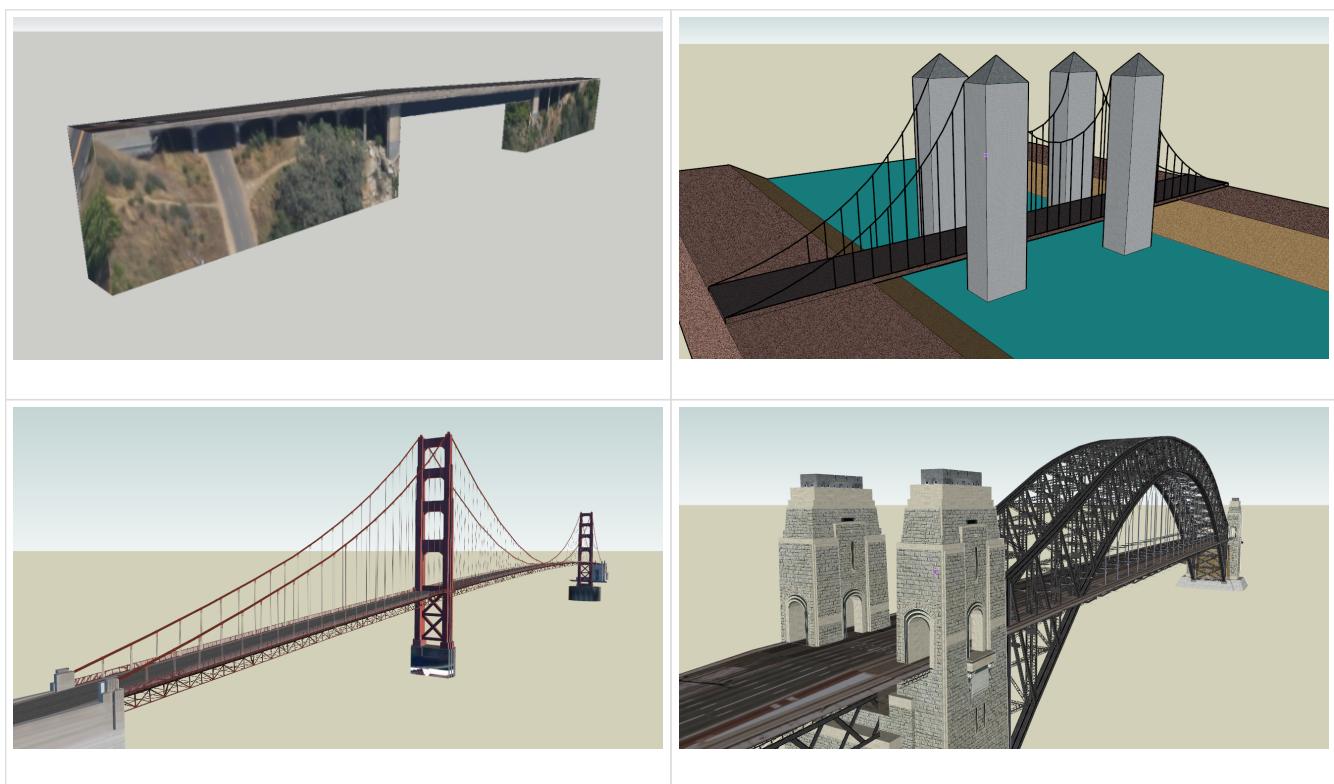


Figure 34. Examples for bridge models in LOD1 (upper left), LOD2 (upper right), LOD3 (lower left) and LOD4 (lower right) (source: Google 3D warehouse).

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 [UML diagram of the bridge model, part one](#).

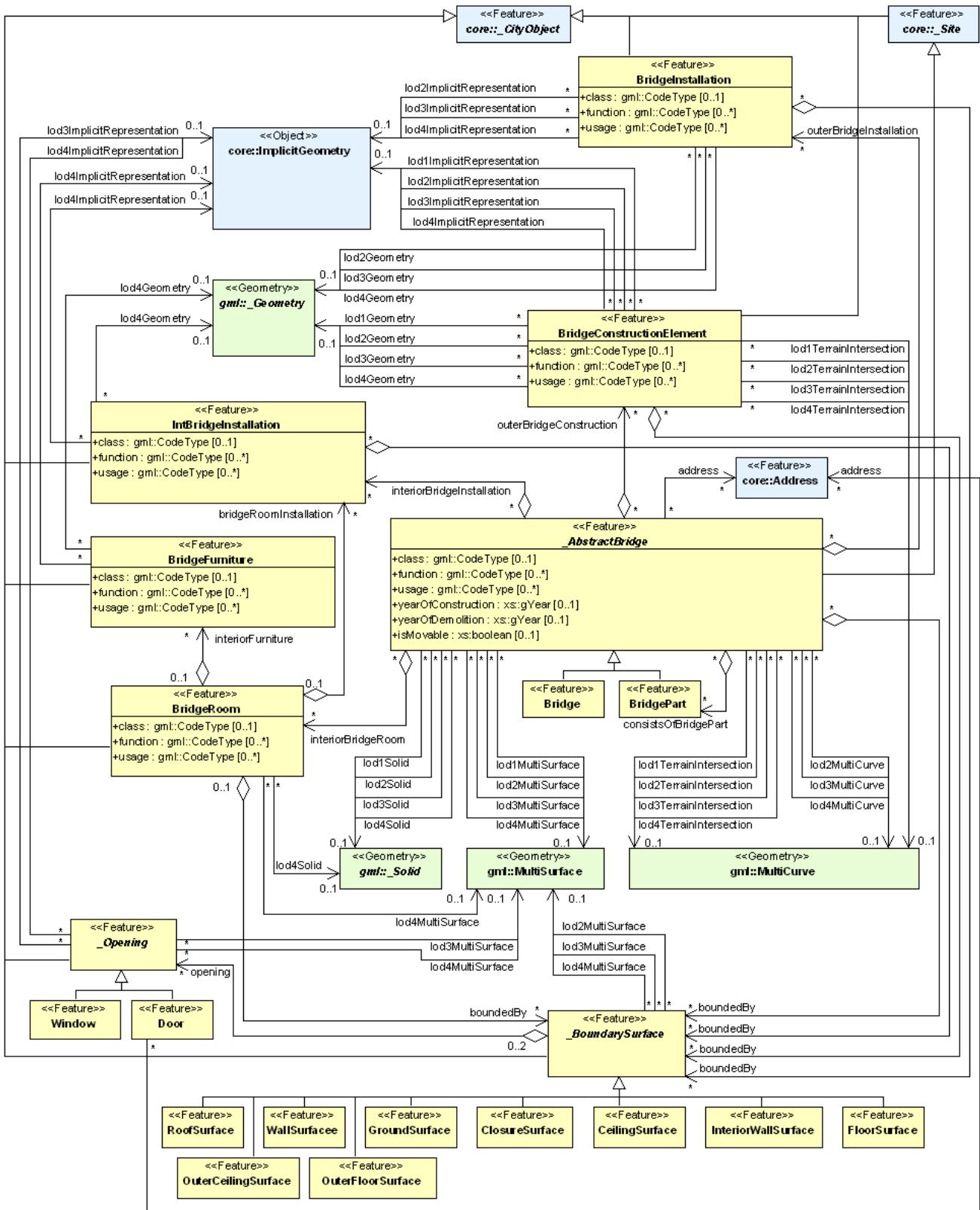


Figure 35. UML diagram of the bridge model, part one

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 BridgeInstallation or BridgeConstructionElement can be used (see section 10.5.2). All relations to semantic objects and geo-metric properties are listed in Tab. 7.

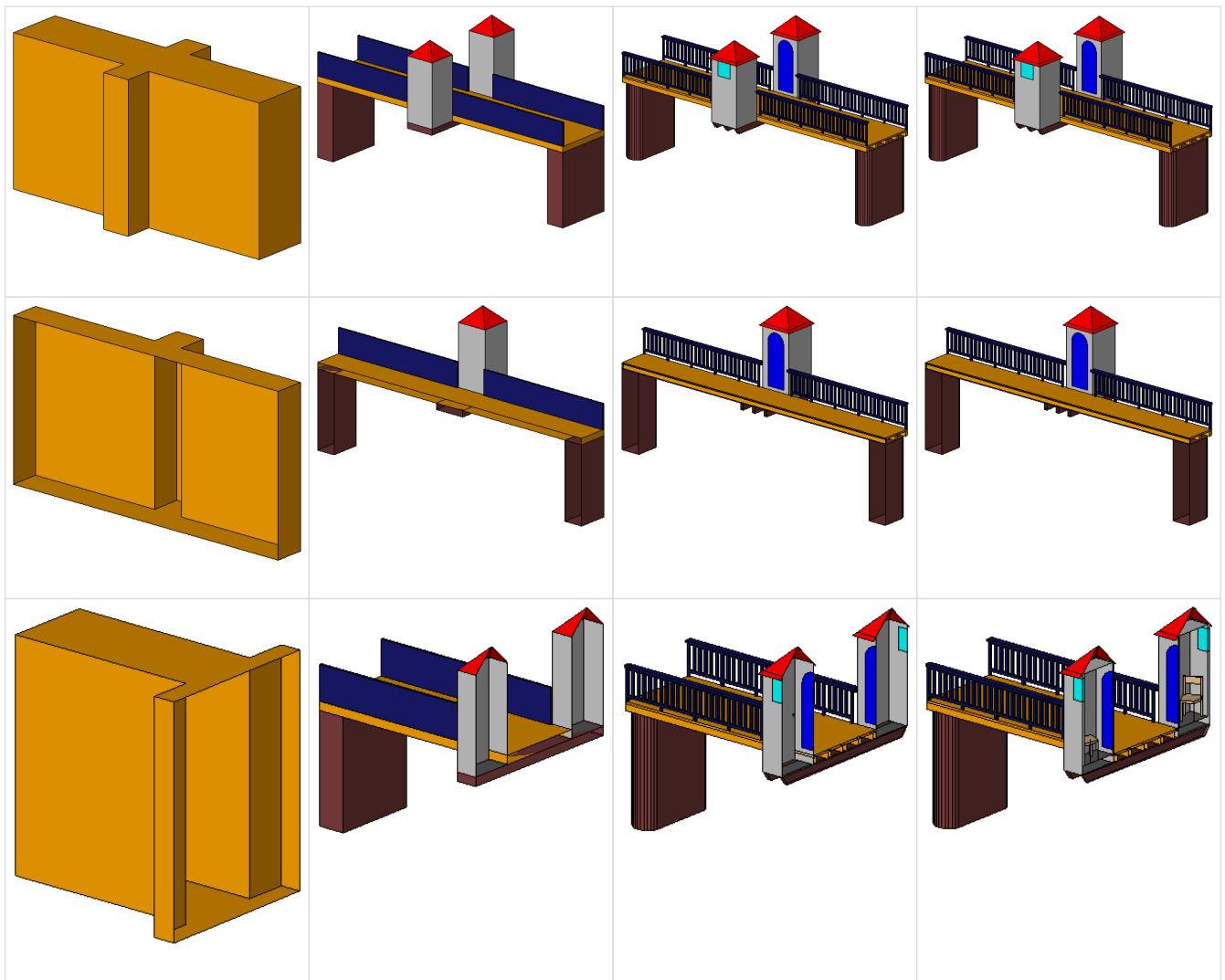


Figure 36. Bridge model in LOD1 – LOD4. (source: Karlsruhe Institute of Technology (KIT)).

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 5. 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		•	•	•
Bridge construction elements (chapter 10.5.2)	BridgeConstruction-ElementType	•	•	•	•
Openings (chapter 10.5.4)	AbstractOpeningType			•	•
Bridge rooms (chapter 10.5.5)	BridgeRoomType				•
Interior bridge installations	IntBridgeInstallationType				•

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 animated GIFs. Currently these do not render for PDF.

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

[[align="center"] | *figures/inwork/Figure_47_2.gif*

[[align="center"] | *figures/inwork/Figure_47_3.gif*

[[align="center"] | *figures/inwork/Figure_47_4.gif*

[[align="center"] | *figures/inwork/Figure_47_5.gif*

Figure 37. Examples for movable bridges (source: ISO 6707).

8.3.4. Bridge and bridge part

Bridge

BridgePart

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 attributes and relations. The relation consistOfBridgePart represents the aggregation hierarchy between a Bridge (or a BridgePart) and its 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.

AbstractBridge

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.

8.3.5. Bridge construction elements and bridge installations

BridgeConstructionElement

BridgeInstallation

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 ImplicitGeometry 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). BridgeInstallations occur in LOD 2 to 4 only and are geometrically represented as gml:_Geometry. Again, the concept of ImplicitGeometry can be applied to BridgeInstallations alternatively, and their visible surfaces can be semantically 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

8.3.6. Boundary surfaces

BoundarySurface (AbstractConstructionSurface?)

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 RoofSurface, 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 represented 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 IntBridgeInstallations 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.

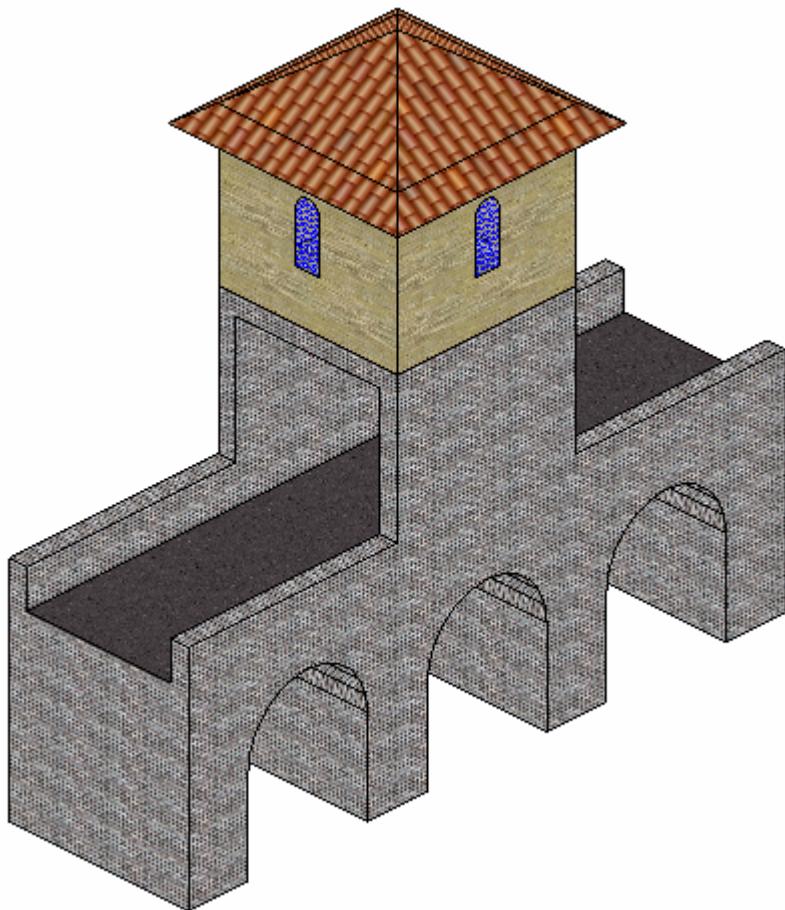


Figure 38. Different BoundarySurfaces of a bridge.

NOTE need to add annotations to Figure 50

GroundSurface

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 pointing downwards.

OuterCeilingSurface

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

WallSurface

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

OuterFloorSurface

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

RoofSurface

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

ClosureSurface

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.

FloorSurface

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

InteriorWallSurface

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

CeilingSurface

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

8.3.7. Openings

Opening

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 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).

Window

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.

Door

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).

8.3.8. 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.

BridgeRoom

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).

BridgeFurniture

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 coordinate system and referenced by other bridge furniture features (see chapter 8.2).

IntBridgeInstallation

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. Alternatively, 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 referenced 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).

8.3.9. UML Model

The UML diagram of the Bridge module is depicted in [UML diagram of the Bridge Model](#). The Bridge module inherits concepts from the Construction module (cf. [\[ug_construction_section\]](#)). The Construction module defines objects that are common to all types of construction, such as the different surface types and the openings.

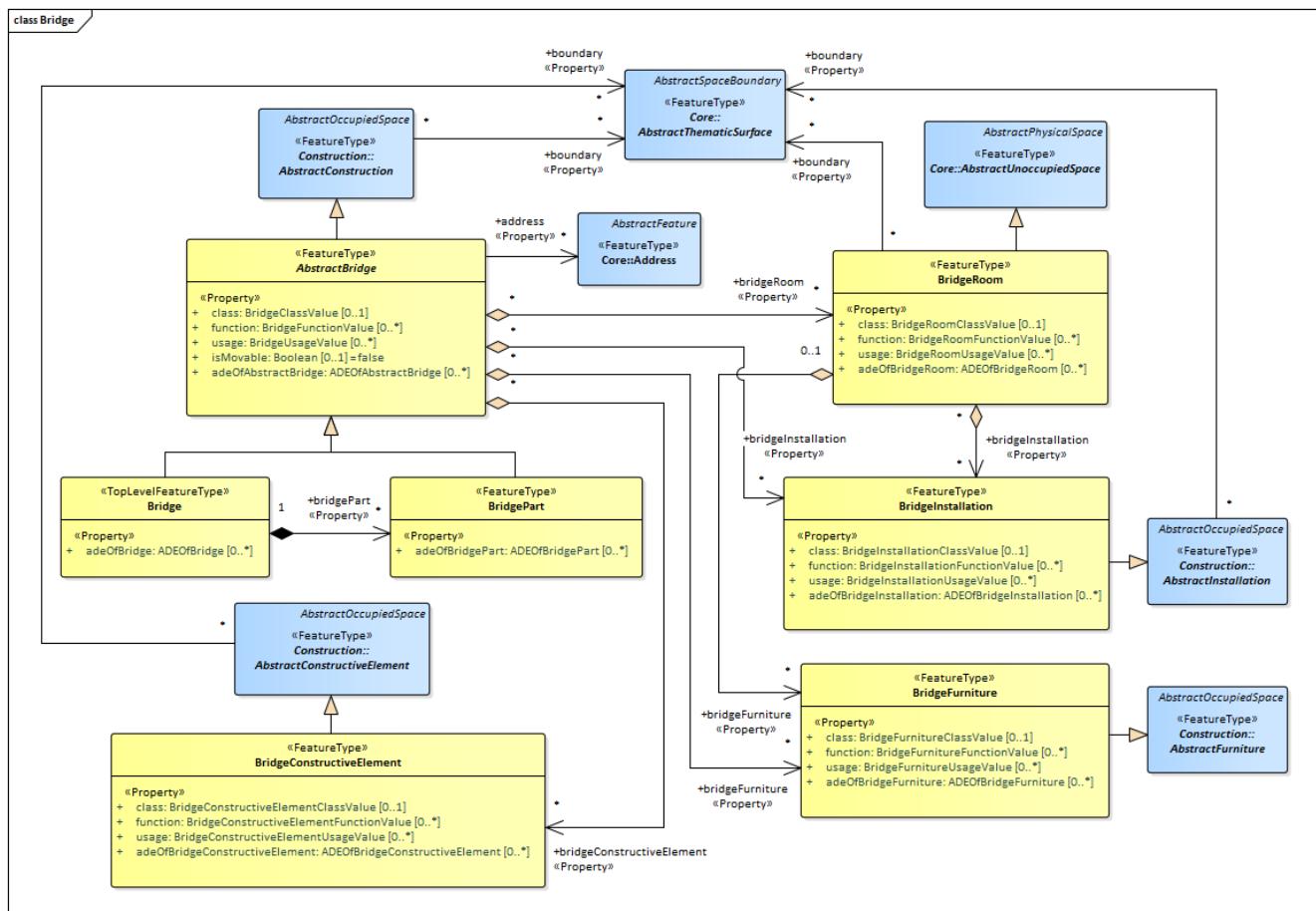


Figure 39. UML diagram of the Bridge Model.

The ADE data types provided for the Bridge module are illustrated in [ADE classes of the CityGML Bridge module..](#)

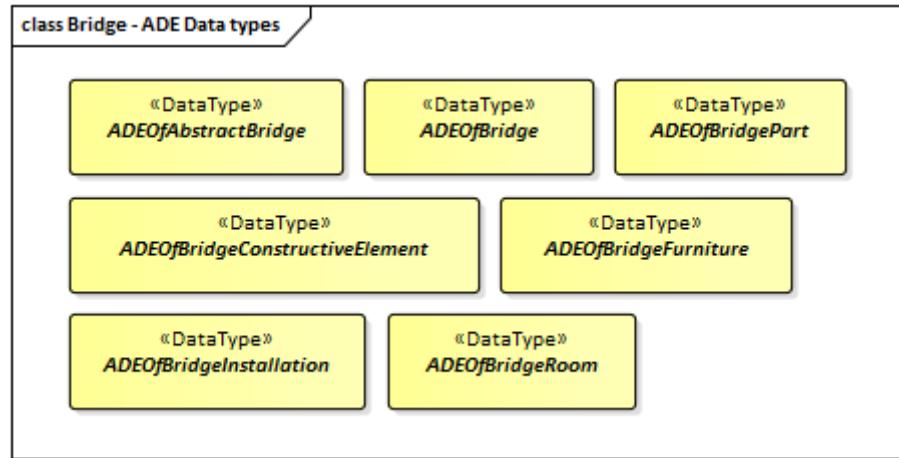


Figure 40. ADE classes of the CityGML Bridge module.

The Code Lists provided for the Bridge module are illustrated in [Codelists from the CityGML Bridge module..](#)

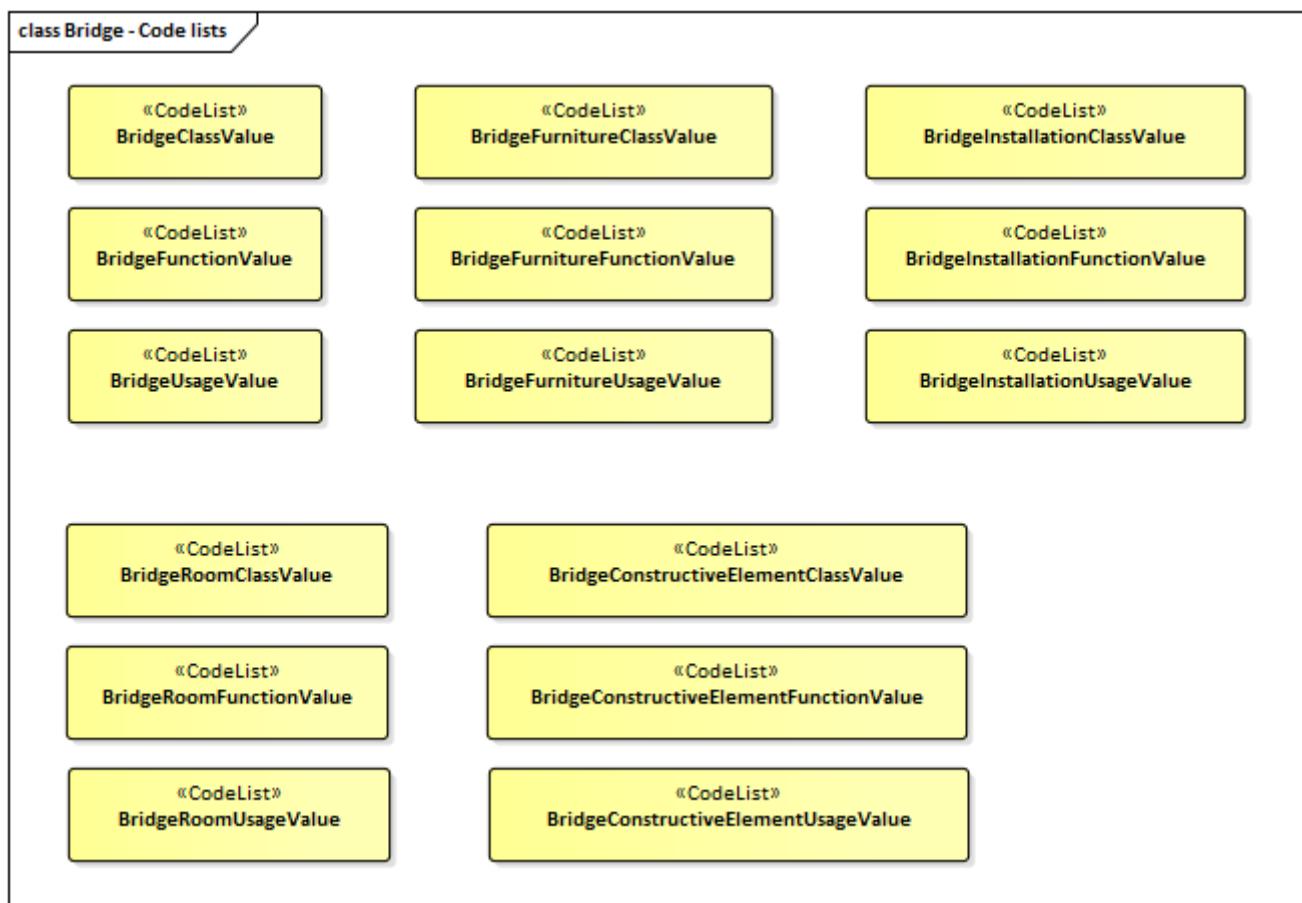


Figure 41. Codelists from the CityGML Bridge module.

8.3.10. 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).

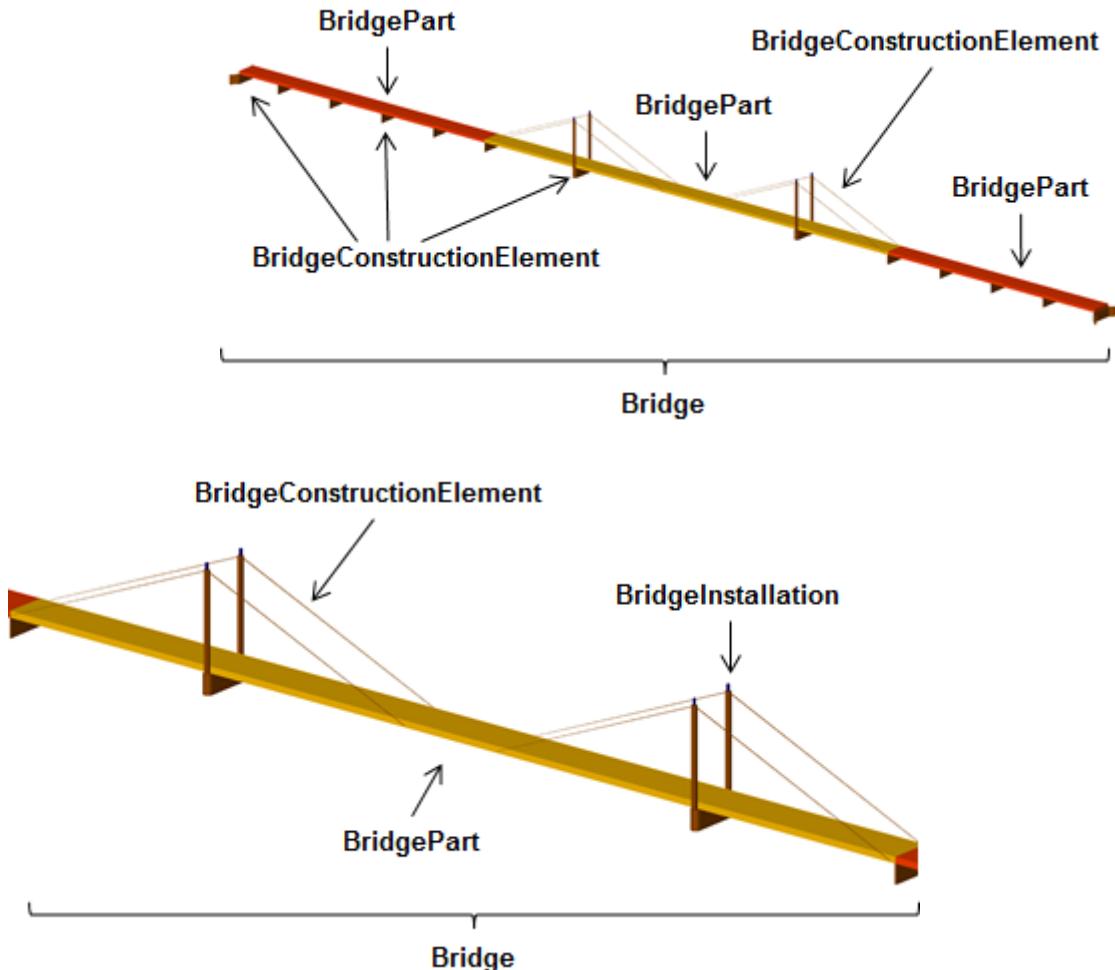


Figure 42. The bridge of Rees, consisting of a Bridge feature and three BridgePart features (left). The bridge contains BridgeConstructionElement and BridgeInstallation features (right).

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 right side (source: District of Recklinghausen / KIT).

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



Figure 43. 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).

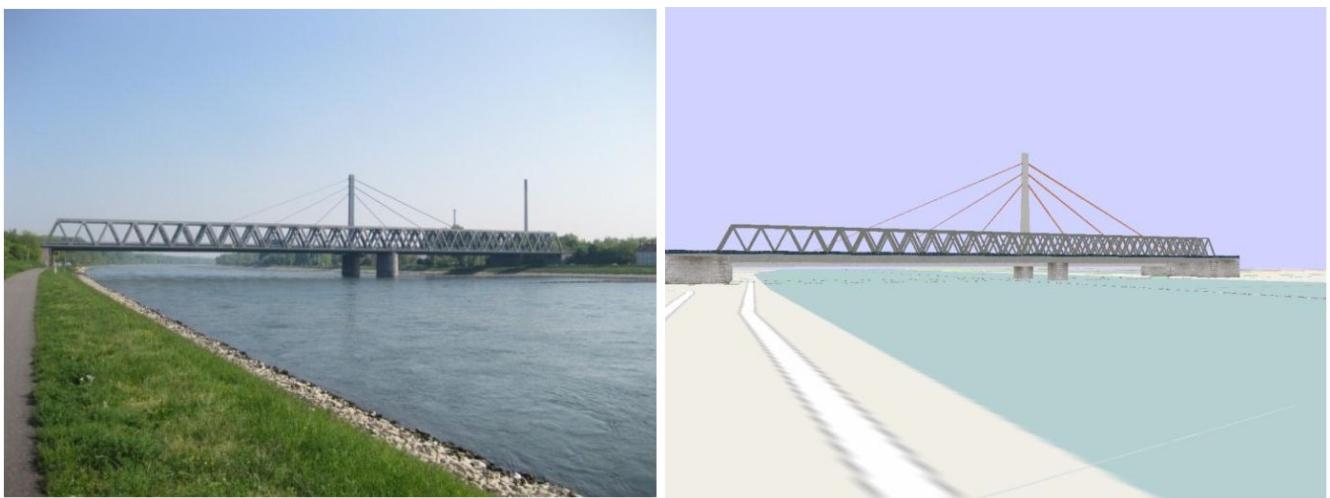


Figure 44. Bridge over the river Rhine at Karlsruhe (left a photo, right the 3D CityGML model) (source: Karlsruhe Institute of Technology (KIT), courtesy of City of Karlsruhe).

In CityGML both bridges are modeled as single Bridge objects with BridgeConstructionElements and BridgeInstallations. 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.

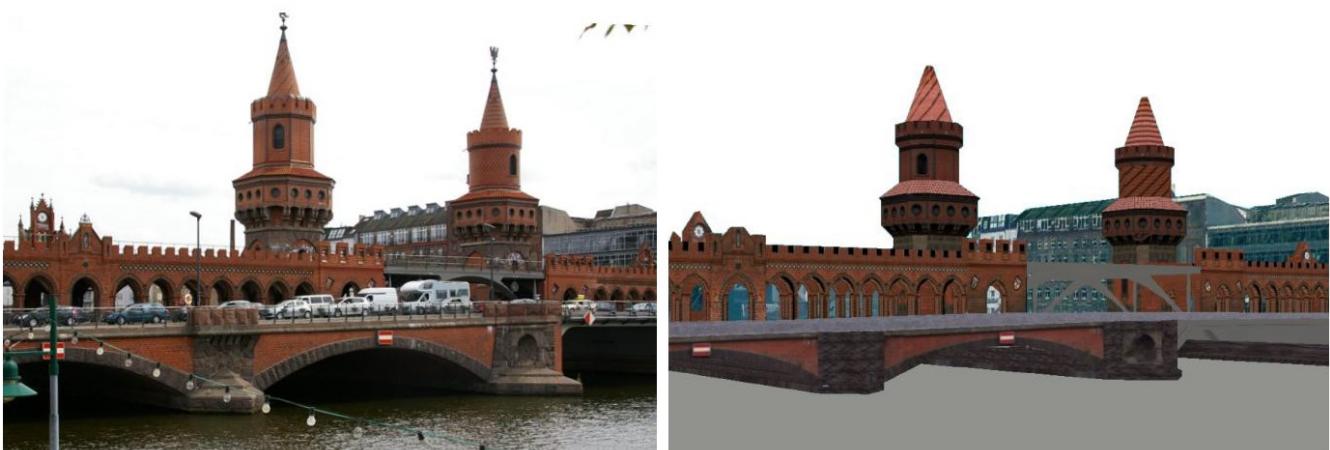


Figure 45. The bridge “Oberbaumbrücke” in Berlin represented as bridge model in LOD4 (left a photo, right the 3D CityGML model) (source: Berlin Senate of Business, Technology and Women; Business Location Center, Berlin; Technische Universität Berlin; Karlsruhe Institute of Technology (KIT)).

8.4. Building Model

Contributors

C. Heazel - first draft - working on better integration with Construction.

8.4.1. Synopsis

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 four levels of detail, LOD0 to LOD3. [Figure 28](#) provides examples of 3D city and building models in LOD1 – 3.

NOTE Figure needs to be updated

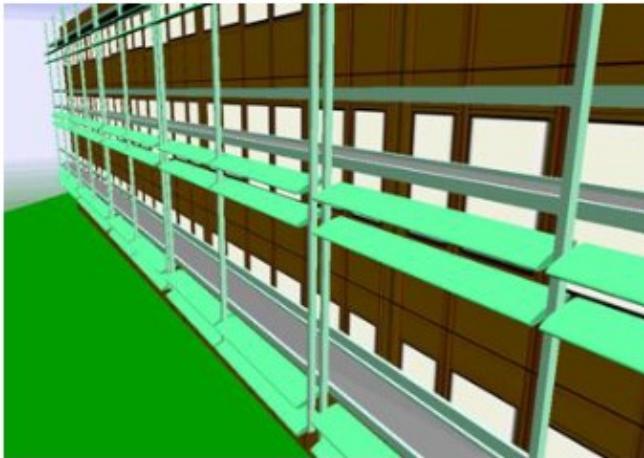
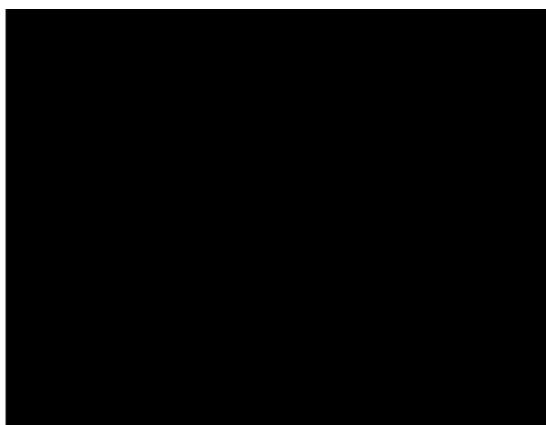


Figure 46. Examples for city or building models in LOD1 (upper left), LOD2 (upper right), and LOD3 (lower left). (source: District of Recklinghausen, m-g-h ingenieure+architekten GmbH).

8.4.2. Key Concepts

AbstractBuilding: An abstract superclass representing the common attributes and associations of the classes **Building** and **BuildingPart**.

A type of [AbstractConstruction](#).

Building: A free-standing, self-supporting construction that is roofed, usually walled, and can be entered by humans and is normally designed to stand permanently in one place. It is intended for

human occupancy (e.g. a place of work or recreation), habitation and/or shelter of humans, animals or things.

A type of [AbstractBuilding](#).

BuildingPart: A physical or functional subdivision of a Building. It would be considered a Building, if it were not part of a collection of other BuildingParts.

A type of [AbstractBuilding](#).

AbstractBuildingSubdivision: The abstract superclass for different kinds of logical building subdivisions.

A type of [AbstractLogicalSpace](#).

BuildingUnit: A logical subdivision of a Building. BuildingUnits are formed according to some homogeneous property like function, ownership, management, or accessibility. They may be separately sold, rented out, inherited, managed, etc.

A type of [AbstractBuildingSubdivision](#).

Storey: A horizontal section of a Building. Storeys are not always defined according to the building structure, but can also be defined according to logical considerations.

A type of [AbstractBuildingSubdivision](#).

BuildingRoom: A space within a Building or BuildingPart intended for human occupancy (e.g. a place of work or recreation) and/or containment of animals or things. A BuildingRoom is bounded physically and/or virtually (e.g. by ClosureSurfaces or GenericSurfaces).

A type of [AbstractUnoccupiedSpace](#).

BulidingInstallation: A permanent part of a Building (inside and/or outside) which has not the significance of a BuildingPart. Examples are stairs, antennas, balconies or small roofs.

A type of [AbstractInstallation](#).

BulidingFurniture: An equipment for occupant use, usually not fixed to the building.

A type of [AbstractFurniture](#).

BuildingConstructiveElement: An element of a Building which is essential from a structural point of view. Examples are walls, slabs, staircases, beams.

A type of [AbstractConstructiveElement](#).

8.4.3. Discussion

NOTE From CityGML 3.0

The Building module provides the representation of thematic and spatial aspects of buildings. Buildings are free-standing, self-supporting constructions that are roofed and usually walled, and that can be entered by humans and are normally designed to stand permanently in one place. Buildings are intended for human occupancy (e.g. a place of work or recreation), habitation and/or shelter of humans, animals or things. Buildings are represented in the UML model by the top-level feature type *Building*, which is the main class of the Building module. Buildings can physically or functionally be subdivided into building parts, and logically into storeys and building units (e.g. apartments). In addition, buildings can be decomposed into structural elements, such as walls,

slabs, staircases, and beams.

The interior of buildings is represented by rooms. This allows a virtual accessibility of buildings such as for visitor information in a museum (“Location Based Services”), the examination of accommodation standards or the presentation of daylight illumination of a building.

Buildings can contain installations and furniture. Installations are permanent parts of a building that strongly affect the outer or inner appearance of the building and that cannot be moved. Examples are balconies, chimneys, dormers or stairs. In contrast, furniture, represents moveable objects inside a building, like tables and chairs.

Buildings can be bounded by different types of surfaces. In this way, the outer façade of buildings can be differentiated semantically into wall surfaces, roof surfaces, ground surfaces, outer floor surfaces, and outer ceiling surfaces, whereas the visible surface of rooms can be structured into interior wall surfaces, floor surfaces, and ceiling surfaces. Furthermore, the openings of buildings, i.e. windows and doors, can be represented including their corresponding surfaces.

NOTE

The following content is from CityGML 2.0 with some updates to CityGML 3.0. Some of this content probably belongs in the Construction section now (windows, doord, etc.). There are also some changes from 2.0 to 3.0 (ex. Boundary Surfaces) which I am not clear on.

The pivotal class of the model is [AbstractBuilding](#), which is a subclass of [AbstractConstruction](#) and therefore a descendent of [AbstractCityObject](#). [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 a descendent of the [AbstractCityObject](#) class, an [AbstractBuilding](#) inherits all [City Object](#) properties including:

- The standard Feature properties ([name](#) etc.),
- The City Object specific properties like [externalReference](#),
- Spatial properties:
 - [lod\[1,2,3\]Solid](#),
 - [lod0Point](#),
 - [lod\[0,2,3\]MultiSurface](#),
 - [lod\[0,2,3\]MultCurve](#),
 - [lod\[1,2,3\]TerrainIntersectionCurve](#),
 - [lod\[1,2,3\]ImplicitRepresenation](#)

Further properties not explicitly covered by [AbstractBuilding](#) may be modelled as generic attributes provided by the CityGML [Generics module](#) or using the CityGML [Application Domain Extension](#) mechanism.

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](#). 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**.

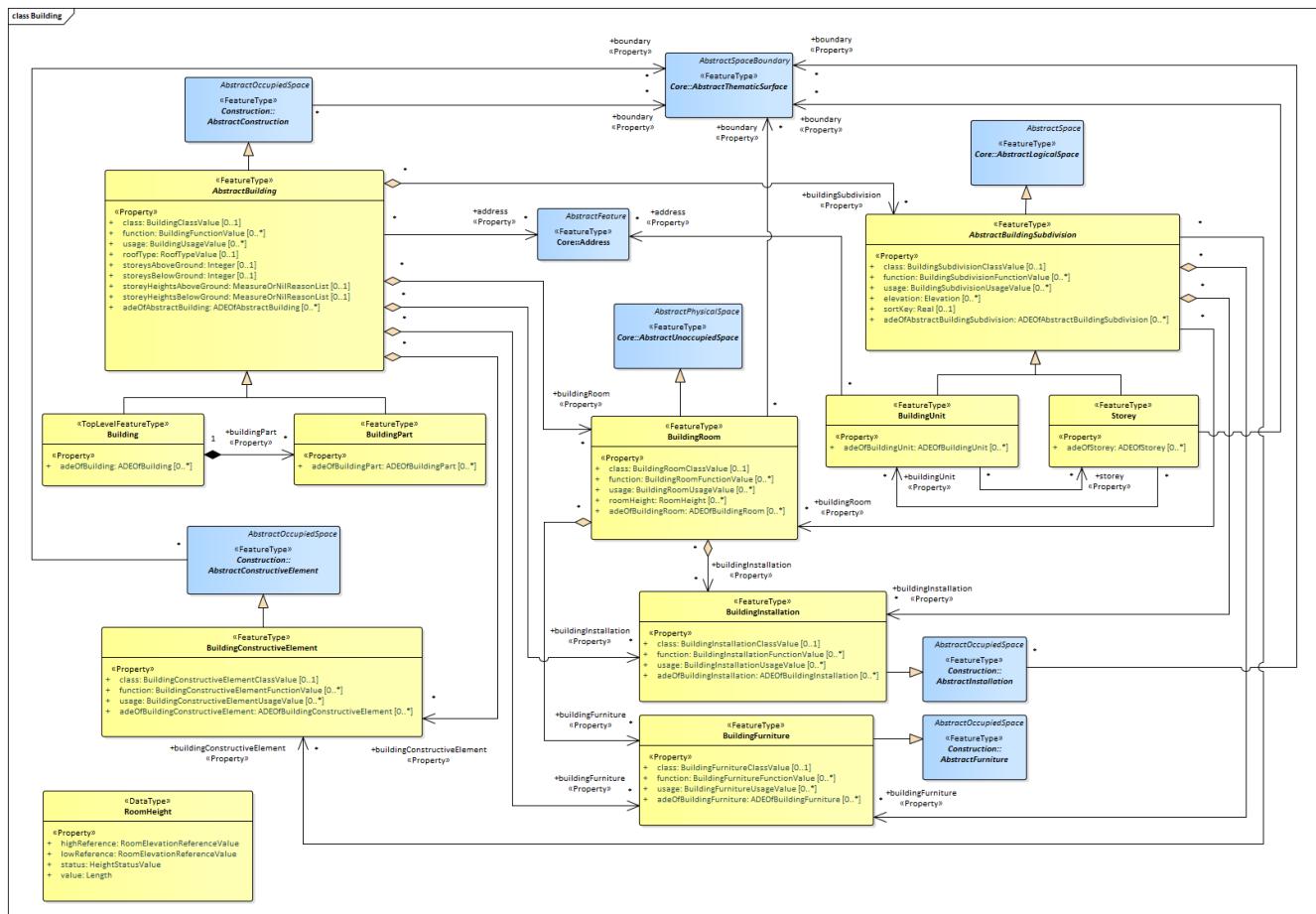


Figure 47. UML diagram of CityGML's building model.

The geometric representation and semantic structure of an **AbstractBuilding** is shown in [Figure-27]. The model is successively refined from LOD0 to LOD3. 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. 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. Cadastre 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 photogrammetrically extracted 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 [GM_MultiCurve](#) representing the [TerrainIntersectionCurve](#) can be specified. This geometric representation is refined in LOD2 by additional [GM_MultiSurface](#) and [GM_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 [BuildingConstructiveElement](#) and [BuildingInstallation](#). A [BuildingConstructiveElement](#) is a part of the building's exterior shell with a special function like wall (WallSurface), roof (RoofSurface), ground plate (GroundSurface), outer floor (OuterFloorSurface), outer ceiling (OuterCeilingSurface) 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. [UML diagram of CityGML's building model](#)).

In LOD3, the openings in [BuildingConstructiveElement](#) objects (doors and windows) can be represented as thematic objects. 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 (CeilingSurface), 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 geometri-topologically sound LOD4 model including the building interior.

8.4.4. Building Part

Building

The Building class is one of the two subclasses of AbstractBuilding. If a building only consists of one (homogeneous) 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 BuildingPart (see [Examples of buildings consisting of one and two building parts \(source: City of Coburg\)](#)). The geometry and non-spatial properties of the central part of the building should be represented in the aggregating Building feature.

Building Part

The class BuildingPart is derived from AbstractBuilding. It is used to model a structural part of a building (see [Examples of buildings consisting of one and two building parts \(source: City of Coburg\)](#)). A BuildingPart object should be uniquely related to exactly one building or building part object.



Figure 48. Examples of buildings consisting of one and two building parts (source: City of Coburg)

AbstractBuilding

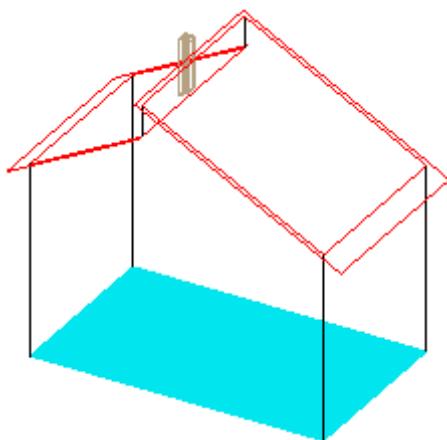
The abstract class AbstractBuilding contains properties for building attributes, purely geometric representations, and geometric/semantic representations of the building or building part in different levels of detail. The attributes describe:

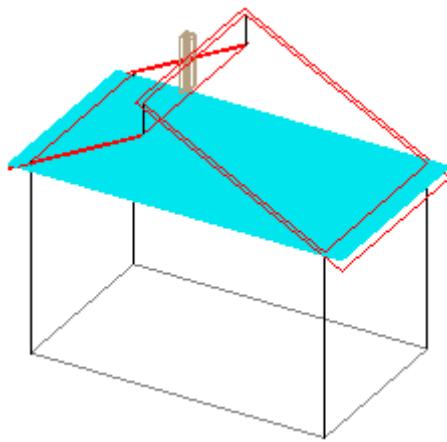
1. classification of the building or building part (class), the different intended usages (function),

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 building 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 geometric representation and the thematic structuring of the model into components with a special semantic meaning. This is illustrated in [\[figure-29\]](#) and [Building model in LOD1 – LOD4 \(source: Karlsruhe Institute of Technology \(KIT\), courtesy of Franz-Josef Kaiser\).<o:p></o:p>](#), showing the same building in five different LODs. The class AbstractBuilding has a number of properties which are associated with certain LODs.





*Figure 49. 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 *lod0RoofEdge* 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).*

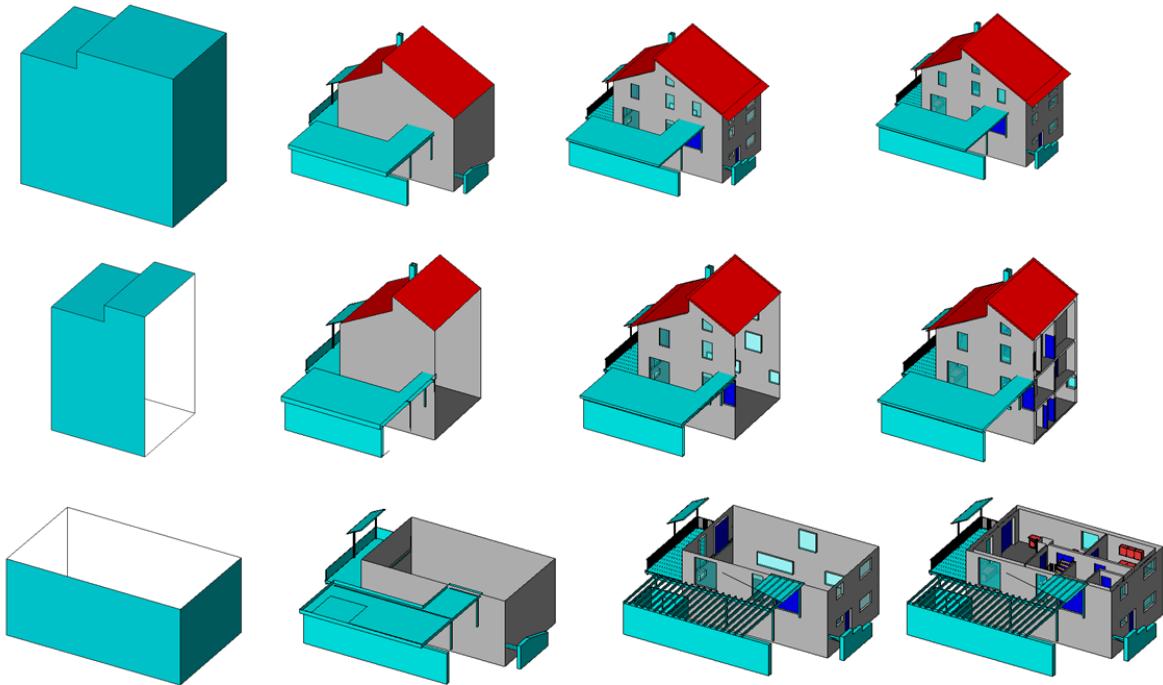


Figure 50. Building model in LOD1 – LOD4 (source: Karlsruhe Institute of Technology (KIT), courtesy of Franz-Josef Kaiser).<o:p></o:p>

Tab. 5 shows the correspondence of the different geometric and semantic themes of the building model to LODs. In LOD1 – 3, the volume of a building can be expressed by a GM_Solid geometry and/or a GM_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 – 3. The TIC can – but does not have to – build closed rings around the building or building parts.

In LOD0 (cf. [figure-29]) the building is represented by horizontal surfaces describing the footprint and the roof edge.

In LOD1 (cf. [Building model in LOD1 – LOD4](#) (source: Karlsruhe Institute of Technology (KIT),

courtesy of Franz-Josef Kaiser).<o:p></o:p>), 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 GM_Solid geometry and/or a GM_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 GM_MultiSurface geometry. If in a building model there is both a geometric representation 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 GM_MultiSurface of the **_BoundarySurface** elements.

Table 6. 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).

8.4.5. Outer building installations

BuildingInstallation

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.

8.4.6. Boundary surfaces

BoundarySurface

NOTE Is this now AbstractConstructionSurface?

_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 illustrated 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 GM_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 represented 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).

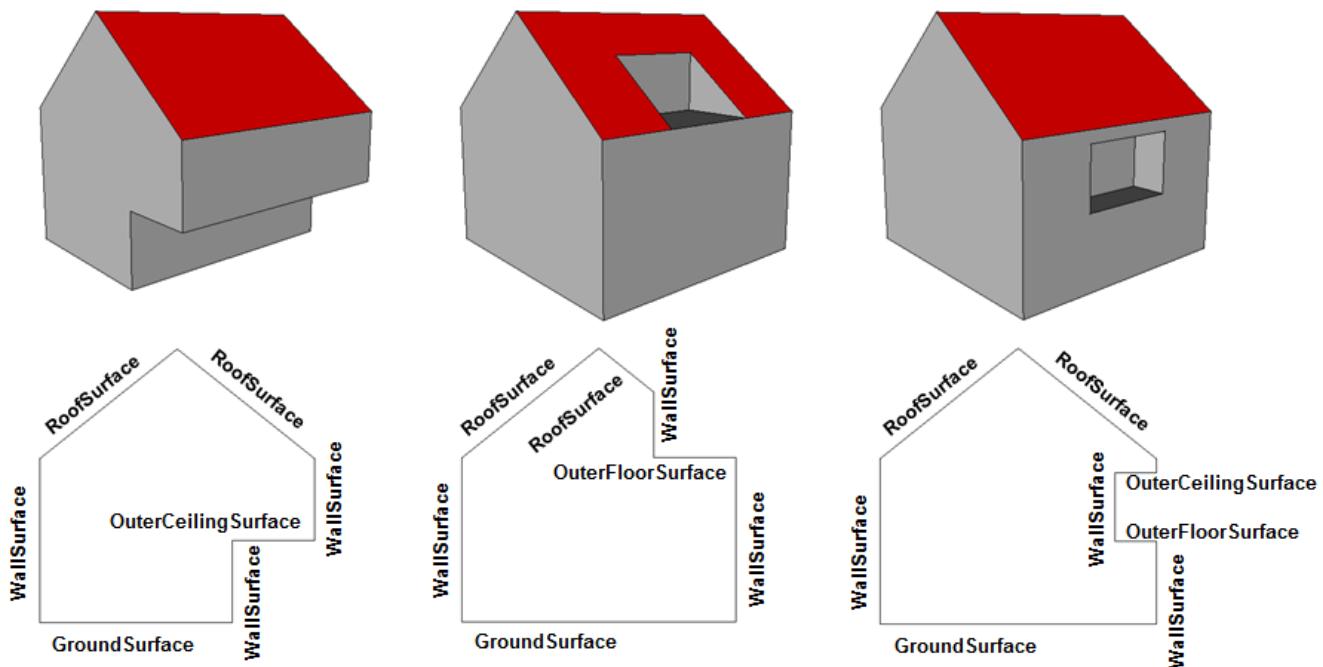
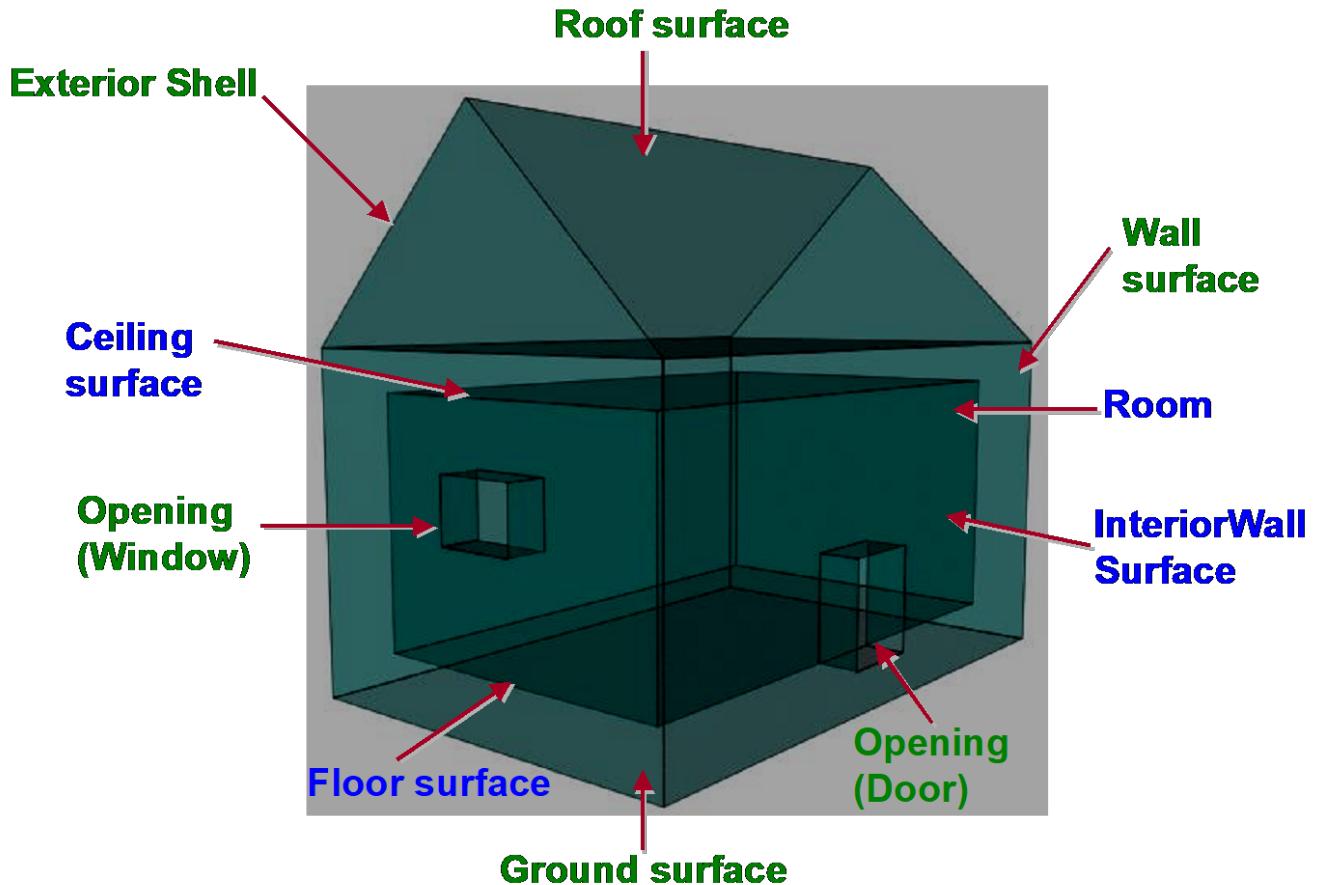


Figure 51. Examples of the classification of _BoundarySurfaces of the outer building shell (source: Karlsruhe Institute of Technology (KIT))



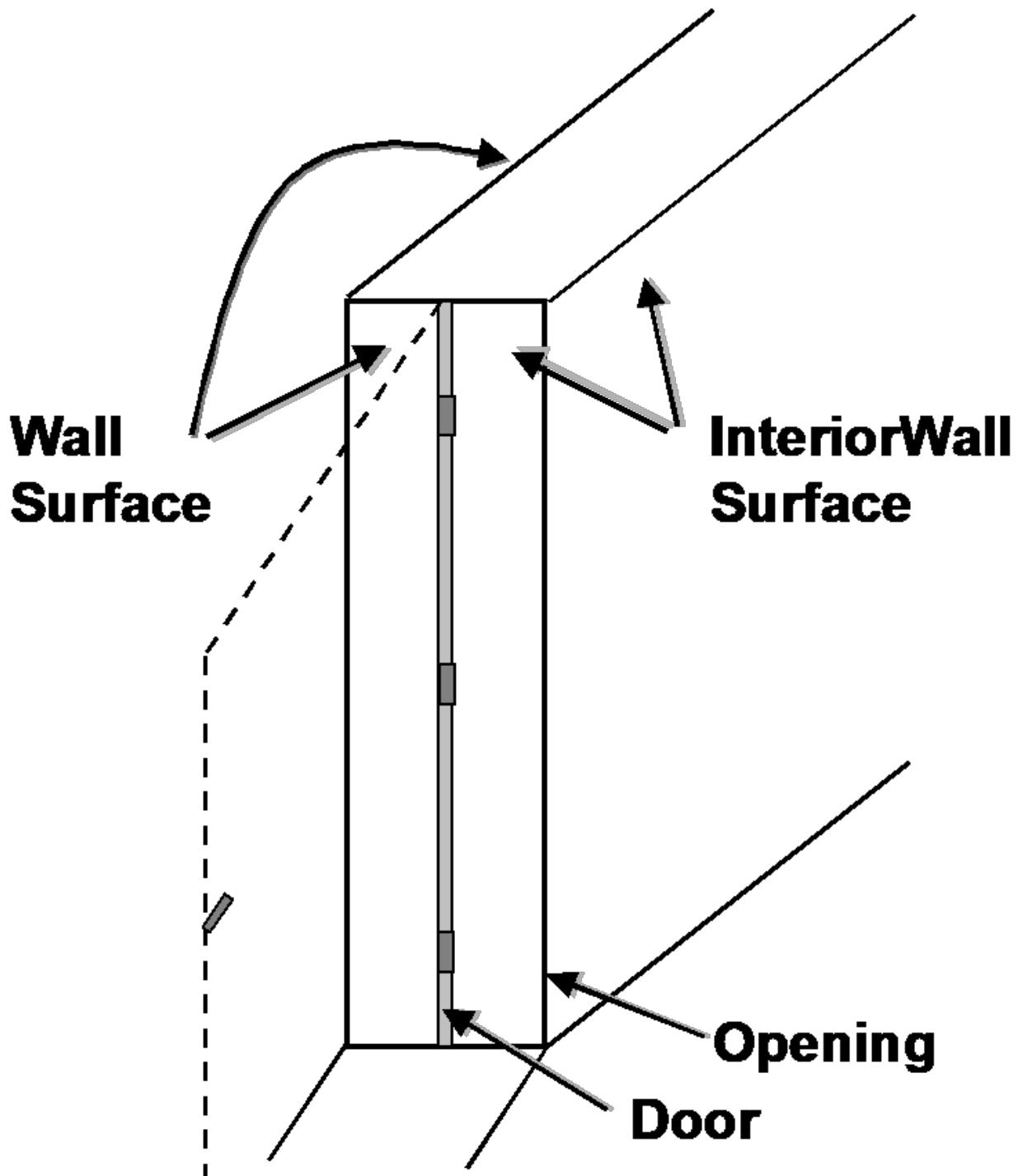


Figure 52. Classification of BoundarySurfaces (left), in particular for Openings (right) (graphic: IGG Uni Bonn).

GroundSurface

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.

OuterCeilingSurface

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.

WallSurface

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

OuterFloorSurface

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.

RoofSurface

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.

ClosureSurface

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 ClosureSurface should be used to separate or connect the volumes of both rooms.

FloorSurface

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

InteriorWallSurface

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

CeilingSurface

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

8.4.7. Openings

Opening

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 `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).

Window

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.

Door

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).

8.4.8. Building Interior

Room

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).

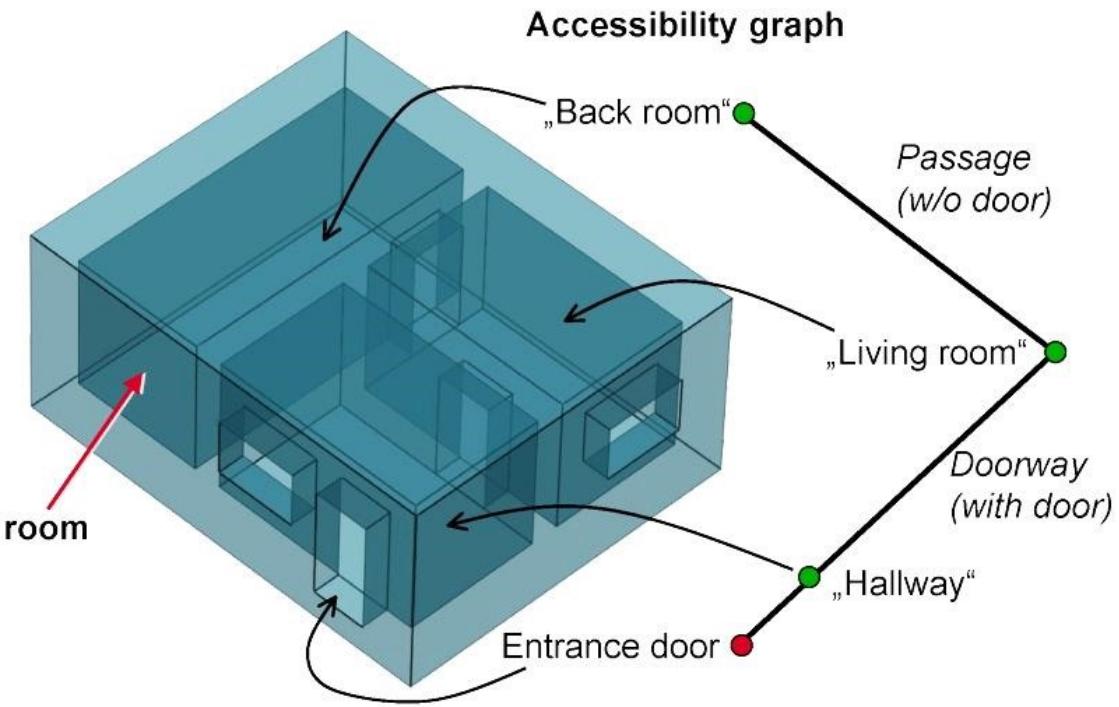


Figure 53. Accessibility graph derived from topological adjacencies of room surfaces (graphic: IGG Uni Bonn).

BuildingFurniture

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 coordinate system and referenced by other building furniture features (see chapter 8.2).

IntBuildingInstallation

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 IntBuildingInstallation 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 coordinate 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).

8.4.9. 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 building 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 fragmentation 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 denote 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 represents 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”.

8.4.10. UML Model

The UML diagram of the building module is depicted in [UML diagram of CityGML's building model..](#) The Building module inherits concepts from the [Construction module](#). The [Construction module](#) defines objects that are common to all types of construction, such as the different surface types and the openings.

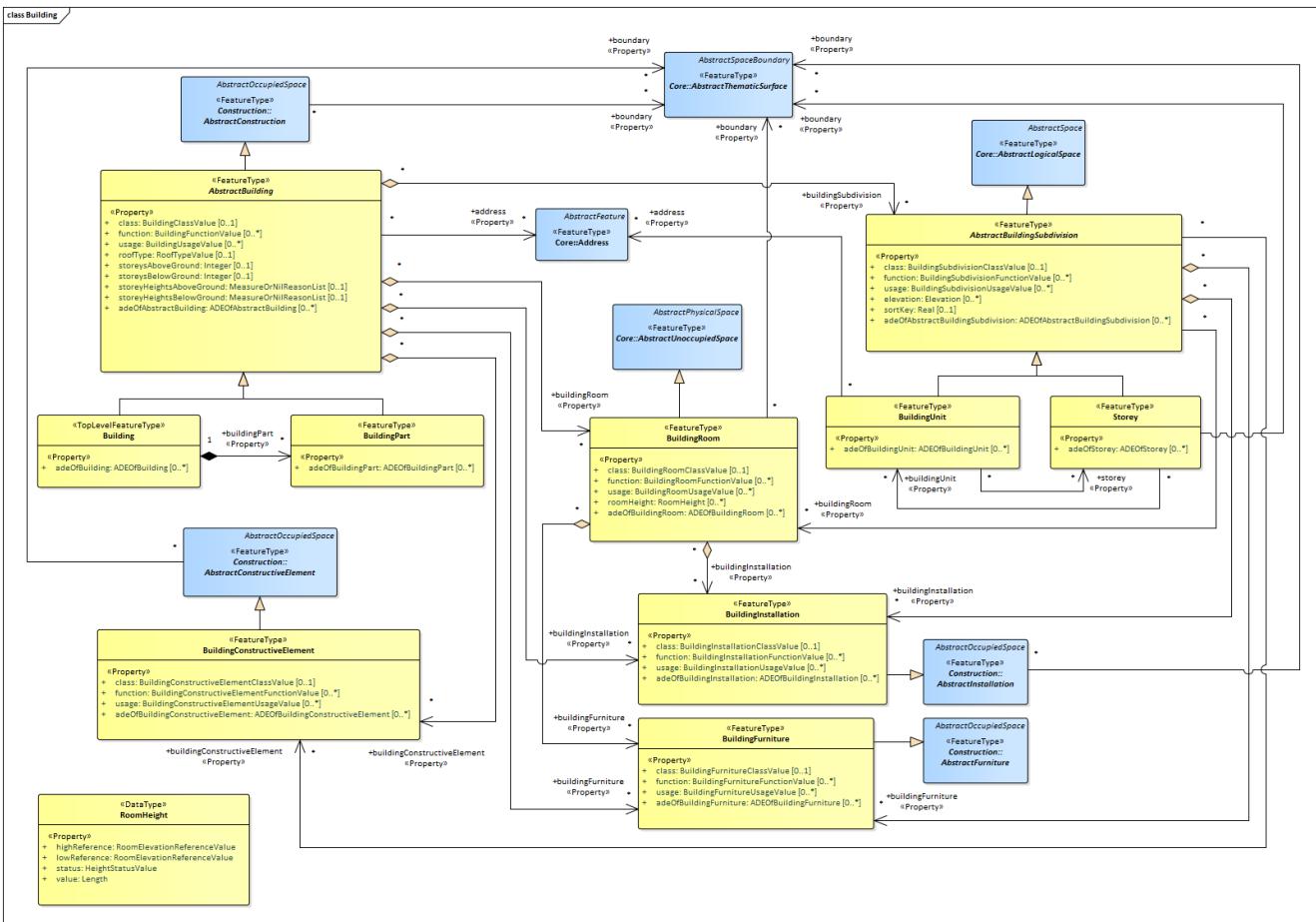


Figure 54. UML diagram of CityGML's building model.

The ADE data types provided for the Building module are illustrated in [ADE classes of the CityGML Building module..](#)

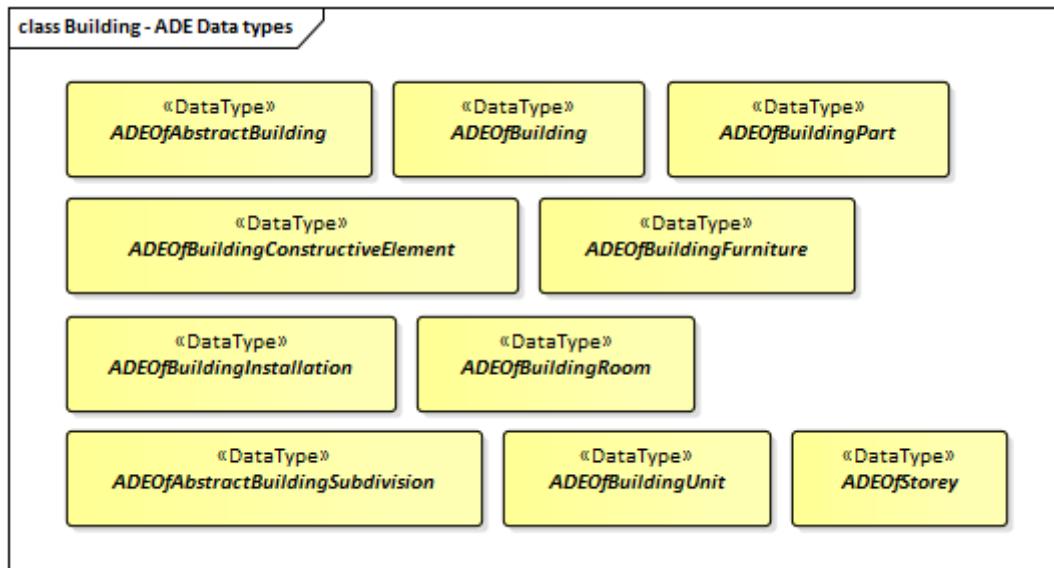


Figure 55. ADE classes of the CityGML Building module.

The Code Lists provided for the Building module are illustrated in [Codelists from the CityGML Building module..](#)

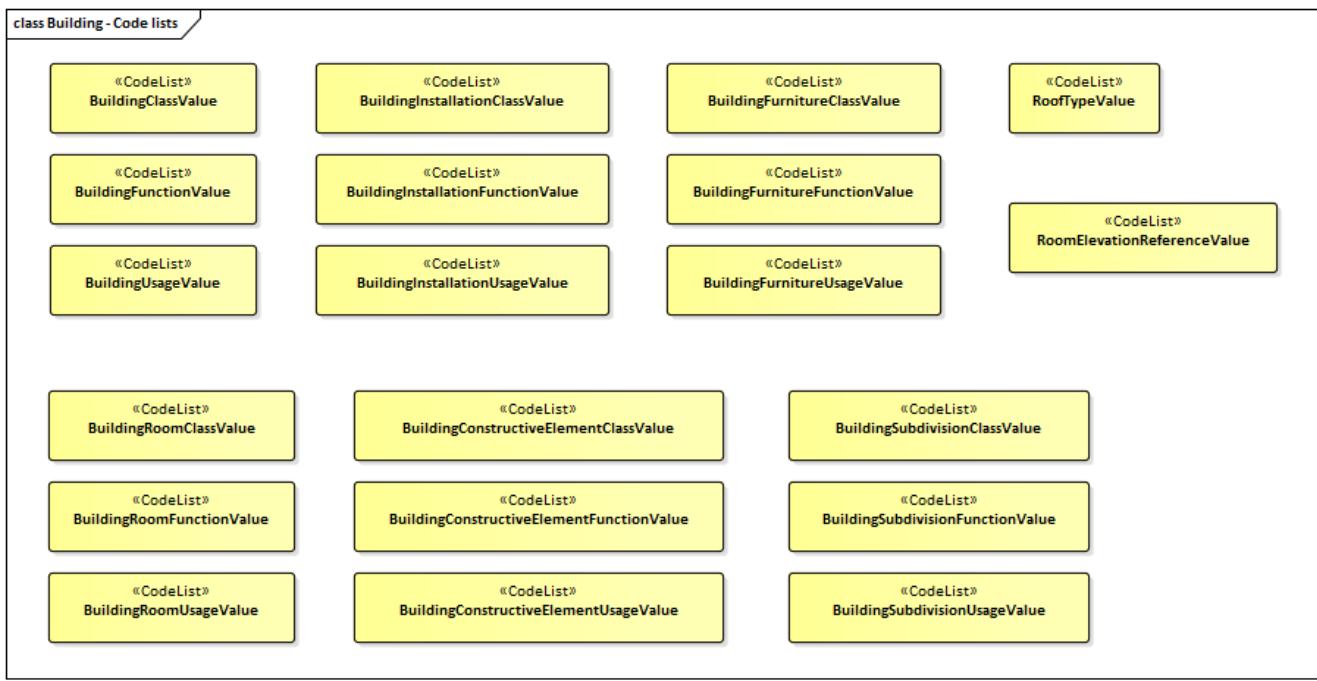


Figure 56. Codelists from the CityGML Building module.

8.4.11. Examples

The LOD1 model of the Campus North of the Karlsruhe 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 57. 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 58. 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.

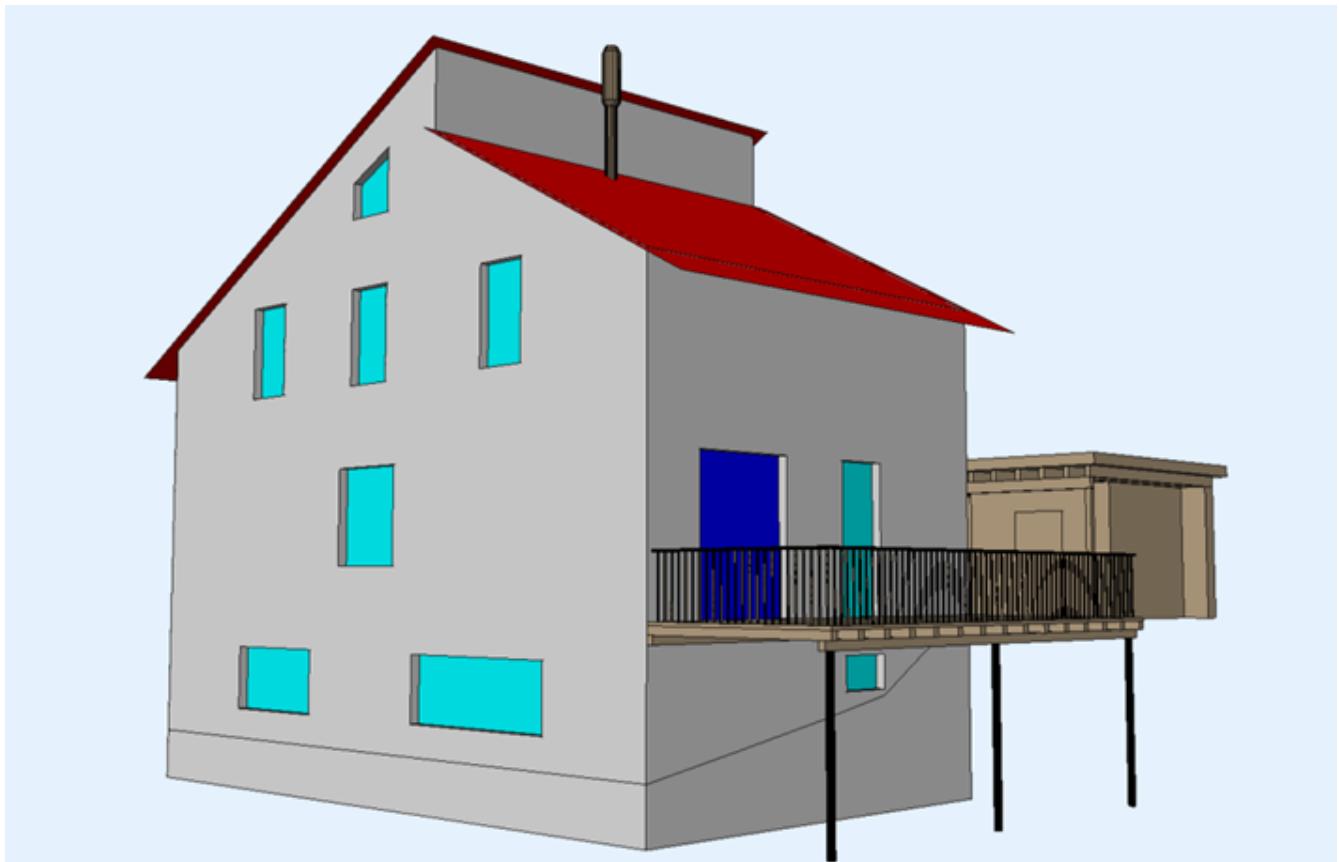


Figure 59. Example of a buildingmodeled in the Level of Detail 3. The chimney, the balcony and the car port are modeled as building installations (source: Karlsruhe Institute of Technology (KIT), courtesy of Franz-Josef Kaiser).

8.5. City Furniture

Contributors

Chuck Heazel: first draft

8.5.1. Synopsis

The CityFurniture module provides the representation of objects or pieces of equipment that are installed in the outdoor environment for various purposes, such as decoration, explanation or control. City furniture objects are relatively small, immovable objects and usually are of stereotypical form. Examples include road signs, traffic signals, bicycle racks, street lamps, fountains, flower buckets, advertising columns, and benches.

8.5.2. Key Concepts

CityFurniture: CityFurniture is an object or piece of equipment installed in the outdoor environment for various purposes. Examples include street signs, traffic signals, street lamps, benches, fountains.

A type of [AbstractOccupiedSpace](#).

8.5.3. Discussion

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 ([Figure 29](#), [Figure 30](#)). 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.

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 60. 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).



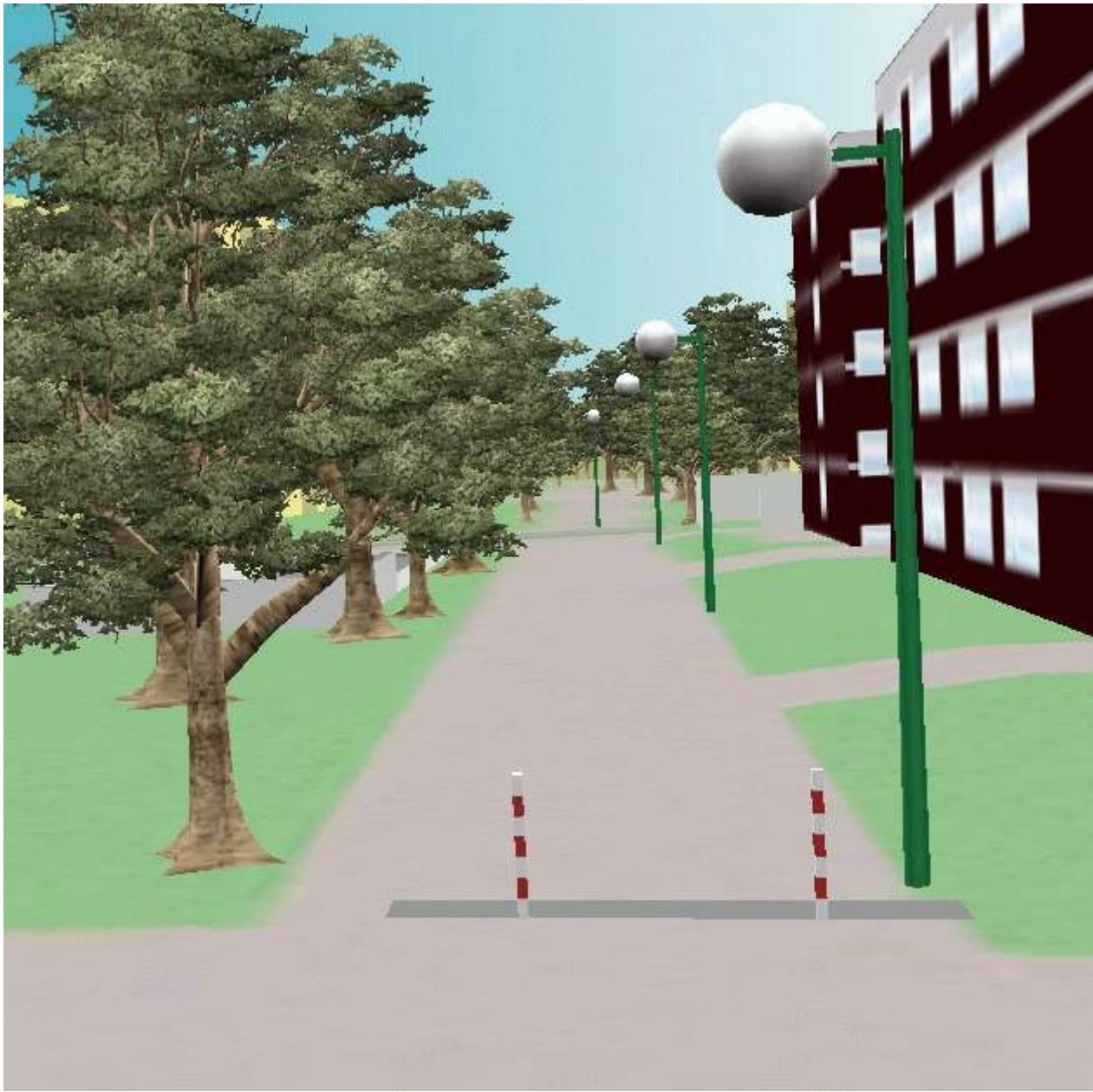


Figure 61. Real situation showing lanterns and delimitation stakes (left). In the right image they are modelled as CityFurniture objects with ImplicitGeometry representations (source: 3D city model of Barkenberg).

The class `CityFurniture` may have the attributes `class`, `function` and `usage`. Their possible values can be specified in corresponding `code lists`. 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.

Since `CityFurniture` is a subclass of `AbstractOccupiedSpace` and hence is a feature, it inherits the attribute `name`. As with any `CityObject`, `CityFurniture` objects may be assigned `ExternalReferences` and may be augmented by generic attributes using CityGML's `Generics` module. 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 1-3 may be represented by an explicit geometry (lodXGeometry where X is between 0 and 3) or an [ImplicitGeometry](#) object. Following the concept of [ImplicitGeometry](#), the geometry of a prototype city furniture is stored only once in a local coordinate system and referenced by other city furniture features. 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. This allows for ensuring a smooth transition between the DTM and the city furniture object.

8.5.4. UML Model

The UML diagram of the CityFurniture module is depicted in [UML diagram of CityGML's City Furniture model..](#)

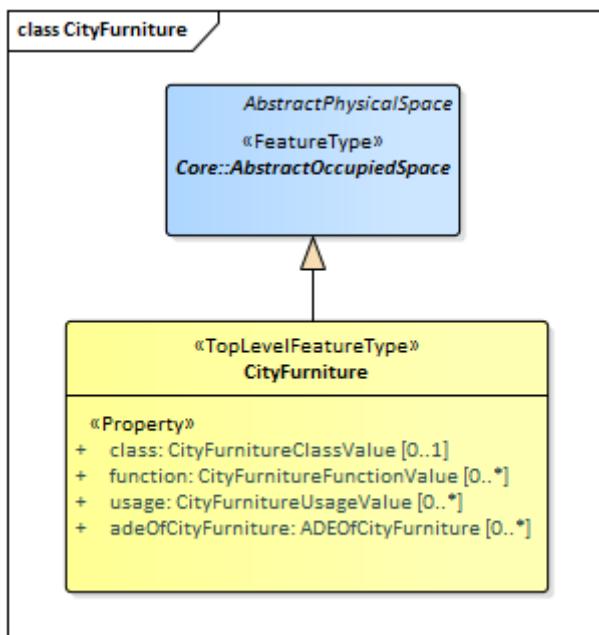


Figure 62. UML diagram of CityGML's City Furniture model.

The ADE data types and Code Lists provided for the CityFurniture module are illustrated in [ADE classes and Code Lists of the CityGML CityFurniture module..](#)

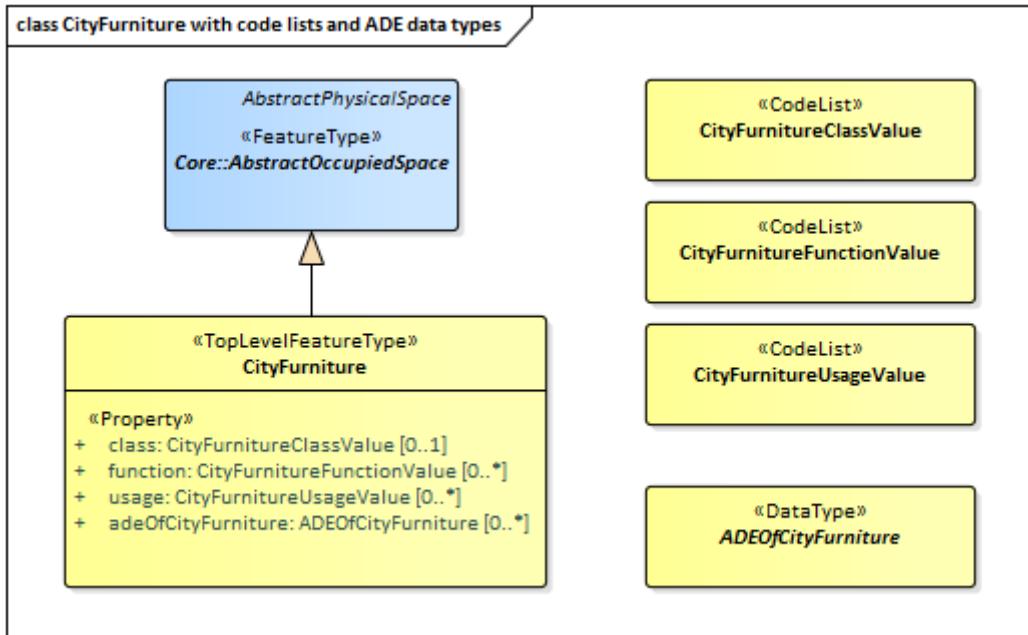


Figure 63. ADE classes and Code Lists of the CityGML CityFurniture module.

8.5.5. Examples

NOTE the following example is from version 2.0

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 coordinates. Both surface appearances are modelled using the CityGML Appearance module (cf. chapter 9).

Cover Surface

Surface left

Texture
stake.gif



Figure 64. Example of a simple city furniture object (source: District of Recklinghausen).

NOTE | insert example - GML?

8.6. City Object Group

Contributors

Chuck Heazel: first draft

8.6.1. Synopsis

Aggregates City Objects into a collection of City Objects. That collection is itself a City Object.

8.6.2. Key Concepts

CityObjectGroup: A CityObjectGroup represents an application-specific aggregation of [city objects](#)

according to some user-defined criteria.

A type of [AbstractLogicalSpace](#).

Role: Role is a UML Association Class tied to the `groupMember` association between [CityObjectGroup](#) and [AbstractCityObject](#). It serves to qualify the function of a city object within the CityObjectGroup.

8.6.3. Discussion

CityObjectGroups are modelled using the Composite Design Pattern from software engineering (cf. Gamma et al. 1995): CityObjectGroups aggregate [City Objects](#) and furthermore are defined as special [City Objects](#). 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.

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 intended to express the main purpose of a group, possibly to which thematic area it belongs (e.g. site, building, transportation, architecture, unknown etc.). The attribute `usage` can be used, if the way the object is actually used differs from the function. The `function` and `usage` attributes can occur multiple times.

Each member of a group may be qualified by a [Role](#), reflecting the role each [City Object](#) plays in the context of the group. Furthermore, a CityObjectGroup can optionally be assigned an arbitrary ISO [geometry object](#). 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 hierarchical 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.

8.6.4. UML Model

The UML diagram of the CityObjectGroup module is depicted in [UML diagram of the City Object Group Model..](#)

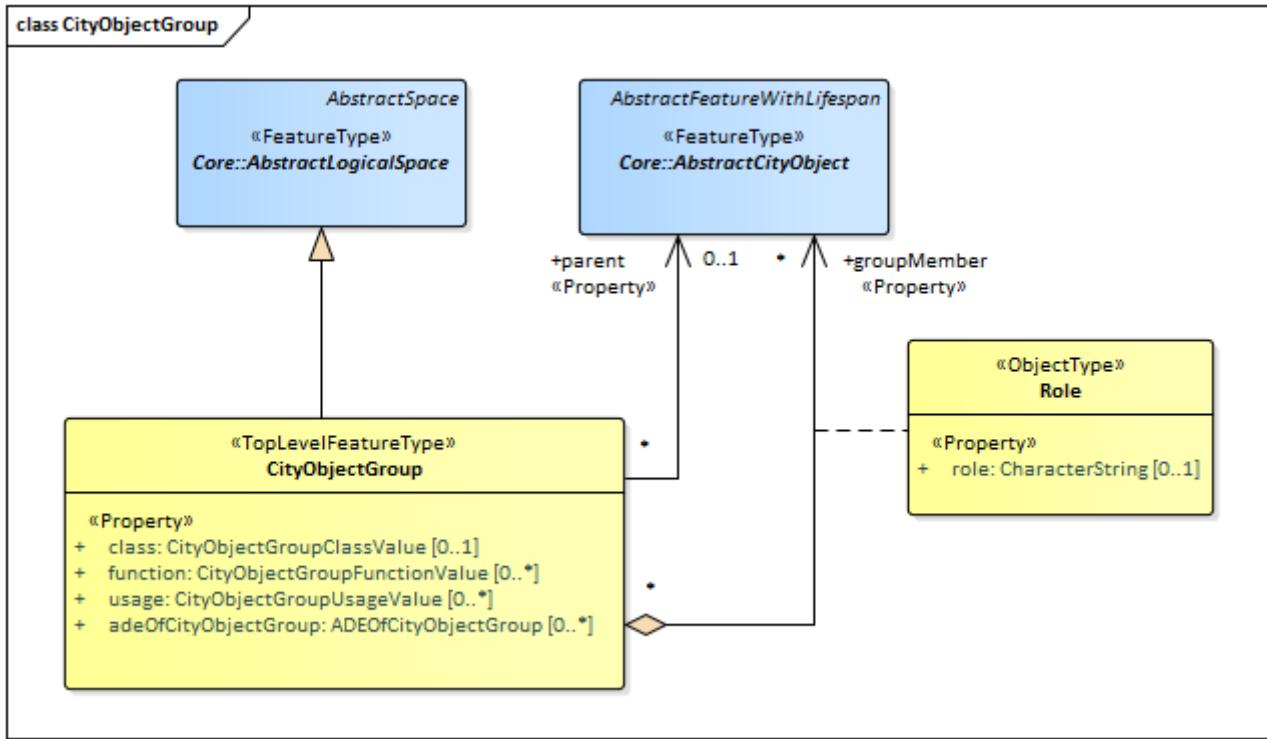


Figure 65. UML diagram of the City Object Group Model.

The ADE data types provided for the `CityObjectGroup` module are illustrated in [ADE classes of the CityGML CityObjectGroup module..](#)

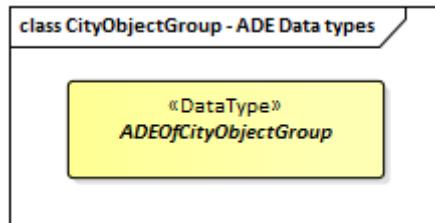


Figure 66. ADE classes of the CityGML CityObjectGroup module.

The Code Lists provided for the `CityObjectGroup` module are illustrated in [Codelists from the CityGML CityObjectGroup module..](#)

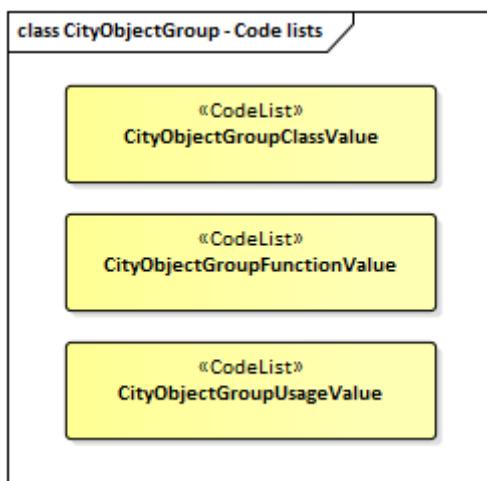


Figure 67. Codelists from the CityGML CityObjectGroup module.

8.6.5. Examples

8.7. Construction

Contributors
Chuck Heazel - first draft

8.7.1. Synopsis

The Construction module defines concepts that are common to all forms of constructions.

8.7.2. Key Concepts

AbstractConstruction: Classes of objects that are manufactured by humans from construction materials, are connected to earth, and are intended to be permanent. A type of [AbstractOccupiedSpace](#)

AbstractConstructionElement: Classes that represent volumetric elements of a construction. Examples are walls, beams, slabs. A type of [AbstractOccupiedSpace](#).

AbstractFillingElement: Classes that represent different kinds of elements that fill the openings of a construction such as windows and doors. A type of [AbstractOccupiedSpace](#).

AbstractFurniture: Classes that represent furniture objects of a construction. A type of [AbstractOccupiedSpace](#).

AbstractInstallation: Classes that represent installation objects of a construction. A type of [AbstractOccupiedSpace](#).

AbstractConstructionSurface: Classes that represent different kinds of surfaces that bound a construction such as roof and wall surfaces. A type of [AbstractThematicSurface](#).

AbstractFillingSurface: Classes that represent different kinds of surfaces that seal openings. The openings are filled by filling elements such as window and door surfaces. A type of [AbstractThematicSurface](#).

8.7.3. Discussion

The Construction module defines concepts that are common to all forms of constructions. Constructions are objects that are manufactured by humans from construction materials, are connected to earth, and are intended to be permanent. The Construction module focuses on as-built representations of constructions and integrates all concepts that are similar over different types of constructions, in particular buildings, bridges, and tunnels. In addition, for representing man-made structures that are neither buildings, nor bridges, nor tunnels so-called other constructions (e.g. large chimneys or city walls) can be defined.

Furniture, installations, and constructive elements are further concepts that are defined in the Construction module. Installations are permanent parts of a construction that strongly affect the outer or inner appearance of the construction and that cannot be moved (e.g. balconies, chimneys,

or stairs), whereas furniture represent moveable objects of a construction (e.g. tables and chairs). Constructive elements allow for decomposing a construction into volumetric components, such as walls, beams, and slabs.

Constructions and constructive elements can be bounded by different types of surfaces. In this way, the outer structure of constructions and constructive elements can be differentiated semantically into wall surfaces, roof surfaces, ground surfaces, outer floor surfaces, and outer ceiling surfaces, whereas the visible surface of interior spaces can be structured into interior wall surfaces, floor surfaces, and ceiling surfaces.

Furthermore, the openings of constructions, i.e. windows and doors, can be represented as so-called filling elements including their corresponding filling surfaces.

The Construction module defines concepts that are inherited and, where necessary, are specialized by the modules Building, Bridge, and Tunnel (cf. [Building Model](#), [Bridge Model](#), and [Tunnel Model](#)).

8.7.4. Construction Spaces

8.7.5. Construction Surfaces

8.7.6. Levels of Detail

The different Levels of Detail are defined in the following way:

- LOD 0: Volumetric real-world objects (Spaces) can be spatially represented by a single point, by a set of curves, or by a set of surfaces.

Areal real-world objects (Space Boundaries) can be spatially represented in LOD0 by a set of curves or a set of surfaces. LOD0 surface representations are typically the result of a projection of the shape of a volumetric object onto a plane parallel to the ground, hence, representing a footprint (e.g. a building footprint or a floor plan of the rooms inside a building). LOD0 curve representations are either the result of a projection of the shape of a vertical surface (e.g. a wall surface) onto a grounding plane or the skeleton of a volumetric shape of longitudinal extent such as a road or river segment.

- LOD 1: Volumetric real-world objects (Spaces) are spatially represented by a vertical extrusion solid, i.e. a solid created from a horizontal footprint by vertical extrusion.

Areal real-world objects (Space Boundaries) can be spatially represented in LOD1 by a set of horizontal or vertical surfaces.

- LOD 2: Volumetric real-world objects (Spaces) can be spatially represented by a set of curves, a set of surfaces, or a single solid geometry.

Areal real-world objects (Space Boundaries) can be spatially represented in LOD2 by a set of surfaces. The shape of the real-world object is generalized in LOD2 and smaller details (e.g. bulges, dents, sills, but also structures like e.g. balconies or dormers of buildings) are typically neglected.

LOD2 curve representations are skeletons of volumetric shapes of longitudinal extent like an antenna or a chimney.

- LOD 3: Volumetric real-world objects (Spaces) can be spatially represented by a set of curves, a set of surfaces, or a single solid geometry.

Areal real-world objects (Space Boundaries) can be spatially represented in LOD3 by a set of surfaces. LOD3 is the highest level of detail and respective geometries include all available shape details.

A **Construction Model** is successively refined from LOD0 to LOD3. Therefore, not all components of a construction 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. An object can be represented simultaneously in different LODs by providing distinct geometries for the corresponding LODs.

LOD0

In LOD0, a **Construction Object** can be represented by two horizontal, 3-dimensional surfaces (**AbstractThematicSurface**). These two surfaces represent the foot-print of a **Construction Object** and a top surface such as a roof edge (see [Figure 31](#)). 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. Cadastre data typically depicts the shape of a **Construction Object** on the ground (footprints). Higher level surfaces are often photogrammetrically extracted from area/satellite images or derived from airborne laser data. Topographic data is often provides a mixture between footprints and geometries at higher levels.

The surface geometries at LOD0 can be represented using [GM_MultiCurve](#) or [GM_MultiSurface](#). 3D coordinates are required for both representations. At LOD0 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.

[Figure 31](#) illustrates an LOD0 representation of a Building Construction. The two horizontal 3D surfaces are colored in cyan. On the left, the building footprint 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 Roof Edge representation is illustrated which results from a horizontal projection of the building's roof and which is located at the eaves height

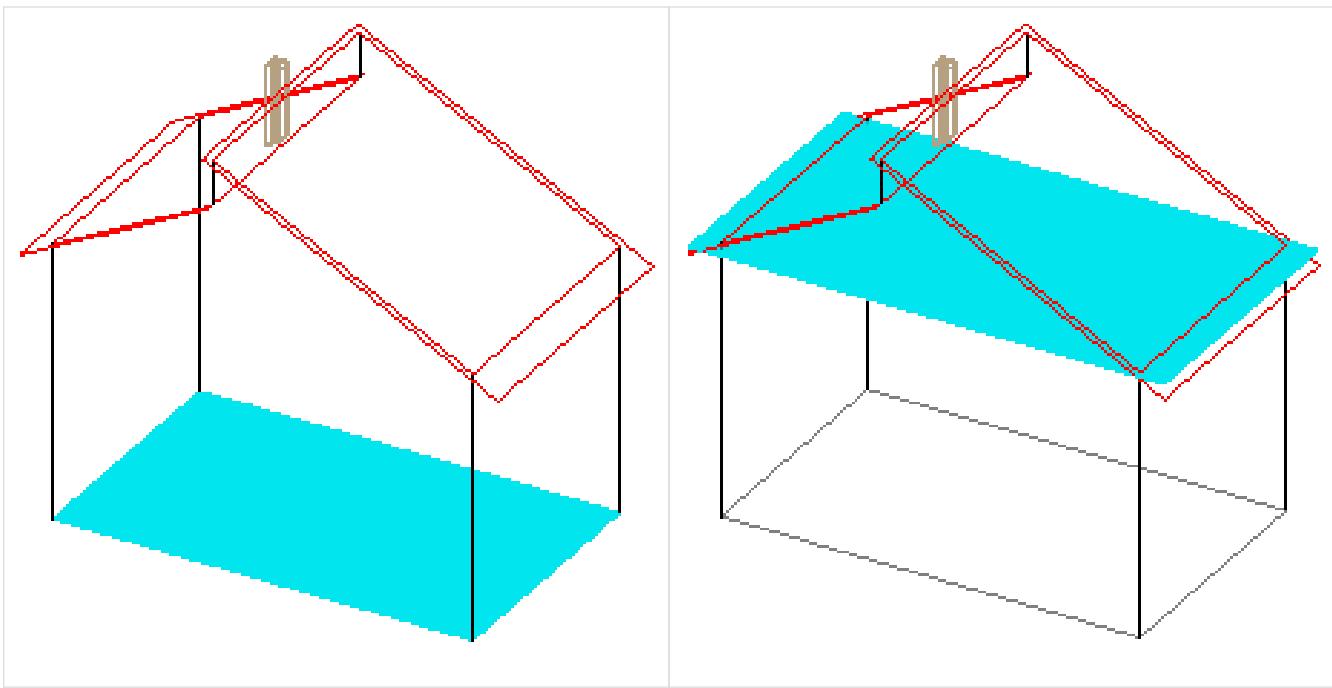


Figure 68. Modeling a construction (building) in LOD0 (source: Karlsruhe Institute of Technology (KIT), courtesy of Franz-Josef Kaiser).

Note that at LOD0 a [construction object](#) also has a [Space](#) geometry but that geometry is limited to no more than one [GM_Point](#).

LOD1

In LOD1, a [construction object](#) consists of the generalized geometric shape of the object ([GM_Solid](#)) as represented by its' boundary surfaces. The boundary surfaces are [AbstractThematicSurfaces](#) identified by the [boundary](#) attribute. Optionally, a [GM_MultiCurve](#) representing the [TerrainIntersectionCurve](#) can also be specified.

LOD2

This geometric representation is refined in LOD2 by additional [GM_MultiSurface](#), [GM_MultiCurve](#), and [GM_Solid](#) geometries, used for modelling architectural details like roof overhangs, columns, or antennas.

In LOD2 and higher LODs the outer facade of a [Construction Object](#) can also be differentiated semantically by the classes [AbstractConstructiveElement](#) and [AbstractInstallation](#). An [AbstractConstructiveElement](#) is a structural part of the [Construction Objects](#) exterior shell. At LOD2 only the [boundary](#) surface is displayed. An [AbstractInstallation](#) is a non-structural part of the [Construction Objects](#) exterior shell. As with [AbstractConstructiveElement](#), only the [boundary](#) surface is displayed at LOD2.

The [boundary](#) surfaces are [abstractThematicSurfaces](#) identified using the [boundary](#) attribute of [AbstractConstructiveElement](#) and [AbstractInstallation](#).

At LOD2 these boundary surfaces are represented as subclasses of the [AbstractConstructionSurface](#) class. These subclasses represent surfaces with a special function. These subclasses are:

- roof (RoofSurface),
- ground plate (GroundSurface),
- external walls (WallSurface),
- interior walls (InteriorWallSurface)
- interior floors (FloorSurface)
- exterior floors (OuterFloorSurface),
- interior ceiling (CeilingSurface),
- exterior ceiling (OuterCeilingSurface),
- Closure Surface ([ClosureSurface](#)).

NOTE

LOD2 and LOD3 needs some work. It would be helpful if we could identify which Thematic Surface subclasses apply at each LOD.

LOD3

In LOD3, the openings in [BuildingConstructiveElement](#) objects (doors and windows) can be represented as thematic objects. 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 (CeilingSurface), 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.

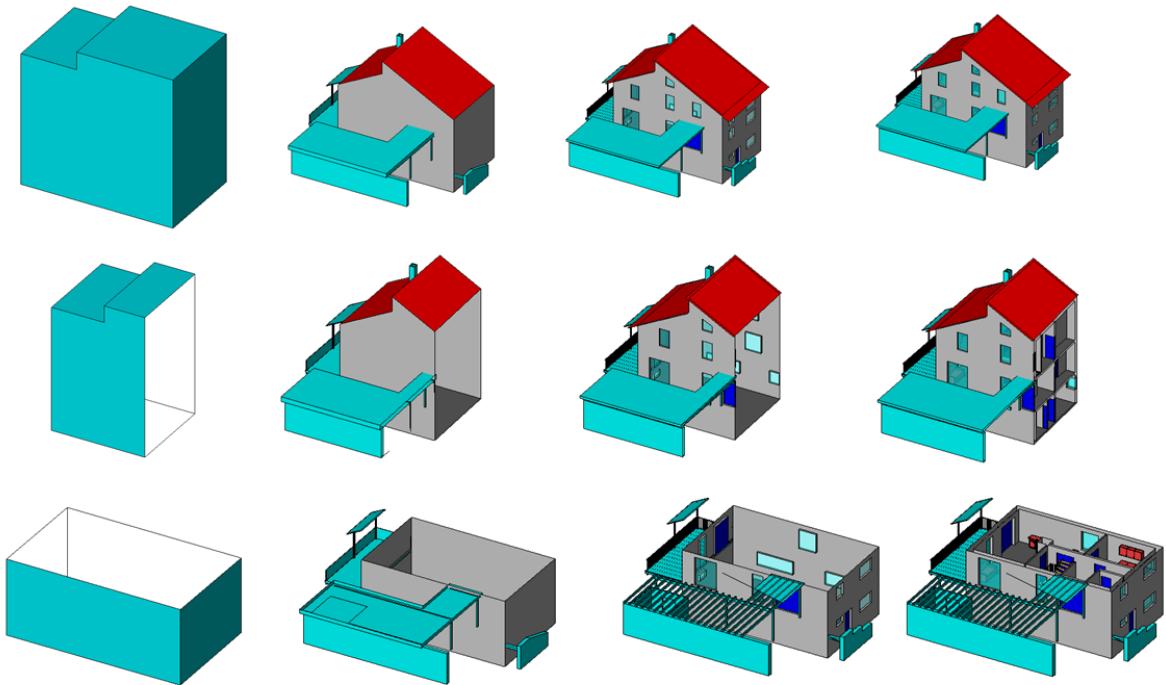


Figure 69. Building model in LOD1 – LOD4 (source: Karlsruhe Institute of Technology (KIT), courtesy of Franz-Josef Kaiser).<o:p></o:p>

8.7.7. Spaces and Surfaces

The semantic structure of a [Construction Model](#) is shown in [Figure 32](#). The «FeatureType» classes in this model fall into two categories; [Spaces](#) and [Surfaces](#). [Spaces](#) are descended from the [AbstractOccupiedSpace](#) class and inherit their geometry from this class. [Surfaces](#) are descended from the [AbstractThematicSurface](#) class and inherit their geometry from that class.

Occupied Spaces (class [AbstractOccupiedSpace](#))

Abstract Construction (class [AbstractConstruction](#))

The [AbstractConstruction](#) class is used for free-standing, self-supporting constructions. It represents the construction as a whole. A construction composed of structural segments differing in, for example, the number of storeys or the roof type has to be separated into one Construction having one or more additional Construction Elements (see [Examples of buildings consisting of one and two building parts \(source: City of Coburg\)](#)). The geometry and non-spatial properties of the central part of the construction should be represented in the aggregating Construction feature.

The abstract class [AbstractConstruction](#) contains properties for [Construction Object](#) attributes. These properties describe:

- [conditionOfConstruction](#):
- `dateOfConstruction`:
- [dateOfDemolition](#):
- [constructionEvent](#):
- [elevation](#):

- **height**:
- **occupancy**:
- **boundary**: An association with the **AbstractThematicSurface** which defines the boundary of this **Construction Object**.

AbstractConstruction is a subclass of **AbstractOccupiedSpace** and inherits the geometry properties from that class.

Other Construction (class **OtherConstruction**)

Under construction

Abstract Construction Element (class **AbstractConstructionElement**)

The class **AbstractConstructionElement** is derived from **AbstractOccupiedSpace**. Subclasses of this class are used to identify the constituent parts of a **Construction Object** (see [Examples of buildings consisting of one and two building parts \(source: City of Coburg\)](#)). A **construction element** object should be uniquely related to exactly one **construction** or **construction element** object.

AbstractConstructionElement includes properties which describe:

- **isStructualElement**:
- **filling**:
- **boundary**: An association with the **AbstractThematicSurface** which defines the boundary of this **Construction Element Object**.

AbstractConstructionElement is a subclass of **AbstractOccupiedSpace** and inherits the geometry properties from that class.



Figure 70. Examples of buildings consisting of one and two building parts (source: City of Coburg)

Abstract Furniture (class **AbstractFurniture**)

Construction Objects may have **Furnitures** and **Installations**. An **AbstractFurniture** is a movable part of a **Construction**, such as a chair or table. An **AbstractFurniture** object should be uniquely

related to exactly one **Construction 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 coordinate system and referenced by other **furniture** features.

Abstract Installation (class AbstractInstallation)

An **AbstractInstallation** is an object inside a **Construction Object** with a specialised function or semantic meaning. In contrast to **Furniture**, **AbstractInstallation** are permanently attached to the structure and cannot be moved. Examples for **AbstractInstallation** are stairways, railings and heaters. An **AbstractInstallation** object should be uniquely related to exactly one **Construction 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 coordinate system and referenced by other **furniture** features.

An **AbstractInstallation** optionally has attributes **class**, **function** and **usage**. The attribute **class**, which can only occur once, represents a general classification of the **Installation Object**. With the attributes **function** and **usage**, nominal and real functions of a **Installation Object** can be described. For all three attributes the list of feasible values can be specified in a code list.

AbstractInstallation is a subclass of **AbstractOccupiedSpace** and inherits the geometry properties from that class.

Abstract Filling Element (class AbsractFillingElement)

The class **AbsractFillingElement** 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. Each **_Opening** is associated with a **GM_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.

===== Window (class Window)

The class **Window** is used for modelling windows in the exterior shell of a **Construction Object**, or between adjacent **Construction Elements**. 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.

===== Door (class Door)

The class **Door** is used for modelling doors in the exterior shell of a **Construction Object**, or between adjacent **Construction Elements**. Doors can be used by people to enter or leave a **Construction Object** or **Construction Element**. 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.

Thematic Surfaces (class AbstractThematicSurface)

Under construction

Filling Surfaces (class AbstractFillingSurface)

Under construction

===== Door Surface (class DoorSurface)

Under construction

===== Window Surface (class WindowSurface)

Under construction

Construction Surfaces (class AbstractConstructionSurface)

Under construction

===== Roof (class RoofSurface),

Under construction

===== Ground Plate (class GroundSurface)

The ground plate of a **Construction** or **Constructive Element** is modelled by the class **GroundSurface**. The polygon defining the ground plate is congruent with the construction's footprint. However, the surface normal of the ground plate is pointing downwards.

===== External Walls (class WallSurface),

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

===== Interior Walls (class InteriorWallSurface)

Under construction

===== Interior Floors (class FloorSurface)

Under construction

===== Exterior Floors (class OuterFloorSurface),

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

===== Interior Ceiling (class CeilingSurface),

Under construction

===== Exterior Ceiling (class OuterCeilingSurface),

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

Closure Surface (class ClosureSurface).

An opening in a **Construction Object** not filled by a **door** or **window** can be sealed by a virtual surface called a **ClosureSurface**. Hence, **Constructions** with open sides can be virtually closed in order to be able to compute their volume.

8.7.8. UML Model

The UML diagram of the Construction module is depicted in [Figure 32](#).

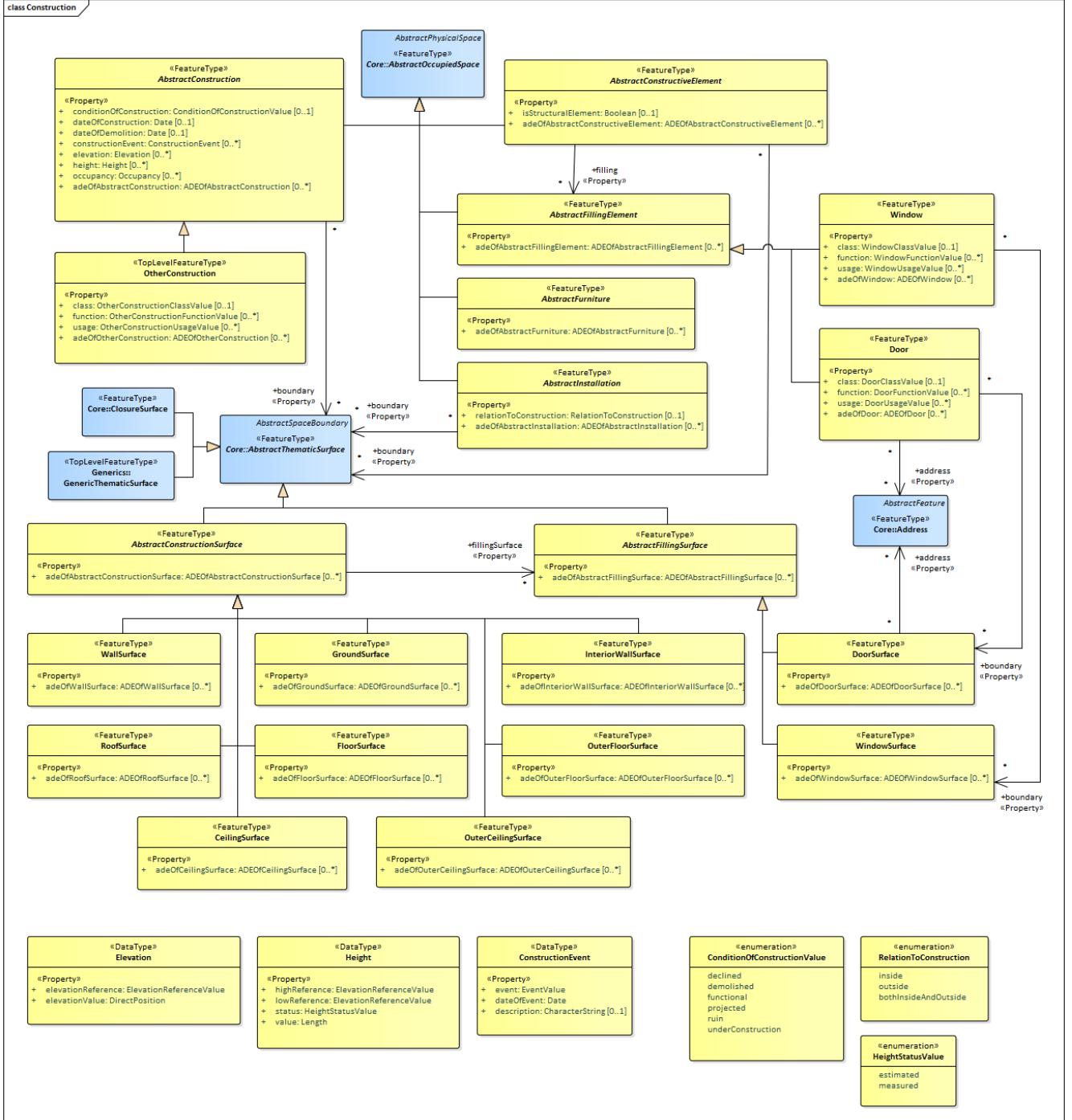


Figure 71. UML diagram of the Construction Model.

The ADE data types provided for the Construction module are illustrated in [Figure 33](#).

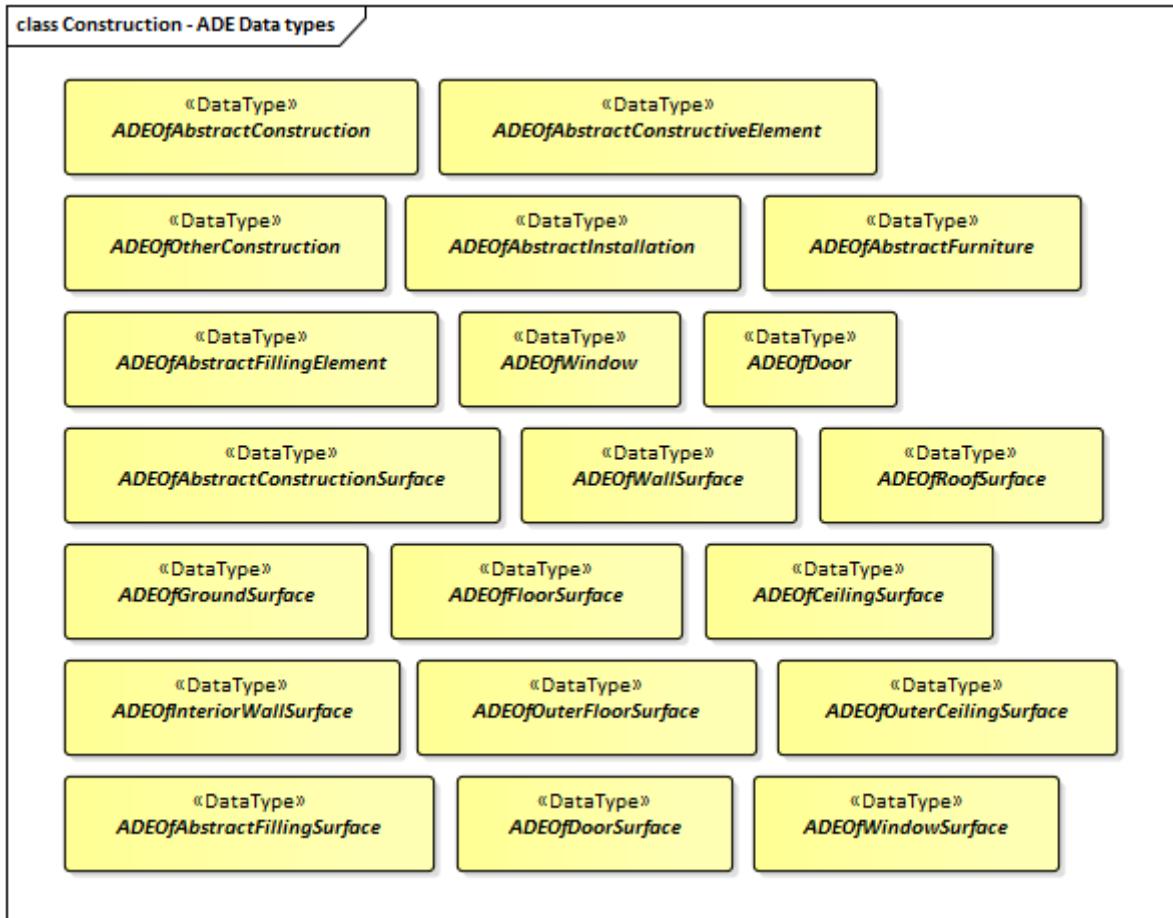


Figure 72. ADE classes of the CityGML Construction module.

The Code Lists provided for the Construction module are illustrated in [Figure 34](#).

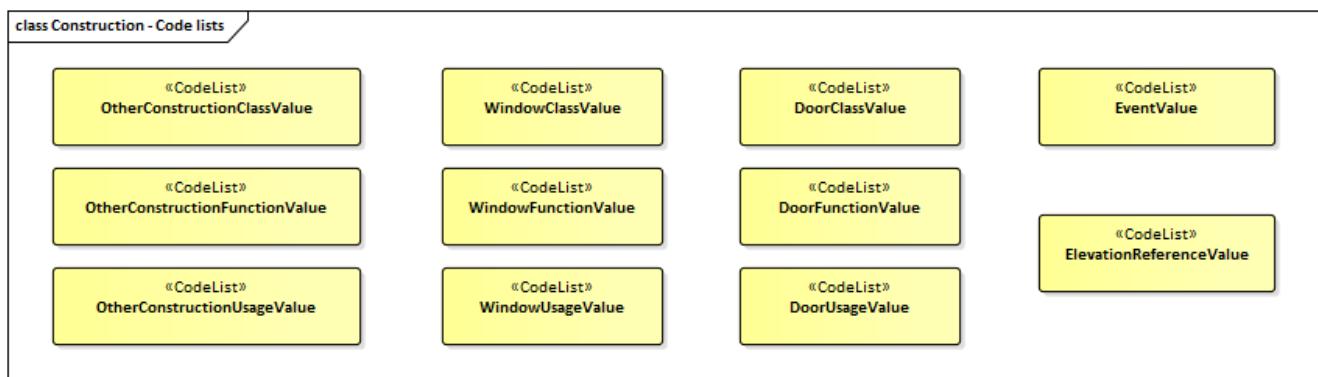


Figure 73. Codelists from the CityGML Construction module.

8.7.9. Examples

8.8. Dynamizer

Contributors

C. Heazel - first draft

8.8.1. 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.

8.8.2. 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.

8.8.3. Discussion

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.

8.8.4. UML Model

The UML diagram of the Dynamizer module is depicted in [UML diagram of the Dynamizer Model..](#)

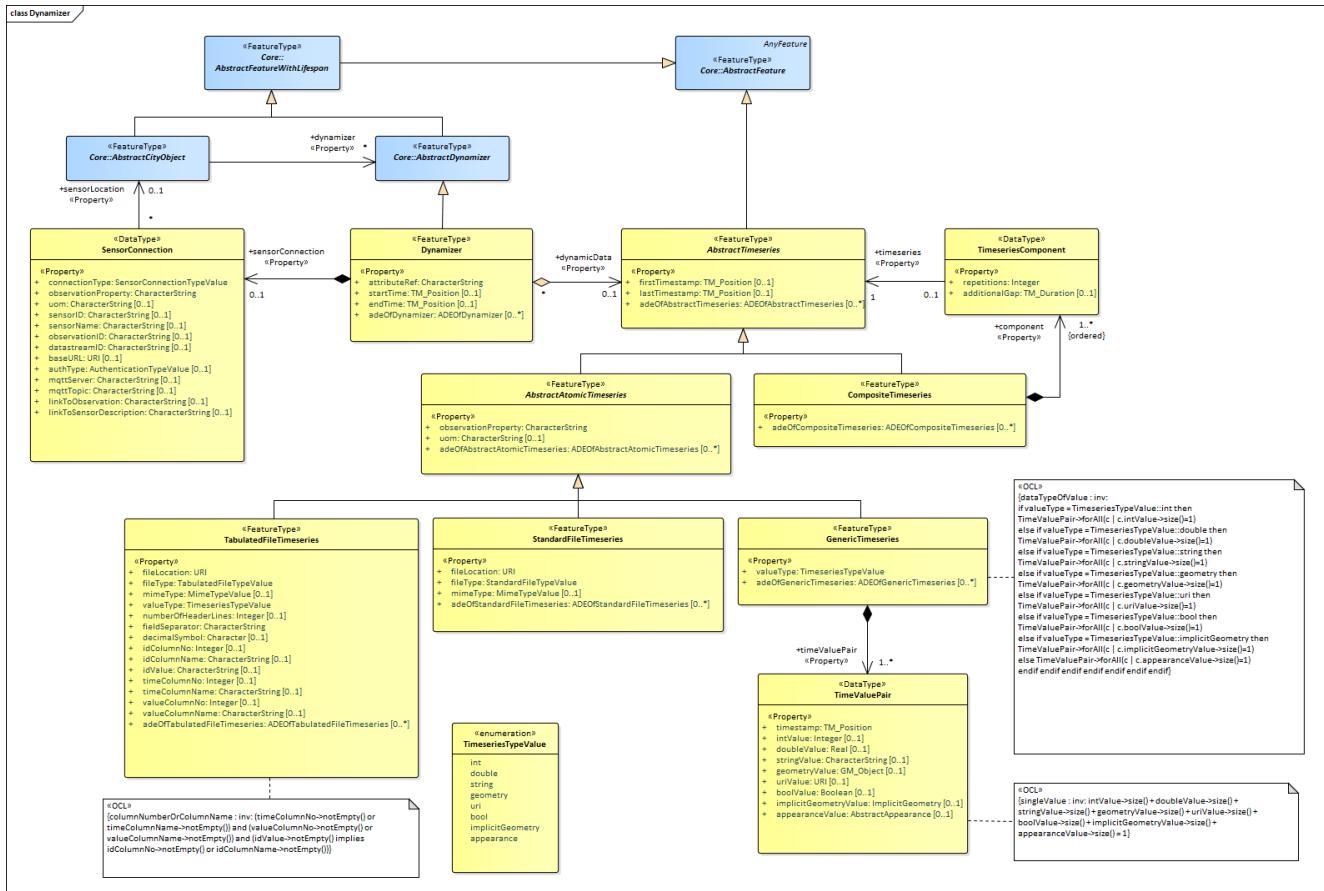


Figure 74. UML diagram of the Dynamizer Model.

The ADE data types provided for the Dynamizer module are illustrated in [ADE classes of the CityGML Dynamizer module..](#)

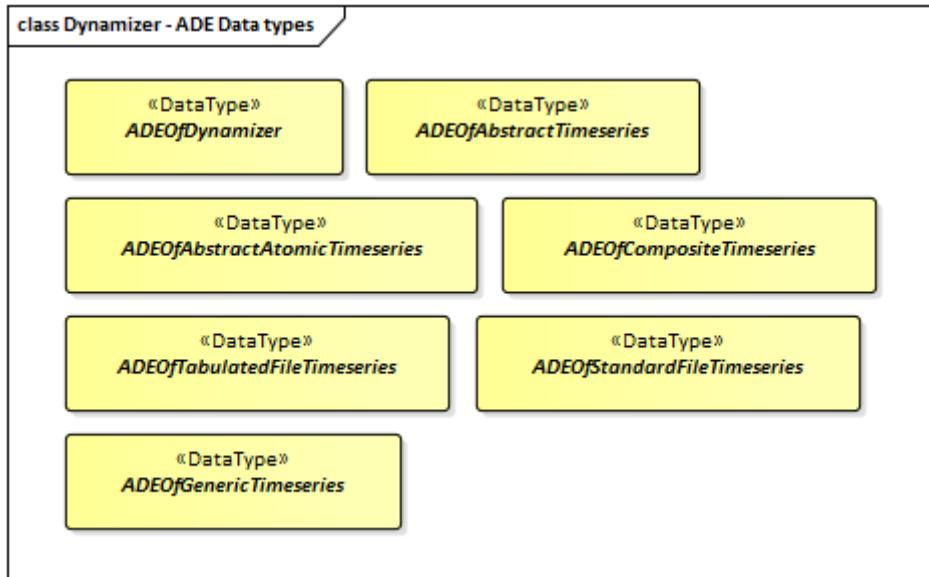


Figure 75. ADE classes of the CityGML Dynamizer module.

The Code Lists provided for the Dynamizer module are illustrated in [Codelists from the CityGML Dynamizer module..](#)

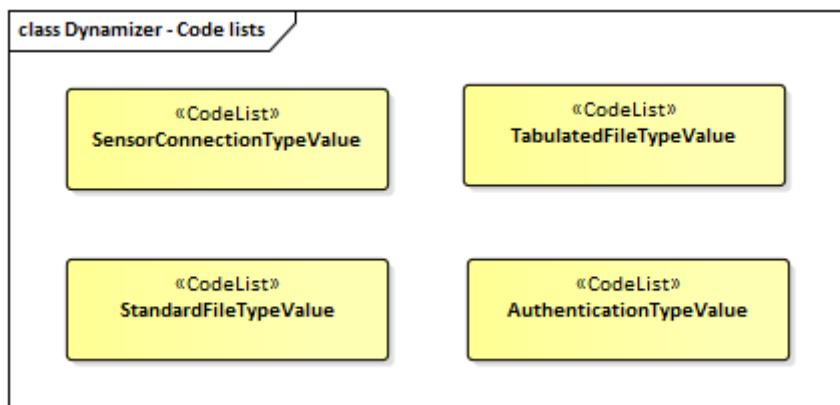


Figure 76. Codelists from the CityGML Dynamizer module.

8.8.5. Examples

8.9. Generics

Contributors
C. Heazel - first draft

8.9.1. 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.

8.9.2. Key Concepts

GenericLogicalSpace: A space that is not represented by any explicitly modelled AbstractLogicalSpace subclass within CityGML.

A type of [AbstractLogicalSpace](#).

GenericOccupiedSpace: A space that is not represented by any explicitly modelled AbstractOccupiedSpace subclass within CityGML.

A type of [AbstractOccupiedSpace](#).

GenericUnoccupiedSpace: A space that is not represented by any explicitly modelled AbstractUnoccupiedSpace subclass within CityGML.

A type of [AbstractUnoccupiedSpace](#).

GenericThematicSurface: A surface that is not represented by any explicitly modelled AbstractThematicSurface subclass within CityGML.

A type of [AbstractThematicSurface](#).

GenericAttributeSet: A named collection of generic attributes.

A type of [AbstractGenericAttribute](#).

<type>Attribute: a data type used to define generic attributes of type <type> where type = "String", "Int", "Double", "Date", "Uri", "Measure", or "Code".

A type of [AbstractGenericAttribute](#).

8.9.3. Discussion

The Generics module provides the representation of generic city objects. These are city objects that are not covered by any explicitly modelled thematic class within the CityGML Conceptual Model. The Generics module also provides the representation of generic attributes which are attributes that are not explicitly represented in the CityGML Conceptual Model. In order to avoid problems concerning semantic interoperability, generic city objects and generic attributes shall only be used if appropriate thematic classes and attributes are not provided by any other CityGML module.

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

Generic attributes are defined as name-value pairs and are always associated with a city object. Generic attributes can be of type String, Integer, Double, Date, URI, Measure, and Code. In addition, generic attributes can be grouped under a common name as generic attribute sets.

8.9.4. UML Model

The UML diagram of the Generics module is depicted in [UML diagram of the Generics Model](#).

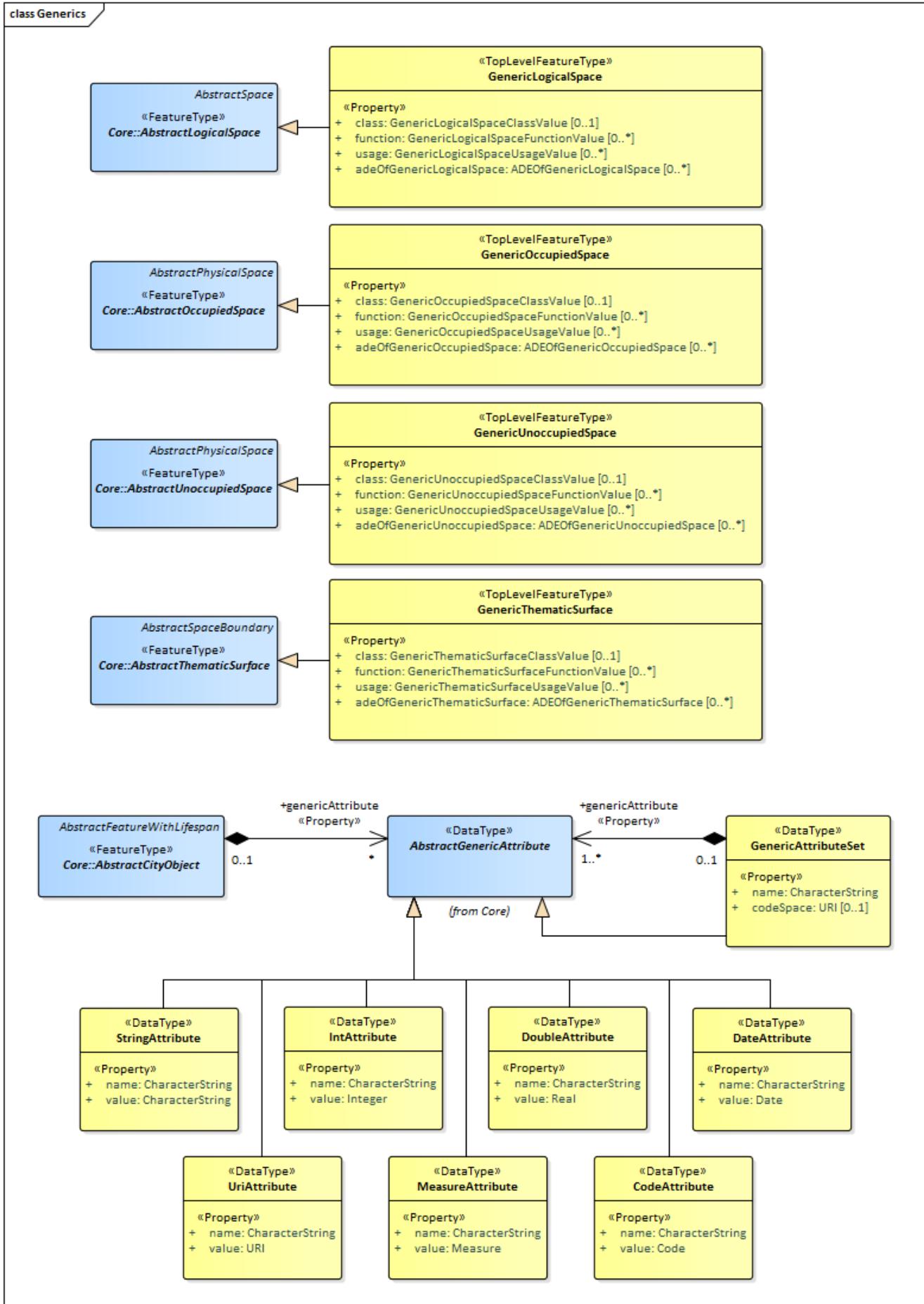


Figure 77. UML diagram of the Generics Model.

The ADE data types provided for the Generics module are illustrated in [ADE classes of the CityGML](#)

Generics module..

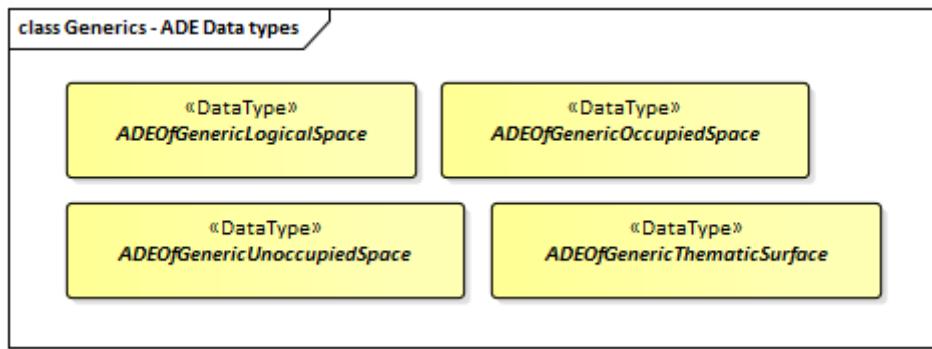


Figure 78. ADE classes of the CityGML Generics module.

The Code Lists provided for the Generics module are illustrated in [Codelists from the CityGML Generics module..](#)

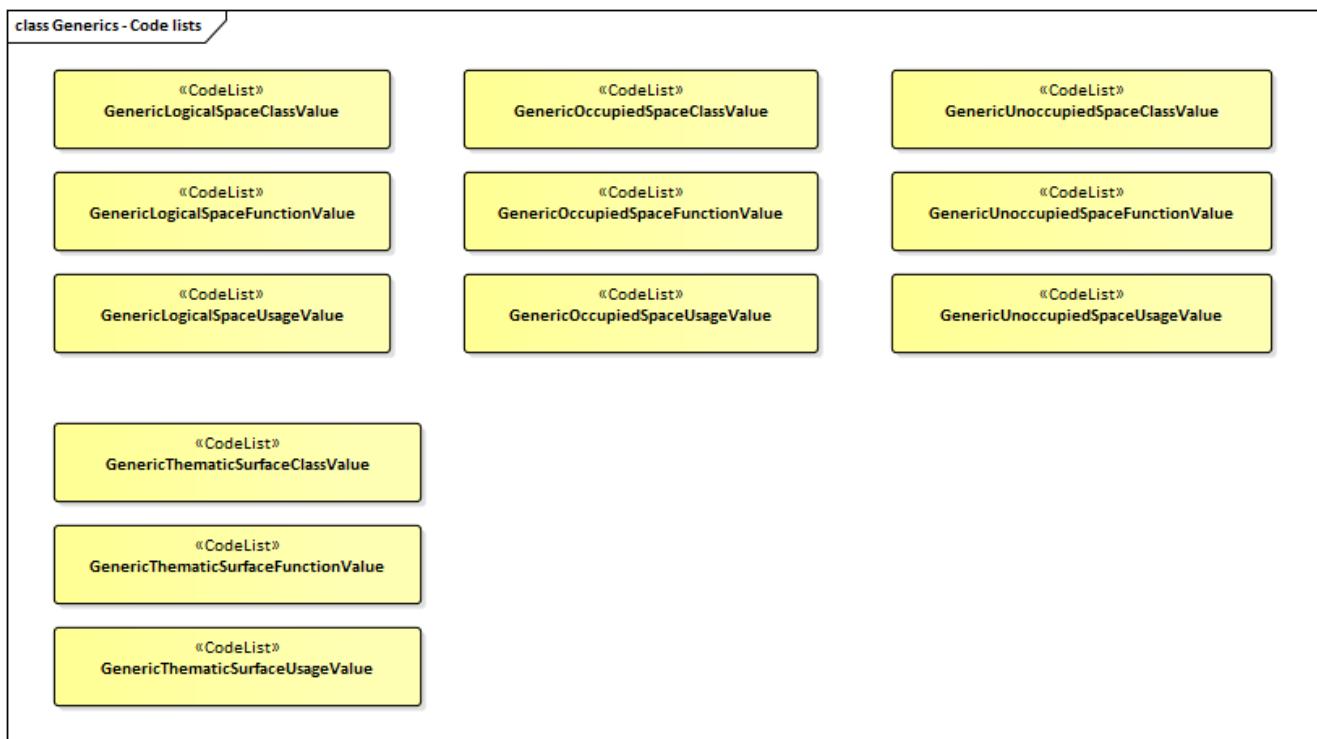


Figure 79. Codelists from the CityGML Generics module.

8.9.5. Examples

8.10. Land Use

Contributors
C. Heazel - first draft

8.10.1. Synopsis

The LandUse module defines objects that can be used to describe areas of the earth's surface dedicated to a specific land use or having a specific land cover with or without vegetation, such as

sand, rock, mud flats, forest, grasslands, or wetlands (i.e. the physical appearance).

8.10.2. Key Concepts

LandUse: An area of the earth's surface dedicated to a specific land use or having a specific land cover with or without vegetation, such as sand, rock, mud flats, forest, grasslands, or wetlands.
A type of [AbstractThematicSurface](#).

8.10.3. Discussion

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 concepts are interlinked and often mixed in practice. Land use objects in CityGML support 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.

Land use objects are represented in the UML model by the top-level feature type [LandUse](#), which is also the only class of the 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. [\[ug_codelist_section\]](#)). 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 the `function` and `usage` attributes can occur multiple times.

The LandUse object is defined for all LOD 0-3 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 [GM_MultiSurface](#), which might be assigned appearance properties like textures or colors (see [\[ug_appearance_section\]](#)).

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 [space boundary](#) only once and by referencing it in the corresponding [City Object](#).

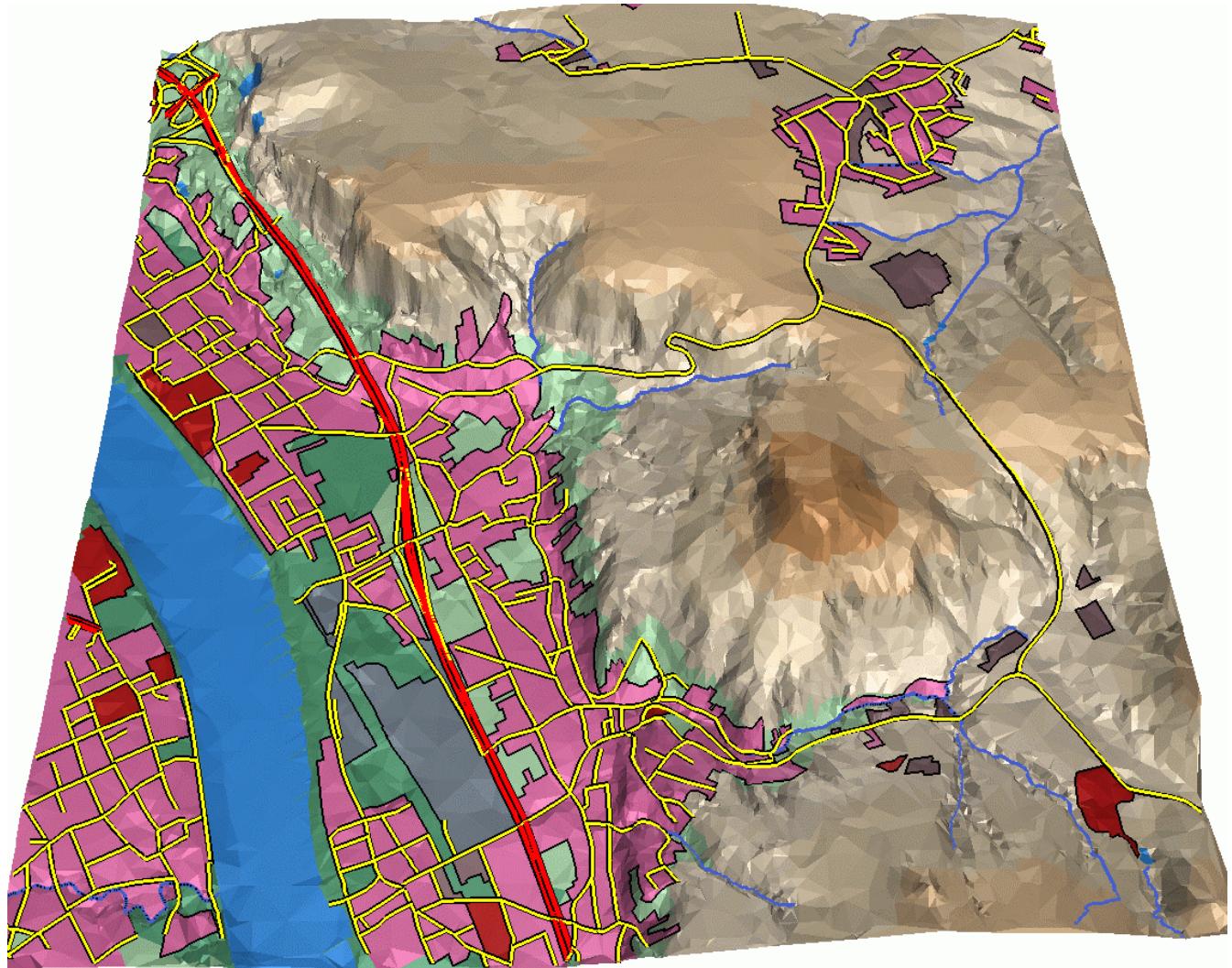


Figure 80. LOD0 regional model consisting of land use objects in CityGML (source: IGG Uni Bonn).

8.10.4. UML Model

The UML diagram of the LandUse module is depicted in [UML diagram of the Land Use Model..](#)

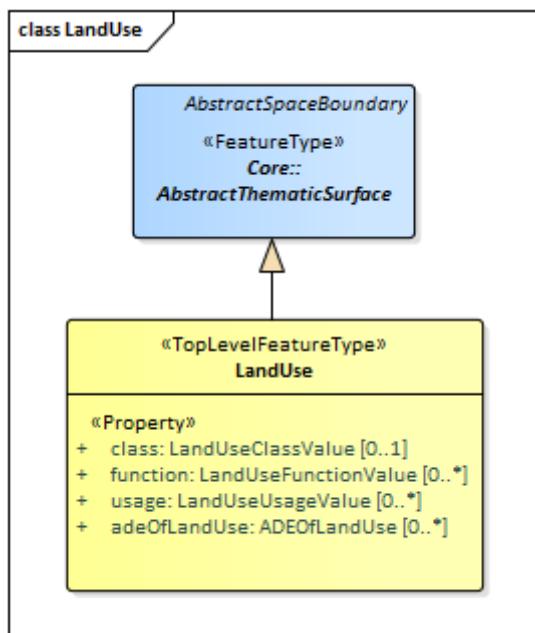


Figure 81. UML diagram of the Land Use Model.

The ADE data types provided for the Land Use module are illustrated in [ADE classes of the CityGML Land Use module..](#)

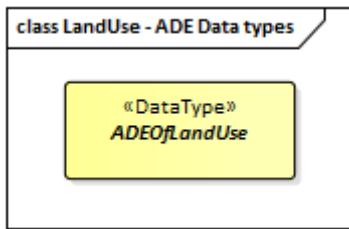


Figure 82. ADE classes of the CityGML Land Use module.

The Code Lists provided for the Land Use module are illustrated in [Codelists from the CityGML Land Use module..](#)

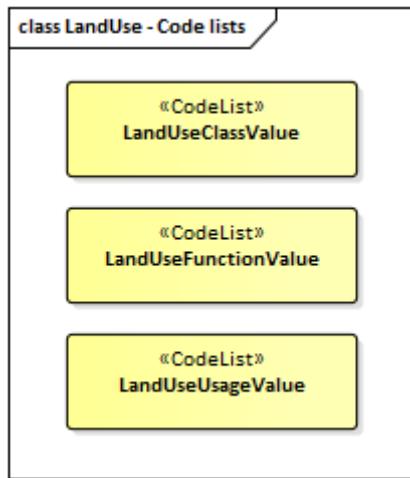


Figure 83. Codelists from the CityGML Land Use module.

8.10.5. Examples

8.11. Point Cloud

Contributors
C. Heazel - first draft

8.11.1. Synopsis

The PointCloud module specifies how to encode the geometry of physical spaces and of thematic surfaces as 3D point clouds. In this way, the building hull, a room within a building or a single wall surface can be spatially represented by a point cloud only.

8.11.2. Key Concepts

PointCloud: A PointCloud is an unordered collection of points that is a sampling of the geometry of a space or space boundary.

A type of [AbstractPointCloud](#).

8.11.3. Discussion

The PointCloud module specifies how to encode the geometry of physical spaces and of thematic surfaces as 3D point clouds. In this way, the building hull, a room within a building or a single wall surface can be spatially represented by a point cloud only. The same applies to all other thematic feature types including transportation objects, vegetation, city furniture, etc. Point clouds can either be provided inline within a CityGML file or as reference to external point cloud files of common file types such as LAS or LAZ.

Point clouds are represented in the UML model by the feature type *PointCloud*, which is also the only class of the PointCloud module.

8.11.4. UML Model

The UML diagram of the PointCloud module is depicted in [UML diagram of the Point Cloud Model..](#)

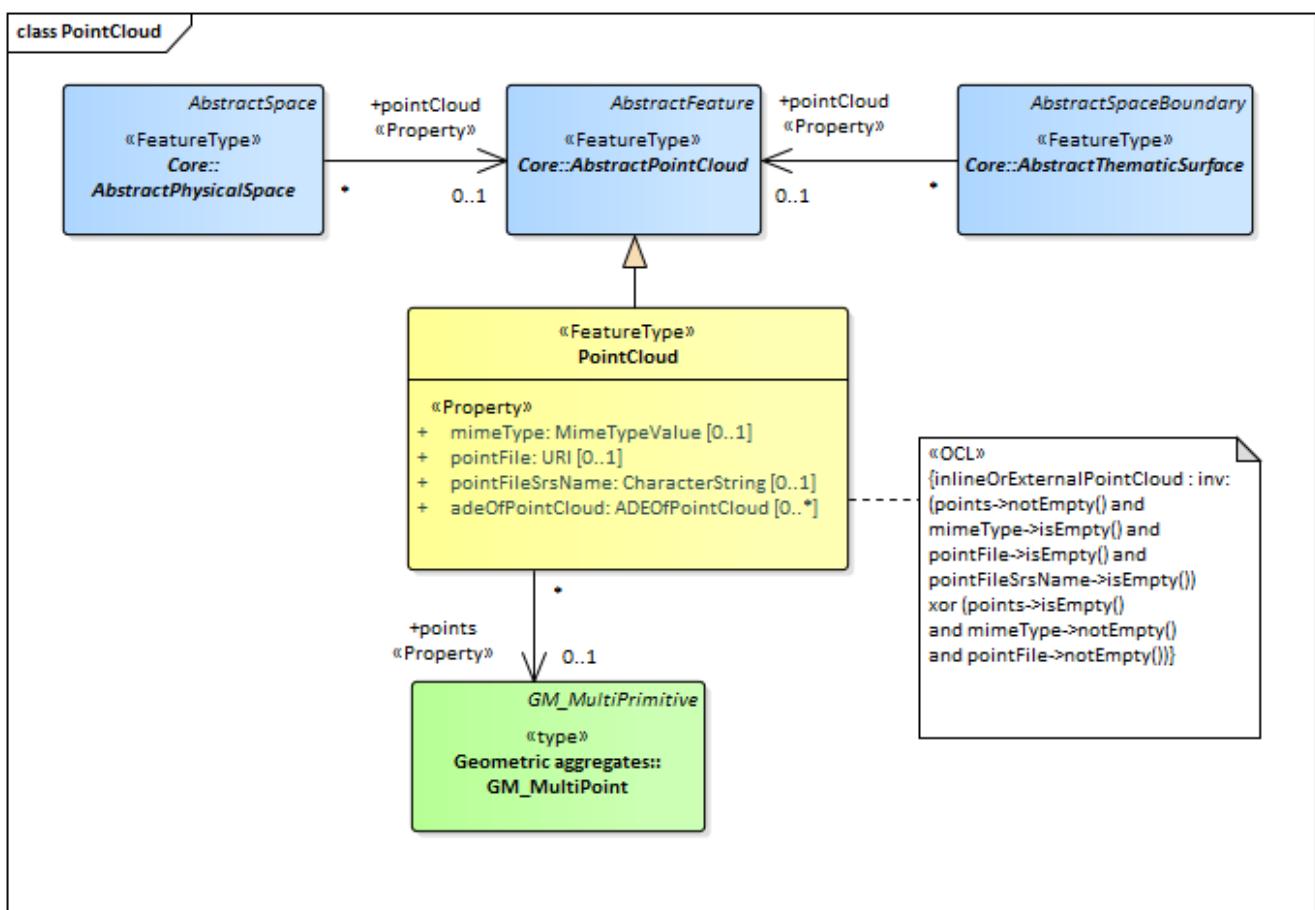


Figure 84. UML diagram of the Point Cloud Model.

The ADE data types provided for the Point Cloud module are illustrated in [ADE classes of the CityGML Point Cloud module..](#)

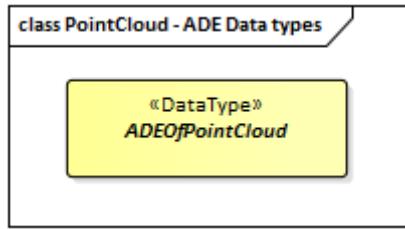


Figure 85. ADE classes of the CityGML Point Cloud module.

8.11.5. Examples

8.12. Relief

Contributors
C. Heazel - first draft

8.12.1. Synopsis

The Relief module supports representation of the terrain. CityGML supports terrain representations at different levels of detail, reflecting different accuracies or resolutions. Terrain may be specified as a regular raster or grid, as a TIN, by break lines, and/or by mass points.

8.12.2. Key Concepts

ReliefFeature: A collection of terrain components representing the Earth's surface, also known as the Digital Terrain Model.

A type of [AbstractSpaceBoundary](#).

AbstractReliefComponent: An element of the terrain surface - either a TIN, a raster or grid, mass points or break lines.

A type of [AbstractSpaceBoundary](#).

TINRelief: A terrain component as a triangulated irregular network.

A type of [AbstractReliefComponent](#).

MassPointRelief: A terrain component as a collection of 3D points.

A type of [AbstractReliefComponent](#).

BreaklineRelief: A terrain component with 3D lines. These lines denote break lines or ridge/valley lines.

A type of [AbstractReliefComponent](#).

RasterRelief: A terrain component as a regular raster or grid.

A type of [AbstractReliefComponent](#).

8.12.3. Discussion

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. In CityGML, the terrain is represented by the

class [ReliefFeature](#) in LOD 0-3. A [ReliefFeature](#) consists of one or more entities of the class [AbstractReliefComponent](#). Its validity may be restricted to a certain [extent](#) defined by an optional [GM_Surface](#). As [ReliefFeature](#) and [AbstractReliefComponent](#) are derivatives of [AbstractCityObject](#), the corresponding attributes and relations are inherited.

The class [ReliefFeature](#) is associated, through [AbstractReliefComponent](#), 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 - [TINRelief](#)),
- by break lines ([BreaklineRelief](#)), or
- by mass points ([MasspointRelief](#)).

The four types are implemented by the corresponding ISO geometries:

- grids by [CV_DiscreteGridPointCoverage](#),
- break lines by [GM_MultiCurve](#),
- mass points by [GM_MultiPoint](#) and
- TINs either by [GM_TriangulatedSurface](#) or by [GM_Tin](#).

In case of [GM_TriangulatedSurfaces](#), the triangles are given explicitly while in case of [GM_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.

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 ([Figure 35](#)). 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.

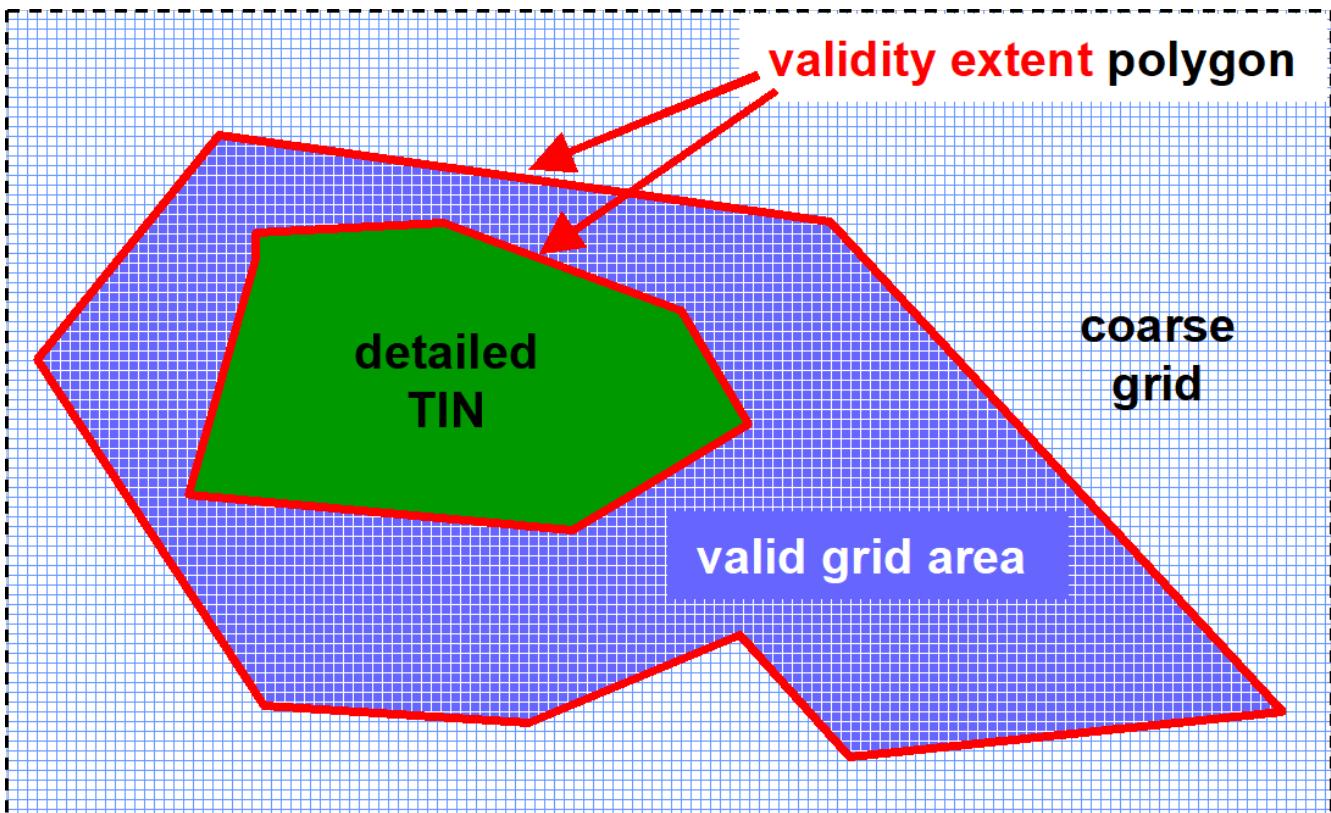


Figure 86. Nested DTMs in CityGML using validity extent polygons (graphic: IGG Uni Bonn).

Accuracy and resolution of the DTM are not necessarily dependent on features of other CityGML extension 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 [TerrainIntersectionCurve](#) TIC. 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.

8.12.4. UML Model

The UML diagram of the Relief module is depicted in [UML diagram of Relief module..](#)

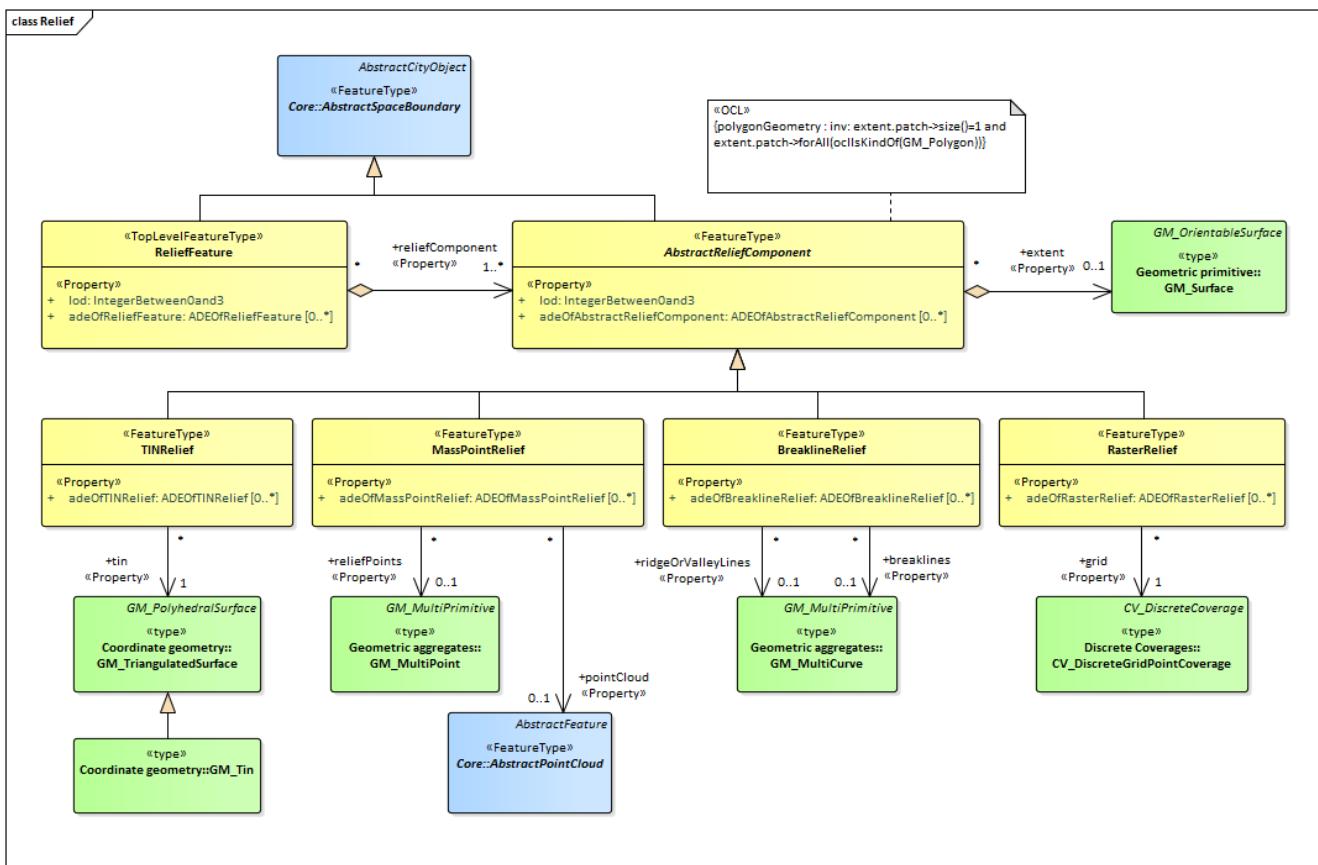


Figure 87. UML diagram of Relief module.

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

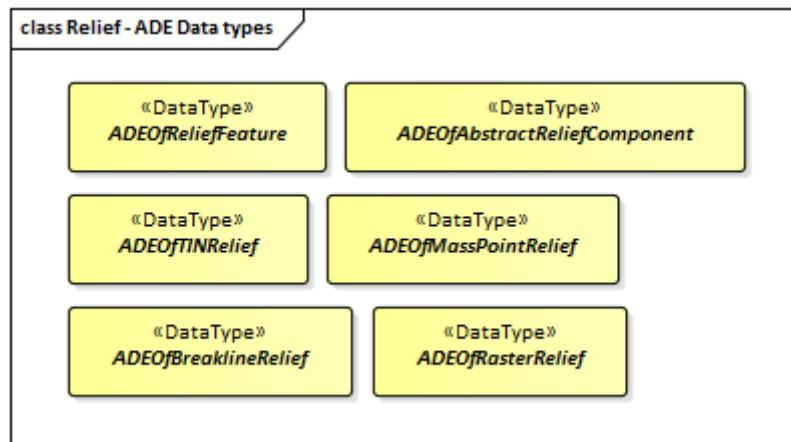


Figure 88. ADE classes of the CityGML Relief module.

8.12.5. Examples

8.13. Transportation

Contributors

C. Heazel - first draft

8.13.1. 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.

8.13.2. Key Concepts

Railway: Railway represents routes that are utilized by rail vehicles like trams or trains. A type of [AbstractTransportationSpace](#).

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. A type of [AbstractTransportationSpace](#).

Square: A Square is an open area commonly found in cities (e.g. a plaza, market square). A type of [AbstractTransportationSpace](#).

Track: A Track is a small path mainly used by pedestrians. A type of [AbstractTransportationSpace](#).

Waterway: A Waterway is a transportation space used for the movement of vessels upon or within a water body. A type of [AbstractTransportationSpace](#).

Abstract Transportation Space: [AbstractTransportationSpace](#) defines the properties and associated concepts which are common to Tracks, Roads, Railways, Squares, and Waterways. The associated concepts include: A type of [AbstractUnoccupiedSpace](#).

- **Traffic Space:** A [TrafficSpace](#) is a space in which traffic (transportation) takes place. This is more than a geometry. It includes properties which may be of interest to anyone who wishes to traverse this space. This includes properties such as overhead clearance, surface type, and direction of travel. A type of [AbstractUnoccupiedSpace](#).
- **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. A type of [AbstractThematicSurface](#).
- **Hole:** 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. A type of [AbstractUnoccupiedSpace](#).
- **Auxillary Traffic Space:** a space within the transportation space not intended for traffic purposes. A type of [AbstractUnoccupiedSpace](#).

8.13.3. 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. [Figure 36](#) illustrates the different transportation classes.

NOTE Need to add a waterway to figure 57.

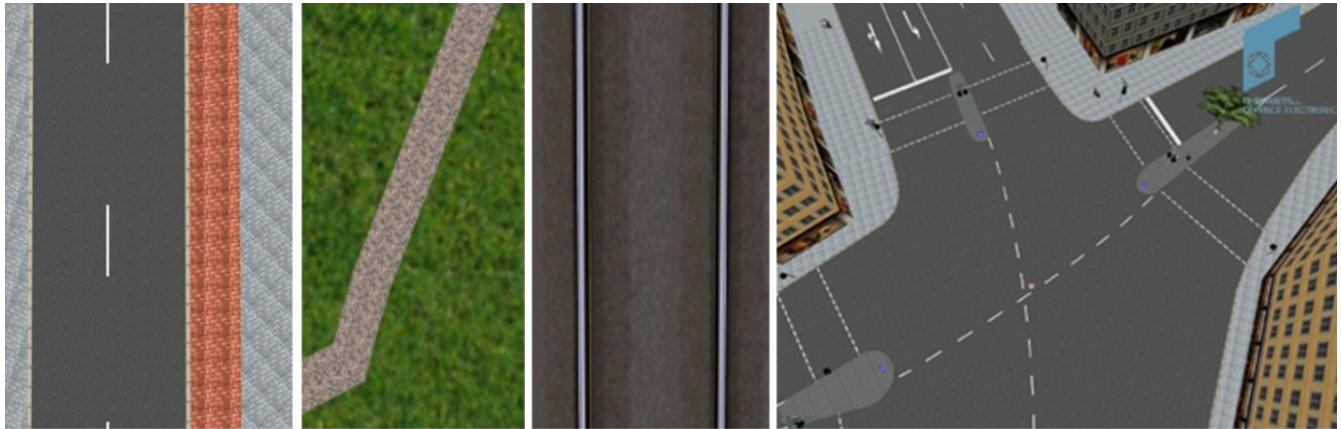


Figure 89. Transportation classes (from left to right: examples of road, track, rail, waterway, and square) (source: Rheinmetall Defence Electronics)

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.

[[figure-58](#)] 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 Auxillary Traffic Spaces below the raised flower beds.

[[[figure-58](#)],[Figure 1](#)] .Example LOD2 Traffic Spaces in CityGML: a road, which is the aggregation of TrafficAreas and AuxiliaryTrafficAreas (source: City of Solingen, IGG Uni Bonn) image::figures/Figure_58.png[align="center"]

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.

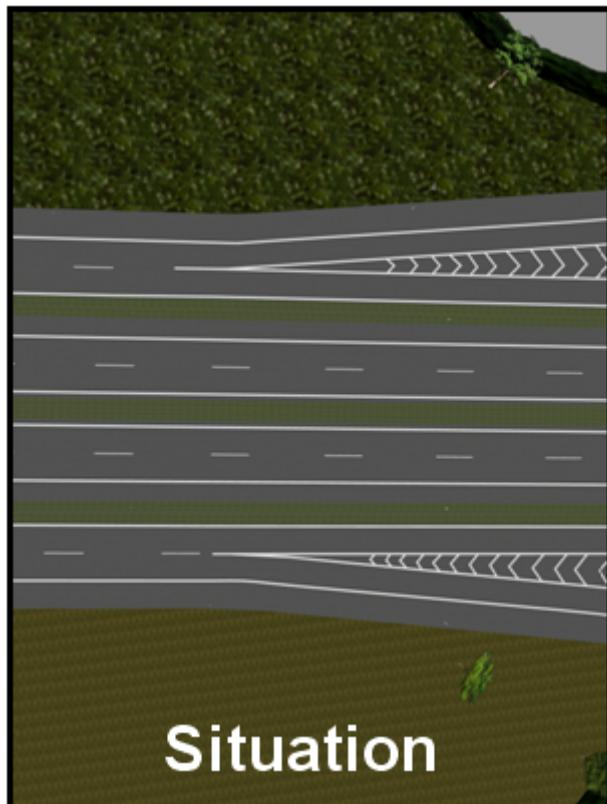
8.13.4. Level of Detail

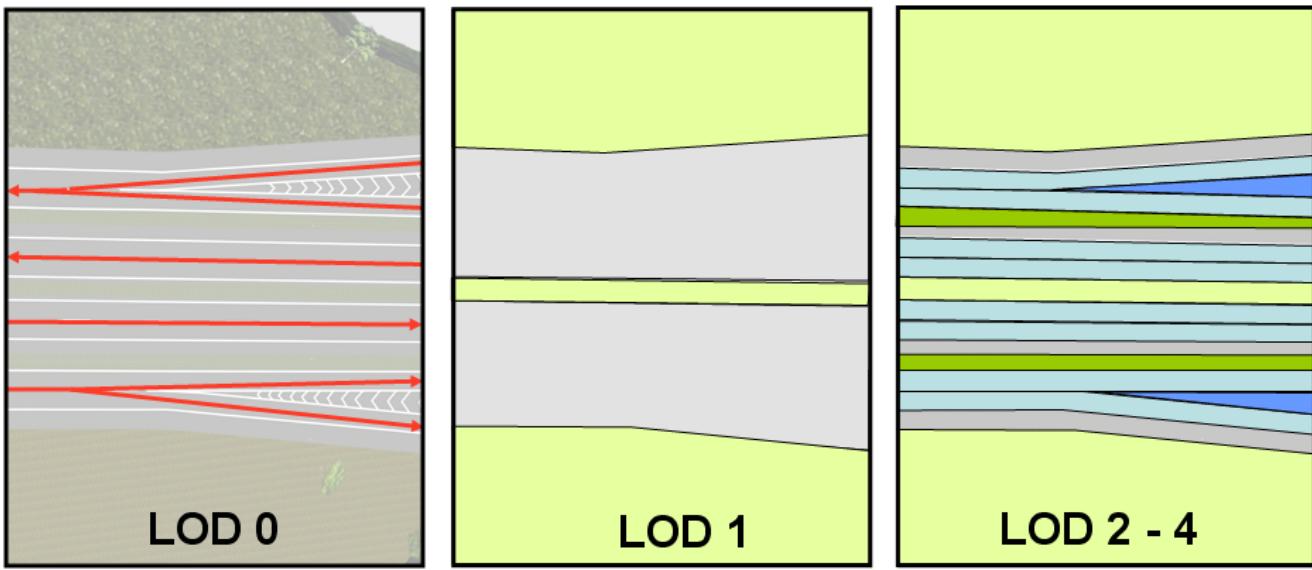
The geometrical representation of the Transportation objects varies through the different levels of detail.

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 LOD3, 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 37](#).





TransportationComplex provides linear network with line objects

→ line objects

TransportationComplex provides surface geometry describing the actual shape of the object

■ TransportationComplex (Surface geometry)
■ Terrain surface

Surface geometry is devided thematically into TrafficAreas, like:

- Traffic – cars
- Traffic – emergency lane
- Traffic – restricted area
- Auxiliary - grass

Figure 90. Transportation Concepts in LOD0, 1, and 2-3 (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](#). 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](#). These extensions may be country or user-specific (especially for country-specific road signs and signals).

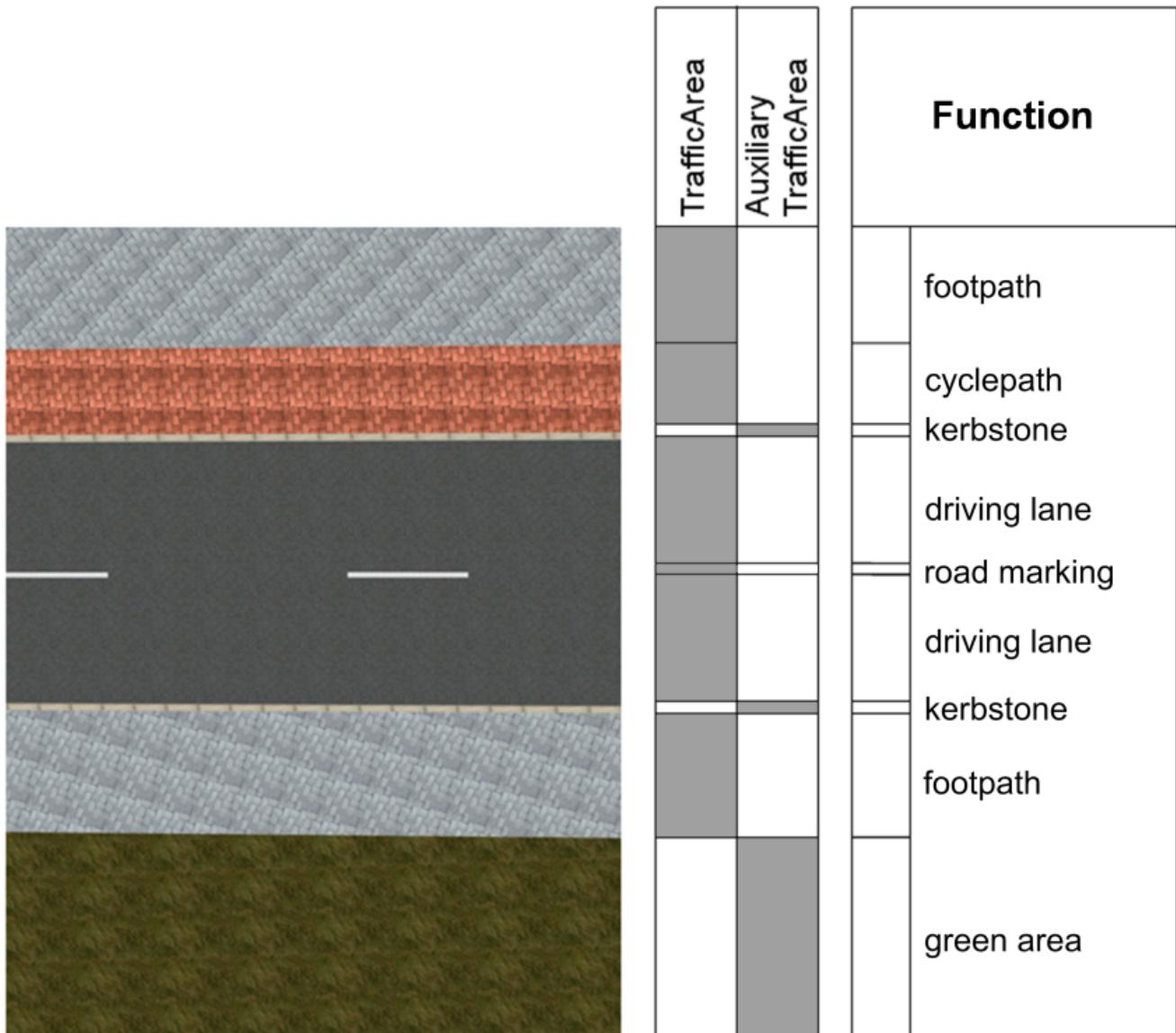


Figure 91. *TransportationComplex* in LOD 2-3: representation of a road with a complex cross-section profile (example shows urban road) (source: Rheinmetall Defence Electronics).

8.13.5. UML Model

The UML diagram of the Transportation module is depicted in [UML diagram of the Transportation Model.](#)

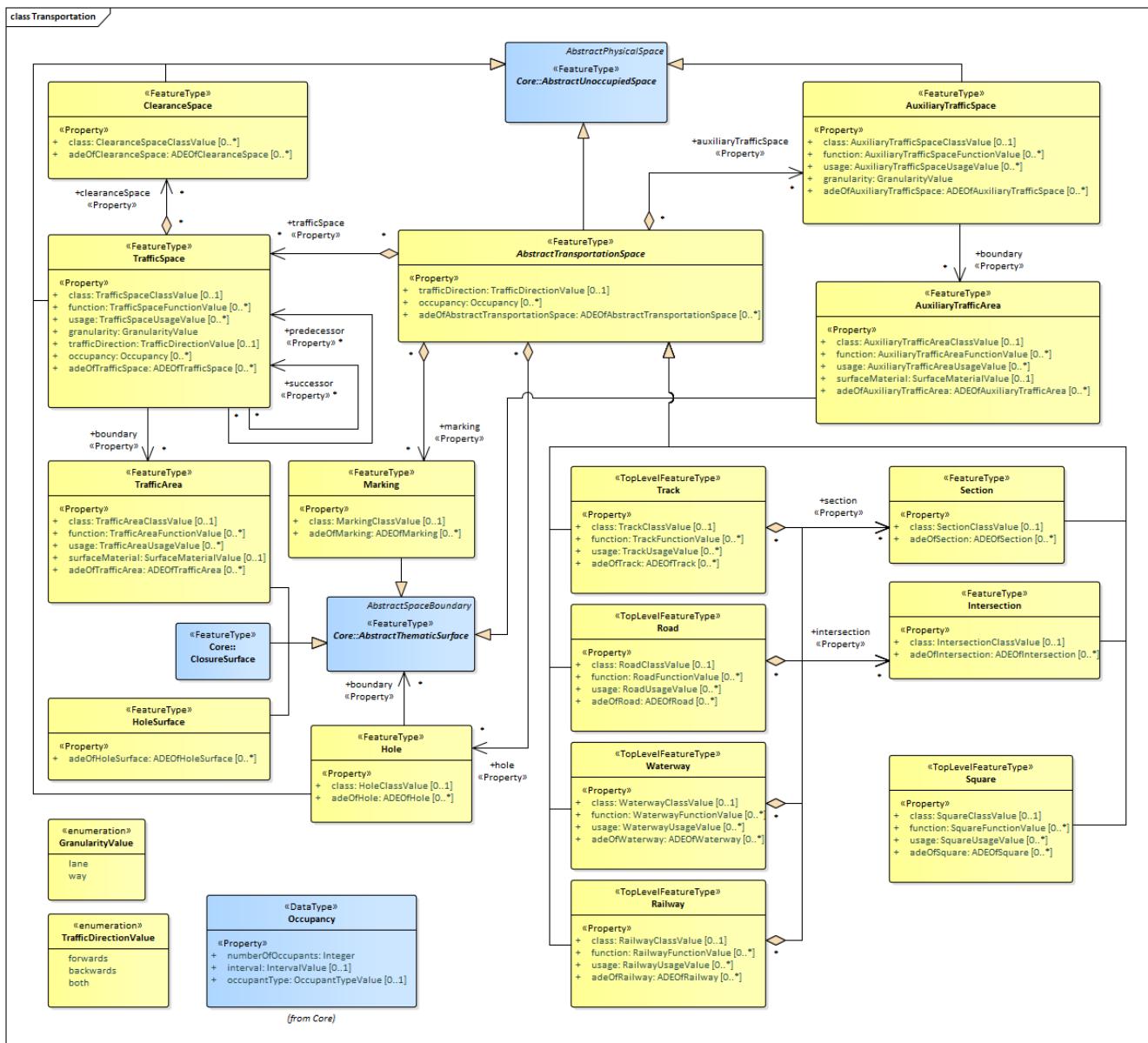


Figure 92. UML diagram of the Transportation Model.

The ADE data types provided for the Transportation module are illustrated in [ADE classes of the CityGML Transportation module..](#)

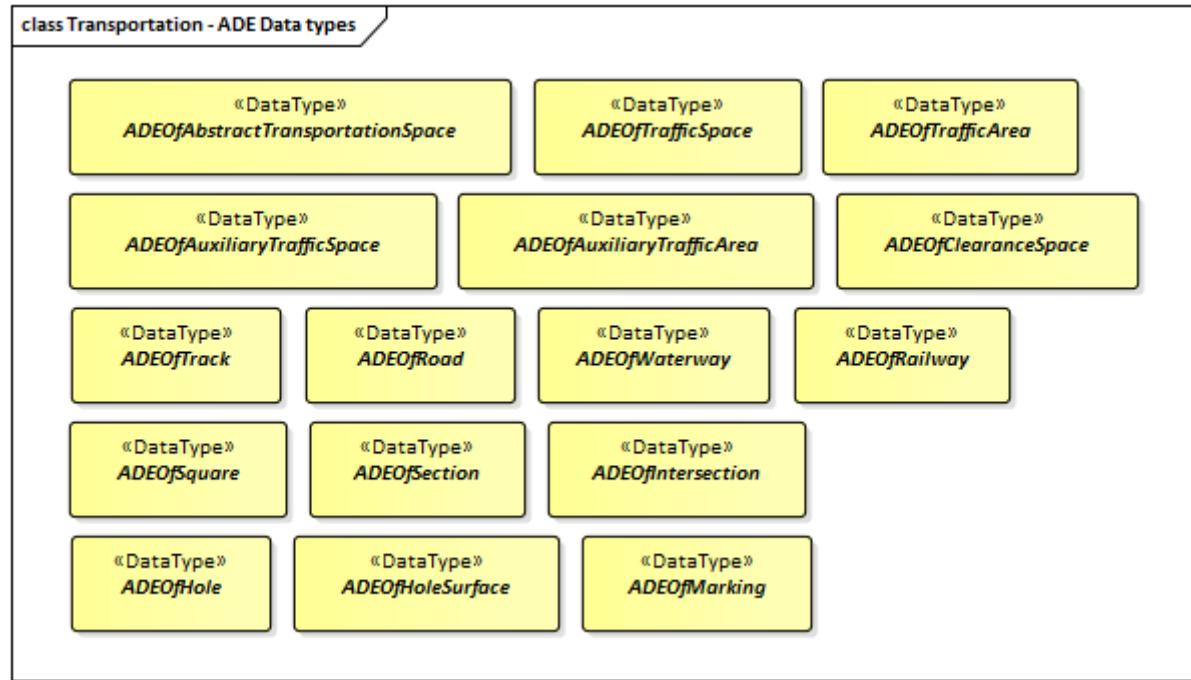


Figure 93. ADE classes of the CityGML Transportation module.

The Code Lists provided for the Transportation module are illustrated in [Codelists from the CityGML Transportation module..](#)

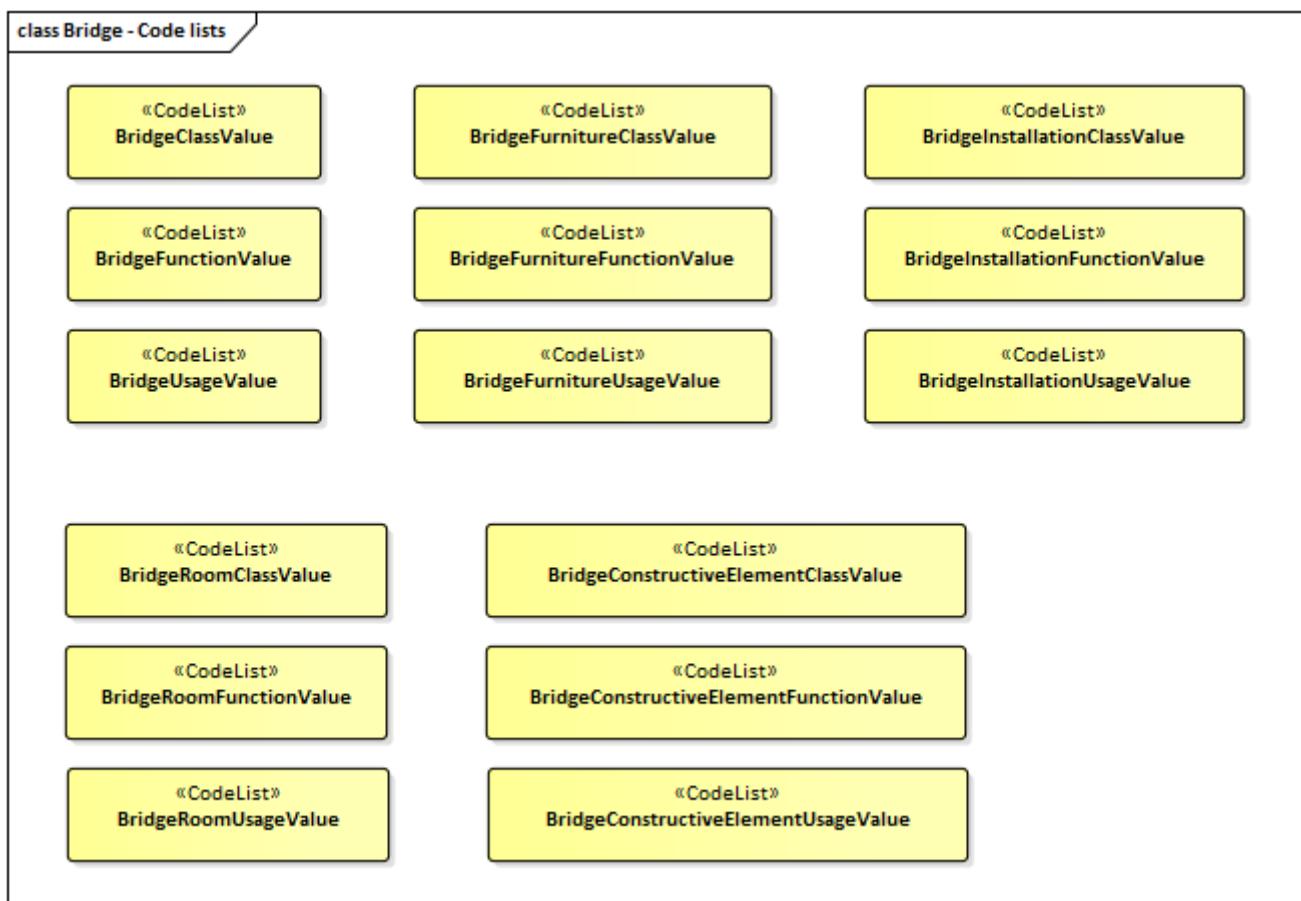


Figure 94. Codelists from the CityGML Transportation module.

8.13.6. Examples

Table 7. 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.

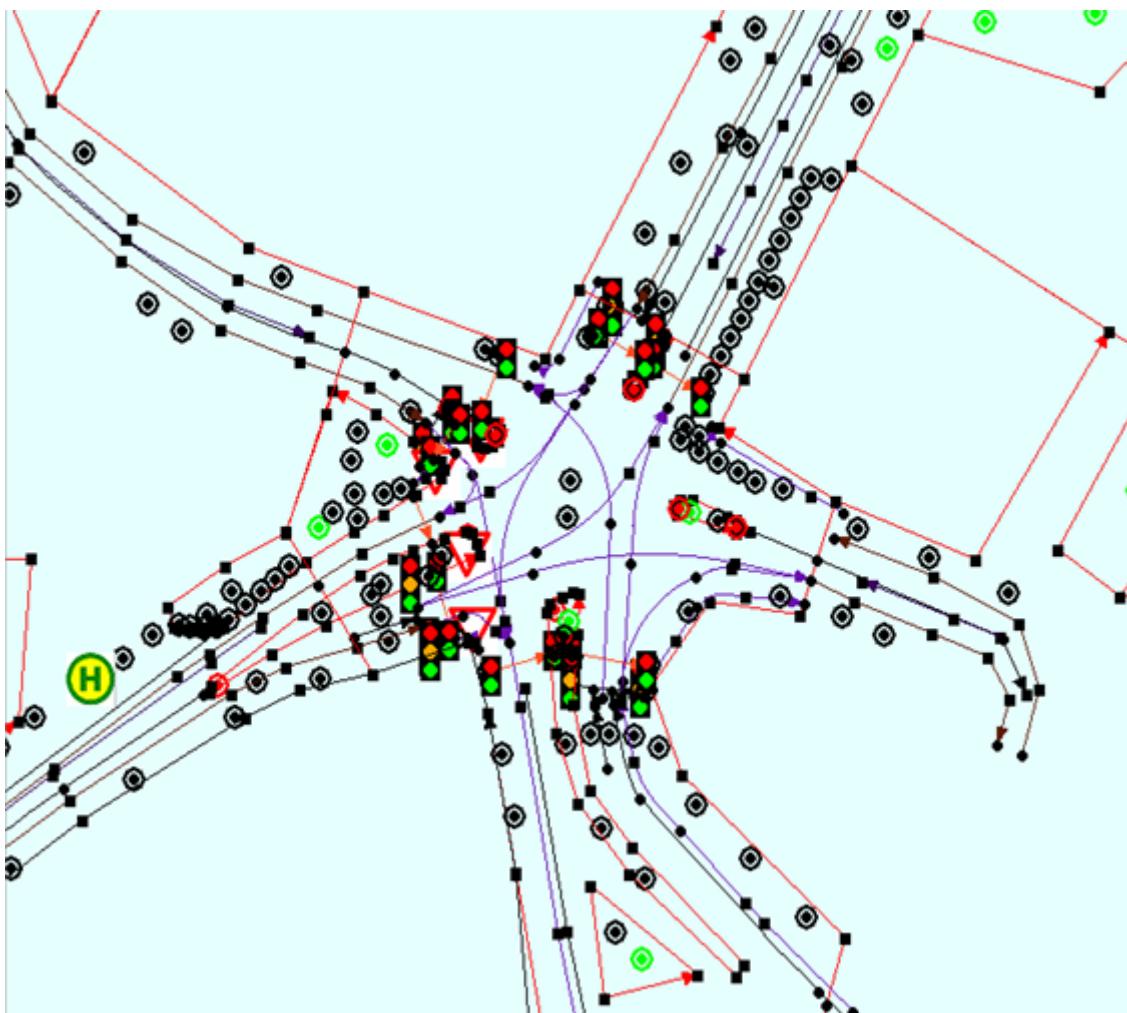




Figure 95. Complex urban intersection (left: linear transportation network with surface descriptions and external references, right: generated scene) (source: Rheinmetall Defence Electronics).

8.14. Tunnel Model

Contributors

C. Heazel - first draft

NOTE This section uses tables to organize multiple images into rows and columns. The null.png image serves to associate the caption with a figure rather than with the table. This is a kludge, but seems to work. Is there a better way?

8.14.1. Synopsis

The Tunnel module supports representation of thematic and spatial aspects of tunnels, tunnel parts, tunnel installations, and interior tunnel structures.

8.14.2. Key Concepts

Abstract Tunnel: AbstractTunnel is an abstract superclass representing the common attributes and associations of the classes Tunnel and TunnelPart.

A type of [AbstractConstruction](#).

Tunnel: A Tunnel represents a horizontal or sloping enclosed passage way of a certain length, mainly underground or underwater.

A type of [AbstractTunnel](#).

Tunnel Part: A TunnelPart is a physical or functional subdivision of a Tunnel. It would be considered a Tunnel, if it were not part of a collection of other TunnelParts.

A type of [AbstractTunnel](#).

Hollow Space: A HollowSpace is a space within a Tunnel or TunnelPart intended for certain functions (e.g. transport or passage ways, service rooms, emergency shelters). A HollowSpace is bounded physically and/or virtually (e.g. by ClosureSurfaces or GenericSurfaces).

A type of [AbstractUnoccupiedSpace](#).

Tunnel Installation: A TunnelInstallation is a permanent part of a Tunnel (inside and/or outside) which does not have the significance of a TunnelPart. In contrast to TunnelConstructiveElement, a TunnelInstallation is not essential from a structural point of view. Examples are stairs, antennas or railings.

A type of [AbstractInstallation](#).

Tunnel Furniture: A TunnelFurniture is an equipment for occupant use, usually not fixed to the tunnel.

A type of [AbstractFurniture](#).

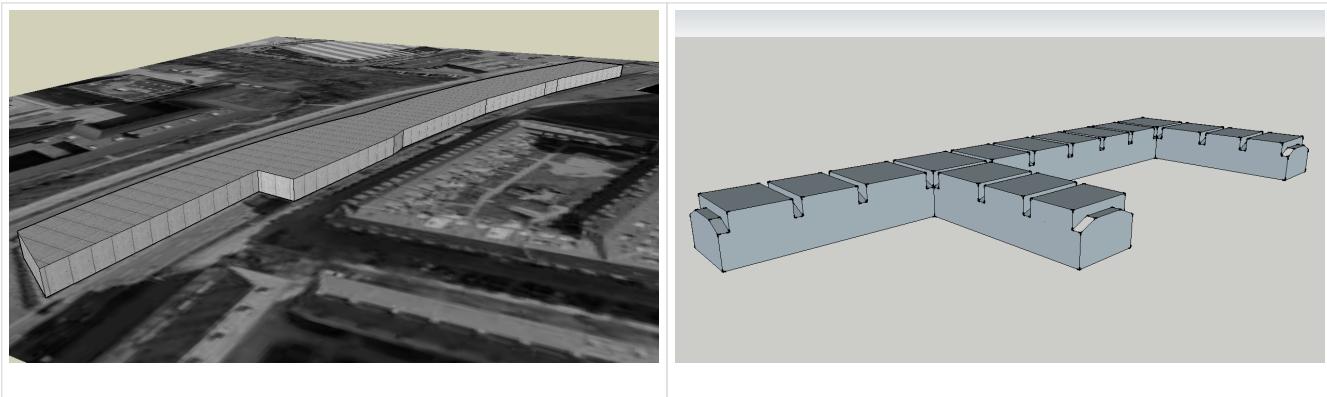
Tunnel Constructive Element: A TunnelConstructiveElement is an element of a Tunnel which is essential from a structural point of view. Examples are walls, slabs, beams.

A type of [AbstractConstructiveElement](#).

8.14.3. Discussion

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, LOD0 to LOD3. The tunnel model of CityGML is defined by the thematic extension module Tunnel. [Figure 39](#) provides examples of tunnel models for each LOD.

NOTE This figure is from CityGML 2.0. It needs to be updated for CityGML 3.0



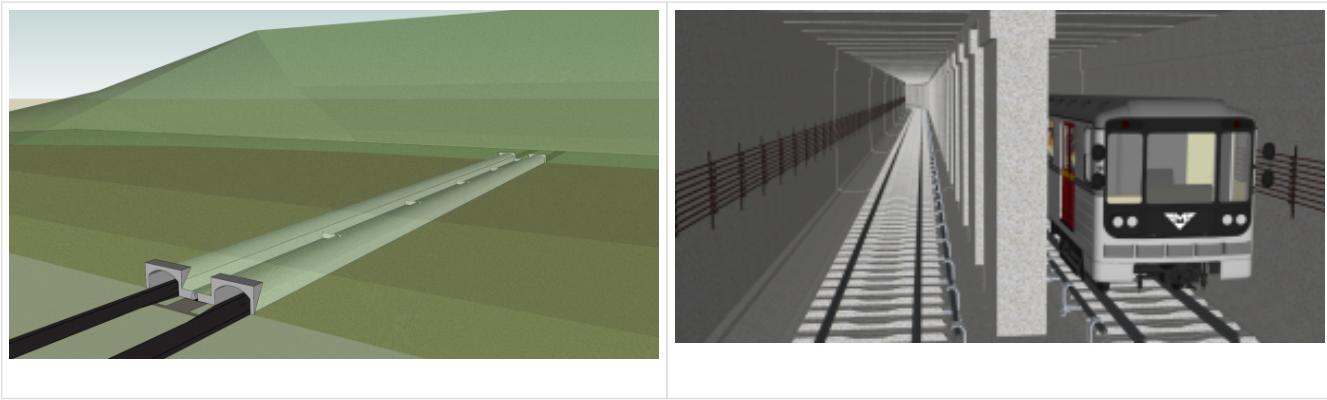


Figure 96. Examples for tunnel models in LOD1 (upper left), LOD2 (upper right), LOD3 (lower left) and LOD4 (lower right) (source: Google 3D warehouse).

Tunnel Class

The Tunnel class is one of the two subclasses of `AbstractTunnel`. If a tunnel only consists of one (homogeneous) 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 `TunnelParts` (see [Figure 40](#)). The geometry and non-spatial properties of the central part of the tunnel should be represented in the aggregating Tunnel feature.

Tunnel Part Class

If sections of a tunnel differ in geometry and / or attributes, the tunnel can be separated into parts (see [Figure 40](#)). Like `Tunnel`, the class `TunnelPart` is derived from `AbstractTunnel` and inherits all attributes of `AbstractTunnel`. A `TunnelPart` object should be uniquely related to exactly one tunnel or tunnel part object.



Figure 97. Example of a tunnel modeled with two tunnel parts (source: Helmut Stracke).

Abstract Tunnel Class

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),
2. the different functions (function), and
3. the usage (usage).

Spanning the different levels of detail, the tunnel model differs in the complexity and granularity of the geometric representation and the thematic structuring of the model into components with a special semantic meaning. This is illustrated in [Figure 41](#), showing the same tunnel in four different LODs. Some properties of the class `AbstractTunnel` are also associated with certain LODs.

NOTE This figure is from CityGML 2.0 and needs to be updated to the CityGML 3.0 LODs.

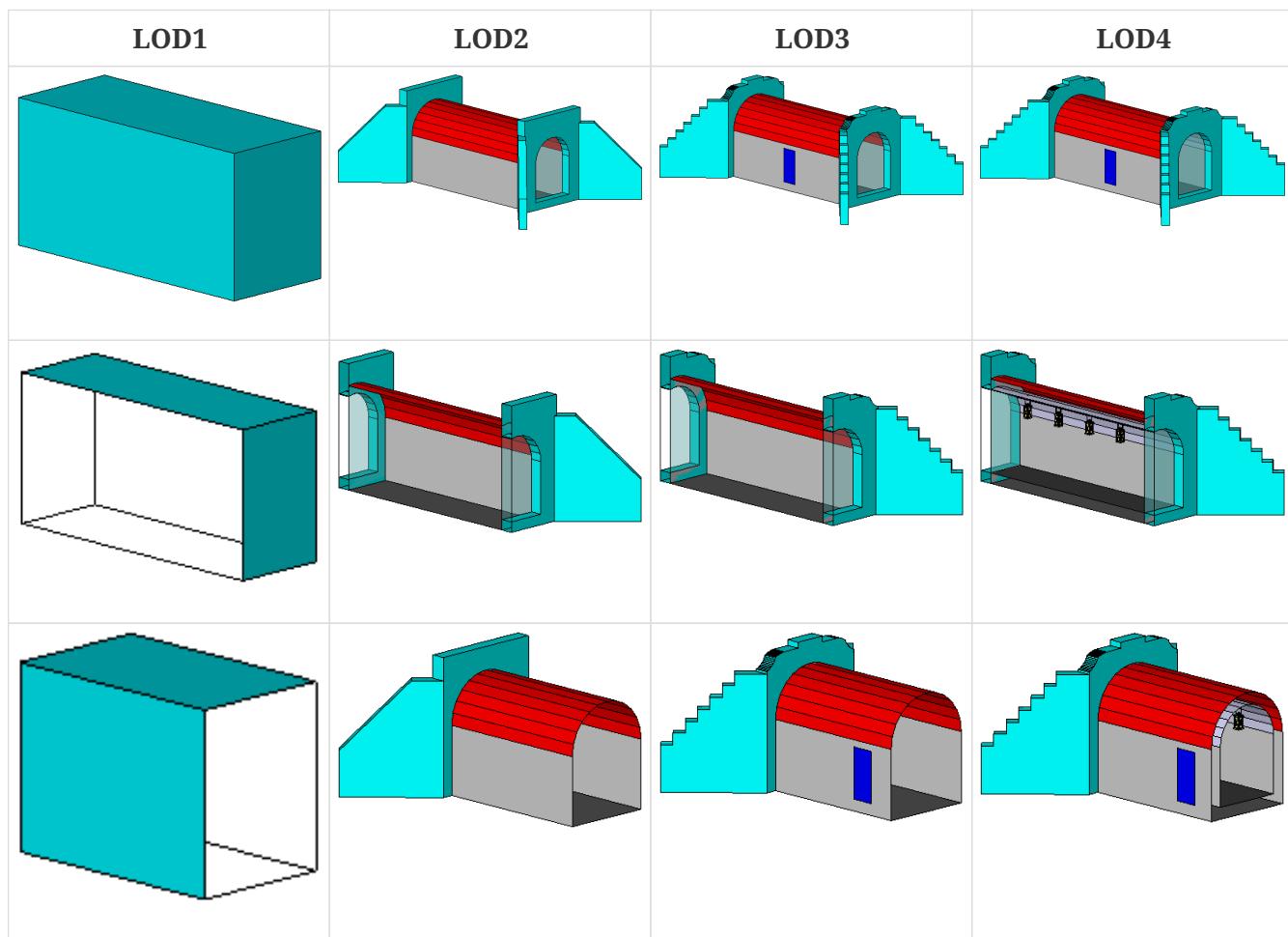


Figure 98. Tunnel model in LOD1 – LOD4 (source: Karlsruhe Institute of Technology (KIT)).

[Table 8](#) 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 `GM_Solid` geometry and/or a `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.

NOTE | Update the LOD to CityGML 3.0 approach.

Table 8. Semantic themes of the class *_AbstractTunnel*

Geometric / semantic theme	Property type	LOD1	LOD2	LOD3	LOD4
Building footprint and roof edge	MultiSurfaceType	•			
	Volume part of the tunnel shell	SolidType	•	•	•
•	Surface part of the tunnel shell	MultiSurfaceType	•	•	•
•	Terrain intersection curve	MultiCurveType	•	•	•
•	Curve part of the tunnel shell	MultiCurveType		•	•
•	Tunnel parts	TunnelPartType	•	•	•
•	Boundary surfaces	AbstractBoundarySurfaceType		•	•
•	Tunnel installations	TunnelInstallationType		•	•
•	Openings	AbstractOpeningType			•
•	Hollow spaces	HollowSpaceType			

Hollow Space Class

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 [GM_Solid](#) (lod3Solid property). However, if the topological correctness of the boundary cannot be guaranteed, the geometry can alternatively be given as a [GM_MultiSurface](#) (lod3MultiSurface property). The surface normals of the outer shell of [GM_Solid](#) must point outwards. This is important if appearances 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 ClosureSurface, cf. [Construction](#)).

Tunnel Installation Class

A TunnelInstallation is permanent part of a Tunnel (inside or outside) which does not have the significance of a TunnelPart. In contrast to TunnelConstructiveElement, a TunnelInstallation is not essential from a structural point of view. Examples are stairs, antennas or railings. 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 be specified in a code list.

For the geometrical representation of a TunnelInstallation, an arbitrary geometry object from the subset of ISO classes shown in [Table 3](#) can be used. Alternatively, the geometry may be given as ImplicitGeometry object (see [Prototypic Objects / Scene Graph Concepts](#)). 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. The visible surfaces of a tunnel installation can be semantically classified using the concept of boundary surfaces. A TunnelInstallation object should be uniquely related to exactly one tunnel or tunnel part object.

Tunnel Furniture Class

Hollow spaces may have TunnelFurniture. A TunnelFurniture is a movable part of a hollow space. A TunnelFurniture 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 furniture features.

Tunnel Constructive Element Class

A TunnelConstructiveElement is an element of a Tunnel which is essential from a structural point of view. Examples are walls, slabs, and beams. A TunnelConstructiveElement optionally has attributes **class**, **function** and **usage**. The attribute **class** - which can only occur once - represents a general classification of the element. With the attributes **function** and **usage**, nominal and real functions of a tunnel constructive element can be described. For all three attributes the list of feasible values can be specified in a code list.

8.14.4. Level of Detail

NOTE The term Boundary Surface is never defined.

The geometric representation and semantic structure of an [AbstractTunnel](#) is shown in [Figure 42](#). The model is successively refined from LOD0 to LOD3. 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. An object can be represented simultaneously in different LODs by providing distinct geometries for the corresponding LODs.

In LOD1, a tunnel model consists of a geometric representation of the tunnel volume. Optionally, a [GM_MultiCurve](#) representing the [TerrainIntersectionCurve](#) can be specified. The geometric representation is refined in LOD2 by additional [GM_MultiSurface](#) and [GM_MultiCurve](#) geometries.

In LOD2 and higher LODs the outer structure of a tunnel can also be differentiated semantically by the class [TunnelInstallation](#). The [TunnelInstallation](#) class is used for non-structural tunnel elements, like outer stairs, which strongly affect the outer appearance of a tunnel. A TunnelInstallation may have the attributes class, function and usage.

The visible surface of a hollow space is represented geometrically as a [GM_MultiCurve](#) or [GM_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.

8.14.5. UML Model

The UML diagram of the Tunnel module is depicted in [Figure 42](#). The Tunnel module inherits concepts from the Construction module (cf. [Construction](#)). The Construction module defines objects that are common to all types of construction, such as the different surface types and the openings.

The UML diagram of the tunnel model is shown in [Figure 42](#). The pivotal class of the model is [AbstractTunnel](#), which is a subclass of the thematic class [AbstractConstruction](#) (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](#) such as the feature properties (`name` etc.) and the CityGML specific properties like [ExternalReferences](#) (cf. [\[ug_model_core_section\]](#)). Further properties not explicitly covered by [AbstractTunnel](#) may be modelled as generic attributes provided by the CityGML Generics module (cf. [Generics](#)) or using the CityGML Application Domain Extension mechanism (cf. [\[ug_model_ade_section\]](#)).

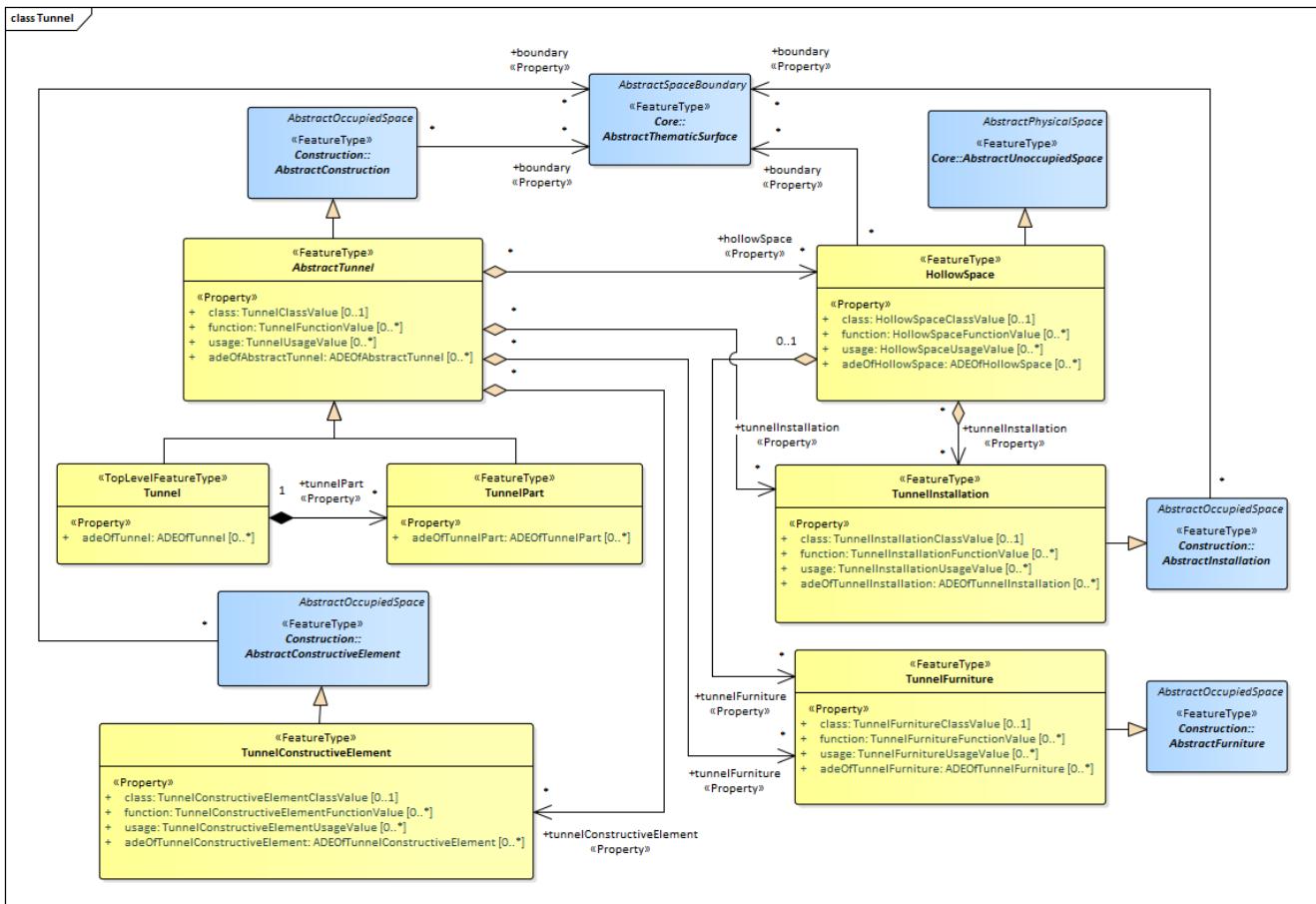


Figure 99. UML diagram of the Tunnel Model.

The ADE data types provided for the Tunnel module are illustrated in Figure 43.

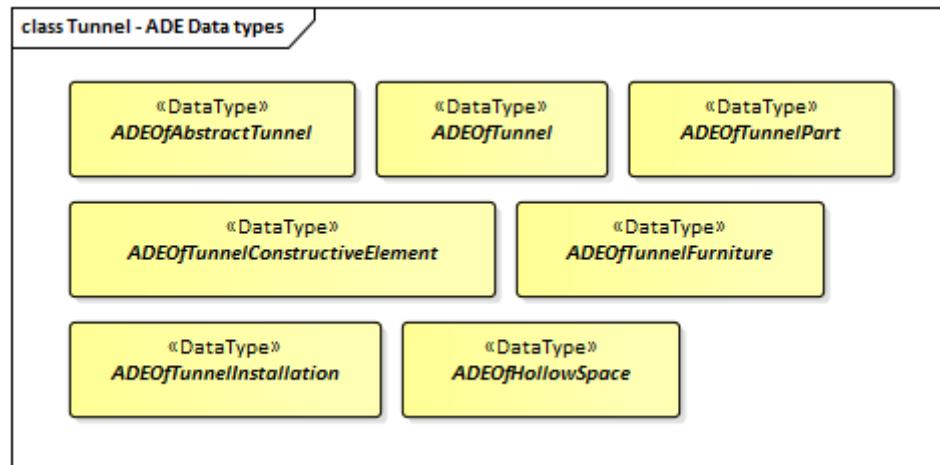


Figure 100. ADE classes of the CityGML Tunnel module.

The Code Lists provided for the Tunnel module are illustrated in Figure 44.

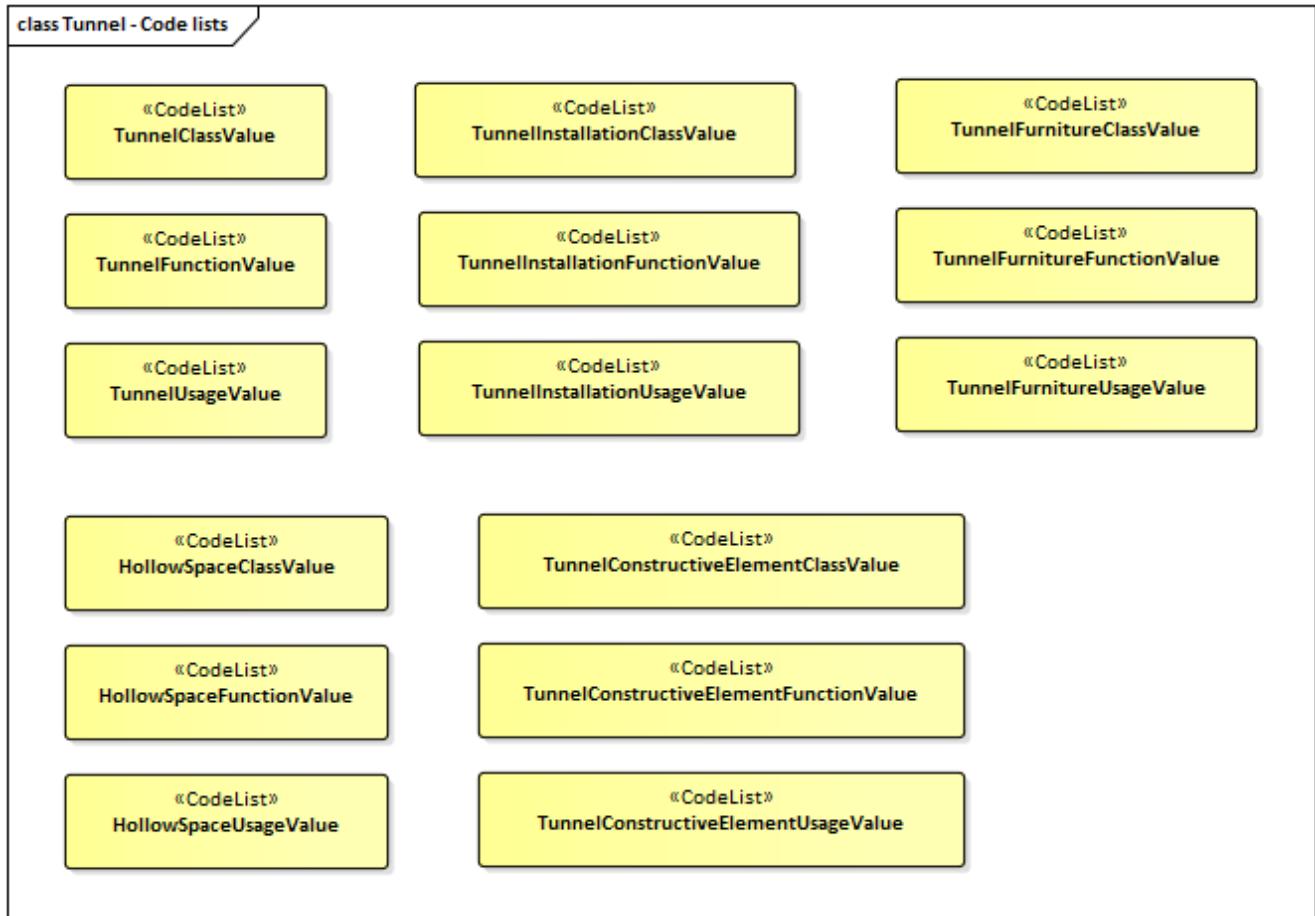


Figure 101. Codelists from the CityGML Tunnel module.

8.14.6. Examples

The example in [Figure 45](#) shows a pedestrian underpass in the city centre of Karlsruhe, Germany. On the left side of [Figure 45](#), 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 [Figure 46](#) from a different perspective. The camera is positioned in front of the left entrance (black dashed rectangle in [Figure 45](#)) and pointing in the direction of the right entrance (white dashed rectangle in [Figure 45](#)). On the right side of [Figure 46](#), 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.

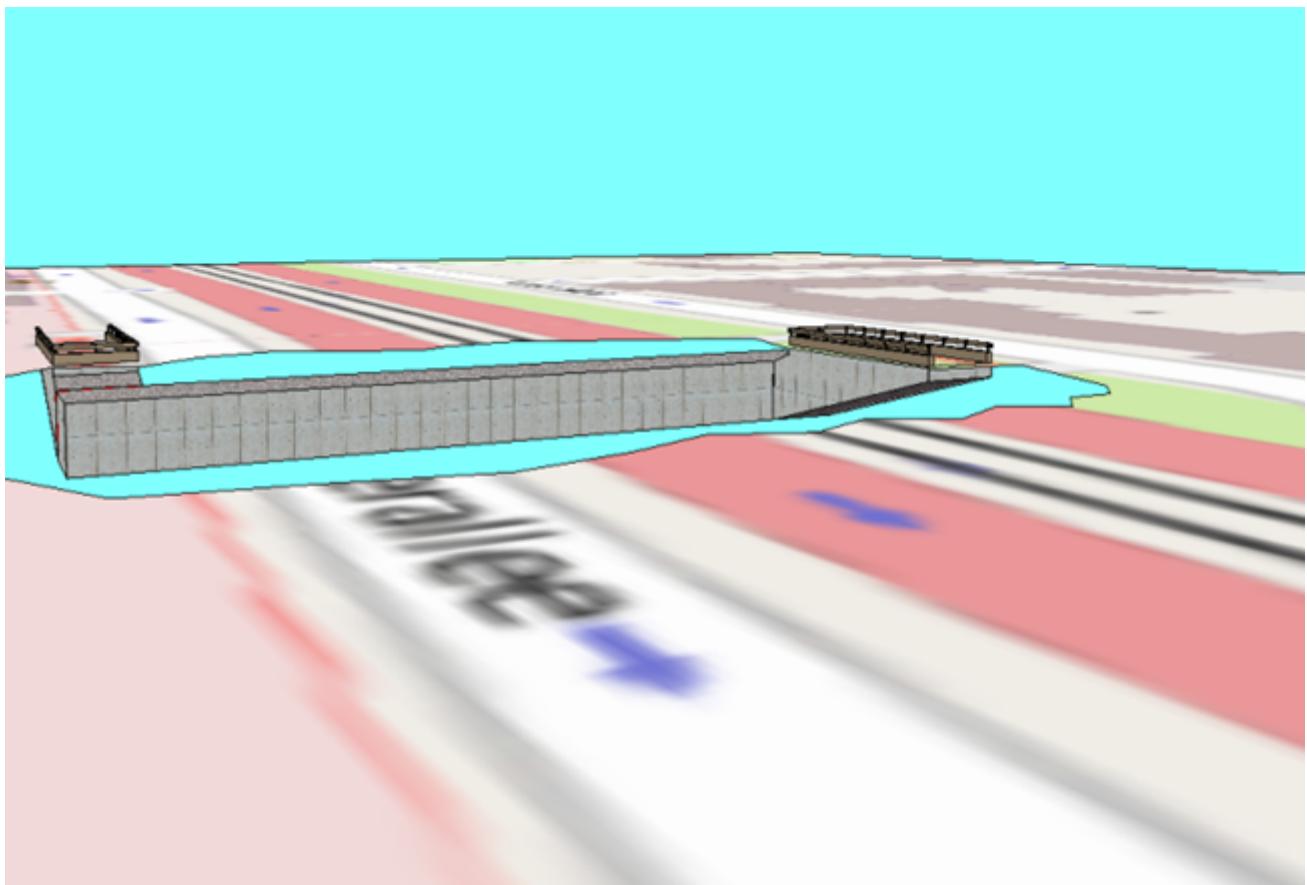


Figure 102. Example of a tunnel modeled in LOD3 (real situation on the left side; CityGML model on the right side) (source: Karlsruhe Institute of Technology (KIT), courtesy of City of Karlsruhe).



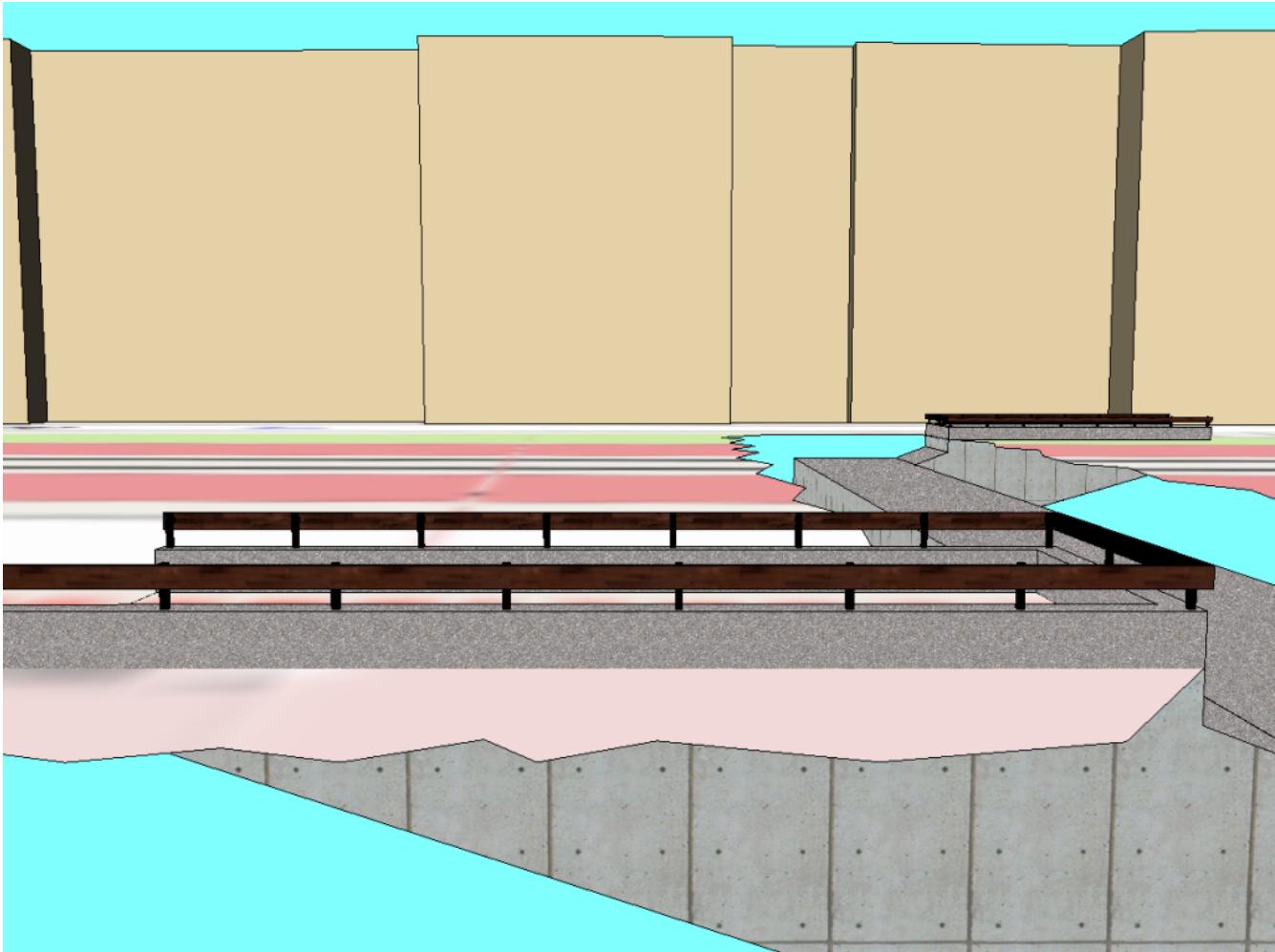


Figure 103. The same LOD3 tunnel shown from a different perspective. The camera is positioned in front of the left entrance and pointing in the direction of the right entrance. (real situation on the left side; CityGML model on the right side). The model on the right also includes an LOD1 representation 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.

8.15. Vegetation

Contributors

C. Heazel - first draft

8.15.1. 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.

8.15.2. Key Concepts

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

A type of [AbstractOccupiedSpace](#).

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

A type of [AbstractOccupiedSpace](#).

8.15.3. 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. concerning 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.

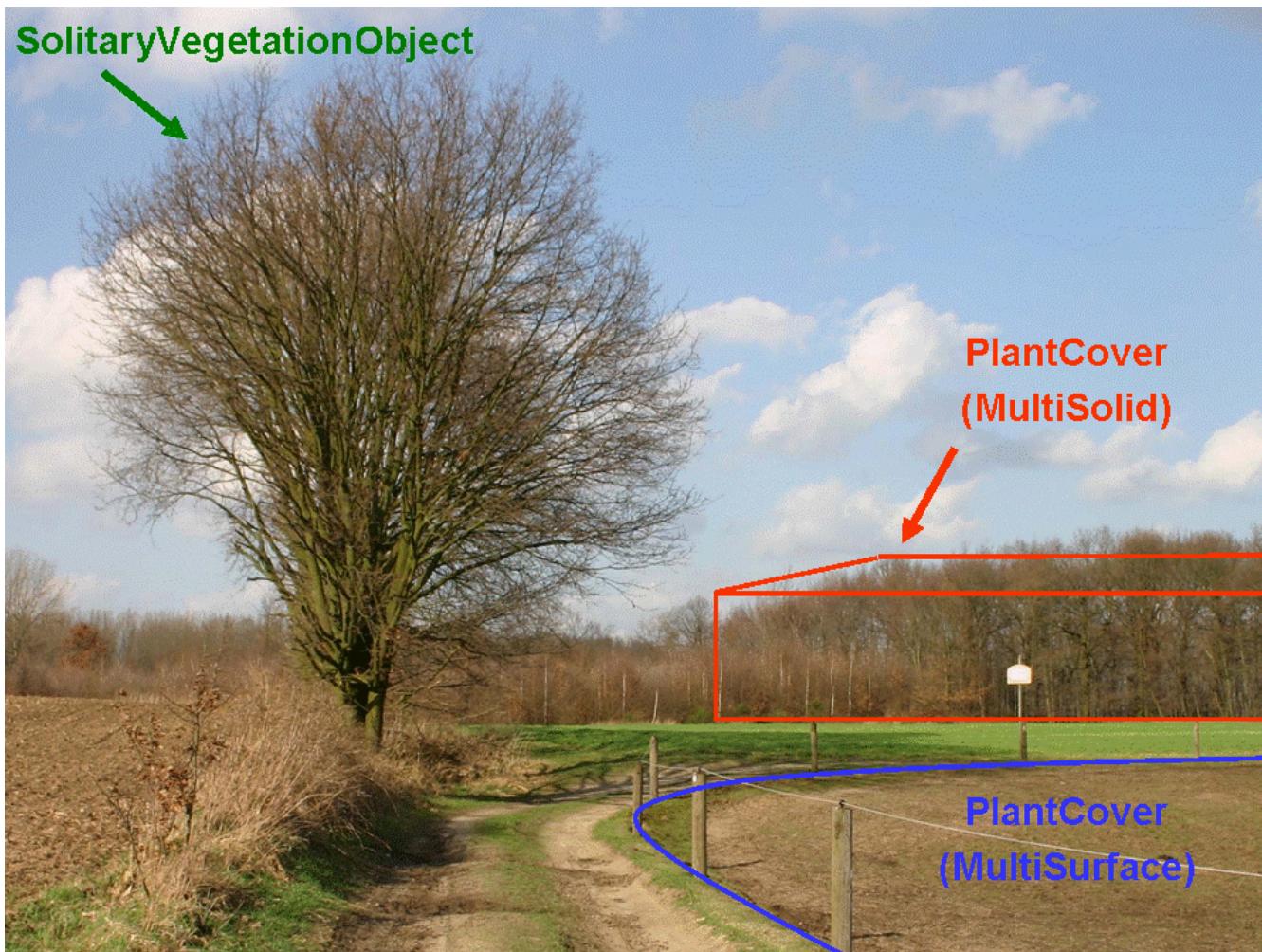


Figure 104. Example for vegetation objects of the classes SolitaryVegetationObject and PlantCover (graphic: District of Recklinghausen).

8.15.4. Level of Detail

The geometry of a SolitaryVegetationObject may be defined in LOD 1-3 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 be 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 [\[figure-65\]](#)).

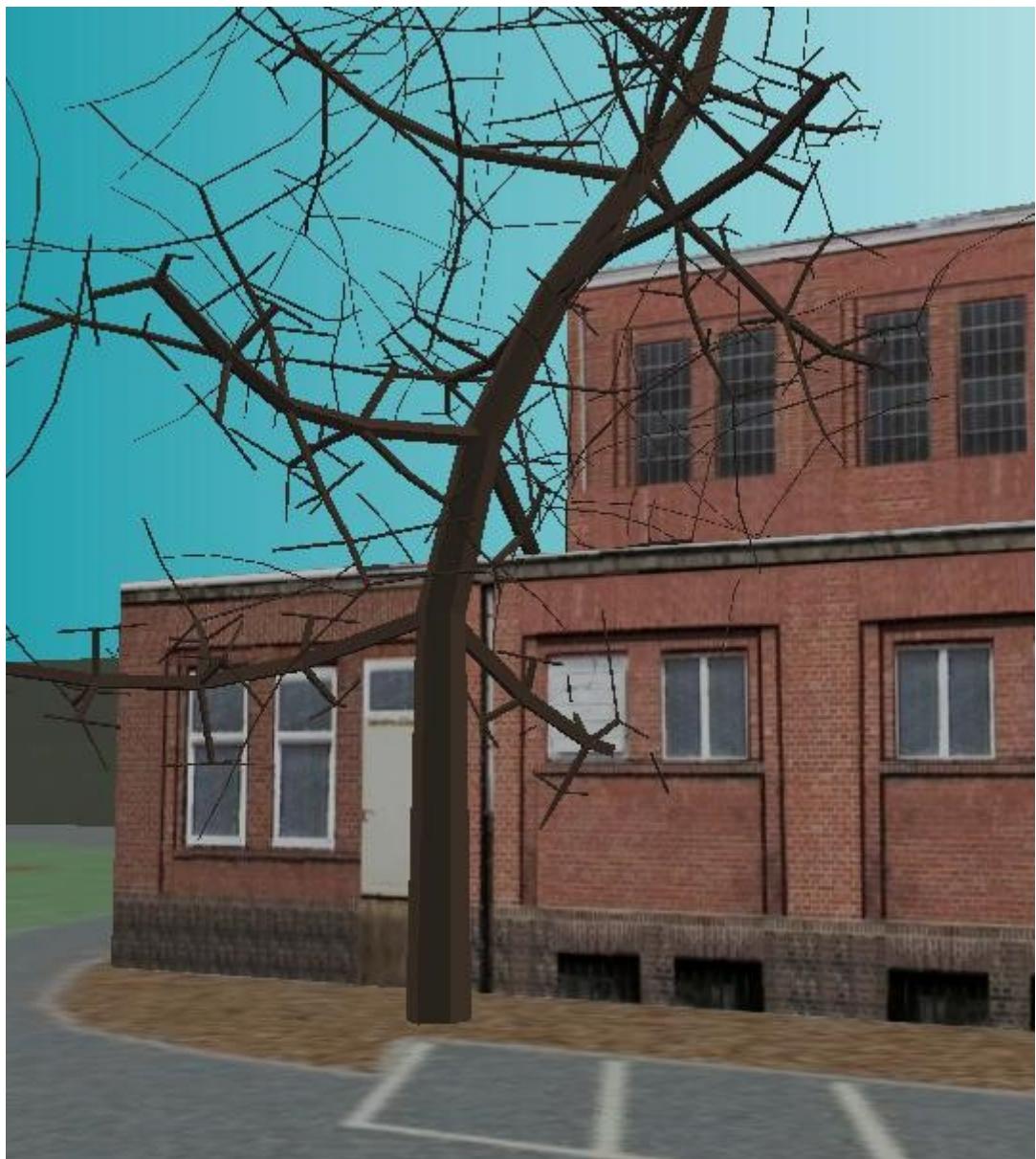




Figure 105. Visualisation of a vegetation object in different seasons (source: District of Recklinghausen).

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 \(source: District of Recklinghausen\)](#)).

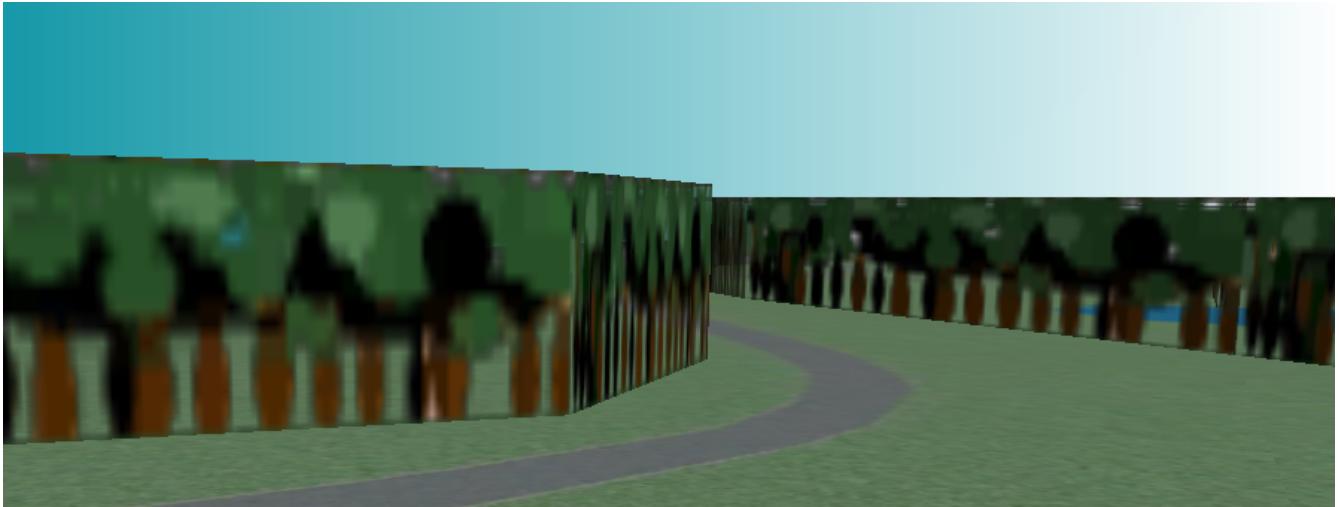


Figure 106. Example for the visualisation/modelling of a solid forest (source: District of Recklinghausen).

8.15.5. 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 contain botanical information from public environmental agencies.

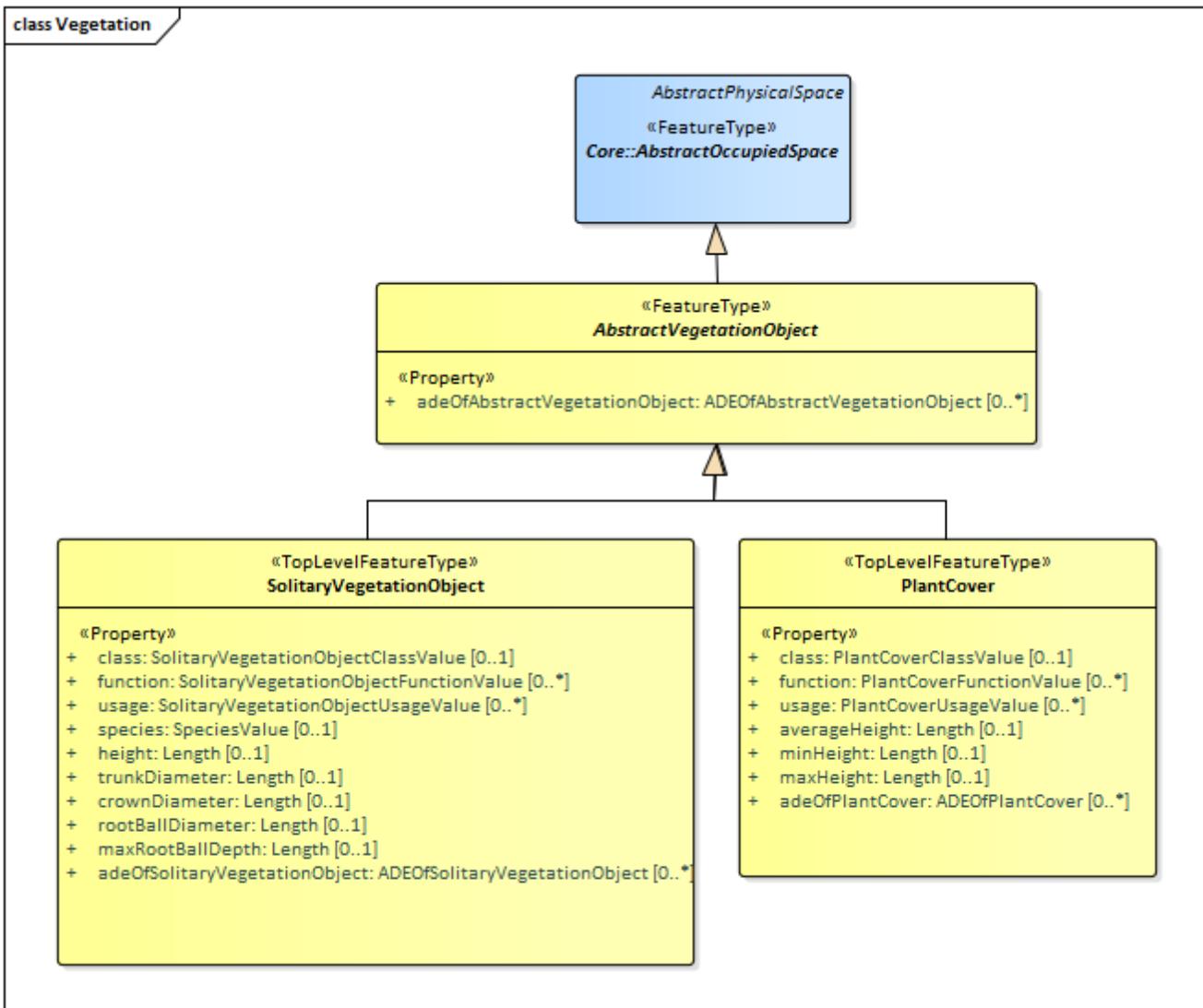


Figure 107. UML diagram of the Vegetation Model.

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

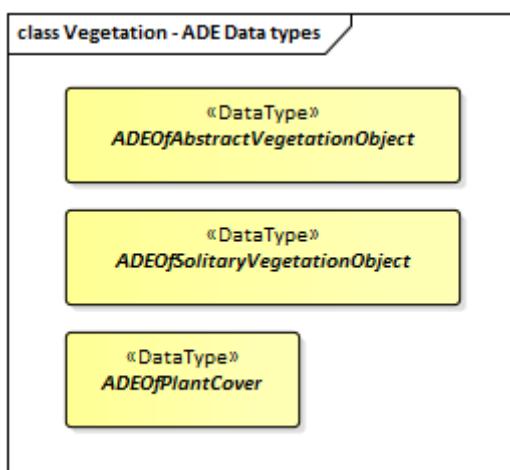


Figure 108. ADE classes of the CityGML Vegetation module.

The Code Lists provided for the Vegetation module are illustrated in [Codelists from the CityGML Vegetation module..](#)

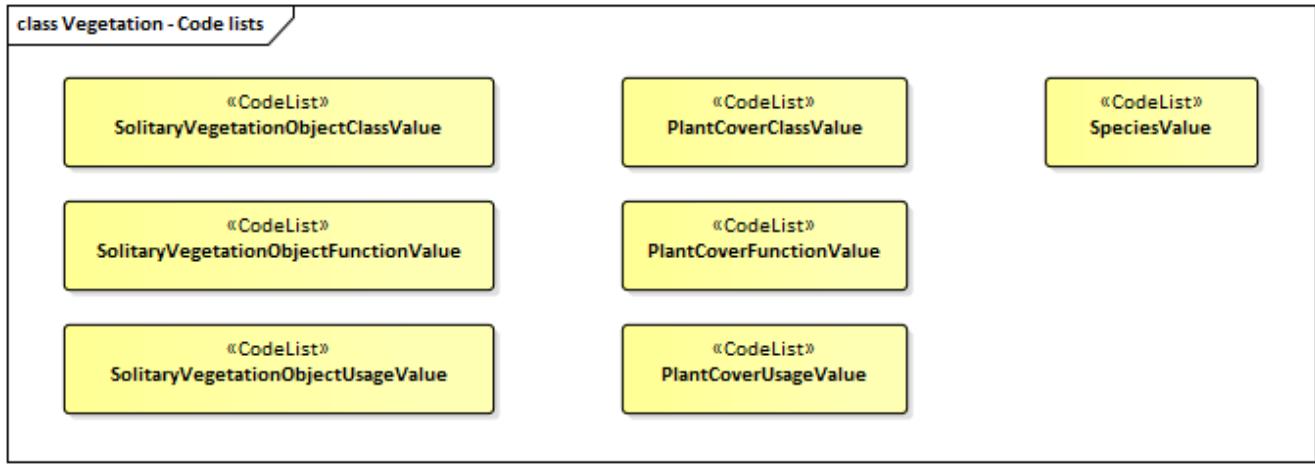


Figure 109. Codelists from the CityGML Vegetation module.

8.15.6. 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?

8.16. Versioning

Contributors
C. Heazel - first draft

8.16.1. Synopsis

The Versioning module defines the concepts that enable encoding representing multiple versions of a city model.

8.16.2. Key Concepts

Version: Version represents a defined state of a city model consisting of the dedicated versions of all city object instances that belong to the respective city model version. Versions can have names, a description and can be labeled with an arbitrary number of user defined tags.

A type of [AbstractVersion](#).

VersionTransition: VersionTransition describes the change of the state of a city model from one version to another. Version transitions can have names, a description and can be further qualified by a type and a reason.

A type of [AbstractVersionTransition](#).

8.16.3. Discussion

The Versioning module defines the concepts that enable encoding representing multiple versions of a city model. A specific version represents a defined state of a city model consisting of the dedicated versions of all city object instances that belong to the respective city model version. Each version can be complemented by version transitions that describe the change of the state of a city model from one version to another and that give the reason for the change and the modifications applied. In addition, the Versioning module introduces bitemporal timestamps for all objects. This allows for providing all objects with information on 1) the time period a specific version of an object is an integral part of the 3D city model and 2) the lifespan a specific version of an object exists in the real world.

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

8.16.4. UML Model

The UML diagram of the Versioning module is depicted in [UML diagram of the Versioning Model..](#)

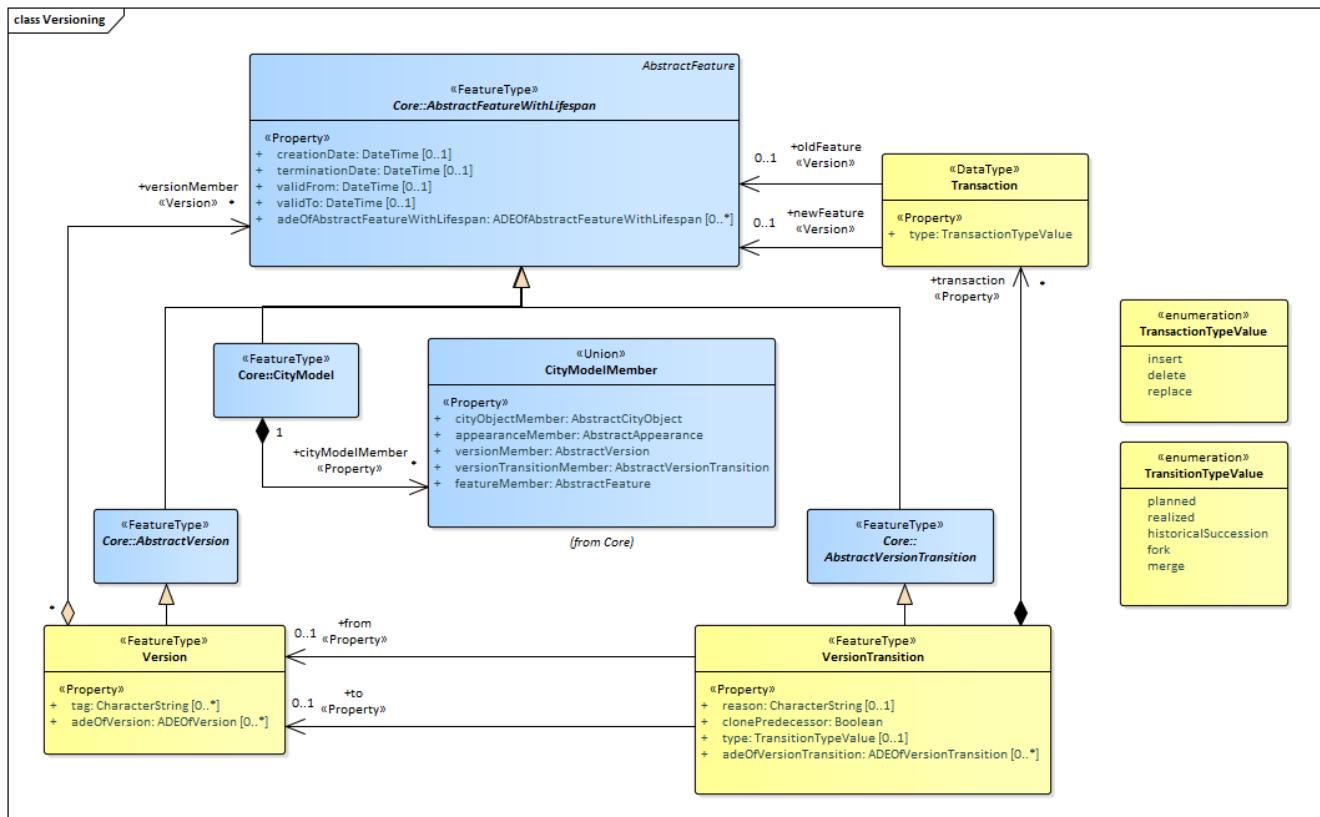


Figure 110. UML diagram of the Versioning Model.

The ADE data types provided for the Versioning module are illustrated in [ADE classes of the CityGML Versioning module..](#)

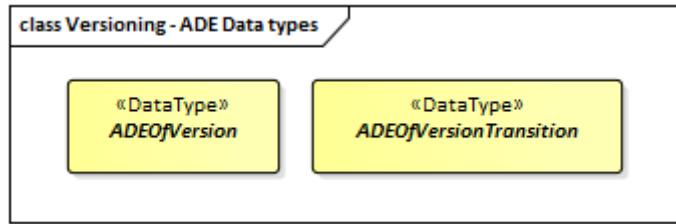


Figure 111. ADE classes of the CityGML Versioning module.

8.16.5. Examples

8.17. Waterbodies

Contributors
C. Heazel - first draft

8.17.1. 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.

8.17.2. 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.

8.17.3. Discussion

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 1-3 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).

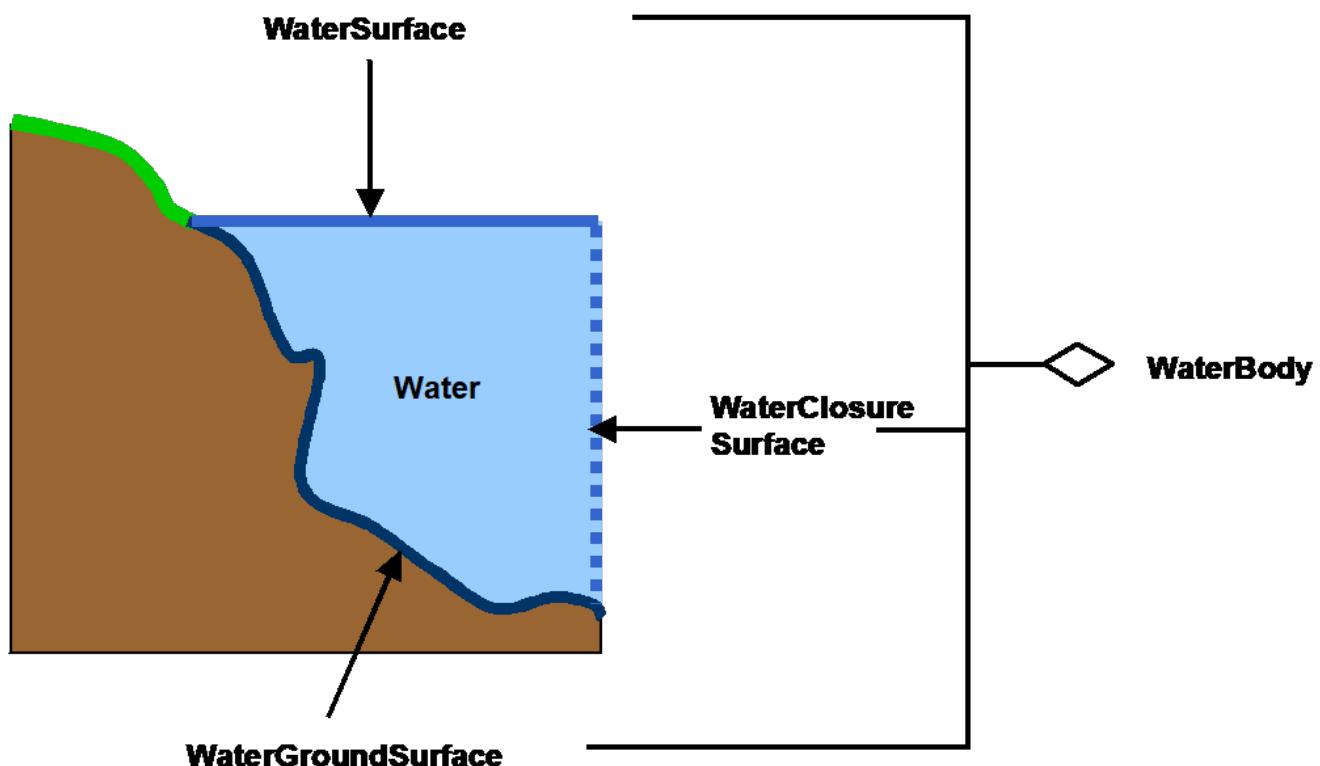


Figure 112. Illustration of a water body defined in CityGML (graphic: IGG Uni Bonn).

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. The [WaterSurface](#) objects together with the [WaterGroundSurface](#) objects enclose the [WaterBody](#) as a volume.

8.17.4. Level of Detail

Both LOD0 and LOD1 represent a low level of illustration and high grade of generalisation. Here the rivers are modelled as [GM_MultiCurve](#) geometry and brooks are omitted. Seas, oceans and lakes

with significant extent are represented as a [GM_MultiSurface](#) ([figure-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 to the last). Areal objects like lakes or seas are represented by 3D surface geometries of the water surface.

Starting from LOD1 water bodies may also be modelled as water filled volumes represented by [GM_Solids](#). If a water body is represented by a [GM_Solids](#) in LOD2 or higher, the surface geometries of the corresponding thematic [WaterGroundSurface](#), and [WaterSurface](#) objects must coincide with the exterior shell of the [GM_Solid](#). This can be ensured, if for each LOD X the respective lodXSolid representation (where X is between 2 and 3) does not redundantly define the geometry, but instead references the corresponding polygons of the lodXSurface elements (where X is between 2 and 3) of [WaterGroundSurface](#) and [WaterSurface](#).

LOD2 to LOD3 demand a higher grade of detail and therefore any [WaterBody](#) can be outlined by <>thematic-surface-concept,thematic surfaces> or a solid composed of the surrounding <>thematic-surface-concept,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: lod1MultiSurface, lod2MultiSurface, lod3MultiSurface.

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.

8.17.5. 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 [AbstractWaterBoundarySurface](#). A [AbstractWaterBoundarySurface](#) is a part of the water body's exterior shell with a special function like WaterSurface or WaterGroundSurface. 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.

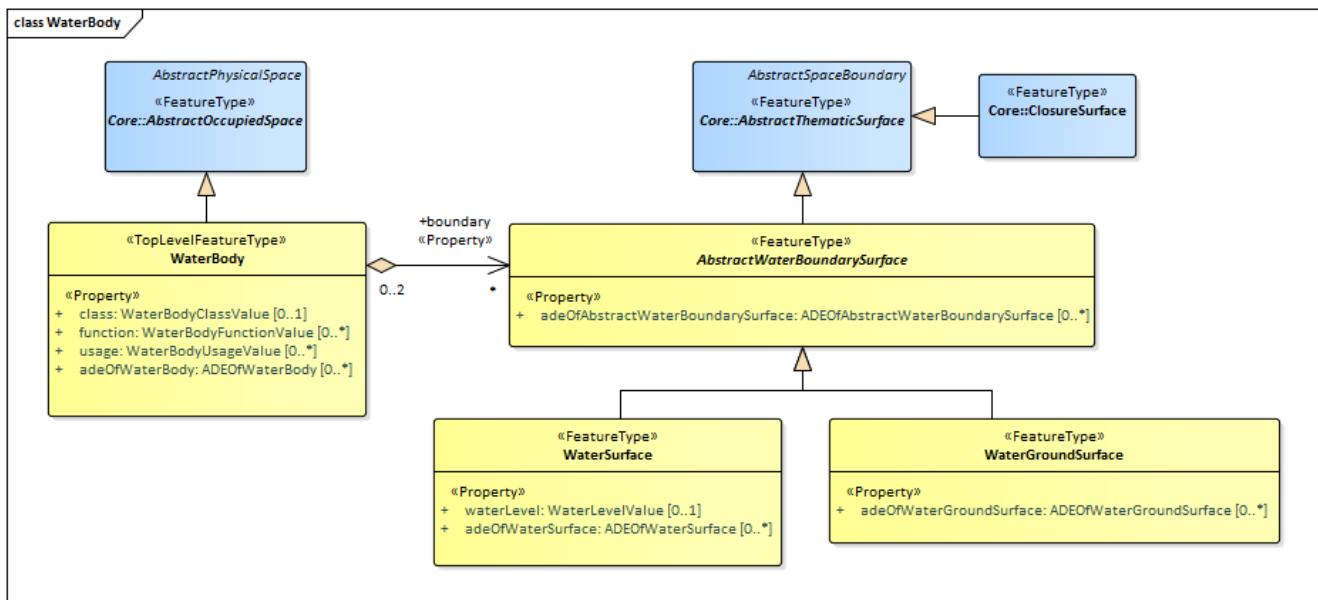


Figure 113. UML diagram of the Water Body Model.

The ADE data types provided for the Water Body module are illustrated in [ADE classes of the CityGML Water Body module..](#)

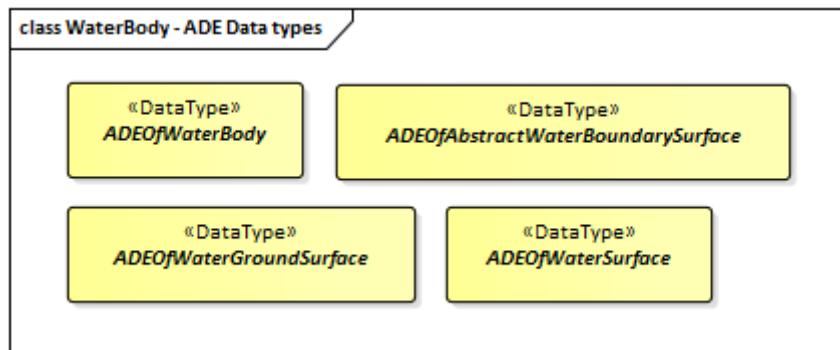


Figure 114. ADE classes of the CityGML Water Body module.

The Code Lists provided for the Water Body module are illustrated in [Codelists from the CityGML Water Body module..](#)

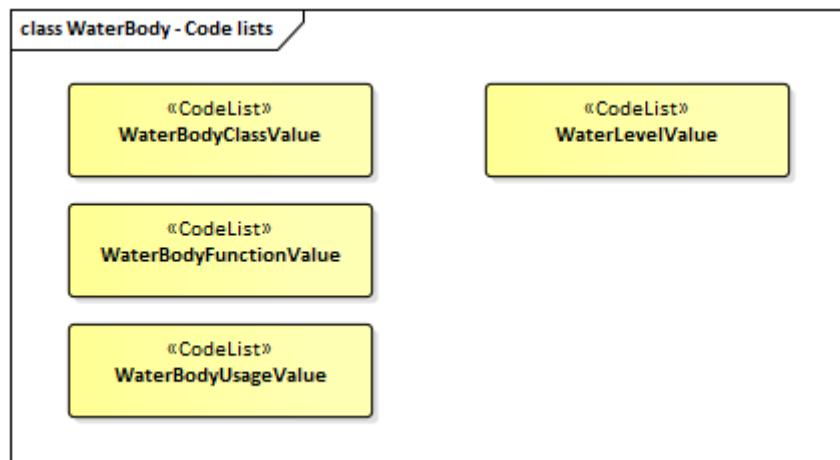


Figure 115. Codelists from the CityGML Water Body module.

8.17.6. Examples

8.18. 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 [\[ug_generics_section\]](#)). 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).

8.18.1. 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 concrete 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 feature 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 property 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 number of additional properties with arbitrary XML content with the name “_GenericApplication-PropertyOf<Featuretypename>”. For example, the last property in the definition of the CityGML feature 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 datatype 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 corresponding 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 communities. Every ADE may add their specific properties to the same CityGML feature type as they all can belong 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 information (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).

8.18.2. 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 correction” 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 substitution 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?)

Chapter 9. CityGML Data Dictionary

The CityGML UML model is the normative definition of the CityGML Conceptual Model. The Data Dictionary tables in this section were software generated from the UML model. As such, this section provides a normative representation of the CityGML Conceptual Model.

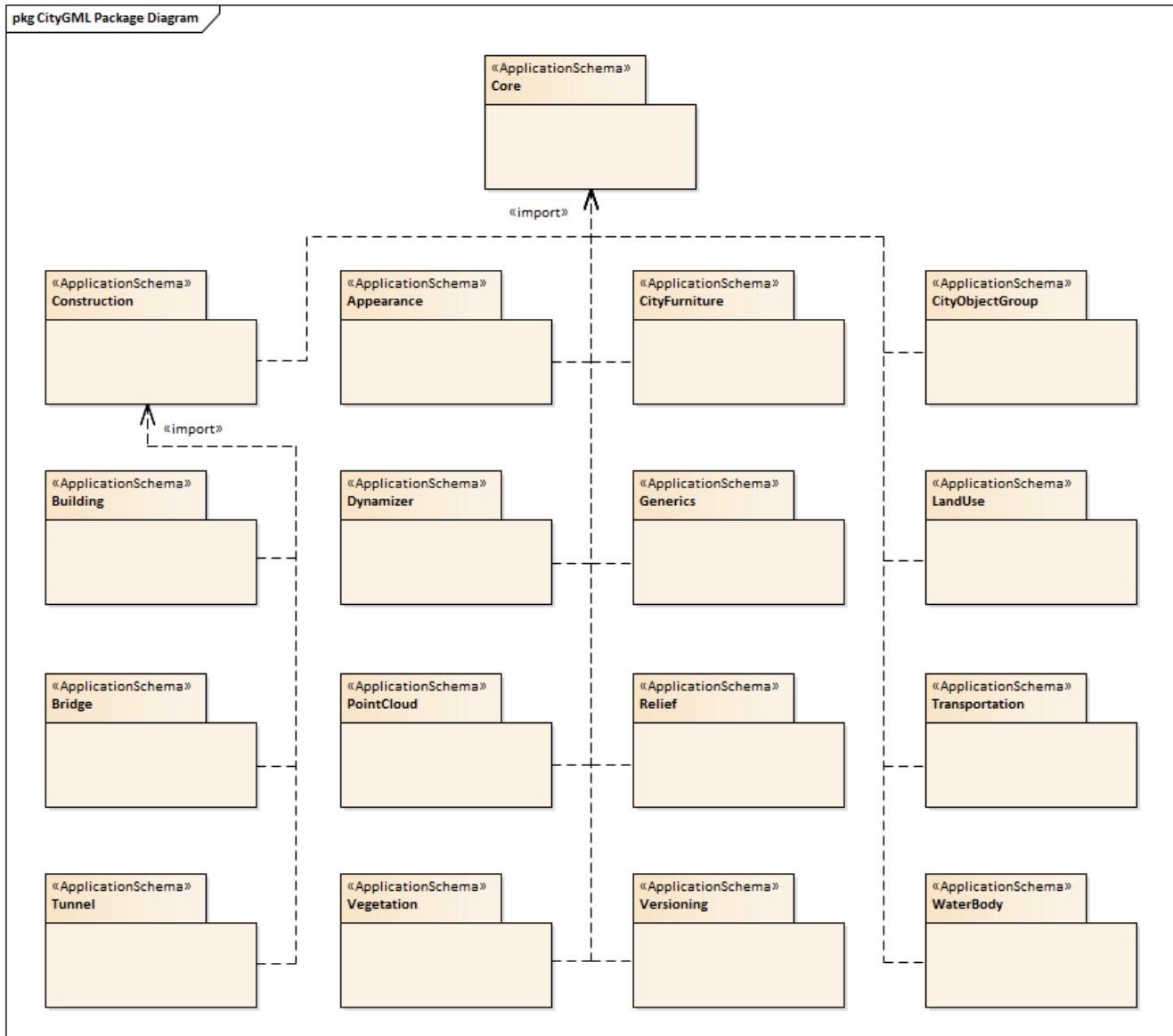


Figure 116. CityGML UML Packages

9.1. ISO Classes

The following classes are defined in ISO standards and used by the CityGML Conceptual Model.

9.1.1. Class AnyFeature (ISO 19109:2015)

AnyFeature

Definition:	AnyFeature is an abstract class that is the generalization of all feature types. AnyFeature is an instance of the «metaclass» FeatureType [cf. ISO 19109].			
Subclass Of:	none			
Stereotype:	«FeatureType»			
<hr/>				
Role name	Target class and multiplicity	Definition		
	FeatureType [1..1]			
<hr/>				
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»				

9.1.2. Class CV_DiscreteGridPointCoverage (ISO 19123:2005)

CV_DiscreteGridPointCoverage				
<hr/>				
Definition:	A coverage that returns the same feature attribute values for every direct position within any single spatial object, temporal object or spatiotemporal object in its domain.			
Subclass Of:	CV_DiscreteCoverage			
Stereotype:	«type»			
<hr/>				
Role name	Target class and multiplicity	Definition		
element	CV_GridPointValue Pair [1..*]			
valueAssignment	CV_GridValuesMatrix			
	ix [1..1]			
<hr/>				
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»				

9.1.3. Class DirectPosition (ISO 19107: 2003)

DirectPosition
<hr/>

Definition:	DirectPosition object data types (Figure 14) hold the coordinates for a position within some coordinate reference system. The coordinate reference system is described in ISO 19111. Since DirectPositions, as data types, will often be included in larger objects (such as GM_Objects) that have references to ISO19111::SC_CRS, the DirectPosition::coordinateReferenceSystem may be left NULL if this particular DirectPosition is included in a larger object with such a reference to a SC_CRS. In this case, the DirectPosition::coordinateReferenceSystem is implicitly assumed to take on the value of the containing object's SC_CRS.	
Subclass Of:	None	
Stereotype:	None	
<hr/>		
Role name	Target class and multiplicity	Definition
CRS	CRS [0..1]	
CRS	SC_CRS [0..1]	
<hr/>		
Attribute	Value type and multiplicity	Definition
coordinate	Sequence< Number > [1..1]	
dimension	Integer [1..1]	
<hr/>		
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»		

9.1.4. Class GM_Object ([ISO 19107: 2003](#))

GM_Object

Definition:	GM_Object is the root class of the geometric object taxonomy and supports interfaces common to all geographically referenced geometric objects. GM_Object instances are sets of direct positions in a particular coordinate reference system. A GM_Object can be regarded as an infinite set of points that satisfies the set operation interfaces for a set of direct positions, TransfiniteSet<DirectPosition>. Since an infinite collection class cannot be implemented directly, a Boolean test for inclusion shall be provided by the GM_Object interface. This international standard concentrates on vector geometry classes, but future work may use GM_Object as a root class without modification. NOTE As a type, GM_Object does not have a well-defined default state or value representation as a data type. Instantiated subclasses of GM_Object will.
Subclass Of:	none
Stereotype:	«type»
Constraint:	dimension() > boundary().dimension (Invariant):
Constraint:	boundary().notEmpty() implies boundary().dimension() = dimension() - 1 (Invariant):
Constraint:	boundary().isEmpty() = isCycle() (Invariant):

Role name	Target class and multiplicity	Definition
	Geometry [1..1]	
	TransfiniteSet<DirectPosition> [1..1]	
	CV_DomainObject [1..1]	
CRS	CRS [0..1]	
CRS	SC_CRS [0..1]	

Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»

9.1.5. Class GM_MultiCurve (ISO 19107: 2003)

GM_MultiCurve

Definition:	An aggregate class containing only instances of GM_OrientableCurve. The association role “element” shall be the set of GM_OrientableCurves contained in this GM_MultiCurve.	
Subclass Of:	GM_MultiPrimitive	
Stereotype:	«type»	
<hr/>		
Attribute	Value type and multiplicity	Definition
length	Length [1..1]	
<hr/>		
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»		

9.1.6. Class GM_MultiPoint ([ISO 19107:2003](#))

GM_MultiPoint		
<hr/>		
Definition:		GM_MultiPoint is an aggregate class containing only points. The association role “element” shall be the set of GM_Points contained in this GM_MultiPoint.
Subclass Of:		GM_MultiPrimitive
Stereotype:		«type»
<hr/>		
Attribute	Value type and multiplicity	Definition
position	Set< DirectPosition > [1..1]	
<hr/>		
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»		

9.1.7. Class GM_MultiSurface ([ISO 19107:2003](#))

GM_MultiSurface		
<hr/>		
Definition:		An aggregate class containing only instances of GM_OrientableSurface. The association role “element” shall be the set of GM_OrientableSurfaces contained in this GM_MultiSurface.
Subclass Of:		GM_MultiPrimitive
Stereotype:		«type»
<hr/>		

Attribute	Value type and multiplicity	Definition
area	Area [1..1]	
perimeter	Length [1..1]	

Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»

9.1.8. Class GM_Point ([ISO 19107:2003](#))

GM_Point		
Role name	Target class and multiplicity	Definition
	Point [1..1]	
composite	GM_CompositePoint t [0..*]	
Attribute	Value type and multiplicity	Definition
position	DirectPosition [1..1]	The attribute "position" shall be the DirectPosition of this GM_Point. GM_Point::position [1] : DirectPosition NOTE In most cases, the state of a GM_Point is fully determined by its position attribute. The only exception to this is if the GM_Point has been subclassed to provide additional non-geometric information such as symbology.

Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»

9.1.9. Class GM_Solid ([ISO 19107:2003](#))

GM_Solid

Definition:	GM_Solid, a subclass of GM_Primitive, is the basis for 3-dimensional geometry. The extent of a solid is defined by the boundary surfaces.							
Subclass Of:	GM_Primitive							
Stereotype:	«type»							
<table border="1"> <thead> <tr> <th>Role name</th><th>Target class and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>composite</td><td> GM_CompositeSolid d [0..*] Solid [1..1] </td><td></td></tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	composite	GM_CompositeSolid d [0..*] Solid [1..1]	
Role name	Target class and multiplicity	Definition						
composite	GM_CompositeSolid d [0..*] Solid [1..1]							
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»								

9.1.10. Class GM_Surface (ISO 19107:2003)

GM_Surface								
Definition:	<p>GM_Surface is a subclass of GM_Primitive and is the basis for 2-dimensional geometry. Unorientable surfaces such as the Möbius band are not allowed. The orientation of a surface chooses an "up" direction through the choice of the upward normal, which, if the surface is not a cycle, is the side of the surface from which the exterior boundary appears counterclockwise. Reversal of the surface orientation reverses the curve orientation of each boundary component, and interchanges the conceptual "up" and "down" direction of the surface. If the surface is the boundary of a solid, the "up" direction is usually outward. For closed surfaces, which have no boundary, the up direction is that of the surface patches, which must be consistent with one another. Its included GM_SurfacePatches describe the interior structure of a GM_Surface. NOTE Other than the restriction on orientability, no other "validity" condition is required for GM_Surface.</p>							
Subclass Of:	GM_OrientableSurface							
Stereotype:	«type»							
<table border="1"> <thead> <tr> <th>Role name</th><th>Target class and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td></td><td> GM_GenericSurface e [1..1] Building [0..*] </td><td></td></tr> </tbody> </table>			Role name	Target class and multiplicity	Definition		GM_GenericSurface e [1..1] Building [0..*]	
Role name	Target class and multiplicity	Definition						
	GM_GenericSurface e [1..1] Building [0..*]							
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»								

9.1.11. Class GM_Tin (ISO 19107:2003)

GM_Tin

Definition: A GM_Tin is a GM_TriangulatedSurface that uses the Delaunay algorithm or a similar algorithm complemented with consideration for breaklines, stoplines and maximum length of triangle sides (Figure 22). These networks satisfy the Delaunay criterion away from the modifications: For each triangle in the network, the circle passing through its vertexes does not contain, in its interior, the vertex of any other triangle.

Subclass Of: [GM_TriangulatedSurface](#)

Stereotype: «type»

Attribute	Value type and multiplicity	Definition
breakLines	Set< GM_LineString > [1..1]	
controlPoint	GM_Position [3..*]	
maxLength	Distance [1..1]	
stopLines	Set< GM_LineString > [1..1]	

Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»

9.1.12. Class GM_TriangulatedSurface (ISO 19107:2003)

GM_TriangulatedSurface

Definition: A GM_TriangulatedSurface is a GM_PolyhedralSurface that is composed only of triangles (GM_Triangle). There is no restriction on how the triangulation is derived.

Subclass Of: [GM_PolyhedralSurface](#)

Stereotype: «type»

Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»

9.1.13. Class SC_CRS (ISO 19111:2019)

SC_CRS

Definition:	Coordinate reference system which is usually single but may be compound.										
Subclass Of:	IO_IdentifiedObjectBase , RS_ReferenceSystem										
Stereotype:	«type»										
<table border="1"> <thead> <tr> <th>Role name</th><th>Target class and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>coordOperation nTo</td><td>CC_CoordinateOper ation [0..*]</td><td>Not-navigable association from a Coordinate Operation that uses ths CRS as its targetCRS.</td></tr> <tr> <td>grid</td><td>CV_ReferenceableGrid rid [0..*]</td><td></td></tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	coordOperation nTo	CC_CoordinateOper ation [0..*]	Not-navigable association from a Coordinate Operation that uses ths CRS as its targetCRS.	grid	CV_ReferenceableGrid rid [0..*]	
Role name	Target class and multiplicity	Definition									
coordOperation nTo	CC_CoordinateOper ation [0..*]	Not-navigable association from a Coordinate Operation that uses ths CRS as its targetCRS.									
grid	CV_ReferenceableGrid rid [0..*]										
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>scope</td><td>CharacterString [1..*]</td><td>Description of usage, or limitations of usage, for which this CRS is valid. If unknown, enter "not known".</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	scope	CharacterString [1..*]	Description of usage, or limitations of usage, for which this CRS is valid. If unknown, enter "not known".			
Attribute	Value type and multiplicity	Definition									
scope	CharacterString [1..*]	Description of usage, or limitations of usage, for which this CRS is valid. If unknown, enter "not known".									
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»											

9.1.14. Class TM_Position (ISO 19108:2006)

TM_Position		
Attribute	Value type and multiplicity	Definition
Definition:	TM_Position is a union class that consists of one of the data types listed as its attributes. Date, Time, and DateTime are basic data types defined in ISO/TS 19103.	
Subclass Of:	None	
Stereotype:	«Union»	
Attribute	Value type and multiplicity	Definition
anyOther	TM_TemporalPosition [1..1]	
date8601	Date [1..1]	
time8601	Time [1..1]	
dateTime8601	DateTime [1..1]	
Note: Unless otherwise specified, all attribute and role names have the stereotype «Property»		

9.2. Core

Description: The Core module defines the basic components of the CityGML data model. The Core module defines abstract base classes that define the core properties of more specialized thematic classes defined in other modules. The Core module also defines concrete classes that are common to other modules, for example basic data types.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.2.1. Classes

AbstractAppearance

Definition: AbstractAppearance is the abstract superclass to represent any kind of appearance objects.

Subclass of: [AbstractFeatureWithLifespan](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfAbstract Appearance	ADEOfAbstractApp [0..*]	Augments AbstractAppearance with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractCityObject

Definition: AbstractCityObject is the abstract superclass of all thematic classes within the CityGML Conceptual Model.

Subclass of: [AbstractFeatureWithLifespan](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
generalizesTo	AbstractCityObject [*]	Relates generalized representations of the same real-world object in different Levels of Detail to the city object. The direction of this relation is from the city object to the corresponding generalized city objects.
genericAttribute	AbstractGenericAttribute [*]	Relates generic attributes to the city object.
dynamizer	AbstractDynamizer [*]	Relates Dynamizer objects to the city object. These allow timeseries data to override static attribute values of the city object.
appearance	AbstractAppearance [*]	Relates appearances to the city object.
externalReference	ExternalReference [*]	References external objects in other information systems that have a relation to the city object.
relatedTo	AbstractCityObject [*]	Relates other city objects to the city object. It also describes how the city objects are related to each other.

Attribute	Value type and multiplicity	Definition
relativeToTerrain	RelativeToTerrain [0..1]	Describes the vertical position of the city object relative to the surrounding terrain.
relativeToWater	RelativeToWater [0..1]	Describes the vertical position of the city object relative to the surrounding water surface.
adeOfAbstractCityObject	ADEOfAbstractCityObject [0..*]	Augments AbstractCityObject with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractDynamizer

Definition: AbstractDynamizer is the abstract superclass to represent Dynamizer objects.

Subclass of: [AbstractFeatureWithLifespan](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfAbstractDynamizer	ADEOfAbstractDynamizer [0..*]	Augments AbstractDynamizer with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractFeature

Definition: AbstractFeature is the abstract superclass of all feature types within the CityGML Conceptual Model.

Subclass of: [AnyFeature](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
featureID	ID [1..1]	Specifies the unique identifier of the feature that is valid in the instance document within which it occurs.
identifier	ScopedName [0..1]	Specifies the unique identifier of the feature that is valid globally.
name	GenericName [0..*]	Specifies the name of the feature.
description	CharacterString [0..1]	Provides further information on the feature.
adeOfAbstract Feature	ADEOfAbstractFeature [0..*]	Augments AbstractFeature with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractFeatureWithLifespan

Definition: AbstractFeatureWithLifespan is the base class for all CityGML features. This class allows the optional specification of the real-world and database times for the existence of each feature.

Subclass of: [AbstractFeature](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
creationDate	DateTime [0..1]	Indicates the date at which a CityGML feature was added to the CityModel.
terminationDate	DateTime [0..1]	Indicates the date at which a CityGML feature was removed from the CityModel.
validFrom	DateTime [0..1]	Indicates the date at which a CityGML feature started to exist in the real world.
validTo	DateTime [0..1]	Indicates the date at which a CityGML feature ended to exist in the real world.
adeOfAbstractFeatureWithLifespan	ADEOfAbstractFeatUreWithLifespan [0..*]	Augments AbstractFeatureWithLifespan with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractLogicalSpace

Definition:	AbstractLogicalSpace is the abstract superclass for all types of logical spaces. Logical space refers to spaces that are not bounded by physical surfaces but are defined according to thematic considerations.
Subclass of:	AbstractSpace
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfAbstractLogicalSpace	ADEOfAbstractLogicalSpace [0..*]	Augments AbstractLogicalSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractOccupiedSpace

Definition:	AbstractOccupiedSpace is the abstract superclass for all types of physically occupied spaces. Occupied space refers to spaces that are partially or entirely filled with matter.
Subclass of:	AbstractPhysicalSpace
Stereotype:	«FeatureType»

Role name	Target class and multiplicity	Definition
lod3ImplicitRepresentation	ImplicitGeometry [0..1]	Relates to an implicit geometry that represents the occupied space in Level of Detail 3.
lod1ImplicitRepresentation	ImplicitGeometry [0..1]	Relates to an implicit geometry that represents the occupied space in Level of Detail 1.
lod2ImplicitRepresentation	ImplicitGeometry [0..1]	Relates to an implicit geometry that represents the occupied space in Level of Detail 2.
Attribute	Value type and multiplicity	Definition
adeOfAbstractOccupiedSpace	ADEOfAbstractOccupiedSpace [0..*]	Augments AbstractOccupiedSpace with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

AbstractPhysicalSpace

Role name	Target class and multiplicity	Definition
Definition:	AbstractPhysicalSpace is the abstract superclass for all types of physical spaces. Physical space refers to spaces that are fully or partially bounded by physical objects.	
Subclass of:	AbstractSpace	
Stereotype:	«FeatureType»	
Role name	Target class and multiplicity	Definition
lod3TerrainIntersectionCurve	GM_MultiCurve [0..1]	Relates to a 3D MultiCurve geometry that represents the terrain intersection curve of the physical space in Level of Detail 3.
lod2TerrainIntersectionCurve	GM_MultiCurve [0..1]	Relates to a 3D MultiCurve geometry that represents the terrain intersection curve of the physical space in Level of Detail 2.
pointCloud	AbstractPointCloud [0..1]	Relates to a 3D PointCloud that represents the physical space.
lod1TerrainIntersectionCurve	GM_MultiCurve [0..1]	Relates to a 3D MultiCurve geometry that represents the terrain intersection curve of the physical space in Level of Detail 1.

Attribute	Value type and multiplicity	Definition
adeOfAbstractPhysicalSpace	ADEOfAbstractPhy sicalSpace [0..*]	Augments AbstractPhysicalSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractPointCloud

Definition: AbstractPointCloud is the abstract superclass to represent PointCloud objects.
 Subclass of: [AbstractFeature](#)
 Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfAbstractPointCloud	ADEOfAbstractPoin tCloud [0..*]	Augments AbstractPointCloud with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractSpace

Definition: AbstractSpace is the abstract superclass for all types of spaces. A space is an entity of volumetric extent in the real world.
 Subclass of: [AbstractCityObject](#)
 Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
lod2MultiCurve e	GM_MultiCurve [0..1]	Relates to a 3D MultiCurve geometry that represents the space in Level of Detail 2.
lod0MultiCurve e	GM_MultiCurve [0..1]	Relates to a 3D MultiCurve geometry that represents the space in Level of Detail 0.
lod0MultiSurface ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the space in Level of Detail 0.
lod2MultiSurface ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the space in Level of Detail 2.
lod3MultiSurface ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the space in Level of Detail 3.
lod0Point	GM_Point [0..1]	Relates to a 3D Point geometry that represents the space in Level of Detail 0.
lod3Solid	GM_Solid [0..1]	Relates to a 3D Solid geometry that represents the space in Level of Detail 3.
lod3MultiCurve e	GM_MultiCurve [0..1]	Relates to a 3D MultiCurve geometry that represents the space in Level of Detail 3.
lod2Solid	GM_Solid [0..1]	Relates to a 3D Solid geometry that represents the space in Level of Detail 2.
boundary	AbstractSpaceBoundary [*]	Relates to surfaces that bound the space.
lod1Solid	GM_Solid [0..1]	Relates to a 3D Solid geometry that represents the space in Level of Detail 1.
Attribute	Value type and multiplicity	Definition
spaceType	SpaceType [0..1]	Specifies the degree of openness of a space.
volume	QualifiedVolume [0..*]	Specifies qualified volumes related to the space.
area	QualifiedArea [0..*]	Specifies qualified areas related to the space.
adeOfAbstract Space	ADEOfAbstractSpace [0..*]	Augments AbstractSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractSpaceBoundary

Definition:	AbstractSpaceBoundary is the abstract superclass for all types of space boundaries. A space boundary is an entity with areal extent in the real world. Space boundaries are objects that bound a Space. They also realize the contact between adjacent spaces.							
Subclass of:	AbstractCityObject							
Stereotype:	«FeatureType»							
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>adeOfAbstract SpaceBoundary</td><td>ADEOfAbstractSpaceBoundary [0..*]</td><td>Augments AbstractSpaceBoundary with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	adeOfAbstract SpaceBoundary	ADEOfAbstractSpaceBoundary [0..*]	Augments AbstractSpaceBoundary with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition						
adeOfAbstract SpaceBoundary	ADEOfAbstractSpaceBoundary [0..*]	Augments AbstractSpaceBoundary with properties defined in an ADE.						
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».								

AbstractThematicSurface

Definition:	AbstractThematicSurface is the abstract superclass for all types of thematic surfaces.																						
Subclass of:	AbstractSpaceBoundary																						
Stereotype:	«FeatureType»																						
<table border="1"> <thead> <tr> <th>Role name</th><th>Target class and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>lod1MultiSurf ace</td><td>GM_MultiSurface [0..1]</td><td>Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 1.</td></tr> <tr> <td>pointCloud</td><td>AbstractPointCloud [0..1]</td><td>Relates to a 3D PointCloud that represents the thematic surface.</td></tr> <tr> <td>lod0MultiCurv e</td><td>GM_MultiCurve [0..1]</td><td>Relates to a 3D MultiCurve geometry that represents the thematic surface in Level of Detail 0.</td></tr> <tr> <td>lod3MultiSurf ace</td><td>GM_MultiSurface [0..1]</td><td>Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 3.</td></tr> <tr> <td>lod0MultiSurf ace</td><td>GM_MultiSurface [0..1]</td><td>Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 0.</td></tr> <tr> <td>lod2MultiSurf ace</td><td>GM_MultiSurface [0..1]</td><td>Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 2.</td></tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	lod1MultiSurf ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 1.	pointCloud	AbstractPointCloud [0..1]	Relates to a 3D PointCloud that represents the thematic surface.	lod0MultiCurv e	GM_MultiCurve [0..1]	Relates to a 3D MultiCurve geometry that represents the thematic surface in Level of Detail 0.	lod3MultiSurf ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 3.	lod0MultiSurf ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 0.	lod2MultiSurf ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 2.
Role name	Target class and multiplicity	Definition																					
lod1MultiSurf ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 1.																					
pointCloud	AbstractPointCloud [0..1]	Relates to a 3D PointCloud that represents the thematic surface.																					
lod0MultiCurv e	GM_MultiCurve [0..1]	Relates to a 3D MultiCurve geometry that represents the thematic surface in Level of Detail 0.																					
lod3MultiSurf ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 3.																					
lod0MultiSurf ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 0.																					
lod2MultiSurf ace	GM_MultiSurface [0..1]	Relates to a 3D MultiSurface geometry that represents the thematic surface in Level of Detail 2.																					

Attribute	Value type and multiplicity	Definition
area	QualifiedArea [0..*]	Specifies qualified areas related to the thematic surface.
adeOfAbstractThematicSurf	ADEOfAbstractThemeSurface [0..*]	Augments AbstractThematicSurface with properties defined in an ADE.
ace		

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractUnoccupiedSpace

Definition:	AbstractUnoccupiedSpace is the abstract superclass for all types of physically unoccupied spaces. Unoccupied space refers to spaces that are entirely or mostly free of matter.
Subclass of:	AbstractPhysicalSpace
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfAbstractUnoccupiedSpace	ADEOfAbstractUnoccupiedSpace [0..*]	Augments AbstractUnoccupiedSpace with properties defined in an ADE.
ace		

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractVersion

Definition:	AbstractVersion is the abstract superclass to represent Version objects.
Subclass of:	AbstractFeatureWithLifespan
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfAbstractVersion	ADEOfAbstractVersion [0..*]	Augments AbstractVersion with properties defined in an ADE.
ion		

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractVersionTransition

Definition: AbstractVersionTransition is the abstract superclass to represent VersionTransition objects.

Subclass of: [AbstractFeatureWithLifespan](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfAbstractVersionTransition	ADEOfAbstractVersionTransition [0..*]	Augments AbstractVersionTransition with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Address

Definition: Address represents an address of a city object.

Subclass of: [AbstractFeature](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
multiPoint	GM_MultiPoint [0..1]	Relates to the MultiPoint geometry of the Address. The geometry relates the address spatially to a city object.
xalAddress	XALAddress [1..1]	Relates an OASIS address object to the Address.

Attribute	Value type and multiplicity	Definition
adeOfAddress	ADEOfAddress [0..*]	Augments the Address with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

CityModel

Definition:	CityModel is the container for all objects belonging to a city model.	
Subclass of:	AbstractFeatureWithLifespan	
Stereotype:	«FeatureType»	
<hr/>		
Role name	Target class and multiplicity	Definition
cityModelMember	CityModelMember [*]	Relates to all objects that are part of the CityModel.
<hr/>		
Attribute	Value type and multiplicity	Definition
engineeringCRS	EngineeringCRS [0..1]	Specifies the local engineering coordinate reference system of the CityModel that can be provided inline the CityModel instead of referencing a well-known CRS definition. 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.
adeOfCityModel	ADEOfCityModel [0..*]	Augments the CityModel with properties defined in an ADE.
<hr/>		
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

CityObjectRelation		
Definition:	CityObjectRelation represents a specific relation from the city object in which the relation is included to another city object.	
Subclass of:	None	
Stereotype:	«ObjectType»	
<hr/>		
Role name	Target class and multiplicity	Definition
genericAttribute	AbstractGenericAttribute [*]	Relates generic attributes to the CityObjectRelation.
<hr/>		
Attribute	Value type and multiplicity	Definition
relationType	RelationTypeValue [1..1]	Indicates the specific type of the CityObjectRelation.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

ClosureSurface

Definition: ClosureSurface is a special type of thematic surface used to close holes in volumetric objects. Closure surfaces are virtual (non-physical) surfaces.

Subclass of: [AbstractThematicSurface](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfClosureSurface	ADEOfClosureSurface ce [0..*]	Augments the ClosureSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

ImplicitGeometry

Definition: ImplicitGeometry is a geometry representation where the shape is stored only once as a prototypical geometry. For example a tree or other vegetation object, a traffic light or a traffic sign. This prototypic geometry object can be re-used or referenced many times, wherever the corresponding feature occurs in the 3D city model.

Subclass of: None

Stereotype: «ObjectType»

Role name	Target class and multiplicity	Definition
relativeGeom	GM_Object [0..1]	Relates to a prototypical geometry in a local coordinate system stored inline with the city model.
referencePoint	GM_Point [1..1]	Relates to a 3D Point geometry that represents the base point of the object in the world coordinate system.
appearance	AbstractAppearance [*]	Relates appearances to the ImplicitGeometry.

Attribute	Value type and multiplicity	Definition
objectID	ID [1..1]	Specifies the unique identifier of the ImplicitGeometry.
transformationMatrix	TransformationMatrix4x4 [1..1]	Specifies the mathematical transformation (translation, rotation, and scaling) between the prototypical geometry and the actual spatial position of the object.
mimeType	MimeTypeValue [0..1]	Specifies the MIME type of the external file that stores the prototypical geometry.
libraryObject	URI [0..1]	Specifies the URI that points to the prototypical geometry stored in an external file.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.2.2. Data Types

AbstractGenericAttribute

- Definition: AbstractGenericAttribute is the abstract superclass for all types of generic attributes.
- Subclass of: None
- Stereotype: «DataType»

ADEOfAbstractAppearance

- Definition: ADEOfAbstractAppearance acts as a hook to define properties within an ADE that are to be added to AbstractAppearance.
- Subclass of: None
- Stereotype: «DataType»

ADEOfAbstractCityObject

- Definition: ADEOfAbstractCityObject acts as a hook to define properties within an ADE that are to be added to AbstractCityObject.
- Subclass of: None
- Stereotype: «DataType»

ADEOfAbstractDynamizer

Definition: ADEOfAbstractDynamizer acts as a hook to define properties within an ADE that are to be added to AbstractDynamizer.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractFeature

Definition: ADEOfAbstractFeature acts as a hook to define properties within an ADE that are to be added to AbstractFeature.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractFeatureWithLifespan

Definition: ADEOfAbstractFeatureWithLifespan acts as a hook to define properties within an ADE that are to be added to AbstractFeatureWithLifespan.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractLogicalSpace

Definition: ADEOfAbstractLogicalSpace acts as a hook to define properties within an ADE that are to be added to AbstractLogicalSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractOccupiedSpace

Definition: ADEOfAbstractOccupiedSpace acts as a hook to define properties within an ADE that are to be added to AbstractOccupiedSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractPhysicalSpace

Definition: ADEOfAbstractPhysicalSpace acts as a hook to define properties within an ADE that are to be added to AbstractPhysicalSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractPointCloud

Definition: ADEOfAbstractPointCloud acts as a hook to define properties within an ADE that are to be added to AbstractPointCloud.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractSpace

Definition: ADEOfAbstractSpace acts as a hook to define properties within an ADE that are to be added to AbstractSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractSpaceBoundary

Definition: ADEOfAbstractSpaceBoundary acts as a hook to define properties within an ADE that are to be added to AbstractSpaceBoundary.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractThematicSurface

Definition: ADEOfAbstractThematicSurface acts as a hook to define properties within an ADE that are to be added to AbstractThematicSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractUnoccupiedSpace

Definition: ADEOfAbstractUnoccupiedSpace acts as a hook to define properties within an ADE that are to be added to AbstractUnoccupiedSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractVersion

Definition: ADEOfAbstractVersion acts as a hook to define properties within an ADE that are to be added to AbstractVersion.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractVersionTransition

Definition: ADEOfAbstractVersionTransition acts as a hook to define properties within an ADE that are to be added to AbstractVersionTransition.

Subclass of: None

Stereotype: «DataType»

ADEOfAddress

Definition: ADEOfAddress acts as a hook to define properties within an ADE that are to be added to an Address.

Subclass of: None

Stereotype: «DataType»

ADEOfCityModel

Definition: ADEOfCityModel acts as a hook to define properties within an ADE that are to be added to a CityModel.

Subclass of: None

Stereotype: «DataType»

ADEOfClosureSurface

Definition: ADEOfClosureSurface acts as a hook to define properties within an ADE that are to be added to a ClosureSurface.

Subclass of: None

Stereotype: «DataType»

ExternalReference

Definition: ExternalReference is a reference to a corresponding object in another information system, for example in the German cadastre (ALKIS), the German topographic information system (ATKIS), or the OS UK MasterMap®.

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
targetResource	URI [1..1]	Specifies the URI that points to the object in the external information system.
informationSystem	URI [0..1]	Specifies the URI that points to the external information system.
relationType	URI [0..1]	Specifies a URI that additionally qualifies the ExternalReference. The URI can point to a definition from an external ontology (e.g. the sameAs relation from OWL) and allows for mapping the ExternalReference to RDF triples.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Occupancy

Definition: Occupancy is an application-dependent indication of what is contained by a feature.

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
numberOfOccupants	Integer [1..1]	Indicates the number of occupants contained by a feature.
interval	IntervalValue [0..1]	Indicates the time period the occupants are contained by a feature.
occupantType	OccupantTypeValue [0..1]	Indicates the specific type of the occupants that are contained by a feature.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

QualifiedArea

Definition:	QualifiedArea is an application-dependent measure of the area of a space or of a thematic surface.
Subclass of:	None
Stereotype:	«DataType»

Attribute	Value type and multiplicity	Definition
area	Area [1..1]	Specifies the value of the QualifiedArea.
typeOfArea	QualifiedAreaTypeValue [1..1]	Indicates the specific type of the QualifiedArea.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

QualifiedVolume

Definition:	QualifiedVolume is an application-dependent measure of the volume of a space.
Subclass of:	None
Stereotype:	«DataType»

Attribute	Value type and multiplicity	Definition
volume	Volume [1..1]	Specifies the value of the QualifiedVolume.
typeOfVolume	QualifiedVolumeTypeValue [1..1]	Indicates the specific type of the QualifiedVolume.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

XALAddress

Definition: XALAddress represents address details according to the OASIS xAL standard.

Subclass of: None

Stereotype: «DataType»

9.2.3. Basic Types

Code

Definition: Code is a basic type for a String-based term, keyword, or name that can additionally have a code space.

Subclass of: None

Stereotype: «BasicType»

Attribute	Value type and multiplicity	Definition
codeSpace	URI [0..1]	Associates the Code with an authority that controls the term, keyword, or name.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

DoubleBetween0and1

Definition: DoubleBetween0and1 is a basic type for values, which are greater or equal than 0 and less or equal than 1. The type is used for color encoding, for example.

Subclass of: None

Stereotype: «BasicType»

Constraint: valueBetween0and1 (OCL): inv: DoubleBetween0and1.allInstances() → forAll(p | p > = 0 and p < = 1)

DoubleBetween0and1List

Definition:	DoubleBetween0and1List is a basic type that represents a list of double values greater or equal than 0 and less or equal than 1. The type is used for color encoding, for example.	
Subclass of:	None	
Stereotype:	«BasicType»	

Attribute	Value type and multiplicity	Definition
-----------	-----------------------------	------------

list	DoubleBetween0an d1 [1..1]	Specifies the list of double values.
------	---	--------------------------------------

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

DoubleList

Definition:	DoubleList is an ordered sequence of double values.	
Subclass of:	None	
Stereotype:	«BasicType»	

Attribute	Value type and multiplicity	Definition
-----------	-----------------------------	------------

list	Real [1..1]	Specifies the list of double values.
------	--------------------	--------------------------------------

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

DoubleOrNilReasonList

Definition:	DoubleOrNilReasonList is a basic type that represents a list of double values and/or nil reasons.	
Subclass of:	None	
Stereotype:	«BasicType»	

Attribute	Value type and multiplicity	Definition
-----------	-----------------------------	------------

list	DoubleOrNilReaso n [1..1]	Specifies the list of double values and/or nil reasons.
------	--	---

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

ID

Definition: ID is a basic type that represents a unique identifier.

Subclass of: None

Stereotype: «BasicType»

IntegerBetween0and3

Definition: IntegerBetween0and3 is a basic type for integer values, which are greater or equal than 0 and less or equal than 3. The type is used for encoding the LOD number.

Subclass of: None

Stereotype: «BasicType»

Constraint: valueBetween0and3 (OCL): inv: IntegerBetween0and3.allInstances() →
forAll(p | p > = 0 and p < = 3)

MeasureOrNilReasonList

Definition: MeasureOrNilReasonList is a basic type that represents a list of double values and/or nil reasons together with a unit of measurement.

Subclass of: [DoubleOrNilReasonList](#)

Stereotype: «BasicType»

Attribute	Value type and multiplicity	Definition
uom	UnitOfMeasure [1..1]	Specifies the unit of measurement of the double values.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TransformationMatrix2x2

Definition: TransformationMatrix2x2 is a 2 by 2 matrix represented as a list of four double values in row major order.

Subclass of: [DoubleList](#)

Stereotype: «BasicType»

Constraint: lengthOfList (OCL): inv: list → size() = 4

TransformationMatrix3x4

Definition: TransformationMatrix3x4 is a 3 by 4 matrix represented as a list of twelve double values in row major order.

Subclass of: [DoubleList](#)

Stereotype: «BasicType»

Constraint: lengthOfList (OCL): inv: list → size() = 12

TransformationMatrix4x4

Definition: TransformationMatrix4x4 is a 4 by 4 matrix represented as a list of sixteen double values in row major order.

Subclass of: [DoubleList](#)

Stereotype: «BasicType»

Constraint: lengthOfList (OCL): inv: list → size() = 16

9.2.4. Unions

CityModelMember

Definition: CityModelMember is a union type that enumerates the different types of objects that can occur as members of a city model.

Stereotype: «Union»

Member name	Type	Definition
cityObjectMember	AbstractCityObject [1..1]	Specifies the city objects that are part of the CityModel.
appearanceMember	AbstractAppearance [1..1]	Specifies the appearances of the CityModel.
versionMember	AbstractVersion [1..1]	Specifies the different versions of the CityModel.
versionTransitionMember	AbstractVersionTransition [1..1]	Specifies the transitions between the different versions of the CityModel.
featureMember	AbstractFeature [1..1]	Specifies the feature objects that are part of the CityModel. It allows to include objects that are not derived from a class defined in the CityGML conceptual model, but from the ISO 19109 class AnyFeature.

DoubleOrNilReason

Definition: DoubleOrNilReason is a union type that allows for choosing between a double value and a nil reason.

Stereotype: «Union»

Member name	Type	Definition
value	Real [1..1]	Specifies the double value.
nilReason	NilReason [1..1]	Specifies the nil reason.

NilReason

Definition: NilReason is a union type that allows for choosing between two different types of nil reason.

Stereotype: «Union»

Member name	Type	Definition
nilReasonEnum	NilReasonEnumeration [1..1]	Indicates a nil reason that is provided in a code list.
URI	URI [1..1]	Specifies a URI that points to a resource that describes the nil reason.

9.2.5. Code Lists

IntervalValue

Definition: IntervalValue is a code list used to specify a time period.

Stereotype: «CodeList»

MimeTypeValue

Definition: MimeTypeValue is a code list used to specify the MIME type of a referenced resource.

Stereotype: «CodeList»

NilReasonEnumeration

Definition: NilReasonEnumeration is a code list that enumerates the different nil reasons.

Stereotype: «CodeList»

OccupantTypeValue

Definition: OccupantTypeValue is a code list used to classify occupants.

Stereotype: «CodeList»

OtherRelationTypeValue

Definition: OtherRelationTypeValue is a code list used to classify other types of city object relations.

Stereotype: «CodeList»

QualifiedAreaTypeValue

Definition: QualifiedAreaTypeValue is a code list used to specify area types.

Stereotype: «CodeList»

QualifiedVolumeTypeValue

Definition: QualifiedVolumeTypeValue is a code list used to specify volume types.

Stereotype: «CodeList»

RelationTypeValue

Definition: RelationTypeValue is a code list used to classify city object relations.

Stereotype: «CodeList»

TemporalRelationTypeValue

Definition: TemporalRelationTypeValue is a code list used to classify temporal city object relations.

Stereotype: «CodeList»

TopologicalRelationTypeValue

Definition: TopologicalRelationTypeValue is a code list used to classify topological city object relations.

Stereotype: «CodeList»

9.2.6. Enumerations

RelativeToTerrain

Definition: RelativeToTerrain enumerates the spatial relations of a city object relative to terrain in a qualitative way.

Stereotype: <>Enumeration>>

Literal value Definition

entirelyAboveTerrain Indicates that the city object is located entirely above the terrain.
ain

substantiallyAboveTerrain Indicates that the city object is for the most part located above the terrain.
eTerrain

substantiallyBelowTerrain Indicates that the city object is located half above the terrain and half below
eAndBelowTerrain the terrain.

n

substantiallyBelowTerrain Indicates that the city object is for the most part located below the terrain.
wTerrain

entirelyBelowTerrain Indicates that the city object is located entirely below the terrain.
ain

RelativeToWater

Definition: RelativeToWater enumerates the spatial relations of a city object relative to the water surface in a qualitative way.

Stereotype: <>Enumeration>>

Literal value	Definition
entirelyAboveWaterSurface	Indicates that the city object is located entirely above the water surface.
substantiallyAboveWaterSurface	Indicates that the city object is for the most part located above the water surface.
substantiallyAboveAndBelowWaterSurface	Indicates that the city object is located half above the water surface and half below the water surface.
substantiallyBelowWaterSurface	Indicates that the city object is for the most part located below the water surface.
entirelyBelowWaterSurface	Indicates that the city object is located entirely below the water surface.
temporarilyAboveAndBelowWaterSurface	Indicates that the city object is temporarily located above or below the water level, because the height of the water surface is varying.

SpaceType

Definition:	SpaceType is an enumeration that characterises a space according to its closure properties.
Stereotype:	<<Enumeration>>

Literal value	Definition
closed	Indicates that the space has boundaries at the bottom, at the top, and on all sides.
open	Indicates that the space has at maximum a boundary at the bottom.
semiOpen	Indicates that the space has a boundary at the bottom and on at least one side.

9.3. Appearance

Description:	The Appearance module supports the modelling of the observable surface properties of CityGML features in the form of textures and material.
Parent Package:	CityGML
Stereotype:	«ApplicationSchema»

9.3.1. Classes

AbstractSurfaceData

Definition: AbstractSurfaceData is the abstract superclass for different kinds of textures and material.

Subclass of: [AbstractFeature](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
isFront	Boolean [0..1]	Indicates whether the texture or material is assigned to the front side or the back side of the surface geometry object.
adeOfAbstract SurfaceData	ADEOfAbstractSurfaceData [0..*]	Augments AbstractSurfaceData with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractTexture

Definition: AbstractTexture is the abstract superclass to represent the common attributes of the classes ParameterizedTexture and GeoreferencedTexture.

Subclass of: [AbstractSurfaceData](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
imageURI	URI [1..1]	Specifies the URI that points to the external image data file.
mimeType	MimeTypeValue [0..1]	Specifies the MIME type of the external point cloud file.
textureType	TextureType [0..1]	Indicates the specific type of the texture.
wrapMode	WrapMode [0..1]	Specifies the behaviour of the texture when the texture is smaller than the surface to which it is applied.
borderColor	ColorPlusOpacity [0..1]	Specifies the color of that part of the surface that is not covered by the texture.
adeOfAbstract Texture	ADEOfAbstractTexture [0..*]	Augments AbstractTexture with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Appearance

Definition: An Appearance is a collection of surface data, i.e. observable properties for surface geometry objects in the form of textures and material.

Subclass of: [AbstractAppearance](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
surfaceData	AbstractSurfaceDat a [*]	Relates to the surface data that are part of the Appearance.
Attribute	Value type and multiplicity	Definition
theme	CharacterString [0..1]	Specifies the topic of the Appearance. Each Appearance contains surface data for one theme only. Examples of themes are infrared radiation, noise pollution, or earthquake-induced structural stress.
adeOfAppeara nce	ADEOfAppearance [0..*]	Augments the Appearance with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

GeoreferencedTexture

Definition: A GeoreferencedTexture is a texture that uses a planimetric projection. It contains an implicit parameterization that is either stored within the image file, an accompanying world file or specified using the orientation and referencePoint elements.

Subclass of: [AbstractTexture](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
referencePoin t	GM_Point [0..1]	Relates to the 2D Point geometry that represents the center of the upper left image pixel in world space.

Attribute	Value type and multiplicity	Definition
preferWorldFi le	Boolean [0..1]	Indicates whether the georeference from the image file or the accompanying world file should be preferred.
orientation	TransformationMatrix2x2 [0..1]	Specifies the rotation and scaling of the image in form of a 2x2 matrix.
target	URI [0..*]	Specifies the URI that points to the surface geometry objects to which the texture is applied.
adeOfGeoreferencedTexture	ADEOfGeoreferencedTexture [0..*]	Augments the GeoreferencedTexture with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

ParameterizedTexture

Definition:	A ParameterizedTexture is a texture that uses texture coordinates or a transformation matrix for parameterization.	
Subclass of:	AbstractTexture	
Stereotype:	«FeatureType»	
Role name	Target class and multiplicity	Definition
textureParameterization	AbstractTextureParameterization [*]	Relates to the texture coordinates or transformation matrices used for parameterization.
Attribute	Value type and multiplicity	Definition
adeOfParameterizedTexture	ADEOfParameterizedTexture [0..*]	Augments the ParameterizedTexture with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TextureAssociation

Definition:	TextureAssociation denotes the relation of a texture to a surface geometry object.
Subclass of:	None
Stereotype:	«ObjectType»

Attribute	Value type and multiplicity	Definition
target	URI [1..1]	Specifies the URI that points to the surface geometry object to which the texture is applied.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

X3DMaterial

Definition:	X3DMaterial defines properties for surface geometry objects based on the material definitions from the X3D and COLLADA standards.	
Subclass of:	AbstractSurfaceData	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
ambientIntensity	DoubleBetween0and1 [0..1]	Specifies the minimum percentage of diffuseColor that is visible regardless of light sources.
diffuseColor	Color [0..1]	Specifies the color of the light diffusely reflected by the surface geometry object.
emissiveColor	Color [0..1]	Specifies the color of the light emitted by the surface geometry object.
specularColor	Color [0..1]	Specifies the color of the light directly reflected by the surface geometry object.
shininess	DoubleBetween0and1 [0..1]	Specifies the sharpness of the specular highlight.
transparency	DoubleBetween0and1 [0..1]	Specifies the degree of transparency of the surface geometry object.
isSmooth	Boolean [0..1]	Specifies which interpolation method is used for the shading of the surface geometry object. If the attribute 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	URI [0..*]	Specifies the URI that points to the surface geometry objects to which the material is applied.
adeOfX3DMaterial	ADEOfX3DMaterial [0..*]	Augments the X3DMaterial with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

9.3.2. Data Types

AbstractTextureParameterization

Definition: AbstractTextureParameterization is the abstract superclass for different kinds of texture parameterizations.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractSurfaceData

Definition: ADEOfAbstractSurfaceData acts as a hook to define properties within an ADE that are to be added to AbstractSurfaceData.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractTexture

Definition: ADEOfAbstractTexture acts as a hook to define properties within an ADE that are to be added to AbstractTexture.

Subclass of: None

Stereotype: «DataType»

ADEOfAppearance

Definition: ADEOfAppearance acts as a hook to define properties within an ADE that are to be added to an Appearance.

Subclass of: None

Stereotype: «DataType»

ADEOfGeoreferencedTexture

Definition: ADEOfGeoreferencedTexture acts as a hook to define properties within an ADE that are to be added to a GeoreferencedTexture.

Subclass of: None

Stereotype: «DataType»

ADEOfParameterizedTexture

Definition: ADEOfParameterizedTexture acts as a hook to define properties within an ADE that are to be added to a ParameterizedTexture.

Subclass of: None

Stereotype: «DataType»

ADEOfX3DMaterial

Definition: ADEOfX3DMaterial acts as a hook to define properties within an ADE that are to be added to an X3DMaterial.

Subclass of: None

Stereotype: «DataType»

TexCoordGen

Definition: TexCoordGen defines texture parameterization using a transformation matrix.

Subclass of: None

Stereotype: «DataType»

Role name	Target class and multiplicity	Definition
crs	SC_CRS [0..1]	Relates to the coordinate reference system of the transformation matrix.

Attribute	Value type and multiplicity	Definition
worldToTextu re	TransformationMa trix3x4 [1..1]	Specifies the 3x4 transformation matrix that defines the transformation between world coordinates and texture coordinates.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TexCoordList

Definition: TexCoordList defines texture parameterization using texture coordinates.

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
textureCoordinates	DoubleList [1..*]	Specifies the coordinates of texture used for parameterization. The texture coordinates are provided separately for each LinearRing of the surface geometry object.
ring	URI [1..*]	Specifies the URIs that point to the LinearRings that are parameterized using the given texture coordinates.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

9.3.3. Basic Types

Color	
Definition:	Color is a list of three double values between 0 and 1 defining an RGB color value.
Subclass of:	DoubleBetween0and1List
Stereotype:	«BasicType»
Constraint:	<code>lengthOfList (OCL): inv: list → size() = 3</code>

ColorPlusOpacity	
Definition:	Color is a list of four double values between 0 and 1 defining an RGBA color value. Opacity value of 0 means transparent.
Subclass of:	DoubleBetween0and1List
Stereotype:	«BasicType»
Constraint:	<code>lengthOfList (OCL): inv: list → size() = 3 or list → size() = 4</code>

9.3.4. Unions

none

9.3.5. Code Lists

none

9.3.6. Enumerations

TextureType

Definition: TextureType enumerates the different texture types.

Stereotype: <>Enumeration>>

Literal value	Definition
specific	Indicates that the texture is specific to a single surface.
typical	Indicates that the texture is characteristic of a surface and can be used repeatedly.
unknown	Indicates that the texture type is not known.

WrapMode

Definition: WrapMode enumerates the different fill modes for textures.

Stereotype: <>Enumeration>>

Literal value	Definition
none	Indicates that the texture is applied to the surface "as is". The part of the surface that is not covered by the texture is shown fully transparent. [cf. COLLADA]
wrap	Indicates that the texture is repeated until the surface is fully covered. [cf. COLLADA]
mirror	Indicates that the texture is repeated and mirrored. [cf. COLLADA]
clamp	Indicates that the texture is stretched to the edges of the surface. [cf. COLLADA]
border	Indicates that the texture is applied to the surface "as is". The part of the surface that is not covered by the texture is filled with the RGBA color that is specified in the attribute borderColor. [cf. COLLADA]

9.4. CityFurniture

Description: The CityFurniture module supports representation of city furniture objects. City furniture objects are immovable objects like lanterns, traffic signs, advertising columns, benches, or bus stops that can be found in traffic areas, residential areas, on squares, or in built-up areas.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.4.1. Classes

CityFurniture

Definition: CityFurniture is an object or piece of equipment installed in the outdoor environment for various purposes. Examples include street signs, traffic signals, street lamps, benches, fountains.

Subclass of: [AbstractOccupiedSpace](#)

Stereotype: «TopLevelFeatureType»

Attribute	Value type and multiplicity	Definition
class	CityFurnitureClass Value [0..1]	Indicates the specific type of the CityFurniture.
function	CityFurnitureFunct ionValue [0..*]	Specifies the intended purposes of the CityFurniture.
usage	CityFurnitureUsage Value [0..*]	Specifies the actual uses of the CityFurniture.
adeOfCityFurniture	ADEOfCityFurniture e [0..*]	Augments the CityFurniture with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.4.2. Data Types

ADEOfCityFurniture

Definition: ADEOfCityFurniture acts as a hook to define properties within an ADE that are to be added to a CityFurniture.

Subclass of: None

Stereotype: «DataType»

9.4.3. Basic Types

none

9.4.4. Unions

none

9.4.5. Code Lists

CityFurnitureClassValue

Definition: CityFurnitureClassValue is a code list used to further classify a CityFurniture.
Stereotype: «CodeList»

CityFurnitureFunctionValue

Definition: CityFurnitureFunctionValue is a code list that enumerates the different purposes of a CityFurniture.
Stereotype: «CodeList»

CityFurnitureUsageValue

Definition: CityFurnitureUsageValue is a code list that enumerates the different uses of a CityFurniture.
Stereotype: «CodeList»

9.4.6. Enumerations

none

9.5. CityObjectGroup

Description: The CityObjectGroup module supports grouping of city objects. Arbitrary city objects may be aggregated in groups according to user-defined criteria. A group may be further classified by application-specific attributes.
Parent Package: CityGML
Stereotype: «ApplicationSchema»

9.5.1. Classes

CityObjectGroup

Definition:	A CityObjectGroup represents an application-specific aggregation of city objects according to some user-defined criteria. Examples for groups are the buildings in a specific region, the result of a query, or objects put together for visualization purposes. Each member of a group may be qualified by a role name, reflecting the role each city object plays in the context of the group.	
Subclass of:	AbstractLogicalSpace	
Stereotype:	«TopLevelFeatureType»	
<hr/>		
Role name	Target class and multiplicity	Definition
parent	AbstractCityObject [0..1]	Relates to a city object to which the CityObjectGroup belongs.
groupMember	AbstractCityObject [*]	Relates to the city objects that are part of the CityObjectGroup.
<hr/>		
Attribute	Value type and multiplicity	Definition
class	CityObjectGroupClass [0..1]	Indicates the specific type of the CityObjectGroup.
function	CityObjectGroupFunction [0..*]	Specifies the intended purposes of the CityObjectGroup.
usage	CityObjectGroupUsage [0..*]	Specifies the actual usages of the CityObjectGroup.
adeOfCityObjectGroup	ADEOfCityObjectGroup [0..*]	Augments the CityObjectGroup with properties defined in an ADE.
<hr/>		
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

Role		
Definition:	Role qualifies the function of a city object within the CityObjectGroup.	
Subclass of:	None	
Stereotype:	«ObjectType»	
<hr/>		
Attribute	Value type and multiplicity	Definition
role	CharacterString [0..1]	Describes the role the city object plays within the CityObjectGroup.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.5.2. Data Types

ADEOfCityObjectGroup

Definition:	ADEOfCityObjectGroup acts as a hook to define properties within an ADE that are to be added to a CityObjectGroup.
Subclass of:	None
Stereotype:	«DataType»

9.5.3. Basic Types

none

9.5.4. Unions

none

9.5.5. Code Lists

CityObjectGroupClassValue

Definition:	CityObjectGroupClassValue is a code list used to further classify a CityObjectGroup.
Stereotype:	«CodeList»

CityObjectGroupFunctionValue

Definition:	CityObjectGroupFunctionValue is a code list that enumerates the different purposes of a CityObjectGroup.
Stereotype:	«CodeList»

CityObjectGroupUsageValue

Definition:	CityObjectGroupUsageValue is a code list that enumerates the different uses of a CityObjectGroup.
Stereotype:	«CodeList»

9.5.6. Enumerations

none

9.6. Dynamizer

- Description: The Dynamizer module supports the injection of timeseries data for individual attributes of CityGML features. Timeseries data can either be retrieved from external Sensor APIs (e.g. OGC SensorThings API, OGC Sensor Observation Services, MQTT, proprietary platforms), external standardized timeseries files (e.g. OGC TimeseriesML or OGC Observations & Measurements), external tabulated files (e.g CSV) or can be represented inline as basic time-value pairs.
- Parent Package: CityGML
- Stereotype: «ApplicationSchema»

9.6.1. Classes

AbstractAtomicTimeseries

- Definition: AbstractAtomicTimeseries represents the attributes and relationships that are common to all kinds of atomic timeseries (GenericTimeseries, TabulatedFileTimeseries, StandardFileTimeseries). An atomic timeseries represents time-varying data of a specific data type for a single contiguous time interval.
- Subclass of: [AbstractTimeseries](#)
- Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
observationProperty	CharacterString [1..1]	Specifies the phenomenon for which the atomic timeseries provides observation values.
uom	CharacterString [0..1]	Specifies the unit of measurement of the observation values.
adeOfAbstractAtomicTimeseries	ADEOfAbstractAtomicTimeseries [0..*]	Augments AbstractAtomicTimeseries with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractTimeseries

Definition:	AbstractTimeseries is the abstract superclass representing any type of timeseries data.													
Subclass of:	AbstractFeature													
Stereotype:	«FeatureType»													
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>firstTimestamp p</td><td>TM_Position [0..1]</td><td>Specifies the beginning of the timeseries.</td></tr> <tr> <td>lastTimestamp p</td><td>TM_Position [0..1]</td><td>Specifies the end of the timeseries.</td></tr> <tr> <td>adeOfAbstractTimeseries</td><td>ADEOfAbstractTimeseries [0..*]</td><td>Augments AbstractTimeseries with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	firstTimestamp p	TM_Position [0..1]	Specifies the beginning of the timeseries.	lastTimestamp p	TM_Position [0..1]	Specifies the end of the timeseries.	adeOfAbstractTimeseries	ADEOfAbstractTimeseries [0..*]	Augments AbstractTimeseries with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition												
firstTimestamp p	TM_Position [0..1]	Specifies the beginning of the timeseries.												
lastTimestamp p	TM_Position [0..1]	Specifies the end of the timeseries.												
adeOfAbstractTimeseries	ADEOfAbstractTimeseries [0..*]	Augments AbstractTimeseries with properties defined in an ADE.												

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

CompositeTimeseries								
Definition:	A CompositeTimeseries is a (possibly recursive) aggregation of atomic and composite timeseries. The components of a composite timeseries must have non-overlapping time intervals.							
Subclass of:	AbstractTimeseries							
Stereotype:	«FeatureType»							
<table border="1"> <thead> <tr> <th>Role name</th><th>Target class and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>component</td><td>TimeseriesComponent [1..*]</td><td>Relates to the atomic and composite timeseries that are part of the CompositeTimeseries. The referenced timeseries are sequentially ordered.</td></tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	component	TimeseriesComponent [1..*]	Relates to the atomic and composite timeseries that are part of the CompositeTimeseries. The referenced timeseries are sequentially ordered.
Role name	Target class and multiplicity	Definition						
component	TimeseriesComponent [1..*]	Relates to the atomic and composite timeseries that are part of the CompositeTimeseries. The referenced timeseries are sequentially ordered.						
Attribute	Value type and multiplicity	Definition						
adeOfCompositeTimeseries	ADEOfCompositeTimeseries [0..*]	Augments the CompositeTimeseries with properties defined in an ADE.						
<p>Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».</p>								

Dynamizer

Definition:	A Dynamizer is an object that injects timeseries data for an individual attribute of the city object in which it is included. The timeseries data overrides the static value of the referenced city object attribute in order to represent dynamic (time-dependent) variations of its value.	
Subclass of:	AbstractDynamizer	
Stereotype:	«FeatureType»	
Role name	Target class and multiplicity	Definition
dynamicData	AbstractTimeseries [0..1]	Relates to the timeseries data that is given either inline within a CityGML dataset or by a link to an external file containing timeseries data.
sensorConnection	SensorConnection [0..1]	Relates to the sensor API that delivers timeseries data.
Attribute	Value type and multiplicity	Definition
attributeRef	CharacterString [1..1]	Specifies the attribute of a CityGML feature whose value is overridden or replaced by the (dynamic) values specified by the Dynamizer.
startTime	TM_Position [0..1]	Specifies the beginning of the time span for which the Dynamizer provides dynamic values.
endTime	TM_Position [0..1]	Specifies the end of the time span for which the Dynamizer provides dynamic values.
adeOfDynamizer	ADEOfDynamizer [0..*]	Augments the Dynamizer with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

GenericTimeseries

Definition:	A GenericTimeseries represents time-varying data in the form of embedded time-value-pairs of a specific data type for a single contiguous time interval.
Subclass of:	AbstractAtomicTimeseries
Stereotype:	«FeatureType»
Constraint:	<pre>dataTypeOfValue (OCL): inv: if valueType = TimeseriesTypeValue::integer then TimeValuePair → forAll(c c.intValue → size()=1) else if valueType = TimeseriesTypeValue::double then TimeValuePair → forAll(c c.doubleValue → size()=1) else if valueType = TimeseriesTypeValue::string then TimeValuePair → forAll(c c.stringValue → size()=1) else if valueType = TimeseriesTypeValue::geometry then TimeValuePair → forAll(c c.geometryValue → size()=1) else if valueType = TimeseriesTypeValue::uri then TimeValuePair → forAll(c c.uriValue → size()=1) else if valueType = TimeseriesTypeValue::bool then TimeValuePair → forAll(c c.boolValue → size()=1) else if valueType = TimeseriesTypeValue::implicitGeometry then TimeValuePair → forAll(c c.implicitGeometryValue → size()=1) else TimeValuePair → forAll(c c.appearanceValue → size()=1)</pre>

Role name	Target class and multiplicity	Definition
timeValuePair	TimeValuePair [1..*]	Relates to the time-value-pairs that are part of the GenericTimeseries.
Attribute	Value type and multiplicity	Definition
valueType	TimeseriesTypeValue [1..1]	Indicates the specific type of all time-value-pairs that are part of the GenericTimeseries.
adeOfGenericTimeseries	ADEOfGenericTimeseries [0..*]	Augments the GenericTimeseries with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

StandardFileTimeseries

Definition:	A StandardFileTimeseries represents time-varying data for a single contiguous time interval. The data is provided in an external file referenced in the StandardFileTimeseries. The data within the external file is encoded according to a dedicated format for the representation of timeseries data such as using the OGC TimeseriesML or OGC Observations & Measurements Standard. The data type of the data has to be specified within the external file.																
Subclass of:	AbstractAtomicTimeseries																
Stereotype:	«FeatureType»																
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>fileLocation</td><td>URI [1..1]</td><td>Specifies the URI that points to the external timeseries file.</td></tr> <tr> <td>fileType</td><td>StandardFileTypeValue blue [1..1]</td><td>Specifies the format used to represent the timeseries data.</td></tr> <tr> <td> mimeType</td><td>MimeTypeValue [0..1]</td><td>Specifies the MIME type of the external timeseries file.</td></tr> <tr> <td>adeOfStandar dFileTimeseri es</td><td>ADEOfStandardFile Timeseries [0..*]</td><td>Augments the StandardFileTimeseries with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	fileLocation	URI [1..1]	Specifies the URI that points to the external timeseries file.	fileType	StandardFileTypeValue blue [1..1]	Specifies the format used to represent the timeseries data.	mimeType	MimeTypeValue [0..1]	Specifies the MIME type of the external timeseries file.	adeOfStandar dFileTimeseri es	ADEOfStandardFile Timeseries [0..*]	Augments the StandardFileTimeseries with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition															
fileLocation	URI [1..1]	Specifies the URI that points to the external timeseries file.															
fileType	StandardFileTypeValue blue [1..1]	Specifies the format used to represent the timeseries data.															
mimeType	MimeTypeValue [0..1]	Specifies the MIME type of the external timeseries file.															
adeOfStandar dFileTimeseri es	ADEOfStandardFile Timeseries [0..*]	Augments the StandardFileTimeseries with properties defined in an ADE.															
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».																	

TabulatedFileTimeseries

Definition:	A TabulatedFileTimeseries represents time-varying data of a specific data type for a single contiguous time interval. The data is provided in an external file referenced in the TabulatedFileTimeseries. The file contains table structured data using an appropriate file format such as comma-separated values (CSV), Microsoft Excel (XLSX) or Google Spreadsheet. The timestamps and the values are given in specific columns of the table. Each row represents a single time-value-pair. A subset of rows can be selected using the idColumn and idValue attributes.
Subclass of:	AbstractAtomicTimeseries
Stereotype:	«FeatureType»
Constraint:	columnNumberOrColumnName (OCL): inv: $(timeColumnNo \rightarrow \text{notEmpty}()) \text{ or } (\text{timeColumnName} \rightarrow \text{notEmpty}()) \text{ and } (valueColumnNo \rightarrow \text{notEmpty}()) \text{ or } (\text{valueColumnName} \rightarrow \text{notEmpty}()) \text{ and } (\text{idValue} \rightarrow \text{notEmpty}()) \text{ implies } (\text{idColumnNo} \rightarrow \text{notEmpty}()) \text{ or } (\text{idColumnName} \rightarrow \text{notEmpty}())$

Attribute	Value type and multiplicity	Definition
fileLocation	URI [1..1]	Specifies the URI that points to the external timeseries file.
fileType	TabulatedFileType Value [1..1]	Specifies the format used to represent the timeseries data.
mimeType	MimeTypeValue [0..1]	Specifies the MIME type of the external timeseries file.
valueType	TimeseriesTypeVal ue [1..1]	Indicates the specific type of the timeseries data.
numberOfHea derLines	Integer [0..1]	Indicates the number of lines at the beginning of the tabulated file that represent headers.
fieldSeparator	CharacterString [1..1]	Indicates which symbol is used to separate the individual values in the tabulated file.
decimalSymb ol	Character [0..1]	Indicates which symbol is used to separate the integer part from the fractional part of a decimal number.
idColumnNo	Integer [0..1]	Specifies the number of the column that stores the identifier of the time-value-pair.
idColumnName e	CharacterString [0..1]	Specifies the name of the column that stores the identifier of the time-value-pair.
idValue	CharacterString [0..1]	Specifies the value of the identifier for which the time-value-pairs are to be selected.
timeColumnN o	Integer [0..1]	Specifies the number of the column that stores the timestamp of the time-value-pair.
timeColumnName ame	CharacterString [0..1]	Specifies the name of the column that stores the timestamp of the time-value-pair.
valueColumn No	Integer [0..1]	Specifies the number of the column that stores the value of the time-value-pair.
valueColumnName Name	CharacterString [0..1]	Specifies the name of the column that stores the value of the time-value-pair.
adeOfTabulat edFileTimeser ies	ADEOfTabulatedFil eTimeseries [0..*]	Augments the TabulatedFileTimeseries with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.6.2. Data Types

ADEOfAbstractAtomicTimeseries

Definition: ADEOfAbstractAtomicTimeseries acts as a hook to define properties within an ADE that are to be added to AbstractAtomicTimeseries.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractTimeseries

Definition: ADEOfAbstractTimeseries acts as a hook to define properties within an ADE that are to be added to AbstractTimeseries.

Subclass of: None

Stereotype: «DataType»

ADEOfCompositeTimeseries

Definition: ADEOfCompositeTimeseries acts as a hook to define properties within an ADE that are to be added to a CompositeTimeseries.

Subclass of: None

Stereotype: «DataType»

ADEOfDynamizer

Definition: ADEOfDynamizer acts as a hook to define properties within an ADE that are to be added to a Dynamizer.

Subclass of: None

Stereotype: «DataType»

ADEOfGenericTimeseries

Definition: ADEOfGenericTimeseries acts as a hook to define properties within an ADE that are to be added to a GenericTimeseries.

Subclass of: None

Stereotype: «DataType»

ADEOfStandardFileTimeseries

Definition: ADEOfStandardFileTimeseries acts as a hook to define properties within an ADE that are to be added to a StandardFileTimeseries.

Subclass of: None

Stereotype: «DataType»

ADEOfTabulatedFileTimeseries

Definition: ADEOfTabulatedFileTimeseries acts as a hook to define properties within an ADE that are to be added to a TabulatedFileTimeseries.

Subclass of: None

Stereotype: «DataType»

SensorConnection

Definition: A SensorConnection provides all details that are required to retrieve a specific datastream from an external sensor web service. This data type comprises the service type (e.g. OGC SensorThings API, OGC Sensor Observation Services, MQTT, proprietary platforms), the URL of the sensor service, the identifier for the sensor or thing, and its observed property as well as information about the required authentication method.

Subclass of: None

Stereotype: «DataType»

Role name	Target class and multiplicity	Definition
sensorLocation	AbstractCityObject n [0..1]	Relates the sensor to the city object where it is located.

Attribute	Value type and multiplicity	Definition
connectionType	SensorConnectionType [1..1]	Indicates the type of Sensor API to which the SensorConnection refers.
observationProperty	CharacterString [1..1]	Specifies the phenomenon for which the SensorConnection provides observations.
uom	CharacterString [0..1]	Specifies the unit of measurement of the observations.
sensorID	CharacterString [0..1]	Specifies the unique identifier of the sensor from which the SensorConnection retrieves observations.
sensorName	CharacterString [0..1]	Specifies the name of the sensor from which the SensorConnection retrieves observations.
observationID	CharacterString [0..1]	Specifies the unique identifier of the observation that is retrieved by the SensorConnection.
datastreamID	CharacterString [0..1]	Specifies the datastream that is retrieved by the SensorConnection.
baseURL	URI [0..1]	Specifies the base URL of the Sensor API request.
authType	AuthenticationType [0..1]	Specifies the type of authentication required to be able to access the Sensor API.
mqttServer	CharacterString [0..1]	Specifies the name of the MQTT Server. This attribute is relevant when the MQTT Protocol is used to connect to a Sensor API.
mqttTopic	CharacterString [0..1]	Names the specific datastream that is retrieved by the SensorConnection. This attribute is relevant when the MQTT Protocol is used to connect to a Sensor API.
linkToObservation	CharacterString [0..1]	Specifies the complete URL to the observation request.
linkToSensorDescription	CharacterString [0..1]	Specifies the complete URL to the sensor description request.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TimeseriesComponent

- Definition: TimeseriesComponent represents an element of a CompositeTimeseries.
- Subclass of: None
- Stereotype: «DataType»

Role name	Target class and multiplicity	Definition
timeseries	AbstractTimeseries [1..1]	Relates a timeseries to the TimeseriesComponent.
Attribute	Value type and multiplicity	Definition
repetitions	Integer [1..1]	Specifies how often the timeseries that is referenced by the TimeseriesComponent should be iterated.
additionalGap	TM_Duration [0..1]	Specifies how much extra time is added after all repetitions as an additional gap.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

TimeValuePair

Definition:	A TimeValuePair represents a value that is valid for a given timepoint. For each TimeValuePair, only one of the value properties can be used mutually exclusive. Which value property has to be provided depends on the selected value type in the GenericTimeSeries feature, in which the TimeValuePair is included.
Subclass of:	None
Stereotype:	«DataType»
Constraint:	<p>singleValue (OCL): inv:</p> $\text{intValue} \rightarrow \text{size}() + \text{doubleValue} \rightarrow \text{size}() + \text{stringValue} \rightarrow \text{size}() + \text{geometryValue} \rightarrow \text{size}() + \text{uriValue} \rightarrow \text{size}() + \text{boolValue} \rightarrow \text{size}() + \text{implicitGeometryValue} \rightarrow \text{size}() + \text{appearanceValue} \rightarrow \text{size}() = 1$

Attribute	Value type and multiplicity	Definition
timestamp	TM_Position [1..1]	Specifies the timepoint at which the value of the TimeValuePair is valid.
intValue	Integer [0..1]	Specifies the "Integer" value of the TimeValuePair.
doubleValue	Real [0..1]	Specifies the "Double" value of the TimeValuePair.
stringValue	CharacterString [0..1]	Specifies the "String" value of the TimeValuePair.
geometryValue	GM_Object [0..1]	Specifies the geometry value of the TimeValuePair.
uriValue	URI [0..1]	Specifies the "URI" value of the TimeValuePair.
boolValue	Boolean [0..1]	Specifies the "Boolean" value of the TimeValuePair.
implicitGeometryValue	ImplicitGeometry [0..1]	Specifies the "ImplicitGeometry" value of the TimeValuePair.
appearanceValue	AbstractAppearance [0..1]	Specifies the "Appearance" value of the TimeValuePair.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.6.3. Basic Types

none

9.6.4. Unions

none

9.6.5. Code Lists

AuthenticationTypeValue

Definition: AuthenticationTypeValue is a code list used to specify the authentication method to be used to access the referenced sensor service. Each value provides enough information such that a software application could determine the required access credentials.

Stereotype: «CodeList»

SensorConnectionTypeValue

Definition: SensorConnectionTypeValue is a code list used to specify the type of the referenced sensor service. Each value provides enough information such that a software application would be able to identify the API type and version.

Stereotype: «CodeList»

StandardFileTypeValue

Definition: StandardFileTypeValue is a code list used to specify the type of the referenced external timeseries data file. Each value provides information about the standard and version.

Stereotype: «CodeList»

TabulatedFileTypeValue

Definition: TabulatedFileTypeValue is a code list used to specify the data format of the referenced external tabulated data file.

Stereotype: «CodeList»

9.6.6. Enumerations

TimeseriesTypeValue

Definition: TimeseriesTypeValue enumerates the possible value types for GenericTimeseries and TimeValuePair.

Stereotype: <>Enumeration>>

Literal value	Definition
int	Indicates that the values of the GenericTimeseries are of type "Integer".
double	Indicates that the values of the GenericTimeseries are of type "Double".
string	Indicates that the values of the GenericTimeseries are of type "String".
geometry	Indicates that the values of the GenericTimeseries are geometries.
uri	Indicates that the values of the GenericTimeseries are of type "URI".
bool	Indicates that the values of the GenericTimeseries are of type "Boolean".
implicitGeometry	Indicates that the values of the GenericTimeseries are of type "ImplicitGeometry".
appearance	Indicates that the values of the GenericTimeseries are of type "Appearance".

9.7. Generics

- Description: The Generics module supports application-specific extensions to the CityGML data model. These extensions may be used to model and exchange additional attributes and features not covered by the predefined thematic classes of CityGML. Generic extensions shall only be used if appropriate thematic classes or attributes are not provided by any other CityGML module.
- Parent Package: CityGML
- Stereotype: «ApplicationSchema»

9.7.1. Classes

GenericLogicalSpace

- Definition: A GenericLogicalSpace is a space that is not represented by any explicitly modelled AbstractLogicalSpace subclass within CityGML.
- Subclass of: [AbstractLogicalSpace](#)
- Stereotype: «TopLevelFeatureType»

Attribute	Value type and multiplicity	Definition
class	GenericLogicalSpace eClassValue [0..1]	Indicates the specific type of the GenericLogicalSpace.
function	GenericLogicalSpace eFunctionValue [0..*]	Specifies the intended purposes of the GenericLogicalSpace.
usage	GenericLogicalSpace eUsageValue [0..*]	Specifies the actual uses of the GenericLogicalSpace.
adeOfGenericLogicalSpace	ADEOfGenericLogicalSpace [0..*]	Augments the GenericLogicalSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

GenericOccupiedSpace

- Definition: A GenericOccupiedSpace is a space that is not represented by any explicitly modelled AbstractOccupiedSpace subclass within CityGML.
- Subclass of: [AbstractOccupiedSpace](#)
- Stereotype: «TopLevelFeatureType»

Attribute	Value type and multiplicity	Definition
class	GenericOccupiedSp aceClassValue [0..1]	Indicates the specific type of the GenericOccupiedSpace.
function	GenericOccupiedSp aceFunctionValue [0..*]	Specifies the intended purposes of the GenericOccupiedSpace.
usage	GenericOccupiedSp aceUsageValue [0..*]	Specifies the actual uses of the GenericOccupiedSpace.
adeOfGeneric OccupiedSpace	ADEOfGenericOccupiedSpace [0..*]	Augments the GenericOccupiedSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

GenericThematicSurface

Definition:	A GenericThematicSurface is a surface that is not represented by any explicitly modelled AbstractThematicSurface subclass within CityGML.
Subclass of:	AbstractThematicSurface
Stereotype:	«TopLevelFeatureType»

Attribute	Value type and multiplicity	Definition
class	GenericThematicSurface rfaceClassValue [0..1]	Indicates the specific type of the GenericThematicSurface.
function	GenericThematicSurface rfaceFunctionValue [0..*]	Specifies the intended purposes of the GenericThematicSurface.
usage	GenericThematicSurface rfaceUsageValue [0..*]	Specifies the actual uses of the GenericThematicSurface.
adeOfGeneric ThematicSurface	ADEOfGenericThematicSurface [0..*]	Augments the GenericThematicSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

GenericUnoccupiedSpace

Definition: A GenericUnoccupiedSpace is a space that is not represented by any explicitly modelled AbstractUnoccupiedSpace subclass within CityGML.

Subclass of: [AbstractUnoccupiedSpace](#)

Stereotype: «TopLevelFeatureType»

Attribute	Value type and multiplicity	Definition
class	GenericUnoccupied SpaceClassValue [0..1]	Indicates the specific type of the GenericUnoccupiedSpace.
function	GenericUnoccupied SpaceFunctionValue e [0..*]	Specifies the intended purposes of the GenericUnoccupiedSpace.
usage	GenericUnoccupied SpaceUsageValue [0..*]	Specifies the actual uses of the GenericUnoccupiedSpace.
adeOfGenericUnoccupiedSpace	ADEOfGenericUnoc [0..*]	Augments the GenericUnoccupiedSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.7.2. Data Types

ADEOfGenericLogicalSpace

Definition: ADEOfGenericLogicalSpace acts as a hook to define properties within an ADE that are to be added to a GenericLogicalSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfGenericOccupiedSpace

Definition: ADEOfGenericOccupiedSpace acts as a hook to define properties within an ADE that are to be added to a GenericOccupiedSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfGenericThematicSurface

Definition: ADEOfGenericThematicSurface acts as a hook to define properties within an ADE that are to be added to a GenericThematicSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfGenericUnoccupiedSpace

Definition: ADEOfGenericUnoccupiedSpace acts as a hook to define properties within an ADE that are to be added to a GenericUnoccupiedSpace.

Subclass of: None

Stereotype: «DataType»

CodeAttribute

Definition: CodeAttribute is a data type used to define generic attributes of type "Code".

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
name	CharacterString [1..1]	Specifies the name of the CodeAttribute.
value	Code [1..1]	Specifies the "Code" value.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

DateAttribute

Definition: DateAttribute is a data type used to define generic attributes of type "Date".

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
name	CharacterString [1..1]	Specifies the name of the DateAttribute.
value	Date [1..1]	Specifies the "Date" value.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

DoubleAttribute

Definition: DoubleAttribute is a data type used to define generic attributes of type "Double".

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
name	CharacterString [1..1]	Specifies the name of the DoubleAttribute.
value	Real [1..1]	Specifies the "Double" value.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

GenericAttributeSet

Definition: A GenericAttributeSet is a named collection of generic attributes.

Subclass of: None

Stereotype: «DataType»

Role name	Target class and multiplicity	Definition
genericAttribute	AbstractGenericAttribute [1..*]	Relates to the generic attributes that are part of the GenericAttributeSet.

Attribute	Value type and multiplicity	Definition
name	CharacterString [1..1]	Specifies the name of the GenericAttributeSet.
codeSpace	URI [0..1]	Associates the GenericAttributeSet with an authority that maintains the collection of generic attributes.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

IntAttribute

Definition: IntAttribute is a data type used to define generic attributes of type "Integer".
 Subclass of: None
 Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
name	CharacterString [1..1]	Specifies the name of the IntAttribute.
value	Integer [1..1]	Specifies the "Integer" value.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

MeasureAttribute

Definition: MeasureAttribute is a data type used to define generic attributes of type "Measure".
 Subclass of: None
 Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
name	CharacterString [1..1]	Specifies the name of the MeasureAttribute.
value	Measure [1..1]	Specifies the value of the MeasureAttribute. The value is of type "Measure", which can additionally provide the units of measure. [cf. ISO 19103]
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

StringAttribute

Definition: StringAttribute is a data type used to define generic attributes of type "String".

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
name	CharacterString [1..1]	Specifies the name of the StringAttribute.
value	CharacterString [1..1]	Specifies the "String" value.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

UriAttribute

Definition: UriAttribute is a data type used to define generic attributes of type "URI".

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
name	CharacterString [1..1]	Specifies the name of the UriAttribute.
value	URI [1..1]	Specifies the "URI" value.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.7.3. Basic Types

none

9.7.4. Unions

none

9.7.5. Code Lists

GenericLogicalSpaceClassValue

Definition: GenericLogicalSpaceClassValue is a code list used to further classify a GenericLogicalSpace.

Stereotype: «CodeList»

GenericLogicalSpaceFunctionValue

Definition: GenericLogicalSpaceFunctionValue is a code list that enumerates the different purposes of a GenericLogicalSpace.

Stereotype: «CodeList»

GenericLogicalSpaceUsageValue

Definition: GenericLogicalSpaceUsageValue is a code list that enumerates the different uses of a GenericLogicalSpace.

Stereotype: «CodeList»

GenericOccupiedSpaceClassValue

Definition: GenericOccupiedSpaceClassValue is a code list used to further classify a GenericOccupiedSpace.

Stereotype: «CodeList»

GenericOccupiedSpaceFunctionValue

Definition: GenericOccupiedSpaceFunctionValue is a code list that enumerates the different purposes of a GenericOccupiedSpace.

Stereotype: «CodeList»

GenericOccupiedSpaceUsageValue

Definition: GenericOccupiedSpaceUsageValue is a code list that enumerates the different uses of a GenericOccupiedSpace.

Stereotype: «CodeList»

GenericThematicSurfaceClassValue

Definition: GenericThematicSurfaceClassValue is a code list used to further classify a GenericThematicSurface.

Stereotype: «CodeList»

GenericThematicSurfaceFunctionValue

Definition: GenericThematicSurfaceFunctionValue is a code list that enumerates the different purposes of a GenericThematicSurface.

Stereotype: «CodeList»

GenericThematicSurfaceUsageValue

Definition: GenericThematicSurfaceUsageValue is a code list that enumerates the different uses of a GenericThematicSurface.

Stereotype: «CodeList»

GenericUnoccupiedSpaceClassValue

Definition: GenericUnoccupiedSpaceClassValue is a code list used to further classify a GenericUnoccupiedSpace.

Stereotype: «CodeList»

GenericUnoccupiedSpaceFunctionValue

Definition: GenericUnoccupiedSpaceFunctionValue is a code list that enumerates the different purposes of a GenericUnoccupiedSpace.

Stereotype: «CodeList»

GenericUnoccupiedSpaceUsageValue

Definition: GenericUnoccupiedSpaceUsageValue is a code list that enumerates the different uses of a GenericUnoccupiedSpace.

Stereotype: «CodeList»

9.7.6. Enumerations

none

9.8. LandUse

Description: The LandUse module supports representation of areas of the earth's surface dedicated to a specific land use.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.8.1. Classes

LandUse		
Attribute	Value type and multiplicity	Definition
class	LandUseClassValue [0..1]	Indicates the specific type of the LandUse.
function	LandUseFunctionValue [0..*]	Specifies the intended purposes of the LandUse.
usage	LandUseUsageValue [0..*]	Specifies the actual uses of the LandUse.
adeOfLandUse	ADEOfLandUse [0..*]	Augments the LandUse with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.8.2. Data Types

ADEOfLandUse

Definition:	ADEOfLandUse acts as a hook to define properties within an ADE that are to be added to a LandUse.
Subclass of:	None
Stereotype:	«DataType»

9.8.3. Basic Types

none

9.8.4. Unions

none

9.8.5. Code Lists

LandUseClassValue

Definition:	LandUseClassValue is a code list used to further classify a LandUse.
Stereotype:	«CodeList»

LandUseFunctionValue

Definition:	LandUseFunctionValue is a code list that enumerates the different purposes of a LandUse.
Stereotype:	«CodeList»

LandUseUsageValue

Definition:	LandUseUsageValue is a code list that enumerates the different uses of a LandUse.
Stereotype:	«CodeList»

9.8.6. Enumerations

none

9.9. PointCloud

Description: The PointCloud module supports representation of CityGML features by a collection of points.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.9.1. Classes

PointCloud		
Role name	Target class and multiplicity	Definition
points	GM_MultiPoint [0..1]	Relates to the 3D MultiPoint geometry of the PointCloud stored inline with the city model.
Attribute	Value type and multiplicity	Definition
mimeType	MimeTypeValue [0..1]	Specifies the MIME type of the external point cloud file.
pointFile	URI [0..1]	Specifies the URI that points to the external point cloud file.
pointFileSrsName	CharacterString [0..1]	Indicates the coordinate reference system used by the external point cloud file.
adeOfPointCloud	ADEOfPointCloud [0..*]	Augments the PointCloud with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.9.2. Data Types

ADEOfPointCloud

Definition:	ADEOfPointCloud acts as a hook to define properties within an ADE that are to be added to a PointCloud.
Subclass of:	None
Stereotype:	«DataType»

9.9.3. Basic Types

none

9.9.4. Unions

none

9.9.5. Code Lists

none

9.9.6. Enumerations

none

9.10. Relief

Description:	The Relief module supports representation of the terrain. CityGML supports terrain representations at different levels of detail, reflecting different accuracies or resolutions. Terrain may be specified as a regular raster or grid, as a TIN, by break lines, and/or by mass points.
Parent Package:	CityGML
Stereotype:	«ApplicationSchema»

9.10.1. Classes

AbstractReliefComponent

Definition:	An AbstractReliefComponent represents an element of the terrain surface - either a TIN, a raster or grid, mass points or break lines.
Subclass of:	AbstractSpaceBoundary
Stereotype:	«FeatureType»
Constraint:	<code>polygonGeometry (OCL): inv: extent.patch → size()=1 and extent.patch → forAll(oclIsKindOf(GM_Polygon))</code>

Role name	Target class and multiplicity	Definition
extent	GM_Surface [0..1]	Indicates the geometrical extent of the terrain component. The geometrical extent is provided as a 2D Surface geometry.
Attribute	Value type and multiplicity	Definition
lod	IntegerBetween0and3 [1..1]	Indicates the Level of Detail of the terrain component.
adeOfAbstractReliefComponent	ADEOfAbstractReliefComponent [0..*]	Augments AbstractReliefComponent with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

BreaklineRelief

Definition:	A BreaklineRelief represents a terrain component with 3D lines. These lines denote break lines or ridge/valley lines.	
Subclass of:	AbstractReliefComponent	
Stereotype:	«FeatureType»	
Role name	Target class and multiplicity	Definition
breaklines	GM_MultiCurve [0..1]	Relates to the 3D MultiCurve geometry of the MassPointRelief. This association role is used to represent break lines.
ridgeOrValleyLines	GM_MultiCurve [0..1]	Relates to the 3D MultiCurve geometry of the MassPointRelief. This association role is used to represent ridge or valley lines.
Attribute	Value type and multiplicity	Definition
adeOfBreaklineRelief	ADEOfBreaklineRelief [0..*]	Augments the BreaklineRelief with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

MassPointRelief

Definition:	A MassPointRelief represents a terrain component as a collection of 3D points.										
Subclass of:	AbstractReliefComponent										
Stereotype:	«FeatureType»										
<table border="1"> <thead> <tr> <th>Role name</th><th>Target class and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>pointCloud</td><td>AbstractPointCloud [0..1]</td><td>Relates to the 3D PointCloud of the MassPointRelief.</td></tr> <tr> <td>reliefPoints</td><td>GM_MultiPoint [0..1]</td><td>Relates to the 3D MultiPoint geometry of the MassPointRelief.</td></tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	pointCloud	AbstractPointCloud [0..1]	Relates to the 3D PointCloud of the MassPointRelief.	reliefPoints	GM_MultiPoint [0..1]	Relates to the 3D MultiPoint geometry of the MassPointRelief.
Role name	Target class and multiplicity	Definition									
pointCloud	AbstractPointCloud [0..1]	Relates to the 3D PointCloud of the MassPointRelief.									
reliefPoints	GM_MultiPoint [0..1]	Relates to the 3D MultiPoint geometry of the MassPointRelief.									
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>adeOfMassPoi ntRelief</td><td>ADEOfMassPointRe lief [0..*]</td><td>Augments the MassPointRelief with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	adeOfMassPoi ntRelief	ADEOfMassPointRe lief [0..*]	Augments the MassPointRelief with properties defined in an ADE.			
Attribute	Value type and multiplicity	Definition									
adeOfMassPoi ntRelief	ADEOfMassPointRe lief [0..*]	Augments the MassPointRelief with properties defined in an ADE.									
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».											

RasterRelief								
Definition:	A RasterRelief represents a terrain component as a regular raster or grid.							
Subclass of:	AbstractReliefComponent							
Stereotype:	«FeatureType»							
<table border="1"> <thead> <tr> <th>Role name</th><th>Target class and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>grid</td><td>CV_DiscreteGridPoi ntCoverage [1]</td><td>Relates to the DiscreteGridPointCoverage of the RasterRelief.</td></tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	grid	CV_DiscreteGridPoi ntCoverage [1]	Relates to the DiscreteGridPointCoverage of the RasterRelief.
Role name	Target class and multiplicity	Definition						
grid	CV_DiscreteGridPoi ntCoverage [1]	Relates to the DiscreteGridPointCoverage of the RasterRelief.						
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>adeOfRasterR elief</td><td>ADEOfRasterRelief [0..*]</td><td>Augments the RasterRelief with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	adeOfRasterR elief	ADEOfRasterRelief [0..*]	Augments the RasterRelief with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition						
adeOfRasterR elief	ADEOfRasterRelief [0..*]	Augments the RasterRelief with properties defined in an ADE.						
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».								

ReliefFeature		

Definition:	A ReliefFeature is a collection of terrain components representing the Earth's surface, also known as the Digital Terrain Model.										
Subclass of:	AbstractSpaceBoundary										
Stereotype:	«TopLevelFeatureType»										
<table border="1"> <thead> <tr> <th>Role name</th><th>Target class and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>reliefComponent</td><td>AbstractReliefComponent [1..*]</td><td>Relates to the terrain components that are part of the ReliefFeature.</td></tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	reliefComponent	AbstractReliefComponent [1..*]	Relates to the terrain components that are part of the ReliefFeature.			
Role name	Target class and multiplicity	Definition									
reliefComponent	AbstractReliefComponent [1..*]	Relates to the terrain components that are part of the ReliefFeature.									
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>lod</td><td>IntegerBetween0and3 [1..1]</td><td>Indicates the Level of Detail of the ReliefFeature.</td></tr> <tr> <td>adeOfReliefFeature</td><td>ADEOfReliefFeature [0..*]</td><td>Augments the ReliefFeature with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	lod	IntegerBetween0and3 [1..1]	Indicates the Level of Detail of the ReliefFeature.	adeOfReliefFeature	ADEOfReliefFeature [0..*]	Augments the ReliefFeature with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition									
lod	IntegerBetween0and3 [1..1]	Indicates the Level of Detail of the ReliefFeature.									
adeOfReliefFeature	ADEOfReliefFeature [0..*]	Augments the ReliefFeature with properties defined in an ADE.									
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».											

TINRelief

Definition:	A TINRelief represents a terrain component as a triangulated irregular network.							
Subclass of:	AbstractReliefComponent							
Stereotype:	«FeatureType»							
<table border="1"> <thead> <tr> <th>Role name</th><th>Target class and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>tin</td><td>GM_TriangulatedSurface [1]</td><td>Relates to the triangulated surface of the TINRelief.</td></tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	tin	GM_TriangulatedSurface [1]	Relates to the triangulated surface of the TINRelief.
Role name	Target class and multiplicity	Definition						
tin	GM_TriangulatedSurface [1]	Relates to the triangulated surface of the TINRelief.						
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>adeOfTINRelief</td><td>ADEOfTINRelief [0..*]</td><td>Augments the TINRelief with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	adeOfTINRelief	ADEOfTINRelief [0..*]	Augments the TINRelief with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition						
adeOfTINRelief	ADEOfTINRelief [0..*]	Augments the TINRelief with properties defined in an ADE.						
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».								

9.10.2. Data Types

ADEOfAbstractReliefComponent

Definition: ADEOfAbstractReliefComponent acts as a hook to define properties within an ADE that are to be added to AbstractReliefComponent.

Subclass of: None

Stereotype: «DataType»

ADEOfBreaklineRelief

Definition: ADEOfBreaklineRelief acts as a hook to define properties within an ADE that are to be added to a BreaklineRelief.

Subclass of: None

Stereotype: «DataType»

ADEOfMassPointRelief

Definition: ADEOfMassPointRelief acts as a hook to define properties within an ADE that are to be added to a MassPointRelief.

Subclass of: None

Stereotype: «DataType»

ADEOfRasterRelief

Definition: ADEOfRasterRelief acts as a hook to define properties within an ADE that are to be added to a RasterRelief.

Subclass of: None

Stereotype: «DataType»

ADEOfReliefFeature

Definition: ADEOfReliefFeature acts as a hook to define properties within an ADE that are to be added to a ReliefFeature.

Subclass of: None

Stereotype: «DataType»

ADEOfTINRelief

Definition:	ADEOfTINRelief acts as a hook to define properties within an ADE that are to be added to a TINRelief.
Subclass of:	None
Stereotype:	«DataType»

9.10.3. Basic Types

none

9.10.4. Unions

none

9.10.5. Code Lists

none

9.10.6. Enumerations

none

9.11. Transportation

Description:	The Transportation module supports representation of the transportation infrastructure. Transportation features include roads, tracks, waterways, railways, and squares. Transportation features may be represented as a network and/or as a collection of spaces or surface elements embedded in a three-dimensional space.
Parent Package:	CityGML
Stereotype:	«ApplicationSchema»

9.11.1. Classes

AbstractTransportationSpace

Definition:	AbstractTransportationSpace is the abstract superclass of transportation objects such as Roads, Tracks, Railways, Waterways or Squares.
Subclass of:	AbstractUnoccupiedSpace
Stereotype:	«FeatureType»

Role name	Target class and multiplicity	Definition
marking	Marking [*]	Relates to the markings that are part of the transportation space.
trafficSpace	TrafficSpace [*]	Relates to the traffic spaces that are part of the transportation space.
auxiliaryTrafficSpace	AuxiliaryTrafficSpace [*]	Relates to the auxiliary traffic spaces that are part of the transportation space.
hole	Hole [*]	Relates to the holes that are part of the transportation space.
Attribute	Value type and multiplicity	Definition
trafficDirection	TrafficDirectionValue [0..1]	Indicates the direction of traffic flow relative to the corresponding linear geometry representation.
occupancy	Occupancy [0..*]	Provides information on the residency of persons, vehicles, or other moving features in the transportation space.
adeOfAbstractTransportationSpace	ADEOfAbstractTransportationSpace [0..*]	Augments AbstractTransportationSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AuxiliaryTrafficArea

- Definition: An AuxiliaryTrafficArea is the ground surface of an AuxiliaryTrafficSpace.
- Subclass of: [AbstractThematicSurface](#)
- Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
class	AuxiliaryTrafficAre aClassValue [0..1]	Indicates the specific type of the AuxiliaryTrafficArea.
function	AuxiliaryTrafficAre aFunctionValue [0..*]	Specifies the intended purposes of the AuxiliaryTrafficArea.
usage	AuxiliaryTrafficAre aUsageValue [0..*]	Specifies the actual uses of the AuxiliaryTrafficArea.
surfaceMaterial	SurfaceMaterialVal [0..1]	Specifies the type of pavement of the AuxiliaryTrafficArea.
adeOfAuxiliaryTrafficArea	ADEOfAuxiliaryTrafficArea [0..*]	Augments the AuxiliaryTrafficArea with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AuxiliaryTrafficSpace

Definition:	An AuxiliaryTrafficSpace is a space within the transportation space not intended for traffic purposes.
Subclass of:	AbstractUnoccupiedSpace
Stereotype:	«FeatureType»

Role name	Target class and multiplicity	Definition
boundary	AuxiliaryTrafficAre a [*]	Relates to the auxiliary traffic areas that bound the AuxiliaryTrafficSpace. This relation is inherited from the Core module.

Attribute	Value type and multiplicity	Definition
class	AuxiliaryTrafficSpace <code>ceClassValue [0..1]</code>	Indicates the specific type of the AuxiliaryTrafficSpace.
function	AuxiliaryTrafficSpace <code>ceFunctionValue [0..*]</code>	Specifies the intended purposes of the AuxiliaryTrafficSpace.
usage	AuxiliaryTrafficSpace <code>ceUsageValue [0..*]</code>	Specifies the actual uses of the AuxiliaryTrafficSpace.
granularity	GranularityValue <code>[1..1]</code>	Defines whether auxiliary traffic spaces are represented by individual ways or by individual lanes, depending on the desired level of spatial and semantic decomposition.
adeOfAuxiliar yTrafficSpace	ADEOfAuxiliaryTrafficSpace <code>[0..*]</code>	Augments the AuxiliaryTrafficSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

ClearanceSpace

Definition:	A ClearanceSpace represents the actual free space above a TrafficArea within which a mobile object can move without contacting an obstruction.
Subclass of:	AbstractUnoccupiedSpace
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
class	ClearanceSpaceClass <code>ssValue [0..*]</code>	Indicates the specific type of the ClearanceSpace.
adeOfClearan ceSpace	ADEOfClearanceSpace <code>[0..*]</code>	Augments the ClearanceSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Hole

Definition:	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.
Subclass of:	AbstractUnoccupiedSpace
Stereotype:	«FeatureType»

Role name	Target class and multiplicity	Definition
boundary	AbstractThematics Surface [*]	Relates to the surfaces that bound the Hole. This relation is inherited from the Core module.
Attribute	Value type and multiplicity	Definition
class	HoleClassValue [0..1]	Indicates the specific type of the Hole.
adeOfHole	ADEOfHole [0..*]	Augments the Hole with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

HoleSurface		
Definition:	A HoleSurface is a representation of the ground surface of a hole.	
Subclass of:	AbstractThematicSurface	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
adeOfHoleSurface	ADEOfHoleSurface [0..*]	Augments the HoleSurface with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

Intersection		
Definition:	An Intersection is a transportation space that is a shared segment of multiple Road, Track, Railway, or Waterway objects (e.g. a crossing of two roads or a level crossing of a road and a railway).	
Subclass of:	AbstractTransportationSpace	
Stereotype:	«FeatureType»	

Attribute	Value type and multiplicity	Definition
class	IntersectionClassValue [0..1]	Indicates the specific type of the Intersection.
adeOfIntersection	ADEOfIntersection [0..*]	Augments the Intersection with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Marking

Definition:	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.
Subclass of:	AbstractThematicSurface
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
class	MarkingClassValue [0..1]	Indicates the specific type of the Marking.
adeOfMarking	ADEOfMarking [0..*]	Augments the Marking with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Railway

Definition:	A Railway is a transportation space used by wheeled vehicles on rails.
Subclass of:	AbstractTransportationSpace
Stereotype:	«TopLevelFeatureType»

Role name	Target class and multiplicity	Definition
intersection	Intersection [*]	Relates to the intersections that are part of the Railway.
section	Section [*]	Relates to the sections that are part of the Railway.

Attribute	Value type and multiplicity	Definition
class	RailwayClassValue [0..1]	Indicates the specific type of the Railway.
function	RailwayFunctionValue [0..*]	Specifies the intended purposes of the Railway.
usage	RailwayUsageValue [0..*]	Specifies the actual uses of the Railway.
adeOfRailway	ADEOfRailway [0..*]	Augments the Railway with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Road

Definition:	A Road is a transportation space used by vehicles, bicycles and/or pedestrians.
Subclass of:	AbstractTransportationSpace
Stereotype:	«TopLevelFeatureType»

Role name	Target class and multiplicity	Definition
intersection	Intersection [*]	Relates to the intersections that are part of the Road.
section	Section [*]	Relates to the sections that are part of the Road.
Attribute	Value type and multiplicity	Definition
class	RoadClassValue [0..1]	Indicates the specific type of the Road.
function	RoadFunctionValue [0..*]	Specifies the intended purposes of the Road.
usage	RoadUsageValue [0..*]	Specifies the actual uses of the Road.
adeOfRoad	ADEOfRoad [0..*]	Augments the Road with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Section

Definition:	A Section is a transportation space that is a segment of a Road, Railway, Track, or Waterway.										
Subclass of:	AbstractTransportationSpace										
Stereotype:	«FeatureType»										
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>class</td><td>SectionClassValue [0..1]</td><td>Indicates the specific type of the Section.</td></tr> <tr> <td>adeOfSection</td><td>ADEOfSection [0..*]</td><td>Augments the Section with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	class	SectionClassValue [0..1]	Indicates the specific type of the Section.	adeOfSection	ADEOfSection [0..*]	Augments the Section with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition									
class	SectionClassValue [0..1]	Indicates the specific type of the Section.									
adeOfSection	ADEOfSection [0..*]	Augments the Section with properties defined in an ADE.									

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Square																	
<p>Definition: A Square is a transportation space for unrestricted movement for vehicles, bicycles and/or pedestrians. This includes plazas as well as large sealed surfaces such as parking lots.</p> <p>Subclass of: AbstractTransportationSpace</p> <p>Stereotype: «TopLevelFeatureType»</p>																	
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>class</td><td>SquareClassValue [0..1]</td><td>Indicates the specific type of the Square.</td></tr> <tr> <td>function</td><td>SquareFunctionVal [0..*]</td><td>Specifies the intended purposes of the Square.</td></tr> <tr> <td>usage</td><td>SquareUsageValue [0..*]</td><td>Specifies the actual uses of the Square.</td></tr> <tr> <td>adeOfSquare</td><td>ADEOfSquare [0..*]</td><td>Augments the Square with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	class	SquareClassValue [0..1]	Indicates the specific type of the Square.	function	SquareFunctionVal [0..*]	Specifies the intended purposes of the Square.	usage	SquareUsageValue [0..*]	Specifies the actual uses of the Square.	adeOfSquare	ADEOfSquare [0..*]	Augments the Square with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition															
class	SquareClassValue [0..1]	Indicates the specific type of the Square.															
function	SquareFunctionVal [0..*]	Specifies the intended purposes of the Square.															
usage	SquareUsageValue [0..*]	Specifies the actual uses of the Square.															
adeOfSquare	ADEOfSquare [0..*]	Augments the Square with properties defined in an ADE.															
<p>Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».</p>																	

Track		

Definition:	A Track is a small path mainly used by pedestrians. Tracks can be segmented into Sections and Intersections.	
Subclass of:	AbstractTransportationSpace	
Stereotype:	«TopLevelFeatureType»	
<hr/>		
Role name	Target class and multiplicity	Definition
section	Section [*]	Relates to the sections that are part of the Track.
intersection	Intersection [*]	Relates to the intersections that are part of the Track.
<hr/>		
Attribute	Value type and multiplicity	Definition
class	TrackClassValue [0..1]	Indicates the specific type of the Track.
function	TrackFunctionValue e [0..*]	Specifies the intended purposes of the Track.
usage	TrackUsageValue [0..*]	Specifies the actual uses of the Track.
adeOfTrack	ADEOfTrack [0..*]	Augments the Track with properties defined in an ADE.
<hr/>		
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

TrafficArea

Definition:	A TrafficArea is the ground surface of a TrafficSpace. Traffic areas are the surfaces upon which traffic actually takes place.
Subclass of:	AbstractThematicSurface
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
class	TrafficAreaClassVal [0..1]	Indicates the specific type of the TrafficArea.
function	TrafficAreaFunction [0..*]	Specifies the intended purposes of the TrafficArea.
usage	TrafficAreaUsageValue [0..*]	Specifies the actual uses of the TrafficArea.
surfaceMaterial	SurfaceMaterialValue [0..1]	Specifies the type of pavement of the TrafficArea.
adeOfTrafficArea	ADEOfTrafficArea [0..*]	Augments the TrafficArea with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TrafficSpace

Role name	Target class and multiplicity	Definition
successor	TrafficSpace [*]	Indicates the successor(s) of the TrafficSpace.
clearanceSpace	ClearanceSpace [*]	Relates to the clearance spaces that are part of the TrafficSpace.
predecessor	TrafficSpace [*]	Indicates the predecessor(s) of the TrafficSpace.
boundary	TrafficArea [*]	Relates to the traffic areas that bound the TrafficSpace. This relation is inherited from the Core module.

Attribute	Value type and multiplicity	Definition
class	TrafficSpaceClassValue [0..1]	Indicates the specific type of the TrafficSpace.
function	TrafficSpaceFunctionValue [0..*]	Specifies the intended purposes of the TrafficSpace.
usage	TrafficSpaceUsageValue [0..*]	Specifies the actual uses of the TrafficSpace.
granularity	GranularityValue [1..1]	Defines whether traffic spaces are represented by individual ways or by individual lanes, depending on the desired level of spatial and semantic decomposition.
trafficDirection	TrafficDirectionValue [0..1]	Indicates the direction of traffic flow relative to the corresponding linear geometry representation.
occupancy	Occupancy [0..*]	Provides information on the residency of persons, vehicles, or other moving features in the TrafficSpace.
adeOfTrafficSpace	ADEOfTrafficSpace [0..*]	Augments the TrafficSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Waterway

Definition:	A Waterway is a transportation space used for the movement of vessels upon or within a water body.
Subclass of:	AbstractTransportationSpace
Stereotype:	«TopLevelFeatureType»

Role name	Target class and multiplicity	Definition
intersection	Intersection [*]	Relates to the intersections that are part of the Waterway.
section	Section [*]	Relates to the sections that are part of the Waterway.

Attribute	Value type and multiplicity	Definition
class	WaterwayClassVal [0..1]	Indicates the specific type of the Waterway.
function	WaterwayFunction Value [0..*]	Specifies the intended purposes of the Waterway.
usage	WaterwayUsageVal [0..*]	Specifies the actual uses of the Waterway.
adeOfWaterway	ADEOfWaterway [0..*]	Augments the Waterway with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.11.2. Data Types

ADEOfAbstractTransportationSpace

- Definition: ADEOfAbstractTransportationSpace acts as a hook to define properties within an ADE that are to be added to AbstractTransportationSpace.
- Subclass of: None
- Stereotype: «DataType»

ADEOfAuxiliaryTrafficArea

- Definition: ADEOfAuxiliaryTrafficArea acts as a hook to define properties within an ADE that are to be added to an AuxiliaryTrafficArea.
- Subclass of: None
- Stereotype: «DataType»

ADEOfAuxiliaryTrafficSpace

- Definition: ADEOfAuxiliaryTrafficSpace acts as a hook to define properties within an ADE that are to be added to an AuxiliaryTrafficSpace.
- Subclass of: None
- Stereotype: «DataType»

ADEOfClearanceSpace

Definition: ADEOfClearanceSpace acts as a hook to define properties within an ADE that are to be added to a ClearanceSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfHole

Definition: ADEOfHole acts as a hook to define properties within an ADE that are to be added to a Hole.

Subclass of: None

Stereotype: «DataType»

ADEOfHoleSurface

Definition: ADEOfHoleSurface acts as a hook to define properties within an ADE that are to be added to a HoleSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfIntersection

Definition: ADEOfIntersection acts as a hook to define properties within an ADE that are to be added to an Intersection.

Subclass of: None

Stereotype: «DataType»

ADEOfMarking

Definition: ADEOfMarking acts as a hook to define properties within an ADE that are to be added to a Marking.

Subclass of: None

Stereotype: «DataType»

ADEOfRailway

Definition: ADEOfRailway acts as a hook to define properties within an ADE that are to be added to a Railway.

Subclass of: None

Stereotype: «DataType»

ADEOfRoad

Definition: ADEOfRoad acts as a hook to define properties within an ADE that are to be added to a Road.

Subclass of: None

Stereotype: «DataType»

ADEOfSection

Definition: ADEOfSection acts as a hook to define properties within an ADE that are to be added to a Section.

Subclass of: None

Stereotype: «DataType»

ADEOfSquare

Definition: ADEOfSquare acts as a hook to define properties within an ADE that are to be added to a Square.

Subclass of: None

Stereotype: «DataType»

ADEOfTrack

Definition: ADEOfTrack acts as a hook to define properties within an ADE that are to be added to a Track.

Subclass of: None

Stereotype: «DataType»

ADEOfTrafficArea

Definition: ADEOfTrafficArea acts as a hook to define properties within an ADE that are to be added to a TrafficArea.

Subclass of: None

Stereotype: «DataType»

ADEOfTrafficSpace

Definition: ADEOfTrafficSpace acts as a hook to define properties within an ADE that are to be added to a TrafficSpace.

Subclass of: None

Stereotype: «DataType»

ADEOfWaterway

Definition: ADEOfWaterway acts as a hook to define properties within an ADE that are to be added to a Waterway.

Subclass of: None

Stereotype: «DataType»

9.11.3. Basic Types

none

9.11.4. Unions

none

9.11.5. Code Lists

AuxiliaryTrafficAreaClassValue

Definition: AuxiliaryTrafficAreaClassValue is a code list used to further classify an AuxiliaryTrafficArea.

Stereotype: «CodeList»

AuxiliaryTrafficAreaFunctionValue

Definition:	AuxiliaryTrafficAreaFunctionValue is a code list that enumerates the different purposes of an AuxiliaryTrafficArea.
Stereotype:	«CodeList»

AuxiliaryTrafficAreaUsageValue

Definition:	AuxiliaryTrafficAreaUsageValue is a code list that enumerates the different uses of an AuxiliaryTrafficArea.
Stereotype:	«CodeList»

AuxiliaryTrafficSpaceClassValue

Definition:	AuxiliaryTrafficSpaceClassValue is a code list used to further classify an AuxiliaryTrafficSpace.
Stereotype:	«CodeList»

AuxiliaryTrafficSpaceFunctionValue

Definition:	AuxiliaryTrafficSpaceFunctionValue is a code list that enumerates the different purposes of an AuxiliaryTrafficSpace.
Stereotype:	«CodeList»

AuxiliaryTrafficSpaceUsageValue

Definition:	AuxiliaryTrafficSpaceUsageValue is a code list that enumerates the different uses of an AuxiliaryTrafficSpace.
Stereotype:	«CodeList»

ClearanceSpaceClassValue

Definition:	ClearanceSpaceClassValue is a code list used to further classify a ClearanceSpace.
Stereotype:	«CodeList»

HoleClassValue

Definition: HoleClassValue is a code list used to further classify a Hole.

Stereotype: «CodeList»

IntersectionClassValue

Definition: IntersectionClassValue is a code list used to further classify an Intersection.

Stereotype: «CodeList»

MarkingClassValue

Definition: MarkingClassValue is a code list used to further classify a Marking.

Stereotype: «CodeList»

RailwayClassValue

Definition: RailwayClassValue is a code list used to further classify a Railway.

Stereotype: «CodeList»

RailwayFunctionValue

Definition: RailwayFunctionValue is a code list that enumerates the different purposes of a Railway.

Stereotype: «CodeList»

RailwayUsageValue

Definition: RailwayUsageValue is a code list that enumerates the different uses of a Railway.

Stereotype: «CodeList»

RoadClassValue

Definition: RoadClassValue is a code list used to further classify a Road.

Stereotype: «CodeList»

RoadFunctionValue

Definition: RoadFunctionValue is a code list that enumerates the different purposes of a Road.

Stereotype: «CodeList»

RoadUsageValue

Definition: RoadUsageValue is a code list that enumerates the different uses of a Road.

Stereotype: «CodeList»

SectionClassValue

Definition: SectionClassValue is a code list used to further classify a Section.

Stereotype: «CodeList»

SquareClassValue

Definition: SquareClassValue is a code list used to further classify a Square.

Stereotype: «CodeList»

SquareFunctionValue

Definition: SquareFunctionValue is a code list that enumerates the different purposes of a Square.

Stereotype: «CodeList»

SquareUsageValue

Definition: SquareUsageValue is a code list that enumerates the different uses of a Square.

Stereotype: «CodeList»

SurfaceMaterialValue

Definition: SurfaceMaterialValue is a code list that enumerates the different surface materials.

Stereotype: «CodeList»

TrackClassValue

Definition: TrackClassValue is a code list used to further classify a Track.

Stereotype: «CodeList»

TrackFunctionValue

Definition: TrackFunctionValue is a code list that enumerates the different purposes of a Track.

Stereotype: «CodeList»

TrackUsageValue

Definition: TrackUsageValue is a code list that enumerates the different uses of a Track.

Stereotype: «CodeList»

TrafficAreaClassValue

Definition: TrafficAreaClassValue is a code list used to further classify a TrafficArea.

Stereotype: «CodeList»

TrafficAreaFunctionValue

Definition: TrafficAreaFunctionValue is a code list that enumerates the different purposes of a TrafficArea.

Stereotype: «CodeList»

TrafficAreaUsageValue

Definition: TrafficAreaUsageValue is a code list that enumerates the different uses of a TrafficArea.

Stereotype: «CodeList»

TrafficSpaceClassValue

Definition: TrafficSpaceClassValue is a code list used to further classify a TrafficSpace.

Stereotype: «CodeList»

TrafficSpaceFunctionValue

Definition: TrafficSpaceFunctionValue is a code list that enumerates the different purposes of a TrafficSpace.

Stereotype: «CodeList»

TrafficSpaceUsageValue

Definition: TrafficSpaceUsageValue is a code list that enumerates the different uses of a TrafficSpace.

Stereotype: «CodeList»

WaterwayClassValue

Definition: WaterwayClassValue is a code list used to further classify a Waterway.

Stereotype: «CodeList»

WaterwayFunctionValue

Definition: WaterwayFunctionValue is a code list that enumerates the different purposes of a Waterway.

Stereotype: «CodeList»

WaterwayUsageValue

Definition:	WaterwayUsageValue is a code list that enumerates the different uses of a Waterway.
Stereotype:	«CodeList»

9.11.6. Enumerations

GranularityValue

Definition: GranularityValue enumerates the different levels of granularity in which transportation objects are represented.

Stereotype: <>Enumeration>>

Literal value	Definition
lane	Indicates that the individual lanes of the transportation object are represented.
way	Indicates that the individual (carriage)ways of the transportation object are represented.

TrafficDirectionValue

Definition: TrafficDirectionValue enumerates the allowed directions of travel of a mobile object.

Stereotype: <>Enumeration>>

Literal value	Definition
forwards	Indicates that traffic flows in the direction of the corresponding linear geometry.
backwards	Indicates that traffic flows in the opposite direction of the corresponding linear geometry.
both	Indicates that traffic flows in both directions.

9.12. Vegetation

Description: The Vegetation module supports representation of vegetation objects with vegetation-specific thematic classes. CityGML's vegetation model distinguishes between solitary vegetation objects like trees, and vegetation areas which represent biotopes like forests or other plant communities.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.12.1. Classes

AbstractVegetationObject

Definition: AbstractVegetationObject is the abstract superclass for all kinds of vegetation objects.

Subclass of: [AbstractOccupiedSpace](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
-----------	-----------------------------	------------

adeOfAbstractVegetationObj [0..*] Augments AbstractVegetationObject with properties defined in an ADE.
ect

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

PlantCover

Definition: A PlantCover represents a space covered by vegetation.

Subclass of: [AbstractVegetationObject](#)

Stereotype: «TopLevelFeatureType»

Attribute	Value type and multiplicity	Definition
-----------	-----------------------------	------------

class [0..1] Indicates the specific type of the PlantCover.

function [0..*] Specifies the intended purposes of the PlantCover.

usage [0..*] Specifies the actual uses of the PlantCover.

averageHeight [0..1] Specifies the average height of the PlantCover.

minHeight [0..1] Specifies the minimum height of the PlantCover.

maxHeight [0..1] Specifies the maximum height of the PlantCover.

adeOfPlantCover [0..*] Augments the PlantCover with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

SolitaryVegetationObject

Definition: A SolitaryVegetationObject represents individual vegetation objects, e.g. trees or bushes.

Subclass of: [AbstractVegetationObject](#)

Stereotype: «TopLevelFeatureType»

Attribute	Value type and multiplicity	Definition
class	SolitaryVegetationObjectClassValue [0..1]	Indicates the specific type of the SolitaryVegetationObject.
function	SolitaryVegetationObjectFunctionValue [0..*]	Specifies the intended purposes of the SolitaryVegetationObject.
usage	SolitaryVegetationObjectUsageValue [0..*]	Specifies the actual uses of the SolitaryVegetationObject.
species	SpeciesValue [0..1]	Indicates the botanical name of the SolitaryVegetationObject.
height	Length [0..1]	Distance between the highest point of the vegetation object and the lowest point of the terrain at the bottom of the object.
trunkDiameter	Length [0..1]	Specifies the diameter of the SolitaryCityObject's trunk.
crownDiameter	Length [0..1]	Specifies the diameter of the SolitaryCityObject's crown.
rootBallDiameter	Length [0..1]	Specifies the diameter of the SolitaryCityObject's root ball.
maxRootBallDepth	Length [0..1]	Specifies the vertical distance between the lowest point of the SolitaryVegetationObject's root ball and the terrain surface.
adeOfSolitaryVegetationObject	ADEOfSolitaryVegetationObject [0..*]	Augments the SolitaryVegetationObject with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.12.2. Data Types

ADEOfAbstractVegetationObject

Definition: ADEOfAbstractVegetationObject acts as a hook to define properties within an ADE that are to be added to AbstractVegetationObject.

Subclass of: None

Stereotype: «DataType»

ADEOfPlantCover

Definition: ADEOfPlantCover acts as a hook to define properties within an ADE that are to be added to a PlantCover.

Subclass of: None

Stereotype: «DataType»

ADEOfSolitaryVegetationObject

Definition: ADEOfSolitaryVegetationObject acts as a hook to define properties within an ADE that are to be added to a SolitaryVegetationObject.

Subclass of: None

Stereotype: «DataType»

9.12.3. Basic Types

none

9.12.4. Unions

none

9.12.5. Code Lists

PlantCoverClassValue

Definition: PlantCoverClassValue is a code list used to further classify a PlantCover.

Stereotype: «CodeList»

PlantCoverFunctionValue

Definition: PlantCoverFunctionValue is a code list that enumerates the different purposes of a PlantCover.

Stereotype: «CodeList»

PlantCoverUsageValue

Definition: PlantCoverUsageValue is a code list that enumerates the different uses of a PlantCover.

Stereotype: «CodeList»

SolitaryVegetationObjectClassValue

Definition: SolitaryVegetationObjectClassValue is a code list used to further classify a SolitaryVegetationObject.

Stereotype: «CodeList»

SolitaryVegetationObjectFunctionValue

Definition: SolitaryVegetationObjectFunctionValue is a code list that enumerates the different purposes of a SolitaryVegetationObject.

Stereotype: «CodeList»

SolitaryVegetationObjectUsageValue

Definition: SolitaryVegetationObjectUsageValue is a code list that enumerates the different uses of a SolitaryVegetationObject.

Stereotype: «CodeList»

SpeciesValue

Definition: A SpeciesValue is a code list that enumerates the species of a SolitaryVegetationObject.

Stereotype: «CodeList»

9.12.6. Enumerations

none

9.13. Versioning

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

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.13.1. Classes

Version		
Definition:	Version represents a defined state of a city model consisting of the dedicated versions of all city object instances that belong to the respective city model version. Versions can have names, a description and can be labeled with an arbitrary number of user defined tags.	
Subclass of:	AbstractVersion	
Stereotype:	«FeatureType»	
Role name	Target class and multiplicity	Definition
versionMember «Version»	AbstractFeatureWithLifespan [*]	Relates to all city objects that are part of the city model version.
Attribute	Value type and multiplicity	Definition
tag	CharacterString [0..*]	Allows for adding keywords to the city model version.
adeOfVersion	ADEOfVersion [0..*]	Augments the Version with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

VersionTransition

Definition: VersionTransition describes the change of the state of a city model from one version to another. Version transitions can have names, a description and can be further qualified by a type and a reason.

Subclass of: [AbstractVersionTransition](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
from	Version [0..1]	Relates to the predecessor version of the VersionTransition.
transaction	Transaction [*]	Relates to all transactions that have been applied as part of the VersionTransition.
to	Version [0..1]	Relates to the successor version of the VersionTransition.
Attribute	Value type and multiplicity	Definition
reason	CharacterString [0..1]	Specifies why the VersionTransition has been carried out.
clonePrecedes	Boolean [1..1]	Indicates whether the set of city object instances belonging to the successor version of the city model is either explicitly enumerated within the successor version object (attribute clonePredecessor=false), or has to be derived from the modifications of the city model provided as a list of transactions on the city object versions contained in the predecessor version (attribute clonePredecessor=true).
sor		
type	TransitionTypeVal [0..1]	Indicates the specific type of the VersionTransition.
adeOfVersion	ADEOfVersionTransition [0..*]	Augments the VersionTransition with properties defined in an ADE.
Transition		

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.13.2. Data Types

ADEOfVersion

- Definition: ADEOfVersion acts as a hook to define properties within an ADE that are to be added to a Version.
- Subclass of: None
- Stereotype: «DataType»

ADEOfVersionTransition

Definition: ADEOfVersionTransition acts as a hook to define properties within an ADE that are to be added to a VersionTransition.

Subclass of: None

Stereotype: «DataType»

Transaction

Definition: Transaction represents a modification of the city model by the creation, termination, or replacement of a specific city object. While the creation of a city object also marks its first object version, the termination marks the end of existence of a real world object and, hence, also terminates the final version of a city object. The replacement of a city object means that a specific version of it is replaced by a new version.

Subclass of: None

Stereotype: «DataType»

Role name	Target class and multiplicity	Definition
newFeature «Version»	AbstractFeatureWithLifespan [0..1]	Relates to the version of the city object subsequent to the Transaction.
oldFeature «Version»	AbstractFeatureWithLifespan [0..1]	Relates to the version of the city object prior to the Transaction.
Attribute	Value type and multiplicity	Definition
type	TransactionTypeValue [1..1]	Indicates the specific type of the Transaction.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.13.3. Basic Types

none

9.13.4. Unions

none

9.13.5. Code Lists

none

9.13.6. Enumerations

TransactionTypeValue

Definition: TransactionTypeValue enumerates the three possible types of transactions: insert, delete, or replace.

Stereotype: <>Enumeration>>

Literal value	Definition
insert	Indicates that the feature referenced from the Transaction via the "newFeature" association has been newly created; the association "oldFeature" is empty in this case.
delete	Indicates that the feature referenced from the Transaction via the "oldFeature" association ceases to exist; the association "newFeature" is empty in this case.
replace	Indicates that the feature referenced from the Transaction via the "oldFeature" association has been replaced by the feature referenced via the "newFeature" association.

TransitionTypeValue

Definition: TransitionTypeValue enumerates the different kinds of version transitions. “planned” and “fork” should be used in cases when from one city model version multiple successor versions are being created. “realized” and “merge” should be used when different city model versions are converging into a common successor version.

Stereotype: <>Enumeration>>

Literal value	Definition
planned	Indicates that the successor version of the city model represents a planning state for a possible future of the city.
realized	Indicates that the predecessor version is the chosen one from a number of possible planning versions.
historicalSuccessor	Indicates that the successor version reflects updates on the city model over time (historical timeline). It shall only be used for at most one version transition outgoing from a city model version.
on fork	Indicates other reasons to create alternative city model versions, for example, when different parties are updating parts of the city model or to reflect the results of different simulation runs.
merge	Indicates other reasons to converge multiple versions back into a common city model version.

9.14. WaterBody

- Description: The WaterBody module supports representation of the thematic aspects and 3D geometry of rivers, canals, lakes, and basins. It does, however, not inherit any hydrological or other dynamic aspects of fluid flow.
- Parent Package: CityGML
- Stereotype: «ApplicationSchema»

9.14.1. Classes

AbstractWaterBoundarySurface

- Definition: AbstractWaterBoundarySurface is the abstract superclass for all kinds of thematic surfaces bounding a water body.
- Subclass of: [AbstractThematicSurface](#)
- Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
-----------	-----------------------------	------------

- | | | |
|-----------------------------------|---|--|
| adeOfAbstractWaterBoundarySurface | ADEOfAbstractWaterBoundarySurface | Augments AbstractWaterBoundarySurface with properties defined in an ADE. |
| rySurface | [0..*] | |

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

WaterBody

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

Subclass of: [AbstractOccupiedSpace](#)

Stereotype: «TopLevelFeatureType»

Role name	Target class and multiplicity	Definition
boundary	AbstractWaterBoundarySurface [*]	
Attribute	Value type and multiplicity	Definition
class	WaterBodyClassVal [0..1]	Indicates the specific type of the WaterBody.
function	WaterBodyFunction [0..*]	Specifies the intended purposes of the WaterBody.
usage	WaterBodyUsageValue [0..*]	Specifies the actual uses of the WaterBody.
adeOfWaterBody	ADEOfWaterBody [0..*]	Augments the WaterBody with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

WaterGroundSurface

Definition: A WaterGroundSurface represents the exterior boundary surface of the submerged bottom of a water body.

Subclass of: [AbstractWaterBoundarySurface](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfWaterGroundSurface	ADEOfWaterGroundSurface [0..*]	Augments the WaterGroundSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

WaterSurface

Definition: A WaterSurface represents the upper exterior interface between a water body and the atmosphere.

Subclass of: [AbstractWaterBoundarySurface](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
waterLevel	WaterLevelValue [0..1]	Specifies the level of the WaterSurface.
adeOfWaterSurface	ADEOfWaterSurface e [0..*]	Augments the WaterSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.14.2. Data Types

ADEOfAbstractWaterBoundarySurface

Definition: ADEOfAbstractWaterBoundarySurface acts as a hook to define properties within an ADE that are to be added to AbstractWaterBoundarySurface.

Subclass of: None

Stereotype: «DataType»

ADEOfWaterBody

Definition: ADEOfWaterBody acts as a hook to define properties within an ADE that are to be added to a WaterBody.

Subclass of: None

Stereotype: «DataType»

ADEOfWaterGroundSurface

Definition: ADEOfWaterGroundSurface acts as a hook to define properties within an ADE that are to be added to a WaterGroundSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfWaterSurface

Definition: ADEOfWaterSurface acts as a hook to define properties within an ADE that are to be added to a WaterSurface.

Subclass of: None

Stereotype: «DataType»

9.14.3. Basic Types

none

9.14.4. Unions

none

9.14.5. Code Lists

WaterBodyClassValue

Definition: WaterBodyClassValue is a code list used to further classify a WaterBody.

Stereotype: «CodeList»

WaterBodyFunctionValue

Definition: WaterBodyFunctionValue is a code list that enumerates the different purposes of a WaterBody.

Stereotype: «CodeList»

WaterBodyUsageValue

Definition: WaterBodyUsageValue is a code list that enumerates the different uses of a WaterBody.

Stereotype: «CodeList»

WaterLevelValue

Definition:	WaterLevelValue is a code list that enumerates the different levels of a water surface.
Stereotype:	«CodeList»

9.14.6. Enumerations

none

9.15. Construction

Description: The Construction module supports representation of key elements of different types of constructions. These key elements include construction surfaces (e.g floor and ceiling), windows and doors, constructive elements (e.g. beams and slabs), installations, and furniture.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.15.1. Classes

AbstractConstruction

Definition: AbstractConstruction is the abstract superclass for objects that are manufactured by humans from construction materials, are connected to earth, and are intended to be permanent. A connection with the ground also exists when the construction rests by its own weight on the ground or is moveable limited on stationary rails or if the construction is intended to be used mainly stationary.

Subclass of: [AbstractOccupiedSpace](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
boundary	AbstractThematicSurface [*]	Relates to the surfaces that bound the construction. This relation is inherited from the Core module.

Attribute	Value type and multiplicity	Definition
conditionOfConstruction	ConditionOfConstructionValue [0..1]	Indicates the life-cycle status of the construction. [cf. INSPIRE]
dateOfConstruction	Date [0..1]	Indicates the date at which the construction was completed.
dateOfDemolition	Date [0..1]	Indicates the date at which the construction was demolished.
constructionEvent	ConstructionEvent [0..*]	Describes specific events in the life-time of the construction.
elevation	Elevation [0..*]	Specifies qualified elevations of the construction in relation to a well-defined surface which is commonly taken as origin (e.g. geoid or water level). [cf. INSPIRE]
height	Height [0..*]	Specifies qualified heights of the construction above ground or below ground. [cf. INSPIRE]
occupancy	Occupancy [0..*]	Provides qualified information on the residency of persons, animals, or other moveable objects in the construction.
adeOfAbstractConstruction	ADEOfAbstractConstruction [0..*]	Augments AbstractConstruction with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractConstructionSurface

Definition:	AbstractConstructionSurface is the abstract superclass for different kinds of surfaces that bound a construction.
Subclass of:	AbstractThematicSurface
Stereotype:	«FeatureType»

Role name	Target class and multiplicity	Definition
fillingSurface	AbstractFillingSurface [*]	Relates to the surfaces that seal the openings of the construction surface.

Attribute	Value type and multiplicity	Definition
adeOfAbstractConstructionSurface	ADEOfAbstractConstructionSurface [0..*]	Augments AbstractConstructionSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractConstructiveElement

Definition: AbstractConstructiveElement is the abstract superclass for the representation of volumetric elements of a construction. Examples are walls, beams, slabs.

Subclass of: [AbstractOccupiedSpace](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
boundary	AbstractThematicsSurface [*]	Relates to the surfaces that bound the constructive element. This relation is inherited from the Core module.
filling	AbstractFillingElement [*]	Relates to the elements that fill the opening of the constructive element.
Attribute	Value type and multiplicity	Definition
isStructuralElement	Boolean [0..1]	Indicates whether the constructive element is essential from a structural point of view. A structural element cannot be omitted without collapsing of the construction. Examples are pylons and anchorages of bridges.
adeOfAbstractConstructiveElement	ADEOfAbstractConstructiveElement [0..*]	Augments AbstractConstructiveElement with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractFillingElement

Definition: AbstractFillingElement is the abstract superclass for different kinds of elements that fill the openings of a construction.

Subclass of: [AbstractOccupiedSpace](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfAbstract FillingElement	ADEOfAbstractFilli ngElement [0..*]	Augments AbstractFillingElement with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

AbstractFillingSurface

Definition: AbstractFillingSurface is the abstract superclass for different kinds of surfaces that seal openings filled by filling elements.

Subclass of: [AbstractThematicSurface](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfAbstract FillingSurface	ADEOfAbstractFilli ngSurface [0..*]	Augments AbstractFillingSurface with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

AbstractFurniture

Definition: AbstractFurniture is the abstract superclass for the representation of furniture objects of a construction.

Subclass of: [AbstractOccupiedSpace](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfAbstract Furniture	ADEOfAbstractFur niture [0..*]	Augments AbstractFurniture with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

AbstractInstallation

Definition:	AbstractInstallation is the abstract superclass for the representation of installation objects of a construction.										
Subclass of:	AbstractOccupiedSpace										
Stereotype:	«FeatureType»										
<table border="1"> <thead> <tr> <th>Role name</th> <th>Target class and multiplicity</th> <th>Definition</th> </tr> </thead> <tbody> <tr> <td>boundary</td> <td>AbstractThematicSurface [*]</td> <td>Relates to the surfaces that bound the installation. This relation is inherited from the Core module.</td> </tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	boundary	AbstractThematicSurface [*]	Relates to the surfaces that bound the installation. This relation is inherited from the Core module.			
Role name	Target class and multiplicity	Definition									
boundary	AbstractThematicSurface [*]	Relates to the surfaces that bound the installation. This relation is inherited from the Core module.									
<table border="1"> <thead> <tr> <th>Attribute</th> <th>Value type and multiplicity</th> <th>Definition</th> </tr> </thead> <tbody> <tr> <td>relationToConstruction</td> <td>RelationToConstruction [0..1]</td> <td>Indicates whether the installation is located inside and/or outside of the construction.</td> </tr> <tr> <td>adeOfAbstractInstallation</td> <td>ADEOfAbstractInstallation [0..*]</td> <td>Augments AbstractInstallation with properties defined in an ADE.</td> </tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	relationToConstruction	RelationToConstruction [0..1]	Indicates whether the installation is located inside and/or outside of the construction.	adeOfAbstractInstallation	ADEOfAbstractInstallation [0..*]	Augments AbstractInstallation with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition									
relationToConstruction	RelationToConstruction [0..1]	Indicates whether the installation is located inside and/or outside of the construction.									
adeOfAbstractInstallation	ADEOfAbstractInstallation [0..*]	Augments AbstractInstallation with properties defined in an ADE.									
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».											

CeilingSurface								
Definition:	A CeilingSurface is a surface that represents the interior ceiling of a construction. An example is the ceiling of a room.							
Subclass of:	AbstractConstructionSurface							
Stereotype:	«FeatureType»							
<table border="1"> <thead> <tr> <th>Attribute</th> <th>Value type and multiplicity</th> <th>Definition</th> </tr> </thead> <tbody> <tr> <td>adeOfCeilingSurface</td> <td>ADEOfCeilingSurface [0..*]</td> <td>Augments the CeilingSurface with properties defined in an ADE.</td> </tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	adeOfCeilingSurface	ADEOfCeilingSurface [0..*]	Augments the CeilingSurface with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition						
adeOfCeilingSurface	ADEOfCeilingSurface [0..*]	Augments the CeilingSurface with properties defined in an ADE.						
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».								

Door

Definition:	A Door is a construction for closing an opening intended primarily for access or egress or both. [cf. ISO 6707-1]	
Subclass of:	AbstractFillingElement	
Stereotype:	«FeatureType»	
<hr/>		
Role name	Target class and multiplicity	Definition
address	Address [*]	Relates to the addresses that are assigned to the Door.
boundary	DoorSurface [*]	Relates to the door surfaces that bound the Door. This relation is inherited from the Core module.
<hr/>		
Attribute	Value type and multiplicity	Definition
class	DoorClassValue [0..1]	Indicates the specific type of the Door.
function	DoorFunctionValue [0..*]	Specifies the intended purposes of the Door.
usage	DoorUsageValue [0..*]	Specifies the actual uses of the Door.
adeOfDoor	ADEOfDoor [0..*]	Augments the Door with properties defined in an ADE.
<hr/>		
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

DoorSurface		
<hr/>		
Definition:	A DoorSurface is either a boundary surface of a Door feature or a surface that seals an opening filled by a door.	
Subclass of:	AbstractFillingSurface	
Stereotype:	«FeatureType»	
<hr/>		
Role name	Target class and multiplicity	Definition
address	Address [*]	Relates to the addresses that are assigned to the DoorSurface.
<hr/>		
Attribute	Value type and multiplicity	Definition
adeOfDoorSurface	ADEOfDoorSurface [0..*]	Augments the DoorSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

FloorSurface

Definition: A FloorSurface is surface that represents the interior floor of a construction.
An example is the floor of a room.

Subclass of: [AbstractConstructionSurface](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfFloorSurface	ADEOfFloorSurface [0..*]	Augments the FloorSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

GroundSurface

Definition: A GroundSurface is a surface that represents the ground plate of a construction. The polygon defining the ground plate is congruent with the footprint of the construction.

Subclass of: [AbstractConstructionSurface](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfGroundSurface	ADEOfGroundSurface [0..*]	Augments the GroundSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

InteriorWallSurface

Definition: An InteriorWallSurface is a surface that is visible from inside a construction.
An example is the wall of a room.

Subclass of: [AbstractConstructionSurface](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfInteriorWallSurface	ADEOfInteriorWallSurface [0..*]	Augments the InteriorWallSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

OtherConstruction

Definition:	An OtherConstruction is a construction that is not covered by any of the other subclasses of AbstractConstruction.
Subclass of:	AbstractConstruction
Stereotype:	«TopLevelFeatureType»

Attribute	Value type and multiplicity	Definition
class	OtherConstruction ClassValue [0..1]	Indicates the specific type of the OtherConstruction.
function	OtherConstruction FunctionValue [0..*]	Specifies the intended purposes of the OtherConstruction.
usage	OtherConstruction UsageValue [0..*]	Specifies the actual uses of the OtherConstruction.
adeOfOtherConstruction	ADEOfOtherConstruction [0..*]	Augments the OtherConstruction with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

OuterCeilingSurface

Definition:	An OuterCeilingSurface is a surface that belongs to the outer building shell with the orientation pointing downwards. An example is the ceiling of a loggia.
Subclass of:	AbstractConstructionSurface
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfOuterCeilingSurface	ADEOfOuterCeilingSurface [0..*]	Augments the OuterCeilingSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

OuterFloorSurface

Definition:	An OuterFloorSurface is a surface that belongs to the outer construction shell with the orientation pointing upwards. An example is the floor of a loggia.
Subclass of:	AbstractConstructionSurface
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfOuterFloorSurface	ADEOfOuterFloorSurface [0..*]	Augments the OuterFloorSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

RoofSurface

Definition:	A RoofSurface is a surface that delimits major roof parts of a construction.
Subclass of:	AbstractConstructionSurface
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
adeOfRoofSurface	ADEOfRoofSurface [0..*]	Augments the RoofSurface with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

WallSurface

Definition:	A WallSurface is a surface that is part of the building facade visible from the outside.							
Subclass of:	AbstractConstructionSurface							
Stereotype:	«FeatureType»							
<table border="1"> <thead> <tr> <th>Attribute</th> <th>Value type and multiplicity</th> <th>Definition</th> </tr> </thead> <tbody> <tr> <td>adeOfWallSurface</td> <td>ADEOfWallSurface [0..*]</td> <td>Augments the WallSurface with properties defined in an ADE.</td> </tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	adeOfWallSurface	ADEOfWallSurface [0..*]	Augments the WallSurface with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition						
adeOfWallSurface	ADEOfWallSurface [0..*]	Augments the WallSurface with properties defined in an ADE.						
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».								

Window

Definition:	A Window is a construction for closing an opening in a wall or roof, primarily intended to admit light and/or provide ventilation. [cf. ISO 6707-1]																
Subclass of:	AbstractFillingElement																
Stereotype:	«FeatureType»																
<table border="1"> <thead> <tr> <th>Role name</th> <th>Target class and multiplicity</th> <th>Definition</th> </tr> </thead> <tbody> <tr> <td>boundary</td> <td>WindowSurface [*]</td> <td>Relates to the window surfaces that bound the Window. This relation is inherited from the Core module.</td> </tr> </tbody> </table>			Role name	Target class and multiplicity	Definition	boundary	WindowSurface [*]	Relates to the window surfaces that bound the Window. This relation is inherited from the Core module.									
Role name	Target class and multiplicity	Definition															
boundary	WindowSurface [*]	Relates to the window surfaces that bound the Window. This relation is inherited from the Core module.															
<table border="1"> <thead> <tr> <th>Attribute</th> <th>Value type and multiplicity</th> <th>Definition</th> </tr> </thead> <tbody> <tr> <td>class</td> <td>WindowClassName [0..1]</td> <td>Indicates the specific type of the Window.</td> </tr> <tr> <td>function</td> <td>WindowFunctionValue [0..*]</td> <td>Specifies the intended purposes of the Window.</td> </tr> <tr> <td>usage</td> <td>WindowUsageValue [0..*]</td> <td>Specifies the actual uses of the Window.</td> </tr> <tr> <td>adeOfWindow</td> <td>ADEOfWindow [0..*]</td> <td>Augments the Window with properties defined in an ADE.</td> </tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	class	WindowClassName [0..1]	Indicates the specific type of the Window.	function	WindowFunctionValue [0..*]	Specifies the intended purposes of the Window.	usage	WindowUsageValue [0..*]	Specifies the actual uses of the Window.	adeOfWindow	ADEOfWindow [0..*]	Augments the Window with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition															
class	WindowClassName [0..1]	Indicates the specific type of the Window.															
function	WindowFunctionValue [0..*]	Specifies the intended purposes of the Window.															
usage	WindowUsageValue [0..*]	Specifies the actual uses of the Window.															
adeOfWindow	ADEOfWindow [0..*]	Augments the Window with properties defined in an ADE.															
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».																	

WindowSurface

Definition:	A WindowSurface is either a boundary surface of a Window feature or a surface that seals an opening filled by a window.							
Subclass of:	AbstractFillingSurface							
Stereotype:	«FeatureType»							
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>adeOfWindow Surface</td><td>ADEOfWindowSurf ace [0..*]</td><td>Augments the WindowSurface with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	adeOfWindow Surface	ADEOfWindowSurf ace [0..*]	Augments the WindowSurface with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition						
adeOfWindow Surface	ADEOfWindowSurf ace [0..*]	Augments the WindowSurface with properties defined in an ADE.						
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».								

9.15.2. Data Types

ADEOfAbstractConstruction
Definition: ADEOfAbstractConstruction acts as a hook to define properties within an ADE that are to be added to AbstractConstruction.
Subclass of: None
Stereotype: «DataType»

ADEOfAbstractConstructionSurface
Definition: ADEOfAbstractConstructionSurface acts as a hook to define properties within an ADE that are to be added to AbstractConstructionSurface.
Subclass of: None
Stereotype: «DataType»

ADEOfAbstractConstructiveElement
Definition: ADEOfAbstractConstructiveElement acts as a hook to define properties within an ADE that are to be added to AbstractConstructiveElement.
Subclass of: None
Stereotype: «DataType»

ADEOfAbstractFillingElement

Definition: ADEOfAbstractFillingElement acts as a hook to define properties within an ADE that are to be added to AbstractFillingElement.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractFillingSurface

Definition: ADEOfAbstractFillingSurface acts as a hook to define properties within an ADE that are to be added to AbstractFillingSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractFurniture

Definition: ADEOfAbstractFurniture acts as a hook to define properties within an ADE that are to be added to AbstractFurniture.

Subclass of: None

Stereotype: «DataType»

ADEOfAbstractInstallation

Definition: ADEOfAbstractInstallation acts as a hook to define properties within an ADE that are to be added to AbstractInstallation.

Subclass of: None

Stereotype: «DataType»

ADEOfCeilingSurface

Definition: ADEOfCeilingSurface acts as a hook to define properties within an ADE that are to be added to a CeilingSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfDoor

Definition: ADEOfDoor acts as a hook to define properties within an ADE that are to be added to a Door.

Subclass of: None

Stereotype: «DataType»

ADEOfDoorSurface

Definition: ADEOfDoorSurface acts as a hook to define properties within an ADE that are to be added to a DoorSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfFloorSurface

Definition: ADEOfFloorSurface acts as a hook to define properties within an ADE that are to be added to a FloorSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfGroundSurface

Definition: ADEOfGroundSurface acts as a hook to define properties within an ADE that are to be added to a GroundSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfInteriorWallSurface

Definition: ADEOfInteriorWallSurface acts as a hook to define properties within an ADE that are to be added to an InteriorWallSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfOtherConstruction

Definition: ADEOfOtherConstruction acts as a hook to define properties within an ADE that are to be added to an OtherConstruction.

Subclass of: None

Stereotype: «DataType»

ADEOfOuterCeilingSurface

Definition: ADEOfOuterCeilingSurface acts as a hook to define properties within an ADE that are to be added to an OuterCeilingSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfOuterFloorSurface

Definition: ADEOfOuterFloorSurface acts as a hook to define properties within an ADE that are to be added to an OuterFloorSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfRoofSurface

Definition: ADEOfRoofSurface acts as a hook to define properties within an ADE that are to be added to a RoofSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfWallSurface

Definition: ADEOfWallSurface acts as a hook to define properties within an ADE that are to be added to a WallSurface.

Subclass of: None

Stereotype: «DataType»

ADEOfWindow

Definition: ADEOfWindow acts as a hook to define properties within an ADE that are to be added to a Window.

Subclass of: None

Stereotype: «DataType»

ADEOfWindowSurface

Definition: ADEOfWindowSurface acts as a hook to define properties within an ADE that are to be added to a WindowSurface.

Subclass of: None

Stereotype: «DataType»

ConstructionEvent

Definition: A ConstructionEvent is a data type used to describe a specific event that is associated with a construction. Examples are the issuing of a building permit or the renovation of a building.

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
event	EventValue [1..1]	Indicates the specific event type.
dateOfEvent	Date [1..1]	Specifies the date at which the event took or will take place.
description	CharacterString [0..1]	Provides additional information on the event.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Elevation

Definition: Elevation is a data type that includes the elevation value itself and information on how this elevation was measured. [cf. INSPIRE]

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
elevationRefere nce	ElevationReference Value [1..1]	Specifies the level from which the elevation was measured. [cf. INSPIRE]
elevationValu e	DirectPosition [1..1]	Specifies the value of the elevation. [cf. INSPIRE]

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Height

Definition:	Height represents a vertical distance (measured or estimated) between a low reference and a high reference. [cf. INSPIRE]
Subclass of:	None
Stereotype:	«DataType»

Attribute	Value type and multiplicity	Definition
highReference	ElevationReference Value [1..1]	Indicates the high point used to calculate the value of the height. [cf. INSPIRE]
lowReference	ElevationReference Value [1..1]	Indicates the low point used to calculate the value of the height. [cf. INSPIRE]
status	HeightStatusValue [1..1]	Indicates the way the height has been captured. [cf. INSPIRE]
value	Length [1..1]	Specifies the value of the height above or below ground. [cf. INSPIRE]

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.15.3. Basic Types

none

9.15.4. Unions

none

9.15.5. Code Lists

DoorClassValue

Definition: DoorClassValue is a code list used to further classify a Door.

Stereotype: «CodeList»

DoorFunctionValue

Definition: DoorFunctionValue is a code list that enumerates the different purposes of a Door.

Stereotype: «CodeList»

DoorUsageValue

Definition: DoorUsageValue is a code list that enumerates the different uses of a Door.

Stereotype: «CodeList»

ElevationReferenceValue

Definition: ElevationReferenceValue is a code list that enumerates the different elevation reference levels used to measure construction heights.

Stereotype: «CodeList»

EventValue

Definition: EventValue is a code list that enumerates the different events of a construction.

Stereotype: «CodeList»

OtherConstructionClassValue

Definition: OtherConstructionClassValue is a code list used to further classify an OtherConstruction.

Stereotype: «CodeList»

OtherConstructionFunctionValue

Definition: OtherConstructionFunctionValue is a code list that enumerates the different purposes of an OtherConstruction.

Stereotype: «CodeList»

OtherConstructionUsageValue

Definition: OtherConstructionUsageValue is a code list that enumerates the different uses of an OtherConstruction.

Stereotype: «CodeList»

WindowClassValue

Definition: WindowClassValue is a code list used to further classify a Window.

Stereotype: «CodeList»

WindowFunctionValue

Definition: WindowFunctionValue is a code list that enumerates the different purposes of a Window.

Stereotype: «CodeList»

WindowUsageValue

Definition: WindowUsageValue is a code list that enumerates the different uses of a Window.

Stereotype: «CodeList»

9.15.6. Enumerations

ConditionOfConstructionValue

Definition: ConditionOfConstructionValue enumerates different conditions of a construction. [cf. INSPIRE]

Stereotype: <<Enumeration>>

Literal value	Definition
declined	Indicates that the construction cannot be used under normal conditions, though its main elements (walls, roof) are still present. [cf. INSPIRE]
demolished	Indicates that the construction has been demolished. There are no more visible remains. [cf. INSPIRE]
functional	Indicates that the construction is functional. [cf. INSPIRE]
projected	Indicates that the construction is being designed. Construction works have not yet started. [cf. INSPIRE]
ruin	Indicates that the construction has been partly demolished and some main elements (roof, walls) have been destroyed. There are some visible remains of the construction. [cf. INSPIRE]
underConstruction	Indicates that the construction is under construction and not yet functional. This applies only to the initial construction works of the construction and not to maintenance work. [cf. INSPIRE]

HeightStatusValue

Definition: HeightStatusValue enumerates the different methods used to capture a height. [cf. INSPIRE]

Stereotype: <<Enumeration>>

Literal value	Definition
estimated	Indicates that the height has been estimated and not measured. [cf. INSPIRE]
measured	Indicates that the height has been (directly or indirectly) measured. [cf. INSPIRE]

RelationToConstruction

Definition: RelationToConstruction is an enumeration used to describe whether an installation is positioned inside and/or outside of a construction.

Stereotype: <<Enumeration>>

Literal value	Definition
inside	Indicates that the installation is positioned inside of the construction.
outside	Indicates that the installation is positioned outside of the construction.
bothInsideAndOut	Indicates that the installation is positioned inside as well as outside of the construction.

9.16. Bridge

Description: The Bridge module supports representation of thematic and spatial aspects of bridges, bridge parts, bridge installations, and interior bridge structures.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.16.1. Classes

AbstractBridge

Definition: AbstractBridge is an abstract superclass representing the common attributes and associations of the classes Bridge and BridgePart.

Subclass of: [AbstractConstruction](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
bridgeConstructiveElement	BridgeConstructive Element [*]	Relates the constructive elements to the Bridge or BridgePart.
bridgeInstallation	BridgeInstallation [*]	Relates the installation objects to the Bridge or BridgePart.
bridgeFurniture	BridgeFurniture [*]	Relates the furniture objects to the Bridge or BridgePart.
bridgeRoom	BridgeRoom [*]	Relates the rooms to the Bridge or BridgePart.
address	Address [*]	Relates the addresses to the Bridge or BridgePart.
Attribute	Value type and multiplicity	Definition
class	BridgeClassValue [0..1]	Indicates the specific type of the Bridge or BridgePart.
function	BridgeFunctionValue [0..*]	Specifies the intended purposes of the Bridge or BridgePart.
usage	BridgeUsageValue [0..*]	Specifies the actual uses of the Bridge or BridgePart.
isMovable	Boolean [0..1]	Indicates whether the Bridge or BridgePart can be moved to allow for watercraft to pass.
adeOfAbstractBridge	ADEOfAbstractBridge [0..*]	Augments AbstractBridge with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Bridge

Definition:	A Bridge represents a structure that affords the passage of pedestrians, animals, vehicles, and service(s) above obstacles or between two points at a height above ground. [cf. ISO 6707-1]
Subclass of:	AbstractBridge
Stereotype:	«TopLevelFeatureType»

Role name	Target class and multiplicity	Definition
bridgePart	BridgePart [*]	Relates the bridge parts to the Bridge.

Attribute	Value type and multiplicity	Definition
adeOfBridge	ADEOfBridge [0..*]	Augments the Bridge with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

BridgeConstructiveElement

Definition:	A BridgeConstructiveElement is an element of a bridge which is essential from a structural point of view. Examples are pylons, anchorages, slabs, beams.
Subclass of:	AbstractConstructiveElement
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
class	BridgeConstructive ElementClassValue [0..1]	Indicates the specific type of the BridgeConstructiveElement.
function	BridgeConstructive ElementFunctionV alue [0..*]	Specifies the intended purposes of the BridgeConstructiveElement.
usage	BridgeConstructive ElementUsageValue [0..*]	Specifies the actual uses of the BridgeConstructiveElement.
adeOfBridgeC onstructiveEle ment	ADEOfBridgeConstr uctiveElement [0..*]	Augments the BridgeConstructiveElement with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

BridgeFurniture

Definition:	A BridgeFurniture is an equipment for occupant use, usually not fixed to the bridge. [cf. ISO 6707-1]
Subclass of:	AbstractFurniture
Stereotype:	«FeatureType»

Attribute	Value type and multiplicity	Definition
class	BridgeFurnitureCla ssValue [0..1]	Indicates the specific type of the BridgeFurniture.
function	BridgeFurnitureFu ctionValue [0..*]	Specifies the intended purposes of the BridgeFurniture.
usage	BridgeFurnitureUs ageValue [0..*]	Specifies the actual uses of the BridgeFurniture.
adeOfBridgeF urniture	ADEOfBridgeFurnit ure [0..*]	Augments the BridgeFurniture with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

BridgeInstallation

Definition:	A BridgeInstallation is a permanent part of a Bridge (inside and/or outside) which does not have the significance of a BridgePart. In contrast to BridgeConstructiveElements, a BridgeInstallation is not essential from a structural point of view. Examples are stairs, antennas or railways.	
Subclass of:	AbstractInstallation	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
class	BridgeInstallationC lassValue [0..1]	Indicates the specific type of the BridgeInstallation.
function	BridgeInstallationF unctionValue [0..*]	Specifies the intended purposes of the BridgeInstallation.
usage	BridgeInstallationU sageValue [0..*]	Specifies the actual uses of the BridgeInstallation.
adeOfBridgeI nstallation	ADEOfBridgeInstall ation [0..*]	Augments the BridgeInstallation with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

BridgePart

Definition:	A BridgePart is a physical or functional subdivision of a Bridge. It would be considered a Bridge, if it were not part of a collection of other BridgeParts.	
Subclass of:	AbstractBridge	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
adeOfBridgeP art	ADEOfBridgePart [0..*]	Augments the BridgePart with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

BridgeRoom

Definition:	A BridgeRoom is a space within a Bridge or BridgePart intended for human occupancy (e.g. a place of work or recreation) and/or containment (storage) of animals or things. A BridgeRoom is bounded physically and/or virtually (e.g. by ClosureSurfaces or GenericSurfaces).	
Subclass of:	AbstractUnoccupiedSpace	
Stereotype:	«FeatureType»	
<hr/>		
Role name	Target class and multiplicity	Definition
bridgeInstallation	BridgeInstallation [*]	Relates to the installation objects to the BridgeRoom.
boundary	AbstractThematicsSurface [*]	Relates to the surfaces that bound the BridgeRoom. This relation is inherited from the Core module.
bridgeFurniture	BridgeFurniture [*]	Relates the furniture objects to the BridgeRoom.
<hr/>		
Attribute	Value type and multiplicity	Definition
class	BridgeRoomClassValue [0..1]	Indicates the specific type of the BridgeRoom.
function	BridgeRoomFunctionValue [0..*]	Specifies the intended purposes of the BridgeRoom.
usage	BridgeRoomUsageValue [0..*]	Specifies the actual uses of the BridgeRoom.
adeOfBridgeRoom	ADEOfBridgeRoom [0..*]	Augments the BridgeRoom with properties defined in an ADE.
<hr/>		
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

9.16.2. Data Types

ADEOfAbstractBridge		
Definition:	ADEOfAbstractBridge acts as a hook to define properties within an ADE that are to be added to AbstractBridge.	
Subclass of:	None	
Stereotype:	«DataType»	

ADEOfBridge

Definition: ADEOfBridge acts as a hook to define properties within an ADE that are to be added to a Bridge.

Subclass of: None

Stereotype: «DataType»

ADEOfBridgeConstructiveElement

Definition: ADEOfBridgeConstructiveElement acts as a hook to define properties within an ADE that are to be added to a BridgeConstructiveElement.

Subclass of: None

Stereotype: «DataType»

ADEOfBridgeFurniture

Definition: ADEOfBridgeFurniture acts as a hook to define properties within an ADE that are to be added to a BridgeFurniture.

Subclass of: None

Stereotype: «DataType»

ADEOfBridgeInstallation

Definition: ADEOfBridgeInstallation acts as a hook to define properties within an ADE that are to be added to a BridgeInstallation.

Subclass of: None

Stereotype: «DataType»

ADEOfBridgePart

Definition: ADEOfBridgePart acts as a hook to define properties within an ADE that are to be added to a BridgePart.

Subclass of: None

Stereotype: «DataType»

ADEOfBridgeRoom

Definition: ADEOfBridgeRoom acts as a hook to define properties within an ADE that are to be added to a BridgeRoom.

Subclass of: None

Stereotype: «DataType»

9.16.3. Basic Types

none

9.16.4. Unions

none

9.16.5. Code Lists

BridgeClassValue

Definition: BridgeClassValue is a code list used to further classify a Bridge.

Stereotype: «CodeList»

BridgeConstructiveElementClassValue

Definition: BridgeConstructiveElementClassValue is a code list used to further classify a BridgeConstructiveElement.

Stereotype: «CodeList»

BridgeConstructiveElementFunctionValue

Definition: BridgeConstructiveElementFunctionValue is a code list that enumerates the different purposes of a BridgeConstructiveElement.

Stereotype: «CodeList»

BridgeConstructiveElementUsageValue

Definition:	BridgeConstructiveElementUsageValue is a code list that enumerates the different uses of a BridgeConstructiveElement.
Stereotype:	«CodeList»

BridgeFunctionValue

Definition:	BridgeFunctionValue is a code list that enumerates the different purposes of a Bridge.
Stereotype:	«CodeList»

BridgeFurnitureClassValue

Definition:	BridgeFurnitureClassValue is a code list used to further classify a BridgeFurniture.
Stereotype:	«CodeList»

BridgeFurnitureFunctionValue

Definition:	BridgeFurnitureFunctionValue is a code list that enumerates the different purposes of a BridgeFurniture.
Stereotype:	«CodeList»

BridgeFurnitureUsageValue

Definition:	BridgeFurnitureUsageValue is a code list that enumerates the different uses of a BridgeFurniture.
Stereotype:	«CodeList»

BridgeInstallationClassValue

Definition:	BridgeInstallationClassValue is a code list used to further classify a BridgeInstallation.
Stereotype:	«CodeList»

BridgeInstallationFunctionValue

Definition: BridgeInstallationFunctionValue is a code list that enumerates the different purposes of a BridgeInstallation.

Stereotype: «CodeList»

BridgeInstallationUsageValue

Definition: BridgeInstallationUsageValue is a code list that enumerates the different uses of a BridgeInstallation.

Stereotype: «CodeList»

BridgeRoomClassValue

Definition: BridgeRoomClassValue is a code list used to further classify a BridgeRoom.

Stereotype: «CodeList»

BridgeRoomFunctionValue

Definition: BridgeRoomFunctionValue is a code list that enumerates the different purposes of a BridgeRoom.

Stereotype: «CodeList»

BridgeRoomUsageValue

Definition: BridgeRoomUsageValue is a code list that enumerates the different uses of a BridgeRoom.

Stereotype: «CodeList»

BridgeUsageValue

Definition: BridgeUsageValue is a code list that enumerates the different uses of a Bridge.

Stereotype: «CodeList»

9.16.6. Enumerations

none

9.17. Building

Description: The Building module supports representation of thematic and spatial aspects of buildings, building parts, building installations, building subdivisions, and interior building structures.

Parent Package: CityGML

Stereotype: «ApplicationSchema»

9.17.1. Classes

AbstractBuilding

Definition: AbstractBuilding is an abstract superclass representing the common attributes and associations of the classes Building and BuildingPart.

Subclass of: [AbstractConstruction](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
buildingFurniture	BuildingFurniture [*]	Relates the furniture objects to the Building or BuildingPart.
buildingRoom	BuildingRoom [*]	Relates the rooms to the Building or BuildingPart.
buildingInstallation	BuildingInstallation [*]	Relates the installation objects to the Building or BuildingPart.
buildingSubdivision	AbstractBuildingSubdivision [*]	Relates the logical subdivisions to the Building or BuildingPart.
buildingConstructiveElement	BuildingConstructiveElement [*]	Relates the constructive elements to the Building or BuildingPart.
address	Address [*]	Relates the addresses to the Building or BuildingPart.

Attribute	Value type and multiplicity	Definition
class	BuildingClassValue [0..1]	Indicates the specific type of the Building or BuildingPart.
function	BuildingFunctionValue [0..*]	Specifies the intended purposes of the Building or BuildingPart.
usage	BuildingUsageValue [0..*]	Specifies the actual uses of the Building or BuildingPart.
roofType	RoofTypeValue [0..1]	Indicates the shape of the roof of the Building or BuildingPart.
storeysAboveGround	Integer [0..1]	Indicates the number of storeys positioned above ground level.
storeysBelowGround	Integer [0..1]	Indicates the number of storeys positioned below ground level.
storeyHeightsAboveGround	MeasureOrNilReasonList [0..1]	Lists the heights of each storey above ground. The first value in the list denotes the height of the storey closest to the ground level, the last value denotes the height furthest away.
storeyHeightsBelowGround	MeasureOrNilReasonList [0..1]	Lists the height of each storey below ground. The first value in the list denotes the height of the storey closest to the ground level, the last value denotes the height furthest away.
adeOfAbstractBuilding	ADEOfAbstractBuilding [0..*]	Augments AbstractBuilding with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

AbstractBuildingSubdivision

- Definition: AbstractBuildingSubdivision is the abstract superclass for different kinds of logical building subdivisions.
- Subclass of: [AbstractLogicalSpace](#)
- Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
buildingRoom	BuildingRoom [*]	Relates the rooms to the building subdivision.
buildingFurniture	BuildingFurniture [*]	Relates the furniture objects to the building subdivision.
buildingConstructiveElement	BuildingConstructiveElement [*]	Relates the constructive elements to the building subdivision.
buildingInstallation	BuildingInstallation [*]	Relates the installation objects to the building subdivision.
Attribute	Value type and multiplicity	Definition
class	BuildingSubdivision nClassValue [0..1]	Indicates the specific type of the building subdivision.
function	BuildingSubdivision nFunctionValue [0..*]	Specifies the intended purposes of the building subdivision.
usage	BuildingSubdivision nUsageValue [0..*]	Specifies the actual uses of the building subdivision.
elevation	Elevation [0..*]	Specifies qualified elevations of the building subdivision in relation to a well-defined surface which is commonly taken as origin (e.g. geoid or water level). [cf. INSPIRE]
sortKey	Real [0..1]	Defines an order among the objects that belong to the building subdivision. An example is the sorting of storeys.
adeOfAbstractBuildingSubdivision	ADEOfAbstractBuildingSubdivision [0..*]	Augments AbstractBuildingSubdivision with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Building

Definition:	A Building is a free-standing, self-supporting construction that is roofed, usually walled, and can be entered by humans and is normally designed to stand permanently in one place. It is intended for human occupancy (e.g. a place of work or recreation), habitation and/or shelter of humans, animals or things.
Subclass of:	AbstractBuilding
Stereotype:	«TopLevelFeatureType»

Role name	Target class and multiplicity	Definition
buildingPart	BuildingPart [*]	Relates the building parts to the Building.
Attribute	Value type and multiplicity	Definition
adeOfBuilding	ADEOfBuilding [0..*]	Augments the Building with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

BuildingConstructiveElement

Attribute	Value type and multiplicity	Definition
class	BuildingConstructiveElementClassVal [0..1]	Indicates the specific type of the BuildingConstructiveElement.
function	BuildingConstructiveElementFunctionValue [0..*]	Specifies the intended purposes of the BuildingConstructiveElement.
usage	BuildingConstructiveElementUsageValue [0..*]	Specifies the actual uses of the BuildingConstructiveElement.
adeOfBuildingConstructiveElement	ADEOfBuildingConstructiveElement [0..*]	Augments the BuildingConstructiveElement with properties defined in an ADE.
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

BuildingFurniture

Definition:	A BuildingFurniture is an equipment for occupant use, usually not fixed to the building. [cf. ISO 6707-1]																
Subclass of:	AbstractFurniture																
Stereotype:	«FeatureType»																
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>class</td><td>BuildingFurnitureC <code>classValue [0..1]</code></td><td>Indicates the specific type of the BuildingFurniture.</td></tr> <tr> <td>function</td><td>BuildingFurnitureF <code>unctionValue [0..*]</code></td><td>Specifies the intended purposes of the BuildingFurniture.</td></tr> <tr> <td>usage</td><td>BuildingFurnitureU <code>sageValue [0..*]</code></td><td>Specifies the actual uses of the BuildingFurniture.</td></tr> <tr> <td>adeOfBuilding Furniture</td><td>ADEOfBuildingFurniture <code>[0..*]</code></td><td>Augments the BuildingFurniture with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	class	BuildingFurnitureC <code>classValue [0..1]</code>	Indicates the specific type of the BuildingFurniture.	function	BuildingFurnitureF <code>unctionValue [0..*]</code>	Specifies the intended purposes of the BuildingFurniture.	usage	BuildingFurnitureU <code>sageValue [0..*]</code>	Specifies the actual uses of the BuildingFurniture.	adeOfBuilding Furniture	ADEOfBuildingFurniture <code>[0..*]</code>	Augments the BuildingFurniture with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition															
class	BuildingFurnitureC <code>classValue [0..1]</code>	Indicates the specific type of the BuildingFurniture.															
function	BuildingFurnitureF <code>unctionValue [0..*]</code>	Specifies the intended purposes of the BuildingFurniture.															
usage	BuildingFurnitureU <code>sageValue [0..*]</code>	Specifies the actual uses of the BuildingFurniture.															
adeOfBuilding Furniture	ADEOfBuildingFurniture <code>[0..*]</code>	Augments the BuildingFurniture with properties defined in an ADE.															

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

BuildingInstallation		
Definition:	A BuildingInstallation is a permanent part of a Building (inside and/or outside) which has not the significance of a BuildingPart. Examples are stairs, antennas, balconies or small roofs.	
Subclass of:	AbstractInstallation	
Stereotype:	«FeatureType»	
Attribute	Value type and multiplicity	Definition
class	BuildingInstallatio <code>nClassValue [0..1]</code>	Indicates the specific type of the BuildingInstallation.
function	BuildingInstallatio <code>nFunctionValue [0..*]</code>	Specifies the intended purposes of the BuildingInstallation.
usage	BuildingInstallatio <code>nUsageValue [0..*]</code>	Specifies the actual uses of the BuildingInstallation.
adeOfBuilding Installation	ADEOfBuildingInst <code>allation [0..*]</code>	Augments the BuildingInstallation with properties defined in an ADE.
<p>Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».</p>		

BuildingPart

Definition: A BuildingPart is a physical or functional subdivision of a Building. It would be considered a Building, if it were not part of a collection of other BuildingParts.

Subclass of: [AbstractBuilding](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
------------------	------------------------------------	-------------------

adeOfBuilding Part	ADEOfBuildingPart [0..*]	Augments the BuildingPart with properties defined in an ADE.
--------------------	--	--

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

BuildingRoom

Definition: A BuildingRoom is a space within a Building or BuildingPart intended for human occupancy (e.g. a place of work or recreation) and/or containment of animals or things. A BuildingRoom is bounded physically and/or virtually (e.g. by ClosureSurfaces or GenericSurfaces).

Subclass of: [AbstractUnoccupiedSpace](#)

Stereotype: «FeatureType»

Role name	Target class and multiplicity	Definition
------------------	--------------------------------------	-------------------

buildingInstallation	BuildingInstallation n [*]	Relates the installation objects to the BuildingRoom.
----------------------	--	---

buildingFurniture	BuildingFurniture [*]	Relates the furniture objects to the BuildingRoom.
-------------------	---------------------------------------	--

boundary	AbstractThematicSurface [*]	Relates to the surfaces that bound the BuildingRoom. This relation is inherited from the Core module.
----------	---	---

Attribute	Value type and multiplicity	Definition
class	BuildingRoomClass Value [0..1]	Indicates the specific type of the BuildingRoom.
function	BuildingRoomFunc tionValue [0..*]	Specifies the intended purposes of the BuildingRoom.
usage	BuildingRoomUsag eValue [0..*]	Specifies the actual uses of the BuildingRoom.
roomHeight	RoomHeight [0..*]	Specifies qualified heights of the BuildingRoom.
adeOfBuilding Room	ADEOfBuildingRoo m [0..*]	Augments the BuildingRoom with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

BuildingUnit

Definition:	A BuildingUnit is a logical subdivision of a Building. BuildingUnits are formed according to some homogeneous property like function, ownership, management, or accessibility. They may be separately sold, rented out, inherited, managed, etc.	
Subclass of:	AbstractBuildingSubdivision	
Stereotype:	«FeatureType»	
Role name	Target class and multiplicity	Definition
storey	Storey [*]	Relates to the storeys on which the BuildingUnit is located.
address	Address [*]	Relates to the addresses that are assigned to the BuildingUnit.
Attribute	Value type and multiplicity	Definition
adeOfBuilding Unit	ADEOfBuildingUnit [0..*]	Augments the BuildingUnit with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Storey

Definition:	A Storey is typically a horizontal section of a Building. Storeys are not always defined according to the building structure, but can also be defined according to logical considerations.																
Subclass of:	AbstractBuildingSubdivision																
Stereotype:	«FeatureType»																
<table border="1"> <thead> <tr> <th>Role name</th><th>Target class and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>boundary</td><td>AbstractThematicSurface [*]</td><td>Relates to the surfaces that bound the Storey. This relation is inherited from the Core module.</td></tr> <tr> <td>buildingUnit</td><td>BuildingUnit [*]</td><td>Relates to the building units that belong to the Storey.</td></tr> </tbody> </table> <table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>adeOfStorey</td><td>ADEOfStorey [0..*]</td><td>Augments the Storey with properties defined in an ADE.</td></tr> </tbody> </table> <p>Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».</p>			Role name	Target class and multiplicity	Definition	boundary	AbstractThematicSurface [*]	Relates to the surfaces that bound the Storey. This relation is inherited from the Core module.	buildingUnit	BuildingUnit [*]	Relates to the building units that belong to the Storey.	Attribute	Value type and multiplicity	Definition	adeOfStorey	ADEOfStorey [0..*]	Augments the Storey with properties defined in an ADE.
Role name	Target class and multiplicity	Definition															
boundary	AbstractThematicSurface [*]	Relates to the surfaces that bound the Storey. This relation is inherited from the Core module.															
buildingUnit	BuildingUnit [*]	Relates to the building units that belong to the Storey.															
Attribute	Value type and multiplicity	Definition															
adeOfStorey	ADEOfStorey [0..*]	Augments the Storey with properties defined in an ADE.															

9.17.2. Data Types

ADEOfAbstractBuilding

Definition:	ADEOfAbstractBuilding acts as a hook to define properties within an ADE that are to be added to AbstractBuilding.
Subclass of:	None
Stereotype:	«DataType»

ADEOfAbstractBuildingSubdivision

Definition:	ADEOfAbstractBuildingSubdivision acts as a hook to define properties within an ADE that are to be added to AbstractBuildingSubdivision.
Subclass of:	None
Stereotype:	«DataType»

ADEOfBuilding

Definition: ADEOfBuilding acts as a hook to define properties within an ADE that are to be added to a Building.

Subclass of: None

Stereotype: «DataType»

ADEOfBuildingConstructiveElement

Definition: ADEOfBuildingConstructiveElement acts as a hook to define properties within an ADE that are to be added to a BuildingConstructiveElement.

Subclass of: None

Stereotype: «DataType»

ADEOfBuildingFurniture

Definition: ADEOfBuildingFurniture acts as a hook to define properties within an ADE that are to be added to a BuildingFurniture.

Subclass of: None

Stereotype: «DataType»

ADEOfBuildingInstallation

Definition: ADEOfBuildingInstallation acts as a hook to define properties within an ADE that are to be added to a BuildingInstallation.

Subclass of: None

Stereotype: «DataType»

ADEOfBuildingPart

Definition: ADEOfBuildingPart acts as a hook to define properties within an ADE that are to be added to a BuildingPart.

Subclass of: None

Stereotype: «DataType»

ADEOfBuildingRoom

Definition: ADEOfBuildingRoom acts as a hook to define properties within an ADE that are to be added to a BuildingRoom.

Subclass of: None

Stereotype: «DataType»

ADEOfBuildingUnit

Definition: ADEOfBuildingUnit acts as a hook to define properties within an ADE that are to be added to a BuildingUnit.

Subclass of: None

Stereotype: «DataType»

ADEOfStorey

Definition: ADEOfStorey acts as a hook to define properties within an ADE that are to be added to a Storey.

Subclass of: None

Stereotype: «DataType»

RoomHeight

Definition: The RoomHeight represents a vertical distance (measured or estimated) between a low reference and a high reference. [cf. INSPIRE]

Subclass of: None

Stereotype: «DataType»

Attribute	Value type and multiplicity	Definition
highReference	RoomElevationRef [1..1]	Indicates the high point used to calculate the value of the room height.
lowReference	RoomElevationRef [1..1]	Indicates the low point used to calculate the value of the room height.
status	HeightStatusValue [1..1]	Indicates the way the room height has been captured.
value	Length [1..1]	Specifies the value of the room height.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

9.17.3. Basic Types

none

9.17.4. Unions

none

9.17.5. Code Lists

BuildingClassValue

Definition: BuildingClassValue is a code list used to further classify a Building.

Stereotype: «CodeList»

BuildingConstructiveElementClassValue

Definition: BuildingConstructiveElementClassValue is a code list used to further classify a BuildingConstructiveElement.

Stereotype: «CodeList»

BuildingConstructiveElementFunctionValue

Definition: BuildingConstructiveElementFunctionValue is a code list that enumerates the different purposes of a BuildingConstructiveElement.

Stereotype: «CodeList»

BuildingConstructiveElementUsageValue

Definition: BuildingConstructiveElementUsageValue is a code list that enumerates the different uses of a BuildingConstructiveElement.

Stereotype: «CodeList»

BuildingFunctionValue

Definition:	BuildingFunctionValue is a code list that enumerates the different purposes of a Building.
Stereotype:	«CodeList»

BuildingFurnitureClassValue

Definition:	BuildingFurnitureClassValue is a code list used to further classify a BuildingFurniture.
Stereotype:	«CodeList»

BuildingFurnitureFunctionValue

Definition:	BuildingFurnitureFunctionValue is a code list that enumerates the different purposes of a BuildingFurniture.
Stereotype:	«CodeList»

BuildingFurnitureUsageValue

Definition:	BuildingFurnitureUsageValue is a code list that enumerates the different uses of a BuildingFurniture.
Stereotype:	«CodeList»

BuildingInstallationClassValue

Definition:	BuildingInstallationClassValue is a code list used to further classify a BuildingInstallation.
Stereotype:	«CodeList»

BuildingInstallationFunctionValue

Definition:	BuildingInstallationFunctionValue is a code list that enumerates the different purposes of a BuildingInstallation.
Stereotype:	«CodeList»

BuildingInstallationUsageValue

Definition: BuildingInstallationUsageValue is a code list that enumerates the different uses of a BuildingInstallation.

Stereotype: «CodeList»

BuildingRoomClassValue

Definition: BuildingRoomClassValue is a code list used to further classify a BuildingRoom.

Stereotype: «CodeList»

BuildingRoomFunctionValue

Definition: BuildingRoomFunctionValue is a code list that enumerates the different purposes of a BuildingRoom.

Stereotype: «CodeList»

BuildingRoomUsageValue

Definition: BuildingRoomUsageValue is a code list that enumerates the different uses of a BuildingRoom.

Stereotype: «CodeList»

BuildingSubdivisionClassValue

Definition: BuildingSubdivisionClassValue is a code list used to further classify a BuildingSubdivision.

Stereotype: «CodeList»

BuildingSubdivisionFunctionValue

Definition: BuildingSubdivisionFunctionValue is a code list that enumerates the different purposes of a BuildingSubdivision.

Stereotype: «CodeList»

BuildingSubdivisionUsageValue

Definition:	BuildingSubdivisionUsageValue is a code list that enumerates the different uses of a BuildingSubdivision.
Stereotype:	«CodeList»

BuildingUsageValue

Definition:	BuildingUsageValue is a code list that enumerates the different uses of a Building.
Stereotype:	«CodeList»

RoofTypeValue

Definition:	RoofTypeValue is a code list that enumerates different roof types.
Stereotype:	«CodeList»

RoomElevationReferenceValue

Definition:	RoomElevationReferenceValue is a code list that enumerates the different elevation reference levels used to measure room heights.
Stereotype:	«CodeList»

9.17.6. Enumerations

none

9.18. Tunnel

Description:	The Tunnel module supports representation of thematic and spatial aspects of tunnels, tunnel parts, tunnel installations, and interior tunnel structures.
Parent Package:	CityGML
Stereotype:	«ApplicationSchema»

9.18.1. Classes

AbstractTunnel

Definition:	AbstractTunnel is an abstract superclass representing the common attributes and associations of the classes Tunnel and TunnelPart.	
Subclass of:	AbstractConstruction	
Stereotype:	«FeatureType»	
<hr/>		
Role name	Target class and multiplicity	Definition
hollowSpace	HollowSpace [*]	Relates the hollow spaces to the Tunnel or TunnelPart.
tunnelConstructiveElement	TunnelConstructiveElement Element [*]	Relates the constructive elements to the Tunnel or TunnelPart.
tunnelInstallation	TunnelInstallation [*]	Relates the installation objects to the Tunnel or TunnelPart.
tunnelFurniture	TunnelFurniture [*]	Relates the furniture objects to the Tunnel or TunnelPart.
<hr/>		
Attribute	Value type and multiplicity	Definition
class	TunnelClassValue [0..1]	Indicates the specific type of the Tunnel or TunnelPart.
function	TunnelFunctionValue [0..*]	Specifies the intended purposes of the Tunnel or TunnelPart.
usage	TunnelUsageValue [0..*]	Specifies the actual uses of the Tunnel or TunnelPart.
adeOfAbstractTunnel	ADEOfAbstractTunnel [0..*]	Augments AbstractTunnel with properties defined in an ADE.
<hr/>		
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».		

HollowSpace

Definition:	A HollowSpace is a space within a Tunnel or TunnelPart intended for certain functions (e.g. transport or passage ways, service rooms, emergency shelters). A HollowSpace is bounded physically and/or virtually (e.g. by ClosureSurfaces or GenericSurfaces).
Subclass of:	AbstractUnoccupiedSpace
Stereotype:	«FeatureType»

Role name	Target class and multiplicity	Definition
tunnelInstallation	TunnelInstallation [*]	Relates the installation objects to the HollowSpace.
tunnelFurniture	TunnelFurniture [*]	Relates the furniture objects to the HollowSpace.
boundary	AbstractThematicSurface [*]	Relates to the surfaces that bound the HollowSpace. This relation is inherited from the Core module.
Attribute	Value type and multiplicity	Definition
class	HollowSpaceClassValue [0..1]	Indicates the specific type of the HollowSpace.
function	HollowSpaceFunctionValue [0..*]	Specifies the intended purposes of the HollowSpace.
usage	HollowSpaceUsageValue [0..*]	Specifies the actual uses of the HollowSpace.
adeOfHollowSpace	ADEOfHollowSpace [0..*]	Augments the HollowSpace with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Tunnel

Definition:	A Tunnel represents a horizontal or sloping enclosed passage way of a certain length, mainly underground or underwater. [cf. ISO 6707-1]	
Subclass of:	AbstractTunnel	
Stereotype:	«TopLevelFeatureType»	
Role name	Target class and multiplicity	Definition
tunnelPart	TunnelPart [*]	Relates the tunnel parts to the Tunnel.
Attribute	Value type and multiplicity	Definition
adeOfTunnel	ADEOfTunnel [0..*]	Augments the Tunnel with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TunnelConstructiveElement

Definition: A TunnelConstructiveElement is an element of a Tunnel which is essential from a structural point of view. Examples are walls, slabs, beams.

Subclass of: [AbstractConstructiveElement](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
class	TunnelConstructive ElementClassValue [0..1]	Indicates the specific type of the TunnelConstructiveElement.
function	TunnelConstructive ElementFunctionV alue [0..*]	Specifies the intended purposes of the TunnelConstructiveElement.
usage	TunnelConstructive ElementUsageValue [0..*]	Specifies the actual uses of the TunnelConstructiveElement.
adeOfTunnelConstructiveElement	ADEOfTunnelConst ructiveElement [0..*]	Augments the TunnelConstructiveElement with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TunnelFurniture

Definition: A TunnelFurniture is an equipment for occupant use, usually not fixed to the tunnel. [cf. ISO 6707-1]

Subclass of: [AbstractFurniture](#)

Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
class	TunnelFurnitureCl assValue [0..1]	Indicates the specific type of the TunnelFurniture.
function	TunnelFurnitureFu nctionValue [0..*]	Specifies the intended purposes of the TunnelFurniture.
usage	TunnelFurnitureUs ageValue [0..*]	Specifies the actual uses of the TunnelFurniture.
adeOfTunnelF urniture	ADEOfTunnelFurni ture [0..*]	Augments the TunnelFurniture with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TunnelInstallation

- Definition: A TunnelInstallation is a permanent part of a Tunnel (inside and/or outside) which does not have the significance of a TunnelPart. In contrast to TunnelConstructiveElement, a TunnelInstallation is not essential from a structural point of view. Examples are stairs, antennas or railings.
- Subclass of: [AbstractInstallation](#)
- Stereotype: «FeatureType»

Attribute	Value type and multiplicity	Definition
class	TunnelInstallation ClassValue [0..1]	Indicates the specific type of the TunnelInstallation.
function	TunnelInstallation FunctionValue [0..*]	Specifies the intended purposes of the TunnelInstallation.
usage	TunnelInstallation UsageValue [0..*]	Specifies the actual uses of the TunnelInstallation.
adeOfTunnelI nstallation	ADEOfTunnelInstal lation [0..*]	Augments the TunnelInstallation with properties defined in an ADE.

Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».

TunnelPart

Definition:	A TunnelPart is a physical or functional subdivision of a Tunnel. It would be considered a Tunnel, if it were not part of a collection of other TunnelParts.							
Subclass of:	AbstractTunnel							
Stereotype:	«FeatureType»							
<table border="1"> <thead> <tr> <th>Attribute</th><th>Value type and multiplicity</th><th>Definition</th></tr> </thead> <tbody> <tr> <td>adeOfTunnelPart</td><td>ADEOfTunnelPart [0..*]</td><td>Augments the TunnelPart with properties defined in an ADE.</td></tr> </tbody> </table>			Attribute	Value type and multiplicity	Definition	adeOfTunnelPart	ADEOfTunnelPart [0..*]	Augments the TunnelPart with properties defined in an ADE.
Attribute	Value type and multiplicity	Definition						
adeOfTunnelPart	ADEOfTunnelPart [0..*]	Augments the TunnelPart with properties defined in an ADE.						
Note: Unless otherwise specified, all attributes and role names have the stereotype «Property».								

9.18.2. Data Types

ADEOfAbstractTunnel

Definition:	ADEOfAbstractTunnel acts as a hook to define properties within an ADE that are to be added to AbstractTunnel.
Subclass of:	None
Stereotype:	«DataType»

ADEOfHollowSpace

Definition:	ADEOfHollowSpace acts as a hook to define properties within an ADE that are to be added to a HollowSpace.
Subclass of:	None
Stereotype:	«DataType»

ADEOfTunnel

Definition:	ADEOfTunnel acts as a hook to define properties within an ADE that are to be added to a Tunnel.
Subclass of:	None
Stereotype:	«DataType»

ADEOfTunnelConstructiveElement

Definition: ADEOfTunnelConstructiveElement acts as a hook to define properties within an ADE that are to be added to a TunnelConstructiveElement.

Subclass of: None

Stereotype: «DataType»

ADEOfTunnelFurniture

Definition: ADEOfTunnelFurniture acts as a hook to define properties within an ADE that are to be added to a TunnelFurniture.

Subclass of: None

Stereotype: «DataType»

ADEOfTunnelInstallation

Definition: ADEOfTunnelInstallation acts as a hook to define properties within an ADE that are to be added to a TunnelInstallation.

Subclass of: None

Stereotype: «DataType»

ADEOfTunnelPart

Definition: ADEOfTunnelPart acts as a hook to define properties within an ADE that are to be added to a TunnelPart.

Subclass of: None

Stereotype: «DataType»

9.18.3. Basic Types

none

9.18.4. Unions

none

9.18.5. Code Lists

HollowSpaceClassValue

Definition:	HollowSpaceClassValue is a code list used to further classify a HollowSpace.
Stereotype:	«CodeList»

HollowSpaceFunctionValue

Definition:	HollowSpaceFunctionValue is a code list that enumerates the different purposes of a HollowSpace.
Stereotype:	«CodeList»

HollowSpaceUsageValue

Definition:	HollowSpaceUsageValue is a code list that enumerates the different uses of a HollowSpace.
Stereotype:	«CodeList»

TunnelClassValue

Definition:	TunnelClassValue is a code list used to further classify a Tunnel.
Stereotype:	«CodeList»

TunnelConstructiveElementClassValue

Definition:	TunnelConstructiveElementClassValue is a code list used to further classify a TunnelConstructiveElement.
Stereotype:	«CodeList»

TunnelConstructiveElementFunctionValue

Definition:	TunnelConstructiveElementFunctionValue is a code list that enumerates the different purposes of a TunnelConstructiveElement.
Stereotype:	«CodeList»

TunnelConstructiveElementUsageValue

Definition: TunnelConstructiveElementUsageValue is a code list that enumerates the different uses of a TunnelConstructiveElement.

Stereotype: «CodeList»

TunnelFunctionValue

Definition: TunnelFunctionValue is a code list that enumerates the different purposes of a Tunnel.

Stereotype: «CodeList»

TunnelFurnitureClassValue

Definition: TunnelFurnitureClassValue is a code list used to further classify a TunnelFurniture.

Stereotype: «CodeList»

TunnelFurnitureFunctionValue

Definition: TunnelFurnitureFunctionValue is a code list that enumerates the different purposes of a TunnelFurniture.

Stereotype: «CodeList»

TunnelFurnitureUsageValue

Definition: TunnelFurnitureUsageValue is a code list that enumerates the different uses of a TunnelFurniture.

Stereotype: «CodeList»

TunnelInstallationClassValue

Definition: TunnelInstallationClassValue is a code list used to further classify a TunnelInstallation.

Stereotype: «CodeList»

TunnelInstallationFunctionValue

Definition: TunnelInstallationFunctionValue is a code list that enumerates the different purposes of a TunnelInstallation.

Stereotype: «CodeList»

TunnelInstallationUsageValue

Definition: TunnelInstallationUsageValue is a code list that enumerates the different uses of a TunnelInstallation.

Stereotype: «CodeList»

TunnelUsageValue

Definition: TunnelUsageValue is a code list that enumerates the different uses of a Tunnel.

Stereotype: «CodeList»

9.18.6. Enumerations

none

Annex A: Revision History

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

Annex B: Glossary

conformance test class

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

feature

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

feature attribute

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

feature type

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

measurement

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

model

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

observation

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

observation procedure

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

observation result

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

property

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

requirements class

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

schema

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

sensor

type of observation procedure that provides the estimated value of an observed property at its output

[OGC 08-094r1, definition 4.5]

Standardization Target

TBD

timeseries

sequence of data values which are ordered in time

[OGC 15-043r3]

universe of discourse

view of the real or hypothetical world that includes everything of interest

[ISO 19101-1:2014, definition 4.1.38]

version

Particular variation of a spatial object

[INSPIRE Glossary]

B.1. ISO Concepts

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

Area

The measure of the physical extent of any topologically 2-D geometric object. Usually measured in "square" units of length.

[\[ISO 19103:2015\]](#)

Boolean

boolean is the mathematical datatype associated with two-valued logic

[\[ISO 19103:2015\]](#)

CC_CoordinateOperation

mathematical operation on coordinates that transforms or converts coordinates to another coordinate reference system.

[\[ISO 19111:2019\]](#)

Character

symbol from a standard character-set.

[\[ISO 19103:2015\]](#)

CharacterString

Characterstring is a family of datatypes which represent strings of symbols from standard character-sets.

[\[ISO 19103:2015\]](#)

CRS

Coordinate reference system which is usually single but may be compound.

[ISO 19111:2019]

CV_DiscreteCoverage

A subclass of CV_Coverage that returns a single record of values for any direct position within a single geometric object in its spatiotemporal domain.

[ISO 19123:2005]

CV_DomainObject

[ISO 19123:2005]

CV_GridPointValuePair

[ISO 19123:2005]

CV_GridValuesMatrix

The geometry represented by the various offset vectors is in the image plane of the grid.

[ISO 19123:2005]

CV_ReferenceableGrid

[ISO 19123:2005]

Date

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

[ISO 19103:2015]

DateTime

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

[ISO 19103:2015]

Distance

Used as a type for returning distances and possibly lengths.

[ISO 19103:2015]

Engineering CRS

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

1. earth-fixed systems applied to engineering activities on or near the surface of the earth;
2. CRSSs on moving platforms such as road vehicles, vessels, aircraft or spacecraft.

[ISO 19111:2019]

Generic Name

Generic Name is the abstract class for all names in a NameSpace. Each instance of a GenericName is either a LocalName or a ScopedName.

[ISO 19103:2015]

Geometry

[ISO 19107:2003]

GM_CompositePoint

[ISO 19107:2003]

GM_CompositeSolid

set of geometric solids adjoining one another along common boundary geometric surfaces

[ISO 19107:2003]

GM_GenericSurface

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

[ISO 19107:2003]

GM_LineString

consists of sequence of line segments, each having a parameterization like the one for GM_LineSegment

[ISO 19107:2003]

GM_MultiPrimitive

[ISO 19107:2003]

GM_OrientableSurface

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

[ISO 19107:2003]

GM_PolyhedralSurface

a GM_Surface composed of polygon surfaces (GM_Polygon) connected along their common boundary curves.

[ISO 19107:2003]

GM_Position

a union type consisting of either a DirectPosition or of a reference to a GM_Point from which a DirectPosition shall be obtained.

[ISO 19107:2003]

GM_Primitive

The abstract root class of the geometric primitives. Its main purpose is to define the basic "boundary" operation that ties the primitives in each dimension together.

[ISO 19107:2003]

Integer

An exact integer value, with no fractional part.

[ISO 19103:2015]

Internet of Things

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

IO_IdentifiedObjectBase

[\[ISO 19103:2015\]](#)

Length

The measure of distance as an integral, i.e. the limit of an infinite sum of distances between points on a curve.

[\[ISO 19103:2015\]](#)

Measure

The result from performing the act or process of ascertaining the extent, dimensions, or quantity of some entity.

[\[ISO 19103:2015\]](#)

Number

The base type for all number data, giving the basic algebraic operations.

[\[ISO 19103:2015\]](#)

Point

GM_Point is the basic data type for a geometric object consisting of one and only one point.

[\[ISO 19107:2003\]](#)

Real

The common binary Real finite implementation using base 2.

[\[ISO 19103:2015\]](#)

RS_ReferenceSystem

Description of a spatial and temporal reference system used by a dataset.

[\[ISO 19111:2019\]](#)

Scoped Name

ScopedName is a composite of a LocalName for locating another NameSpace and a GenericName valid in that NameSpace. ScopedName contains a LocalName as head and a GenericName, which might be a LocalName or a ScopedName, as tail.

[\[ISO 19103:2015\]](#)

Solid

GM_Solid, a subclass of GM_Primitive, is the basis for 3-dimensional geometry. The extent of a solid is defined by the boundary surfaces.

[\[ISO 19107:2003\]](#)

Time

Time is the designation of an instant on a selected time scale, astronomical or atomic. It is used in the sense of time of day.

[\[ISO 19103:2015\]](#)

TM_Duration

[\[ISO 19108:2006\]](#)

TM_TemporalPosition

The position of a TM_Instant relative to a TM_ReferenceSystem.

[ISO 19108:2006]

Unit of Measure

Any of the systems devised to measure some physical quantity such distance or area or a system devised to measure such things as the passage of time.

[ISO 19103:2015]

URI

Uniform Resource Identifier (URI), is a compact string of characters used to identify or name a resource

[ISO 19103:2015]

Volume

Volume is the measure of the physical space of any 3-D geometric object.

[ISO 19103:2015]

B.2. Abbreviated Terms

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

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

Annex C: Bibliography

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

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

- Springer New York, 2018. ISBN 978-1-4614-8266-6. Available from: 10.1007/978-1-4614-8265-9
- Löwner, M.-O., Gröger, G., Benner, J., Biljecki, F., Nagel, C., 2016: **Proposal for a new LOD and multi-representation concept for CityGML**. In: Proceedings of the 11th 3D Geoinfo Conference 2016, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. IV-2/W1, 3–12. <https://doi.org/10.5194/isprs-annals-IV-2-W1-3-2016>
 - Nouvel, R., Bahu, J. M., Kaden, R., Kaempf, J., Cipriano, P., Lauster, M., Haefele, K.-H., Munoz, E., Tournaire, O., Casper, E., 2015: **Development of the CityGML Application Domain Extension Energy for Urban Energy Simulation**. In: Proceedings of Building Simulation 2015 - 14th Conference of the International Building Performance Simulation Association, IBPSA, 559-564. <http://www.ibpsa.org/proceedings/BS2015/p2863.pdf>
 - Smith, B., Varzi, A. C., 2000: **Fiat and Bona Fide Boundaries**. Philosophy and Phenomenological Research, Vol. 60, No. 2, 401-420. <https://doi.org/10.2307/2653492>
 - Snodgrass, Richard T: **Developing time-oriented database applications in SQL**. San Francisco, California : Morgan Kaufmann Publishers, July 1999. ISBN 1-55860-436-7. Available from: <http://www.cs.arizona.edu/rts/tdbbook.pdf>
 - Stadler, A., Kolbe, T. H., 2007: **Spatio-semantic Coherence in the Integration of 3D City Models**. In: Proceedings of the 5th International ISPRS Symposium on Spatial Data Quality ISSDQ 2007 in Enschede. http://www.isprs.org/proceedings/XXXVI/2-C43/Session1/paper_Stadler.pdf
 - Vretanos, P. A. 2010: **OpenGIS Web Feature Service 2.0 Interface Standard**, Open Geospatial Consortium. [OGC Doc. 09-025r1](#)
 - OASIS MQTT Technical Committee: **MQTT Version 5.0 Standard**, OASIS, March 7, 2019, Available from [OASIS](#).
 - Reed, C., Belayneh T.: **OGC Indexed 3d Scene Layer (I3S) and Scene Layer Package Format Specification**, Open Geospatial Consortium, Available from [OGC Doc. 17-014r7](#)
 - [[3dtiles_citation, OGC 3D Tiles]]Cozzi, P., Lilley, S., Getz, G. **OGC 3D Tiles Specification 1.0** Open Geospatial Consortium, Available from [OGC Doc. 18-053r2](#)
 - Burggraf, D.: **OGC KML 2.3**, Open Geospatial Consortium, Available from [OGC Doc. 12-007r2](#)
 - Bröring, A., Stasch, C., Echterhoff, J.: **OGC® Sensor Observation Service Interface Standard**, Open Geospatial Consortium, Available from [OGC Doc. 12-006](#)
 - Liang, S., Huang, C., Khalafbeigi, T.: **OGC SensorThings API Part 1: Sensing**, Open Geospatial Consortium, Available from [OGC Doc. 15-078r6](#)
 - [[3dps_citation, OGC 3D Portrayal Service]]Hagedorn, B., Thum, S., Reitz, T., Coors, V., Gutbell, R.: **OGC® 3D Portrayal Service 1.0**, Open Geospatial Consortium, Available from [OGC Doc. 15-001r3](#).
 - Bhatia, S., Cozzi, P., Knyazev, A., Parisi, T.: **The GL Transmission Format (glTF)**, The Khronos Group, Available from <https://www.khronos.org/gltf>.