Open Geospatial Consortium

Submission Date: <yyyy-mm-dd>

Approval Date: <yyyy-mm-dd>

Publication Date: <2019-11-08>

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

Internal reference number of this OGC® document: YY-nnnrx

Version: 0.2

Category: OGC® Best Practice

Editor: Charles Heazel

OGC City Geography Markup Language (CityGML) Conceptual Model Best Practice

Warning

This document defines an OGC Best Practice on a particular technology or approach related to an OGC standard. This document is not an OGC Standard and may not be referred to as an OGC Standard. It is subject to change without notice. However, this document is an official position of the OGC membership on this particular technology topic.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type:	OGC® Best Practice
Document subtype:	Conceptual Model
Document stage:	Draft
Document language:	English

License Agreement

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

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

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

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

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

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

Table of Contents

i. Abstract

CityGML is an open data model and XML-based format for the storage and exchange of virtual 3D city models. It is an application schema for the Geography Markup Language version 3.2.1 (GML3), the extendible international standard for spatial data exchange issued by the Open Geospatial Consortium (OGC) and the ISO TC211.

The aim of the development of CityGML is to reach a common definition of the basic entities, attributes, and relations of a 3D city model. This is especially important with respect to the cost-effective sustainable 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, <tags separated by commas>

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.

There are significant changes between CityGML version 2.0.0 and CityGML version 1.0.0 (OGC document no. 08-007r1):

- New thematic modules for the representation of tunnels and bridges;
- Additional boundary surfaces for the semantic classification of the outer shell of buildings and building parts (OuterCeilingSurface and OuterFloorSurface);
- LOD0 representation (footprint and roof egde representations) for buildings and building parts;
- Additional attributes denoting a city object’s location with respect to the surrounding terrain and water surface (relativeToTerrain and relativeToWater);
- Additional generic attributes for measured values and attribute sets; and
- Redesign of the CityGML code list mechanism (enumerative attributes are now of type gml:CodeType which facilitates to provide additional code lists enumerating their possible attribute values).

Migration of existing CityGML 1.0 instances to valid 2.0 instances only requires changing the CityGML namespace and schema location values in the document to the actual 2.0 values.

iv. Submitting organizations

This is the official CityGML logo. For current news on CityGML and information about ongoing projects and fields of research in the area of CityGML see http://www.citygml.org and http://www.citygmlwiki.org

OGC work on CityGML is discussed and coordinated by the OGC 3D Information Management (3DIM) Working Group.CityGMLwas initally implemented and evaluated as part of the OGC Web Services Testbed, Phase 4 (OWS-4) in the CAD/GIS/BIM thread.

Version 2.0 of this standards document was prepared by the OGC CityGML Standards Working Group (SWG). Future discussion and development will be led by the 3DIM Working Group.

For further information see http://www.opengeospatial.org/projects/groups/3dimwg

CityGML also continues to be developed by the members of the Special Interest Group 3D (SIG 3D) of the GDI-DE Geodateninfrastruktur Deutschland (Spatial Data Infrastructure Germany) in joint cooperation with the 3DIM Working Group and the CityGML SWG within OGC.

For further information see http://www.sig3d.org/

The preparation of the English document version and the European discussion has been supported by the European Spatial Data Research Organization (EuroSDR; formerly known as OEEPE) in an EuroSDR Commission III project. For further information see http://www.eurosdr.net

This Document was submitted to the Open Geospatial Consortium (OGC) by the members of the CityGML 3.0 Standards Working Group of the OGC. Amongst others, this comprises the following organizations:

- Autodesk, Inc. (primary submitter)
- Bentley Systems, Inc. (primary submitter)
- Technical University Berlin (submitter of technology)
- Ordnance Survey, UK
- University of Bonn, Germany
- Hasso-Plattner-Institute for IT Systems Engineering, University of Potsdam

- Institute for Applied Computer Science, Karlsruhe Institute of Technology

CityGML was originally developed by the Special Interest Group 3D (SIG 3D), 2002 – 2012 - www.citygml.org.

v. Submitters

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

Table 1. Submission Contact Points

Name	Institution	Email
Prof. Dr. Thomas H. Kolbe, Claus Nagel, Alexandra Lorenz	Institute for Geodesy and Geoinformation Science, Technical University Berlin	thomas.kolbe@tu-berlin.de claus.nagel@tu-berlin.de alexandra.lorenz@tu-berlin.de
Dr. Gerhard Gröger, Prof. Dr. Lutz Plümer, Angela Czerwinski	Institute for Geodesy and Geoinformation, University of Bonn	Groeger@ikg.uni-bonn.de Pluemer@ikg.uni-bonn.de Czerwinski@ikg.uni-bonn.de
Haik Lorenz	Autodesk, Inc.	haik.lorenz@autodesk.com
Alain Lapierre, Stefan Apfel, Paul Scarponcini	Bentley Systems, Inc.	alain.lapierre@bentley.com stefan.apfel@bentley.com paul.scarponcini@bentley.com
Carsten Rönsdorf	Ordnance Survey, Great Britain	carsten.roensdorf@ordnancesurvey.co.uk
Prof. Dr. Jürgen Döllner	Hasso-Plattner-Institute for IT Systems Engineering, University of Potsdam	juergen.doellner@hpi.uni-potsdam.de
Dr. Joachim Benner, Karl-Heinz Häfele	Institute for Applied Computer Science, Karlsruhe Institute of Technology	joachim.benner@kit.edu karl-heinz.haefele@kit.edu

vi. Participants in development

Table 2. Participants in Development

Name	Institution
Ulrich Gruber, Sandra Schlüter	District Administration Recklinghausen, Cadastre Department, Germany
Frank Bildstein	Rheinmetall Defence Electronics, Germany
Rüdiger Drees	T-Systems Enterprise Services GmbH, Bonn, Germany
Andreas Kohlhaas	GIStec GmbH (formerly), Germany

Name	Institution
Frank Thiemann	Institute for Cartography and Geoinformatics, University of Hannover
Martin Degen	Cadastre Department, City of Dortmund
Heinrich Geerling	Architekturbüro Geerling, Germany
Dr. Frank Knospe	Cadastre and Mapping Department, City of Essen,
Hardo Müller	Snowflake Software Ltd., Great Britain
Martin Rechner	rechner logistic, Germany
Jörg Haist, Daniel Holweg	Fraunhofer Institute for Computer Graphics (IGD), Darmstadt, Germany
Prof. Dr. Peter A. Henning	Faculty for Computer Science, University of Applied Sciences, Karlsruhe, Germany
Rolf Wegener, Stephan Heitmann	State Cadastre and Mapping Agency of North-Rhine Westphalia, Germany
Prof. Dr. Marc-O. Löwner	Institute for Geodesy and Photogrammetry, Technical University of Braunschweig
Dr. Egbert Casper	Zerna Ingenieure, Germany
Christian Dahmen	con terra GmbH, Germany
Nobuhiro Ishimaru, Kishiko Maruyama, Eiichiro Umino, Takahiro Hirose	Hitachi, Ltd., Japan
Linda van den Brink	Geonovum, The Netherlands
Ron Lake, David Burggraf	Galdos Systems Inc., Canada
Marie-Lise Vautier, Emmanuel Devys	Institut géographique national, France
Mark Pendlington	Ordnance Survey, Great Britain

vii. Acknowledgements

The SIG 3D wishes to thank the members of the CityGML Standards Working Group and the 3D Information Management (3DIM) Working Group of the OGC as well as all contributors of change requests and comments. In particular: Tim Case, Scott Simmons, Paul Cote, Clemens Portele, Jeffrey Bell, Chris Body, Greg Buehler, François Golay, John Herring, Jury Konga, Kai-Uwe Krause, Gavin Park, Richard Pearsall, George Percivall, Mauro Salvemini, Alessandro Triglia, David Wesloh, Tim Wilson, Greg Yetman, Jim Farley, Cliff Behrens, Lukas Herman, Danny Kita, and Simon Cox.

Further credits for careful reviewing and commenting of this document go to: Ludvig Emgard, Bettina Petzold, Dave Capstick, Mark Pendlington, Alain Lapierre, and Frank Steggink.

Chapter 1. Introduction

1.1. Motivation

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 pedes-trian navigation, facility management and environmental simulations. Furthermore, in the implementation of the European Environmental Noise Directive (END, 2002/49/EC) 3D geoinformation and 3D city models play an important role.

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

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

CityGML is designed as an open data model and XML-based format for the storage and exchange of virtual 3D city models. It is implemented as an application schema of the Geography Markup Language 3 (GML3), the extendible international standard for spatial data exchange and encoding issued by the Open Geospatial Consor-tium (OGC) and the ISO TC211. CityGML is based on a number of standards from the ISO 191xx family, the Open Geospatial Consortium, the W3C Consortium, the Web 3D Consortium, and OASIS.

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

1.2. Historical background

CityGML has been developed since 2002 by the members of the Special Interest Group 3D (SIG 3D). Since 2010, this group is part of the initiative Spatial Data Infrastructure Germany (GDI-DE). Before

2010, the SIG 3D was affiliated to the initiative Geodata Infrastructure North Rhine-Westphalia (GDI NRW). The SIG 3D is an open group consisting of more than 70 companies, municipalities, and research institutions from Germany, Great Britain, Switzerland, and Austria working on the development and commercial exploitation of interoperable 3D city models and geovisualisation. Another result of the work from the SIG 3D is the proposition of the Web 3D Service (W3DS), a 3D portrayal service that is also being discussed in the Open Geospatial Consortium (OGC Doc. No. 05-019 and OGC Doc. No. 09-104r1).

A first successful implementation and evaluation of a subset of CityGML has been performed in the project “Pilot 3D” of the GDI NRW in 2005. Participants came from all over Germany and demonstrated city planning scenarios and tourist applications. By the beginning of 2006, a CityGML project within EuroSDR (European Spatial Data Research) started focusing on the European harmonisation of 3D city modelling. From June to December 2006, CityGML was employed and evaluated in the CAD/GIS/BIM thread of the OpenGIS Web Services Testbed #4 (OWS-4). Since 2008, CityGML (version 1.0.0) is an adopted OGC standard.

From that point in time, CityGML has disseminated worldwide. Many cities in Germany and in other countries in Europe provide their 3D city model in CityGML (Berlin, Cologne, Dresden and Munich, to mention only a few). In France, the project Bâti3D (IGN France) defines a profile of CityGML LOD2 and provides data from Paris and the city centres of Aix-en-Provence, Lille, Nantes and Marseille. CityGML also plays an important role in the pilot 3D project to obtain a 3D geoinformation standard and a 3D infrastructure for The Netherlands. Many cities in Europe like Monaco, Geneva, Zurich, Leewarden use CityGML LOD 2 or 3 to represent and exchange data, as well as cities in Denmark (LOD 2 and 3, partly LOD4). CityGML has strongly influenced the building model (version 2.0) of the INSPIRE initiative of the EU commission, which aims at the creation of an European spatial data infrastructure providing public sector data in an interoperable way. In Asia, the 3D city models of Istanbul (LOD 1 and 2), Doha, Katar (LOD3), and Yokohama (LOD2) are represented and exchanged in CityGML. Moreover, CityGML plays a crucial role for the 3D Spatial data infrastructure in Malaysia.

Today many commercial and academic tools support CityGML by providing import interfaces, export interfaces or both. An example is the 3D City Database which is a free and open source 3D geo database to store, represent, and manage virtual 3D city models on top of Oracle 10g R2 and 11g R1/R2 provided by the Technische Univer-sität Berlin. It fully supports CityGML and is shipped with a tool for the import and export of CityGML models. Furthermore, an open source Java class library and API for the processing of CityGML models (citygml4j) is provided by the Technische Universität Berlin. The conversion tool FME (Feature Manipulation Engine) from Safe Software Inc., which is part of the interoperability extension of ESRI’s ArcGIS, has read and write interfac-es for CityGML. The same applies to CAD tools as BentleyMap from Bentley Systems as well as to GIS tools like SupportGIS from CPA Geo-Information. Many 3D viewers (which all are freely available) provide read interfaces for CityGML: the Aristoteles Viewer from the University of Bonn, LandXplorer CityGML Viewer from Autodesk Inc. (the studio version for authoring and management is not free) and the FZKViewer for IFC and CityGML from KIT Karlsruhe and BS Contact from Bitmanagement Software GmbH which offers a CityGML plugin for the geospatial extension BS Contact Geo. This enumeration of software tools is not exhaus-tive and steadily growing. Please refer to the official website of CityGML at http://www.citygml.org as well as the CityGML Wiki at http://www.citygmlwiki.org for more information.

1.3. Additions in CityGML 2.0

CityGML 2.0 is a major revision of the previous version 1.0 of this International Standard (OGC Doc. No. 08-007r1), and introduces substantial additions and new features to the thematic model of CityGML. The revision was originally planned to be a minor update to version 1.1. The main endeavor of the revision process was to ensure backwards compatibility both on the level of the conceptual model and on the level of CityGML instance documents. However, some changes could not be implemented consistent with directives for minor revisions and backwards compatibility as enforced by OGC policy (cf. OGC Doc. No. 135r11). The major version number change to 2.0 is therefore a consequence of conforming to the OGC versioning policy without having to abandon any changes or additions which reflect requests from the CityGML community.

CityGML 2.0 is backwards compatible with version 1.0 in the following sense: each valid 1.0 instance is a valid 2.0 instance provided that the CityGML namespaces and schema locations in the document are changed to their actual 2.0 values. This step is required because the CityGML version number is encoded in these values, but no further actions have to be taken. Hence, there is a simple migration path from existing CityGML 1.0 instances to valid 2.0 instances.

The following clauses provide an overview of what is new in CityGML 2.0.

New thematic modules for the representation of bridges and tunnels

Bridges and tunnels are important objects in city and landscape models. They are an essential part of the trans-portation infrastructure and are often easily recognizable landmarks of a city. CityGML 1.0 has been lacking thematic modules dedicated to bridges and tunnels, and thus such objects had to be modelled and exchanged using a GenericCityObject as proxy (cf. chapter 10.12). CityGML 2.0 now introduces two new thematic modules for the explicit representation of bridges and tunnels which complement the thematic model of CityGML: the Bridge module (cf. chapter 10.4) and the Tunnel module (cf. chapter 10.5).

Bridges and tunnels can be represented in LOD 1 – 4 and the underlying data models have a coherent structure with the Building model. For example, bridges and tunnels can be decomposed into parts, thematic boundary surfaces with openings are available to semantically classify parts of the shell, and installations as well as interi-or built structures can be represented. This coherent model structure facilitates the similar understanding of semantic entities and helps to reduce software implementation efforts. Both the Bridge and the Tunnel model introduce further concepts and model elements which are specific to bridges and tunnels respectively.

Additions to existing thematic modules

- *CityGML Core module (cf. chapter 10.1)* Two new optional attributes have been added to the abstract base class *core:_CityObject* within the *CityGML Core* module: *relativeToTerrain* and *relativeToWater*. These attributes denote the feature's location with respect to the terrain and water surface in a qualitative way, and thus facilitate simple and efficient queries (e.g., for the number of subsurface buildings) without the need for an additional digital terrain model or a model of the water body.
- *Building module (cf. chapter 10.3)*
    - *LOD0 representation :* Buildings can now be represented in LOD0 by footprint and/or roof

edge polygons. This al-lows the easy integration of existing 2D data and of roof reconstructions from aerial and satel-lite imagery into a 3D city model. The representations are restricted to horizontal, 3-dimensional surfaces.

- *Additional thematic boundary surfaces :* In order to semantically classify parts of the outer building shell which are neither horizontal wall surfaces nor parts of the roof, two additional boundary surfaces are introduced: *OuterFloorSurface* and *OuterCeilingSurface*.
- *Additional relations to thematic boundary surfaces :* In addition to _ *AbstractBuilding* and *Room*, the surface geometries of *BuildingInstallation* and *IntBuildingInstallation* features can now be semantically classified using thematic boundary surfaces. For example, this facilitates the semantic differentiation between roof and wall sur-faces of dormers which are modeled as *BuildingInstallation*.
- *Additional use of implicit geometries :* Implicit geometries (cf. chapter 8.3) are now available for the representation of _ *Opening*, *BuildingInstallation*, and *IntBuildingInstallation* in addition to *BuildingFurniture*. A prototyp-ical geometry for these city objects can thus be stored once and instantiated at different loca-tions in the 3D city model.

- *Generics module (cf. chapter 10.12)* Two generic attributes have been added to the *Generics* module: *MeasureAttribute* and *GenericAttributeSet*. A *MeasureAttribute* facilitates the representation of measured values together with a reference to the employed unit. A *GenericAttributeSet* is a named collection of arbitrary generic attributes. It provides an optional *codeSpace* attribute to denote the authority organization who defined the attribute set.
- *LandUse module (cf. chapter 10.10)* The scope of the feature type *LandUse* has been broadened to comprise both areas of the earth's surface dedicated to a specific land use and areas of the earth's surface having a specific land cover with or without vegetation.
- *Attributes* class, function, *and* usage *(all modules)* In order to harmonize the use of the attributes *class*, *function*, and *usage*, this attribute triplet has been complemented for all feature classes that at least provided one of the attributes in CityGML 1.0.

Additions to the CityGML code list mechanism

In CityGML, code lists providing the allowed values for enumerative attributes such as *class*, *function*, and *usage* can be specified outside the CityGML schema by any organization or information community according to their specific information needs. This mechanism is, however, not fully reflected in the CityGML 1.0 encoding schema, because in a CityGML 1.0 instance document a corresponding attribute cannot point to the dictionary with the used code list values. This has been corrected for CityGML 2.0: All attributes taking values from code lists are now of type *gml:CodeType* following the GML 3.1.1 mechanism for the encoding of code list values (cf.chapter 10.14 for more information). The *gml:CodeType* adds an optional *codeSpace* value to enumerative attributes which allows for providing a persistent URI pointing to the corresponding dictionary.

Changelog for CityGML 2.0

Changes on the level of XML schema components are provided in Annex F.

Further edits to the specification document

- *Accuracy requirements for Levels of Detail (LOD) (cf. chapter 6.2)* The accuracy requirements for the different CityGML LODs proposed in chapter 6.2 are non-normative. The wording of chapter

6.2 in CityGML 1.0 is however inconsistent with regard to this fact and thus has been clarified for CityGML 2.0.

- *Rework of the CityGML example datasets (cf. Annex G)* The CityGML examples provided in Annex G have been reworked and extended. They now show a consistent building model in all five LODs and demonstrate, for example, the semantic and geometric refinement of the building throughout the different LODs as well as the usage of XLinks to share geom-etry elements between features. The datasets are shipped with the CityGML XML Schema package, and are available at http://schemas.opengis.net/citygml/examples/2.0/.
- *New example for the usage of Application Domain Extensions (cf. Annex I)* A second example for the usage of Application Domain Extensions in the field of Ubiquitous Network Robots Services has been added in Annex I.

Chapter 2. Scope

This document is an OGC Encoding Standard for the representation, storage and exchange of virtual 3D city and landscape models. CityGML is implemented as an application schema of the Geography Markup Language version 3.1.1 (GML3).

CityGML models both complex and georeferenced 3D vector data along with the semantics associated with the data. In contrast to other 3D vector formats, CityGML is based on a rich, general purpose information model in addition to geometry and appearance information. For specific domain areas, CityGML also provides an exten-sion mechanism to enrich the data with identifiable features under preservation of semantic interoperability.

Targeted application areas explicitly include urban and landscape planning; architectural design; tourist and leisure activities; 3D cadastres; environmental simulations; mobile telecommunications; disaster management; homeland security; vehicle and pedestrian navigation; training simulators and mobile robotics.

CityGML is considered a source format for 3D portraying. The semantic information contained in the model can be used in the styling process which generates computer graphics represented e.g. as KML/COLLADA or X3D files. The appropriate OGC Portrayal Web Service for this process is the OGC Web 3D Service (W3DS). An image-based 3D portrayal service for virtual 3D landscape and city models is provided by the OGC Web View Service (WVS).

Features of CityGML:

- Geospatial information model (ontology) for urban landscapes based on the ISO 191xx family
- GML3 representation of 3D geometries, based on the ISO 19107 model
- Representation of object surface characteristics (e.g. textures, materials)
- Taxonomies and aggregations
    - Digital Terrain Models as a combination of (including nested) triangulated irregular networks (TINs), regular rasters, break and skeleton lines, mass points
    - Sites (currently buildings, bridges, and tunnels)
    - Vegetation (areas, volumes, and solitary objects with vegetation classification)
    - Water bodies (volumes, surfaces)
    - Transportation facilities (both graph structures and 3D surface data)
    - Land use (representation of areas of the earth’s surface dedicated to a specific land use)
    - City furniture
    - Generic city objects and attributes
    - User-definable (recursive) grouping
- Multiscale model with 5 well-defined consecutive Levels of Detail (LOD):
    - LOD0 – regional, landscape
    - LOD1 – city, region

    - LOD2 – city districts, projects
    - LOD3 – architectural models (outside), landmarks
    - LOD4 – architectural models (interior)
- Multiple representations in different LODs simultaneously; generalisation relations between objects in different LODs
- Optional topological connections between feature (sub)geometries
- Application Domain Extensions (ADE): Specific “hooks” in the CityGML schema allow to define ap-plication specific extensions, for example for noise pollution simulation, or to augment CityGML by properties of the new National Building Information Model Standard (NBIMS) in the U.S.

Chapter 3. References

The following normative documents contain provisions that, through reference in this text, constitute provisions of OGC 12-019. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of OGC 12-019 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies.

The following documents are indispensable for the application of the CityGML standard. The geometry model of GML 3.1.1 is used except for some added concepts like implicit geometries (cf. chapter 8.2). The appearance model (cf. chapter 9) draws concepts from both *X3D* and *COLLADA*. Addresses are represented using the OASIS extensible Address Language *xAL*.

- ISO / TC154: ISO 8601:2004, Data elements and interchange formats – Information interchange – Representation of dates and times
- ISO / TC211: ISO/TS 19103:2005, Geographic Information – Conceptual Schema Language
- ISO / TC211: ISO 19105:2000, Geographic information – Conformance and testing
- ISO / TC211: ISO 19107:2003, Geographic Information – Spatial Schema
- ISO / TC211: ISO 19109:2005, Geographic Information – Rules for Application Schemas
- ISO / TC211: ISO 19111:2003, Geographic information – Spatial referencing by coordinates
- ISO / TC211: ISO 19115:2003, Geographic Information – Metadata
- ISO / TC211: ISO 19123:2005, Geographic Information – Coverages
- ISO / TC211: ISO/TS 19139:2007, Geographic Information – Metadata – XML schema implementation
- ISO / TC211: ISO/IEC 19775:2004, X3D Abstract Specification
- OGC: OpenGIS® Abstract Specification Topic 0, Overview, OGC document 04-084
- OGC: OpenGIS® Abstract Specification Topic 5, The OpenGIS Feature, OGC document 99-105r2
- OGC: OpenGIS® Abstract Specification Topic 8, Relations between Features, OGC document 99-108r2
- OGC: OpenGIS® Abstract Specification Topic 10, Feature Collections, OGC document 99-110
- OGC: OpenGIS® Geography Markup Language Implementation Specification, Version 3.1.1, OGC document 03-105r1
- OGC: OpenGIS® GML 3.1.1 Simple Dictionary Profile, Version 1.0.0, OGC document 05-099r2
- IETF: IETF RFC 2045 & 2046, Multipurpose Internet Mail Extensions (MIME). (November 1996)
- IETF: IETF RFC 2396, Uniform Resource Identifiers (URI): Generic Syntax. (August 1998)
- W3C: W3C XLink, XML Linking Language (XLink) Version 1.0. W3C Recommendation (27 June 2001)
- W3C: W3C XMLName, Namespaces in XML. W3C Recommendation (14 January 1999)
- W3C: W3C XMLSchema-1, XML Schema Part 1: Structures. W3C Recommendation (2 May 2001)

- W3C: W3C XMLSchema-2, XML Schema Part 2: Datatypes. W3C Recommendation (2 May 2001)
- W3C: W3C XPointer, XML Pointer Language (XPointer) Version 1.0. W3C Working Draft (16 August 2002)
- W3C: W3C XML Base, XML Base, W3C Recommendation (27 June 2001)
- W3C: W3C XML, Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation (6 October 2000)
- OASIS (Organization for the Advancement of Structured Information Standards): extensible Address Language (xAL v2.0).
- Khronos Group Inc.: COLLADA – Digital Asset Schema Release 1.4.1
- Jelliffe, R: The Schematron Assertion Language 1.5. (2002-10-01)

Chapter 4. Terms and Definitions

This document uses the terms defined in Sub-clause 5.3 of [OGC 06-121r8], which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this Best Practice.

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

4.1. term name

text of the definition

4.2. Abbreviated Terms

The following abbreviated terms are used in this document:

- 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
- VRML Virtual Reality Modeling Language
- W3C World Wide Web Consortium
- W3DS OGC Web 3D Service
- WFS OGC Web Feature Service
- X3D Open Standards XML-enabled 3D file format of the Web 3D Consortium
- XML Extensible Markup Language
- xAL OASIS extensible Address Language

Chapter 5. Conventions

5.1. Identifiers

The normative provisions in this document are denoted by the URI

http://www.opengis.net/spec/CityGML/3.0

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

5.2. UML Notation

The CityGML standard is presented in this document in 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 below Figure 1.

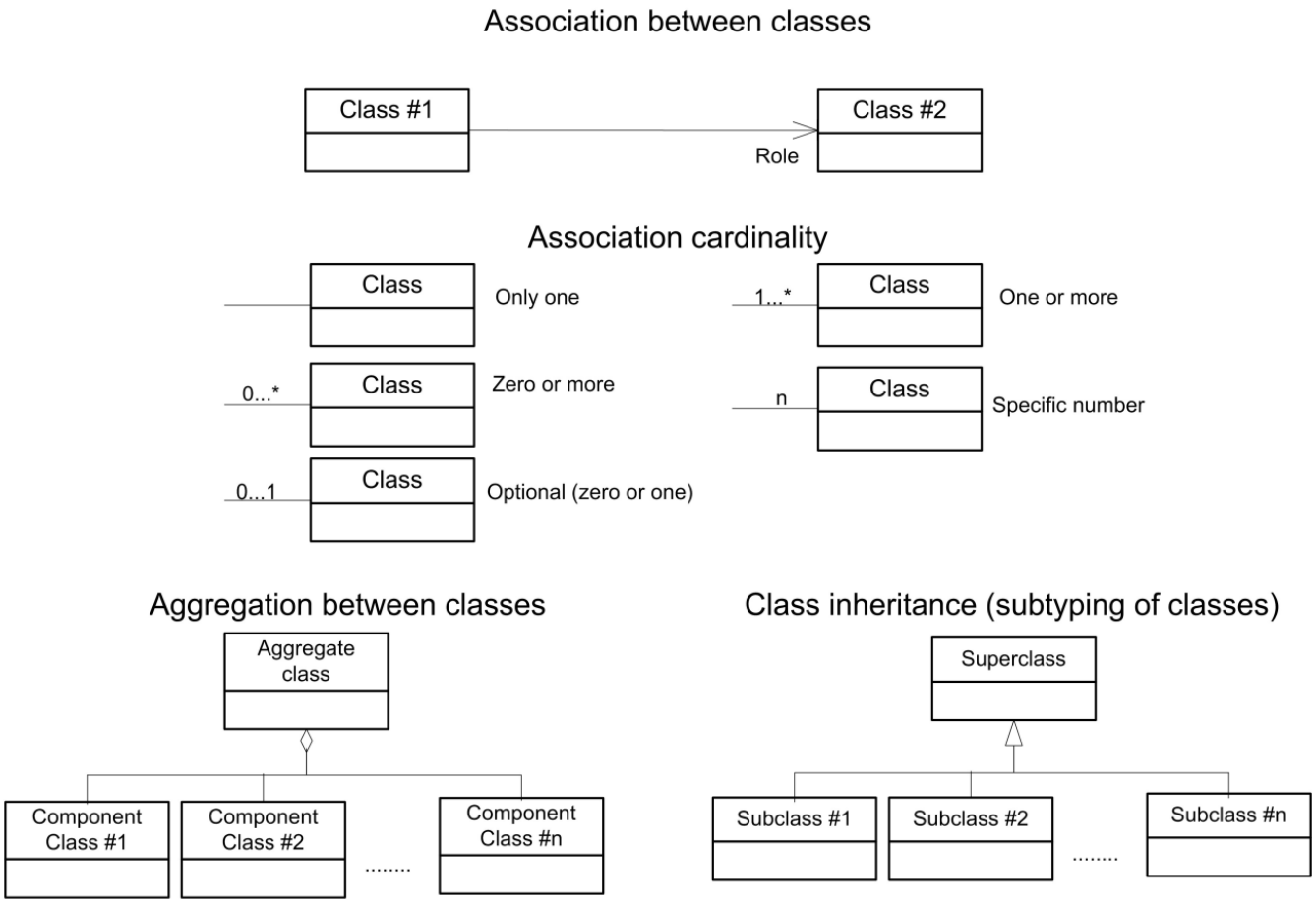


Figure 1. UML notation (see ISO TS 19103, Geographic information - Conceptual schema language).

According to GML3 all associations between model elements in CityGML are uni-directional. Thus, associations in CityGML 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:

- <<Geometry>> represents the geometry of an object. The geometry is an identifiable and distinguishable object that is derived from the abstract GML type *AbstractGeometryType*.
- <<Feature>> represents a thematic feature according to the definition in ISO 19109. A feature is an identifiable and distinguishable object that is derived from the abstract GML type *AbstractFeatureType*.
- <<Object>> represents an identifiable and distinguishable object that is derived from the abstract GML type *AbstractGMLType*.
- <<Enumeration>> enumerates the valid attribute values in a fixed list of named literal values. Enumerations are specified inline the CityGML schema.
- <<CodeList>> enumerates the valid attribute values. In contrast to Enumeration, the list of values is open and, thus, not given inline the CityGML schema. The allowed values can be provided within an external code list. It is recommended that code lists are implemented as simple dictionaries following the GML 3.1.1 Simple Dictionary Profile (cf. chapter 6.6 and chapter 10.14).
- <<Union>> is a list of attributes. The semantics are that only one of the attributes can be present at any time.
- <<PrimitiveType>> is used for representations supported by a primitive type in the implementation.
- <<DataType>> is used as a descriptor of a set of values that lack identity. Data types include primitive prede-fined types and user-definable types. A DataType is thus a class with few or no operations whose primary purpose is to hold the abstract state of another class for transmittal, storage, encoding or persistent storage.
- <<Leaf>> is used within UML package diagrams to indicate model elements that can have no further subtypes.
- <<XSDSchema>> is used within UML package diagrams to denote the root element of an XSD Schema contain-ing all the definitions for a particular namespace. All the package contents or component classes are placed within the one schema.
- <<ApplicationSchema>> is used within UML package diagrams to denote an XML Schema definition funda-mentally dependent on the concepts of another independent Standard within the XML Schema metalan-guage. For example, ApplicationSchema indicates extensions of GML consistent with the GML “rules for application schemas”.

In order to enhance the readability of the CityGML UML diagrams, classes are depicted in different colors if they belong to different UML packages (see Fig. 8 for an overview of UML packages). The following coloring scheme is applied:

- Classes painted in yellow belong to the UML package which is subject of discussion in that clause of the specification in which the UML diagram is given. For example, in the context of chapter 10.1 which introduces the *CityGML Core* module, the yellow color is used to denote classes which are de-fined in the *CityGML Core* UML package. Likewise, the yellow classes shown in UML diagrams in chapter 10.3 are associated with the *Building* module which is subject of discussion in that chapter.

- Classes painted in blue belong to a CityGML UML package different to that associated with the yellow color. In order to explicitly denote the UML package of such classes, their class names carry a namespace prefix which is uniquely associated with a CityGML module throughout this specification (cf. section 4.3 for a list of namespaces and prefixes). For example, in the context of the *Building* module, classes from the *CityGML Core* module are painted in blue and their class names are preceded by the prefix core.
- Classes painted in green are defined in GML3 and their class names are preceded by the prefix *gml*.

The following example UML diagram demonstrates the UML notation and coloring scheme used throughout this specification. In this example, the yellow classes are associated with the *CityGML Building* module, the blue classes are from the *CityGML Core* module, and the green class depicts a geometry element defined by GML3.

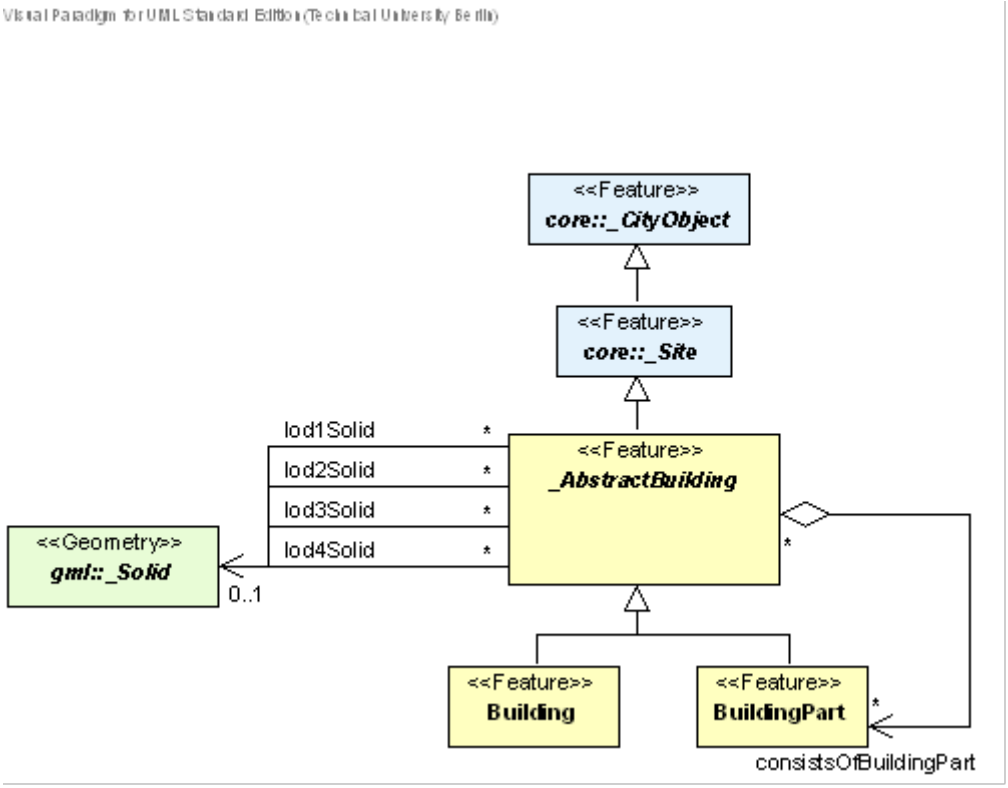


Figure 2. Example UML diagram demonstrating the UML notation and coloring scheme used throughout the CityGML specification.

5.3. XML namespaces and namespace prefixes

The CityGML data model is thematically decomposed into a core module and thematic extension modules. All modules including the core are specified by their own XML schema file, each defining a globally unique XML namespace. The extension modules are based on the core module and, thus, contain (by reference) the CityGML core schema.

Within this document the module namespaces are associated with recommended prefixes. These prefixes are consistently used within the normative parts of this specification, for all UML diagrams and example CityGML instance documents. The CityGML core and extension modules along with their XML namespace identifiers and recommended namespace prefixes are listed in Tab. 1.

Table 3. List of CityGML modules, their associated XML namespace identifiers, and example namespace prefixes.

CityGML module	Namespace identifier	Namespace prefix
CityGML Core	http://www.opengis.net/citygml/2.0	core
Appearance	http://www.opengis.net/citygml/appearance/2.0	app
Bridge	http://www.opengis.net/citygml/bridge/2.0	brid
Building	http://www.opengis.net/citygml/building/2.0	bldg
CityFurniture	http://www.opengis.net/citygml/cityfurniture/2.0	frn
CityObjectGroup	http://www.opengis.net/citygml/cityobjectgroup/2.0	grp
Generics	http://www.opengis.net/citygml/generics/2.0	gen
LandUse	http://www.opengis.net/citygml/landuse/2.0	luse
Relief	http://www.opengis.net/citygml/relief/2.0	dem
Transportation	http://www.opengis.net/citygml/transportation/2.0	tran
Tunnel	http://www.opengis.net/citygml/tunnel/2.0	tun
Vegetation	http://www.opengis.net/citygml/vegetation/2.0	veg
WaterBody	http://www.opengis.net/citygml/waterbody/2.0	wtr
TexturedSurface [deprecated]	http://www.opengis.net/citygml/texturedsurface/2.0	tex

Further XML Schema definitions relevant to this standard are shown in Tab. 2 along with the corresponding XML namespace identifiers and namespace prefixes consistently used within this document.

Table 4. List of XML Schema definitions, their associated XML namespace identifiers, and example namespace prefixes used within this document.

XML Schema Definition	Namespace identifier	Namespace prefix
Geography Markup Language version 3.1.1 (from OGC)	http://www.opengis.net/gml	gml

XML Schema Definition	Namespace identifier	Namespace prefix
Extensible Address Language version 2.0 (from OASIS)	urn:oasis:names:tc:ciq:xsdschema:xAL:2.0	xAL
Schematron Assertion Lan-guage version 1.5	http://www.ascc.net/xml/schematron	sch

5.4. XML-Schema

The normative parts of the standard use the W3C XML schema language to describe the grammar of conformant CityGML data instances. XML schema is a rich language with many capabilities. While a reader who is unfamil-iar with an XML schema may be able to follow the description in a general fashion, this standard is not intended to serve as an introduction to XML schema. In order to have a full understanding of this candidate standard, it is necessary for the reader to have a reasonable knowledge of XML schema.image::images/image.png[]

Chapter 6. Overview of CityGML

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

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

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

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

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

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

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

Chapter 7. General characteristics of CityGML

7.1. Modularisation

The CityGML data model consists of class definitions for the most important types of objects within virtual 3D city models. These classes have been identified to be either required or important in many different application areas. However, implementations are not required to support the overall CityGML data model in order to be conformant to the standard, but may employ a subset of constructs according to their specific information needs. For this purpose, modularisation is applied to the CityGML data model (cf. chapter 7).

The CityGML data model is thematically decomposed into a *core module* and thematic *extension modules*. The core module comprises the basic concepts and components of the CityGML data model and, thus, must be implemented by any conformant system. Based on the core module, each extension covers a specific thematic field of virtual 3D city models. CityGML introduces the following thirteen thematic extension modules: *Appearance*, *Bridge*, *Building*, *CityFurniture*, *CityObjectGroup*, *Generics*, *LandUse*, *Relief*, *Transportation*, *Tunnel*, *Vegetation*, *WaterBody*, and *TexturedSurface* [deprecated].

CityGML compliant implementations may support any combination of extension modules in conjunction with the core module. Such combinations of modules are called CityGML profiles. Therefore, CityGML profiles allow for valid partial implementations of the overall CityGML data model.

7.2. Multi-scale modelling (5 levels of detail, LOD)

CityGML supports different Levels of Detail (LOD). LODs are required to reflect independent data collection processes with differing application requirements. Further, LODs facilitate efficient visualisation and data analysis (see Fig. 3). In a CityGML dataset, the same object may be represented in different LOD simultaneously, enabling the analysis and visualisation of the same object with regard to different degrees of resolution. Furthermore, two CityGML data sets containing the same object in different LOD may be combined and integrated. However, it will be within the responsibility of the user or application to make sure objects in different LODs refer to the same real-world object.

The coarsest level LOD0 is essentially a two and a half dimensional Digital Terrain Model over which an aerial image or a map may be draped. Buildings may be represented in LOD0 by footprint or roof edge polygons. LOD1 is the well-known blocks model comprising prismatic buildings with flat roof structures. In contrast, a building in LOD2 has differentiated roof structures and thematically differentiated boundary surfaces. LOD3 denotes architectural models with detailed wall and roof structures potentially including doors and windows. LOD4 completes a LOD3 model by adding interior structures for buildings. For example, buildings in LOD4 are composed of rooms, interior doors, stairs, and furniture. In all LODs appearance information such as highresolution textures can be mapped onto the structures (cf. 6.9).

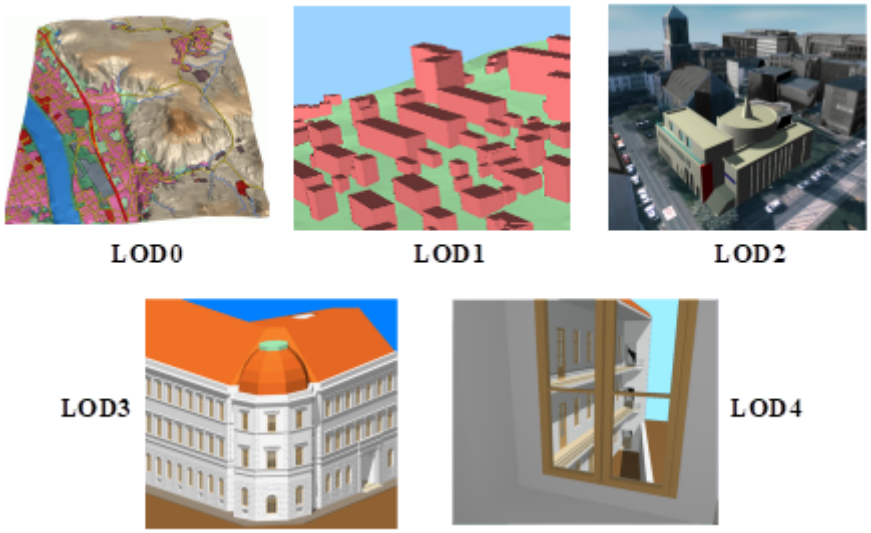


Figure 3. The five levels of detail (LOD) defined by CityGML (source: IGG Uni Bonn)

LODs are also characterised by differing accuracies and minimal dimensions of objects (cf. Tab. 3). The accura-cy requirements given in this standard are debatable and are to be considered as discussion proposals. Accuracy is described as standard deviation of the absolute 3D point coordinates. Relative 3D point accuracy will be added in a future version of CityGML and it is typically much higher than the absolute accuracy. In LOD1, the positional and height accuracy of points should be 5m or less, while all objects with a footprint of at least 6m by 6m should be considered. The positional and height accuracy of LOD2 is proposed to be 2m or better. In this LOD, all objects with a footprint of at least 4m × 4m should be considered. Both types of accuracies in LOD3 should be 0.5m, and the minimal footprint is suggested to be 2m × 2m. Finally, the positional and height accura-cy of LOD4 should be 0.2m or less. By means of these figures, the classification in five LOD may be used to assess the quality of 3D city model datasets. The LOD categorisation makes datasets comparable and provides support for their integration.

Table 5. LOD 0-4 of CityGML with their proposed accuracy requirements (discussion proposal, based on: Albert et al. 2003).

	LOD0	**LOD1**	**LOD2**	**LOD3**	**LOD4**
Model scale description	regional, landscape	city, region	city, city districts, projects	city districts, architectural models (exterior), landmark	architectural models (interi-or), landmark
Class of accuracy	lowest	low	middle	high	very high
Absolute 3D point accuracy (position / height)	lower than LOD1	5/5m	2/2m	0.5/0.5m	0.2/0.2m

	LOD0	LOD1	LOD2	LOD3	LOD4
Generalisation	maximal generalisation	object blocks as generalised features; > 6*6m/3m	objects as generalised features; > 4*4m/2m	object as real features; > 2*2m/1m	constructive elements and openings are represented
Building installations	no	no	yes	representative exterior features	real object form
Roof structure/representation	yes	flat	differentiated roof structures	real object form	real object form
Roof overhanging parts	yes	no	yes, if known	yes	yes
CityFurniture	no	important objects	prototypes, generalized objects	real object form	real object form
SolitaryVegetationObject	no	important objects	prototypes, higher 6m	prototypes, higher 2m	prototypes, real object form
PlantCover	no	>50*50m	>5*5m	< LOD2	<LOD2
…to be continued for the other feature themes					

Whereas in CityGML each object can have a different representation for every LOD, often different objects from the same LOD will be generalised to be represented by an aggregate object in a lower LOD. CityGML supports the aggregation / decomposition by providing an explicit generalisation association between city objects (further details see UML diagram in chapter 10.1).

7.3. Coherent semantical-geometrical modelling

One of the most important design principles for CityGML is the coherent modelling of semantics and geomet-rical/topological properties. At the semantic level, real-world entities are represented by features, such as build-ings, 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 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 wall of a building has two windows and a door on the semantic level, then the geometry representing the wall must contain also the geometry parts of both windows and the door.

7.4. Closure surfaces

Objects, which are not modelled 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 ClosureSurface. These are special surfaces, which are taken into account, when needed to compute volumes and are neglected, when they are irrelevant or not appropriate, for example in visualisations.

The concept of ClosureSurface is also 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, for example in flood simulations. The entrances to subsurface objects also have to be sealed to avoid holes in the digital terrain model Figure 4. However, in close-range visualisations the entrance must be treated as open. Thus, closure surfaces are an adequate way to model those entrances.

NOTE Combine Figures 4

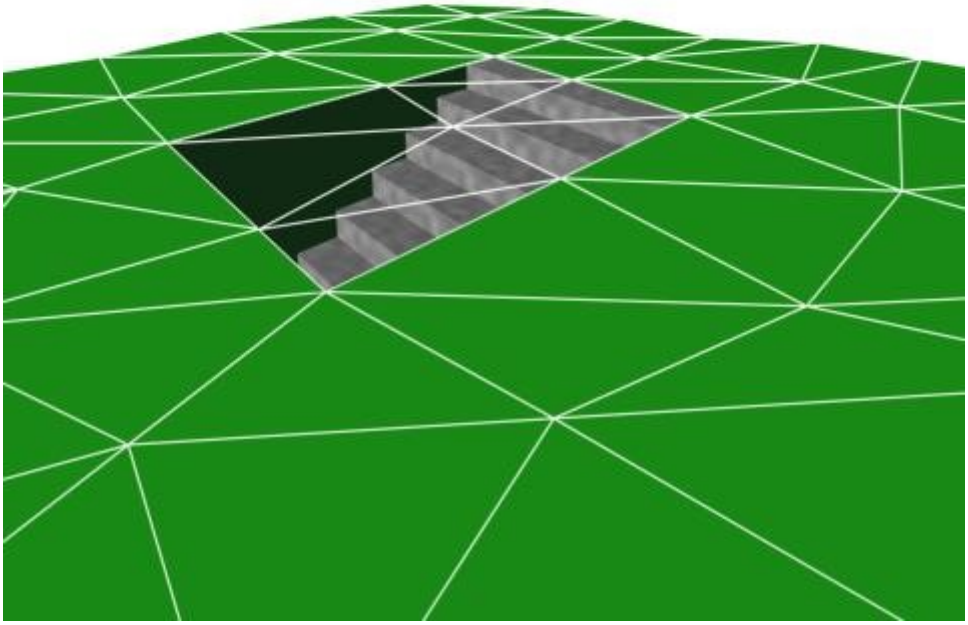


Figure 4. Closure surfaces to seal open structures. Passages are subsurface objects (left). The entrance is sealed by a virtual ClosureSurface, which is both part of the DTM and the subsurface object (right) (graphic: IGG Uni Bonn).

7.5. Terrain Intersection Curve (TIC)

A crucial 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 if terrains and 3D objects in different LOD are combined, or if they come from different providers (Kolbe and 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 Fig. 5). TICs can be applied to buildings and building parts (cf. chapter 10.3), bridge, bridge parts and bridge construction elements (cf. chapter 10.5), tunnel and tunnel parts (cf. chapter 10.4), city furniture objects (cf. chapter 10.9), and generic city objects (cf. chapter 10.12). 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 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 LOD.

NOTE	Combine figures 5

Figure 5. TerrainIntersectionCurve for a building (left, black) and a tunnel object (right, white). The tunnel's hollow space is sealed by a triangulated ClosureSurface (graphic: IGG Uni Bonn).

7.6. Code lists for enumerative attributes

CityGML feature types often include attributes whose values can be enumerated in a list of discrete values. An example is the attribute roof type of a building, whose attribute values typically are saddle back roof, hip roof, semi-hip roof, flat roof, pent roof, or tent roof. If such an attribute is typed as string, misspellings or different names for the same notion obstruct interoperability. Moreover, the list of possible attribute values often is not fixed and may substantially vary for different countries (e.g., due to national law and regulations) and for differ-ent information communities.

In CityGML, such enumerative attributes are of type gml:CodeType and their allowed attribute values can be provided in a code list which is specified outside the CityGML schema. A code list contains coded attribute values and ensures that the same code is used for the same notion or concept. If a code list is provided for an enumerative attribute, the attribute may only take values from this list. This allows applications to validate the attribute value and thus facilitates semantic and syntactic interoperability. It is recommended that code lists are implemented as simple dictionaries following the GML 3.1.1 Simple Dictionary Profile (cf. Whiteside 2005).

The governance of code lists is decoupled from the governance of the CityGML schema and specification. Thus, code lists may be specified by any organisation or information community according to their information needs. There shall be one authority per code list who is in charge of the code list values and the maintenance of the code list. Further information on the CityGML code

list mechanism is provided in chapter 10.14.

Code lists can have references to existing models. For example, room codes defined by the Open Standards Consortium for Real Estate (OSCRE) can be referenced or classifications of buildings and building parts intro-duced by the National Building Information Model Standard (NBIMS) can be used. Annex C contains non-normative code lists proposed by the SIG 3D for almost all enumerative attributes in CityGML. They can be directly referenced in CityGML instance documents and serve as an example for the definition of code lists.

7.7. External references

3D objects are often derived from or have relations to objects in other databases or data sets. For example, a 3D building model may have been constructed from a two-dimensional footprint in a cadastre data set, or may be derived from an architectural model (Fig. 6). The reference of a 3D object to its corresponding object in an external data set is essential, if an update must be propagated or if additional data is required, for example the name and address of a building's owner in a cadastral information system or information on antennas and doors in a facility management system. In order to supply such information, each _CityObject may refer to external data sets (for the UML diagram see Fig. 21; and for XML schema definition see annex A.1) using the concept of ExternalReference. Such a reference denotes the external information system and the unique identifier of the object in this system. Both are specified as a Uniform Resource Identifier (URI), which is a generic format for references to any kind of resources on the internet. The generic concept of external references allows for any _CityObject an arbitrary number of links to corresponding objects in external information systems (e.g. ALKIS, ATKIS, OS MasterMap®, GDF, etc.).

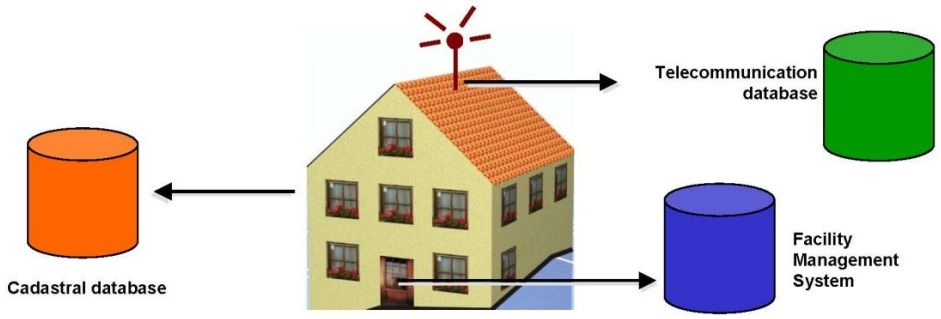


Figure 6. External references (graphic: IGG Uni Bonn).

7.8. City object groups

The grouping concept of CityGML allows for the aggregation of arbitrary city objects according to user-defined criteria, and to represent and transfer these aggregations as part of a city model (for the UML diagram see chapter 10.11; XML schema definition see annex A.6). A group may be assigned one or more names and may be further classified by specific attributes, for example, "escape route from room no. 43 in house no. 1212 in a fire scenario" as a name and "escape route" as type. Each member of the group can optionally be assigned a role name, which specifies the role this particular member plays in the group. This role name may, for example, describe the sequence number of this object in an escape route, or in the case of a building complex, denote the main building.

A group may contain other groups as members, allowing nested grouping of arbitrary depth. The grouping concept is delivered by the thematic extension module CityObjectGroup of CityGML (cf. chapter 10.11).

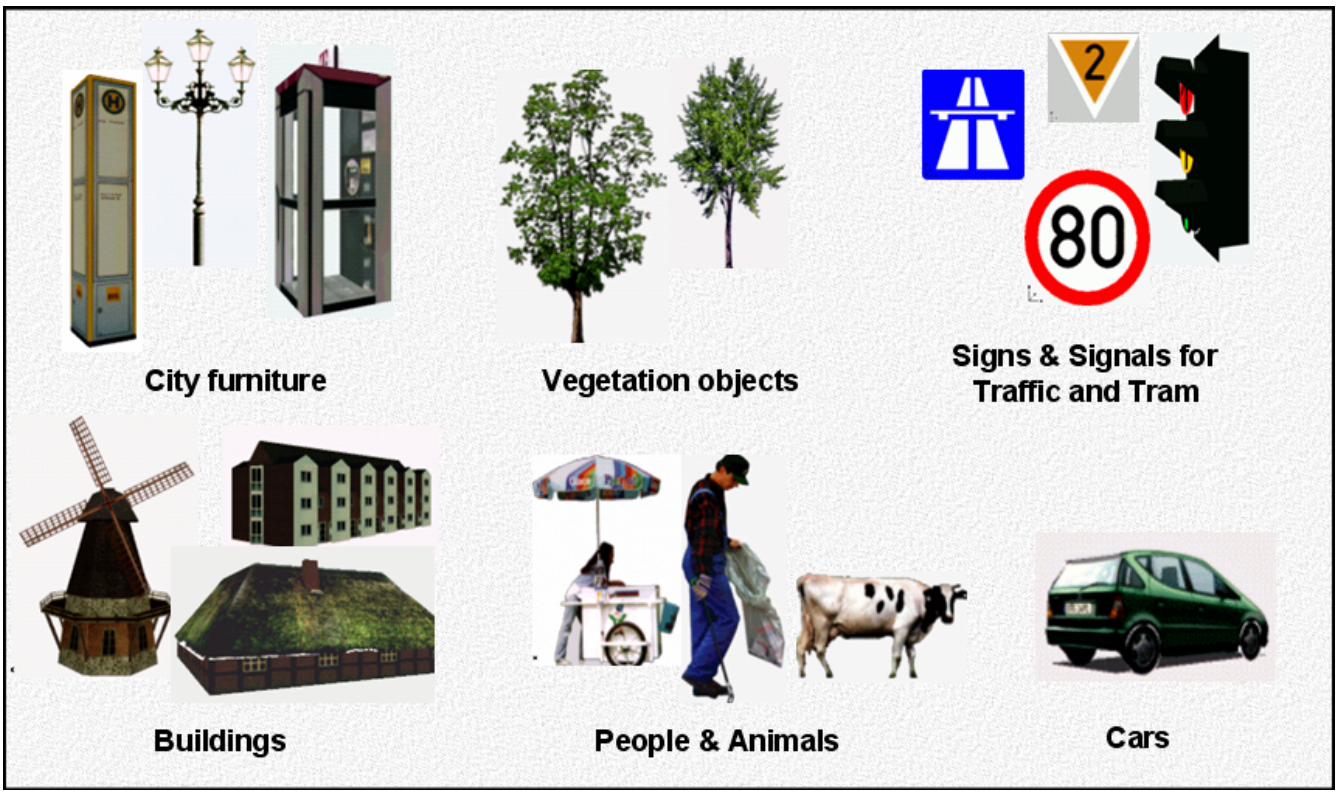
7.9. Appearances

Information about a surface’s appearance, i.e. observable properties of the surface, is considered an integral part of virtual 3D city models in addition to semantics and geometry. Appearance relates to any surface-based theme, e.g. infrared radiation or noise pollution, not just visual properties. Consequently, data provided by appearances can be used as input for both presentation of and analysis in virtual 3D city models.

CityGML 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 its own extension module Appearance (cf. chapter 9).

7.10. 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 (Fig. 7). 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 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 like VRML and X3D. As the GML3 geometry model does not provide support for scene graph concepts, it is implemented as an extension to the GML3 geometry model (for further description cf. chapter 8.2).



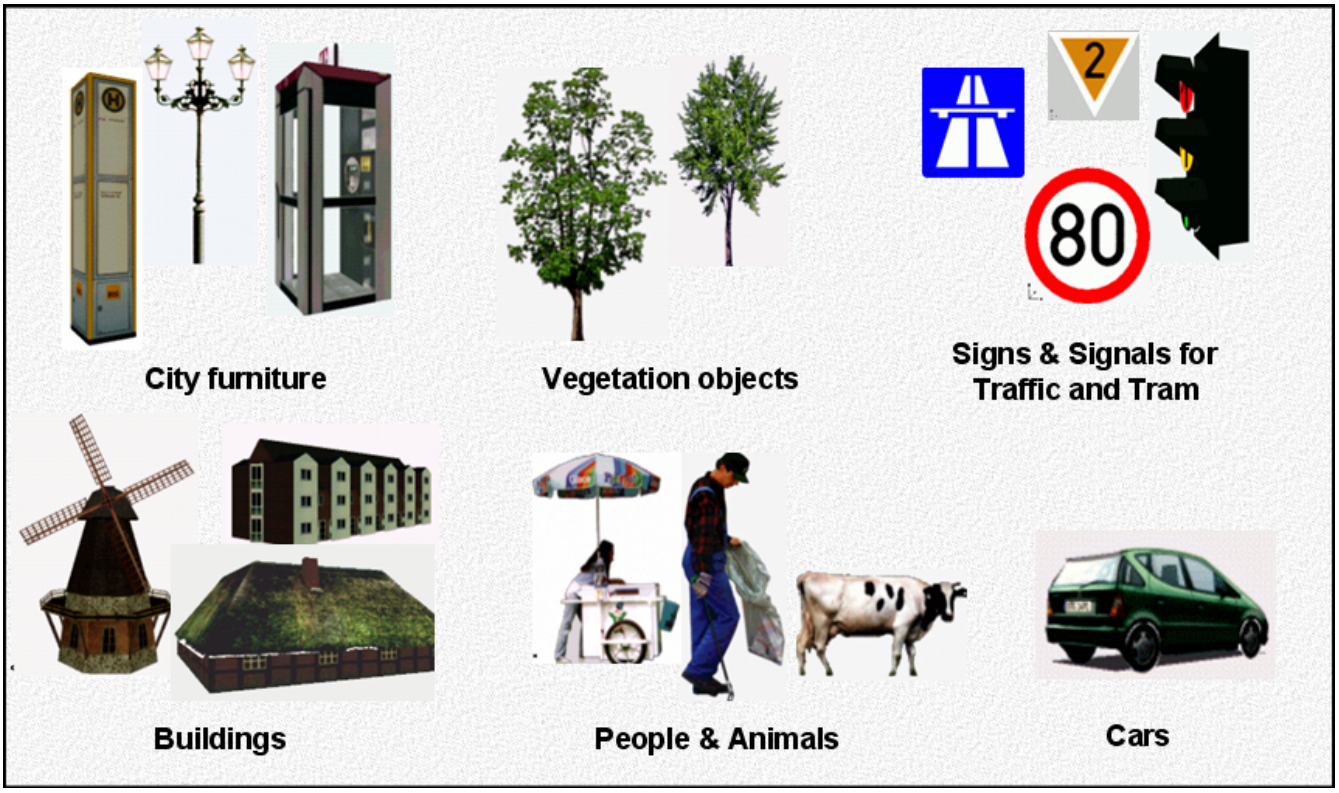


Figure 7. Examples of prototypic shapes (source: Rheinmetall Defence Electronics).

7.11. Generic city objects and attributes

CityGML is being designed as a universal topographic 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 thematic classes of CityGML. CityGML provides two different concepts to support the exchange of such data: 1) generic objects and attributes, and 2) Application Domain Extensions (cf. chapter 6.12).

The concept of generic objects and attributes allows for the extension of CityGML applications during runtime, i.e. any _CityObject may be augmented by additional attributes, whose names, data types, and values can be provided by a running application without any change of the CityGML XML schema. Similarly, features not represented by the predefined thematic classes of the CityGML data model may be modelled and exchanged using generic objects. The generic extensions of CityGML are provided by the thematic extension module Generics (cf. chapter 10.12).

The current version of CityGML does not include, for example, explicit thematic models for embankments, excavations and city walls. These objects may be stored or exchanged using generic objects and attributes.

7.12. Application Domain Extensions (ADE)

Application Domain Extensions (ADE) specify additions to the CityGML data model. Such additions comprise the introduction of new properties to existing CityGML classes like e.g. the number of habitants of a building or the definition of new object types. The difference between ADEs and generic objects and attributes is, that an ADE has to be defined in an extra XML schema definition file with its own namespace. This file has to explicitly import the XML Schema definition of the

extended CityGML modules.

The advantage of this approach is that the extension is formally specified. Extended CityGML instance docu-ments can be validated against the CityGML and the respective ADE schema. ADEs can be defined (and even standardised) by information communities which are interested in specific application fields. More than one ADE can be actively used in the same dataset (further description cf. chapter 10.13).

ADEs may be defined for one or even several CityGML modules providing a high flexibility in adding additional information to the CityGML data model. Thus, the ADE mechanism is orthogonally aligned with the modular-isation approach of CityGML. Consequently, there is no separate extension module for ADEs.

In this specification, two examples for ADEs are included:

- An ADE for Noise Immission Simulation (Annex H) which is employed in the simulation of environmental noise dispersion according to the Environmental Noise Directive of the European Commission (2002/49/EC);
- An ADE for Ubiquitous Network Robots Services (Annex I) which demonstrates the usage of CityGML for the navigation of robots in indoor environments.

Further examples for ADEs are the CAFM ADE (Bleifuß et al., 2009) for facility management, the UtilityNetworkADE (Becker et al., 2011) for the integrated 3D modeling of multi-utility networks and their interdependencies, the HydroADE (Schulte and Coors, 2008) for hydrographical applications and the GeoBIM (IFC) ADE (van Berlo et al., 2011) which combines BIM information from IFC (from bSI) with CityGML and is implemented in the open source modelserver BIMserver.org.

Chapter 8. Modularization

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

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

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

The CityGML core module defines the basic concepts and components of the CityGML data model. It is to be seen as the universal lower bound of the overall CityGML data model and a dependency of all thematic exten-sion modules. Thus, the core module is unique and must be implemented by any conformant system. Based on the CityGML core module, each extension module contains a logically separate thematic component of the CityGML data model. The extensions to the core are derived by vertically slicing the overall CityGML data model. Since the core module is contained (by reference) in each extension module, its general concepts and components are universal to all extension modules. The following thirteen thematic extension modules are introduced by version 2.0 of the CityGML standard. They are directly related to clauses of this document each covering the corresponding thematic field of CityGML:

- Appearance (cf. clause 9),
- Bridge (cf. clause 10.5)
- Building (cf. clause 10.3),
- CityFurniture (cf. clause 10.9),
- CityObjectGroup (cf. clause 10.11),
- Generics (cf. clause 10.12),
- LandUse (cf. clause 10.10),

- Relief (cf. clause 10.2),
- Transportation (cf. clause 10.7),
- Tunnel (cf. clause 10.4)
- Vegetation (cf. clause 10.8),
- WaterBody (cf. clause 10.6), and
- TexturedSurface [deprecated] (cf. clause 9.8).

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

As for future development, each CityGML module may be further developed independently from other modules by expert groups and information communities. Resulting proposals and changes to modules may be introduced into future revisions of the CityGML standard without affecting the validity of other modules. Furthermore, thematic components not covered by the current CityGML data model may be added to future revisions of the standard by additional thematic extension modules. These additional extensions may establish dependency relations to any other existing CityGML module but shall at least be dependent on the CityGML core module. Consequently, the CityGML base profile may vary over time as new extensions are added. However, if a specific application has information needs to be modelled and exchanged which are beyond the scope of the CityGML data model, this application data can also be incorporated within the existing modules using CityGML's Applica-tion Domain Extension mechanism (cf. clause 10.13) or by employing the concepts of generic city objects and attributes (cf. chapter 10.12).

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

8.1. CityGML core and extension modules

Each CityGML module is specified by its own XML Schema definition file and is defined within an individual and globally unique XML target namespace. According to dependency relations between modules, each module may, in addition, import namespaces associated to such related CityGML modules. However, a single namespace shall not be directly included in two modules. Thus, all elements belonging to one module are associ-ated to the module's namespace only. By this means,

module elements are guaranteed to be properly separated and distinguishable in CityGML instance documents.

Compared to CityGML versions before 1.0, the aforementioned namespace conventions introduce an extra level of complexity to data files as there is no single CityGML namespace any more. In contrast, components of different CityGML modules and, thus, of different namespaces may be arbitrarily mixed within the same CityGML instance document. Furthermore, an application might have to parse instance documents containing elements of modules which are not employed by the application itself. These parsing problems though can easily be overcome by non-“schema-aware” applications, i.e. applications that do not parse and interpret GML applica-tion schemas in a generic way. Elements from different namespaces than those declared by the application’s employed CityGML profile could be skipped. Comparable observations have to be made when using CityGML’s Application Domain Extension mechanism (cf. clause 10.13).

As for version 2.0 of the CityGML standard, there are no two thematic extension modules related by dependen-cy. Thus, all extension modules are truly independent from each other and may be separately supported by implementations. However, the CityGML core module is a dependency for any extension module. This means that the XML schema file of the core module is imported by each XML schema file defining an extension.

The dependency relations between CityGML’s modules are illustrated in Fig. 8 using an UML package diagram. Each module is represented by a package. The package names correspond to the module names. A dashed arrow in the figure indicates that the schema at the tail of the arrow depends upon the schema at the head of the arrow. For CityGML modules, a dependency occurs where one schema <import>s another schema and accordingly the corresponding XML namespace. For example, the extension module Building imports the schema of the CityGML Core module. A short description of each module is given in Tab. 4.

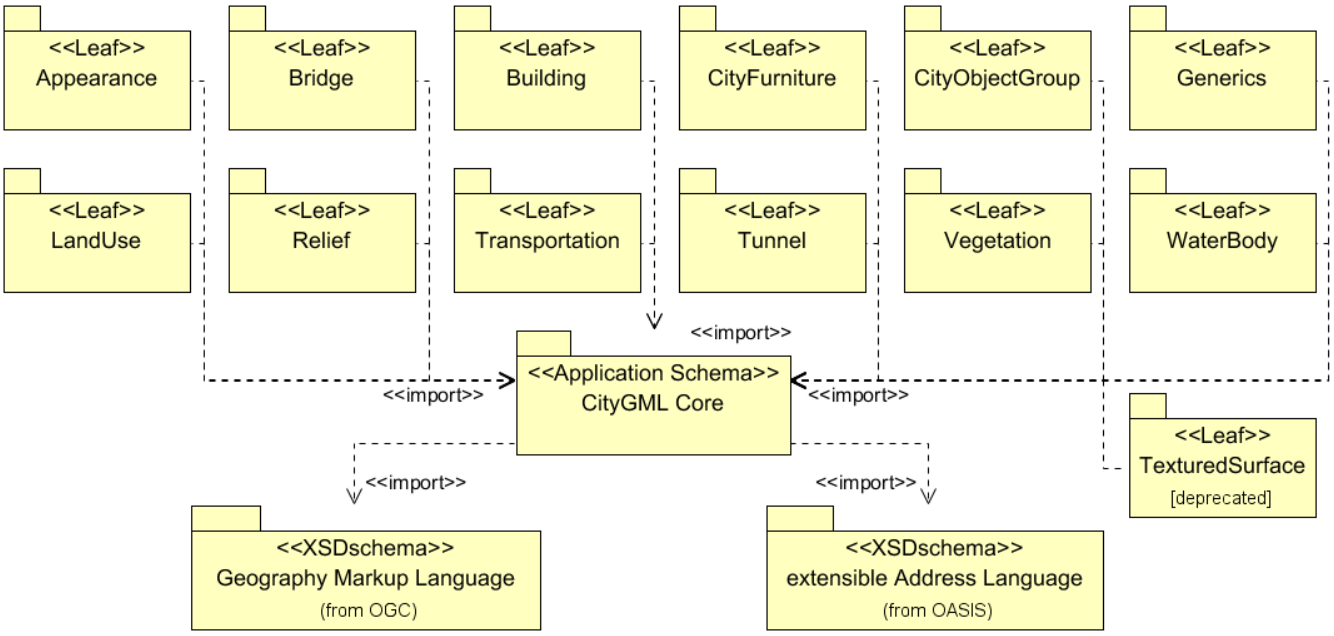


Figure 8. UML package diagram illustrating the separate modules of CityGML and their schema dependencies. Each extension module (indicated by the leaf packages) further imports the GML 3.1.1 schema definition in order to represent spatial properties of its thematic classes. For readability reasons, the corresponding dependencies have been omitted.

Module name	CityGML Core
XML namespace identifier	http://www.opengis.net/citygml/2.0
XML Schema file	cityGML.Base.xsd
Recommended namespace prefix	core
Module description	The CityGML Core module defines the basic components of the CityGML data model. Primarily, this comprises abstract base classes from which all thematic classes are (transitively) derived. But also non-abstract content common to more than one extension module, for example basic data types, is defined within the core module. The core module itself imports the XML schema definition files of GML version 3.1.1 and the OASIS extensible Address Language xAL.

Module name	Appearance
XML namespace identifier	http://www.opengis.net/citygml/appearance/2.0
XML Schema file	appearance.xsd

Recommended namespace prefix	app
Module description	The Appearance module provides the means to model appearances of CityGML features, i.e. observable properties of the feature's surface. Appear-ance data may be stored for each city object. Therefore, the abstract base class _CityObject defined within the core module is augmented by an additional property using CityGML's Application Domain Extension mechanism. Thus, the Appearance module has a deliberate impact on all thematic extension modules.

Module name	Bridge
XML namespace identifier	http://www.opengis.net/citygml/bridge/2.0
XML Schema file	bridge.xsd
Recommended namespace prefix	brid
Module description	The Bridge module allows the representation of thematic and spatial aspects of bridges, bridge parts, bridge installations, and interior bridge structures in four levels of detail (LOD 1 – 4).

Module name	Building
XML namespace identifier	http://www.opengis.net/citygml/building/2.0
XML Schema file	building.xsd
Recommended namespace prefix	bldg
Module description	The Building module allows the representation of thematic and spatial aspects of buildings, building parts, building installations, and interior building struc-tures in five levels of detail (LOD 0 – 4).

Module name	CityFurniture
XML namespace identifier	http://www.opengis.net/citygml/cityfurniture/2.0
XML Schema file	cityFurniture.xsd
Recommended namespace prefix	frn
Module description	The CityFurniture module is used to represent city furniture objects in cities. City furniture objects are immovable objects like lanterns, traffic signs, adver-tising columns, benches, or bus stops that can be found in traffic areas, residen-tial areas, on squares, or in built-up areas.

Module name	CityObjectGroup
XML namespace identifier	http://www.opengis.net/citygml/cityobjectgroup/2.0
XML Schema file	cityObjectGroup.xsd
Recommended namespace prefix	grp
Module description	The CityObjectGroup module provides a grouping concept for CityGML. Arbitrary city objects may be aggregated in groups according to user-defined criteria to represent and transfer these aggregations as part of the city model. A group may be further classified by specific attributes.

Module name	Generics
XML namespace identifier	http://www.opengis.net/citygml/generics/2.0
XML Schema file	generics.xsd
Recommended namespace prefix	gen
Module description	The Generics module provides generic extensions to the CityGML data model that may be used to model and exchange additional attributes and features not covered by the predefined thematic classes of CityGML. However, generic extensions shall only be used if appropriate thematic classes or attributes are not provided by any other CityGML module. In order to represent generic attributes, the Generics module augments the abstract base class _CityObject defined within the core module by an addition-al property using CityGML's Application Domain Extension mechanism. Thus, the Generics module has a deliberate impact on all thematic extension modules.

Module name	LandUse
XML namespace identifier	http://www.opengis.net/citygml/landuse/2.0
XML Schema file	landUse.xsd
Recommended namespace prefix	luse
Module description	The LandUse module allows for the representation of areas of the earth's surface dedicated to a specific land use.

Module name	Relief

XML namespace identifier	http://www.opengis.net/citygml/relief/2.0
XML Schema file	relief.xsd
Recommended namespace prefix	dem
Module description	The Relief module allows for the representation of the terrain in a city model. CityGML supports terrain representations in different levels of detail, reflect-ing different accuracies or resolutions. The terrain may be specified as a regular raster or grid, as a TIN, by break lines, and by mass points.

Module name	Transportation
XML namespace identifier	http://www.opengis.net/citygml/transportation/2.0
XML Schema file	transportation.xsd
Recommended namespace prefix	tran
Module description	The Transportation module is used to represent the transportation features within a city, for example roads, tracks, railways, or squares. Transportation features may be represented as a linear network or by geometrically describing their 3D surfaces.

Module name	Tunnel
XML namespace identifier	http://www.opengis.net/citygml/tunnel/2.0
XML Schema file	tunnel.xsd
Recommended namespace prefix	tun
Module description	The Tunnel module facilitates the representation of thematic and spatial aspects of tunnels, tunnel parts, tunnel installations, and interior tunnel struc-tures in four level of detail (LOD 1 – 4)

Module name	Vegetation
XML namespace identifier	http://www.opengis.net/citygml/vegetation/2.0
XML Schema file	vegetation.xsd
Recommended namespace prefix	veg

Module description	The Vegetation module provides thematic classes to represent vegetation objects. CityGML's vegetation model distinguishes between solitary vegeta-tion objects like trees, and vegetation areas which represent biotopes like forests or other plant communities.

Module name	WaterBody
XML namespace identifier	http://www.opengis.net/citygml/waterbody/2.0
XML Schema file	waterBody.xsd
Recommended namespace prefix	wtr
Module description	The WaterBody module represents the thematic aspects and 3D geometry of rivers, canals, lakes, and basins. It does, however, not inherit any hydrological or other dynamic aspects so far.

Module name	Textured Surface [deprecated]
XML namespace identifier	http://www.opengis.net/citygml/texturedsurface/2.0
XML Schema file	texturedSuface.xsd
Recommended namespace prefix	tex
Module description	The TexturedSurface module allows for assigning visual appearance properties (color, shininess, transparency) and textures to 3D surfaces. Due to inherent limitations of its modelling approach this module has been marked deprecated and is expected to be removed in future CityGML versions. Appearance information provided by this module can be converted to CityGML's Appear-ance module without information loss. Thus, the use of the TexturedSurface module is strongly discouraged.

8.2. CityGML profiles

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

1. CityGML profile definition embedded inline the CityGML instance document A CityGML profile can be bound to an instance document using the schemaLocation attribute defined in the XML Schema instance namespace, http://www.w3.org/2001/XMLSchema-instance (commonly associated with the prefix xsi). The xsi:schemaLocation attribute provides a way to locate the XML Schema definition for namespaces defined in an XML instance document. Its value is a

whitespace-delimited list of pairs of Uniform Resource Identifiers (URIs) where each pair consists of a namespace followed by the location of that namespace's XML Schema definition, which is typically a .xsd file.

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

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

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

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

NOTE replace XML with UML if feasible.

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

NOTE replace XML with UML if feasible

Chapter 9. Spatial Model

Spatial properties of CityGML features are represented by objects of GML3's geometry model. This model is based on the standard ISO 19107 'Spatial Schema' (Herring 2001), representing 3D geometry according to the well-known Boundary Representation (B-Rep, cf. Foley et al. 1995). CityGML actually uses only a subset of the GML3 geometry package, defining a profile of GML3. This subset is depicted in Fig. 9 and Fig. 10. Further-more, GML3's explicit Boundary Representation is extended by scene graph concepts, which allow the represen-tation of the geometry of features with the same shape implicitly and thus more space efficiently (chapter 8.2).

9.1. Geometric-topological model

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

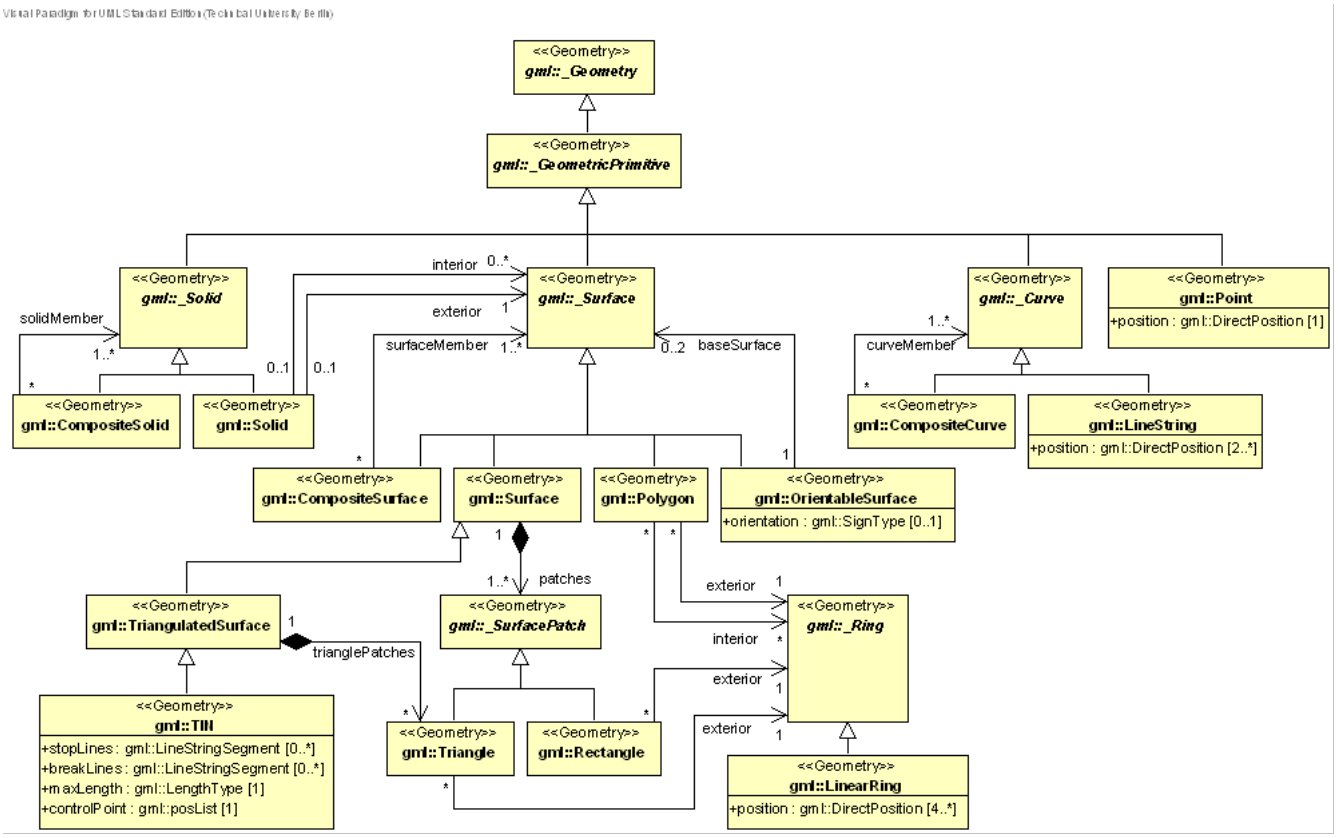


Figure 9. UML diagram of CityGML's geometry model (subset and profile of GML3): Primitives and Composites.

Combined geometries can be aggregates, complexes or composites of primitives (see illustration in Fig. 11). For an aggregate, the spatial relationship between components is not restricted. They may be disjoint, overlapping, touching, or disconnected. GML3 provides a special aggregate for each

dimension, a MultiPoint, a MultiCurve, a MultiSurface, and a MultiSolid (see Fig. 10). In contrast to aggregates, a complex is topologically structured: its parts must be disjoint, must not overlap and are allowed to touch, at most, at their boundaries or share parts of their boundaries. A composite is a special complex provided by GML3. It can only contain elements of the same dimension. Its elements must be disjoint as well, but they must be topologically connected along their bounda-ries. A composite can be a CompositeSolid, a CompositeSurface, or CompositeCurve. (cf. Fig. 9).

NOTE insert fig10 UML

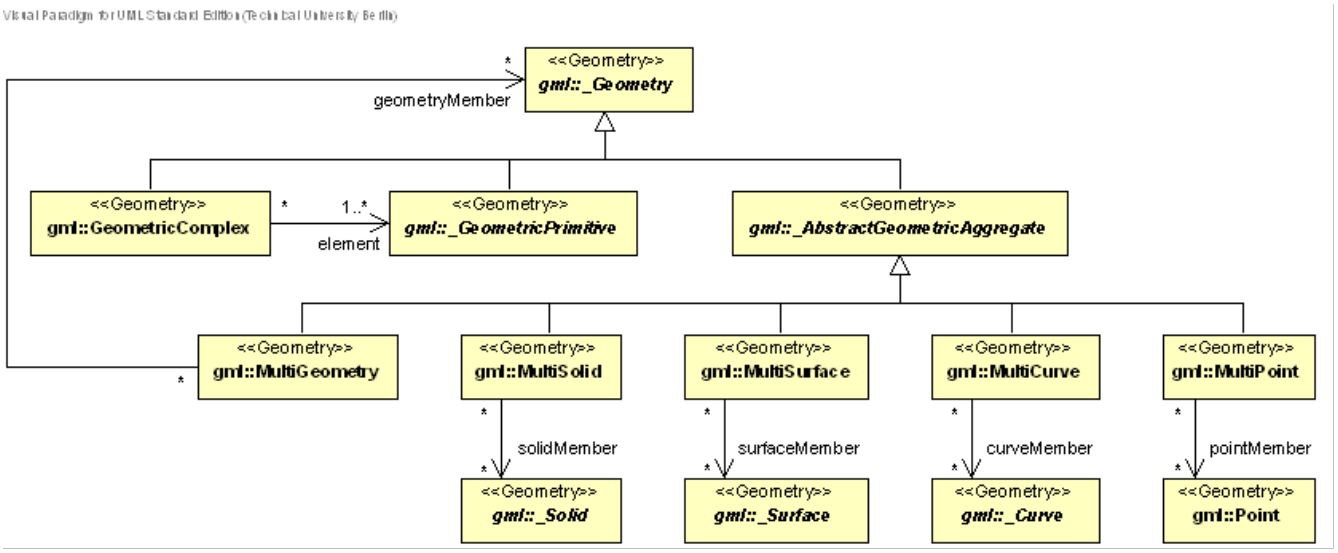


Figure 10. UML diagram of CityGML's geometry model: Complexes and Aggregates

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

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

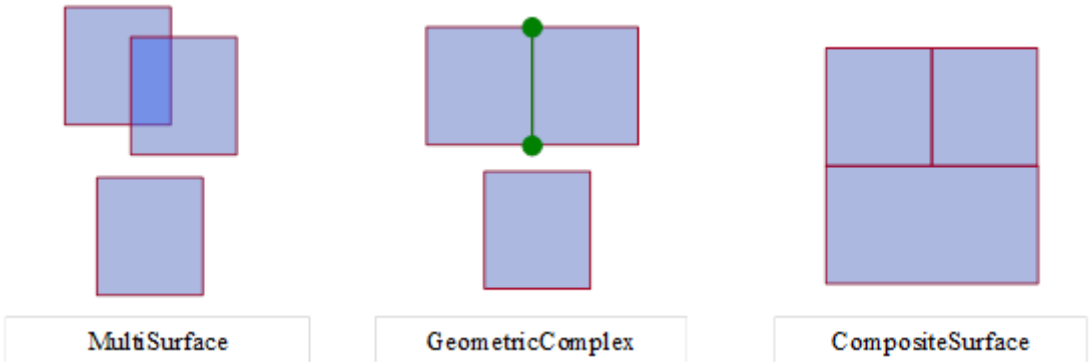


Figure 11. Combined geometries

The GML3 composite model realises a recursive aggregation schema for every primitive type of the corresponding dimension. This aggregation schema allows the definition of nested aggregations (hierarchy of components). For example, a building geometry (CompositeSolid) can be composed of the house geometry (CompositeSolid) and the garage geometry (Solid), while the house's geometry is further decomposed into the roof geometry (Solid) and the geometry of the house body (Solid).

CityGML provides the explicit modelling of topology, for example the sharing of geometry objects between features or other geometries. One part of space is represented only once by a geometry object and is reference by all features or more complex geometries which are defined or bounded by this geometry object. Thus redundancy is avoided and explicit topological relations between parts are maintained. Basically, there are three cases. First, two features may be defined spatially by the same geometry. For example, if a path is both a transportation feature and a vegetation feature, the surface geometry defining the path is referenced both by the transportation object and by the vegetation object. Second, geometry may be shared between a feature and another geometry. A geometry defining a wall of a building may be referenced twice: by the solid geometry defining the geometry of the building, and by the wall feature. Third, two geometries may reference the same geometry, which is in the boundary of both. For example, a building and an adjacent garage may be represented by two solids. The surface describing the area where both solids touch may be represented only once and it is referenced by both solids. As it can be seen from Fig. 12, this requires partitioning of the respective surfaces. In general, Boundary Represen-tation only considers visible surfaces. However, to make topological adjacency explicit and to allow the possibil-ity of deletion of one part of a composed object without leaving holes in the remaining aggregate touching elements are included. Whereas touching is allowed, permeation of objects is not in order to avoid the multiple representation of the same space. However, the use of topology in CityGML is optional.

In order to implement topology, CityGML uses the XML concept of XLinks provided by GML. Each geometry object that should be shared by different geometric aggregates or different thematic features is assigned an unique identifier, which may be referenced by a GML geometry property using a href attribute. CityGML does not deploy the built-in topology package of GML3, which provides separate topology objects accompanying the geometry. This kind of topology is very complex and elaborate. Nevertheless, it lacks flexibility when data sets, which might include or neglect topology, should be covered by the same data model. The XLink topology is simple and flexible and nearly as powerful as the explicit GML3 topology model. However, a disadvantage of the XLink topology is that navigation between topologically connected objects can only be performed in one direction (from an aggregate to its components), not (immediately) bidirectional as it is the case for GML's built-in topology. An example for CityGML's topology representation is given in the dataset listed in annex G.4.

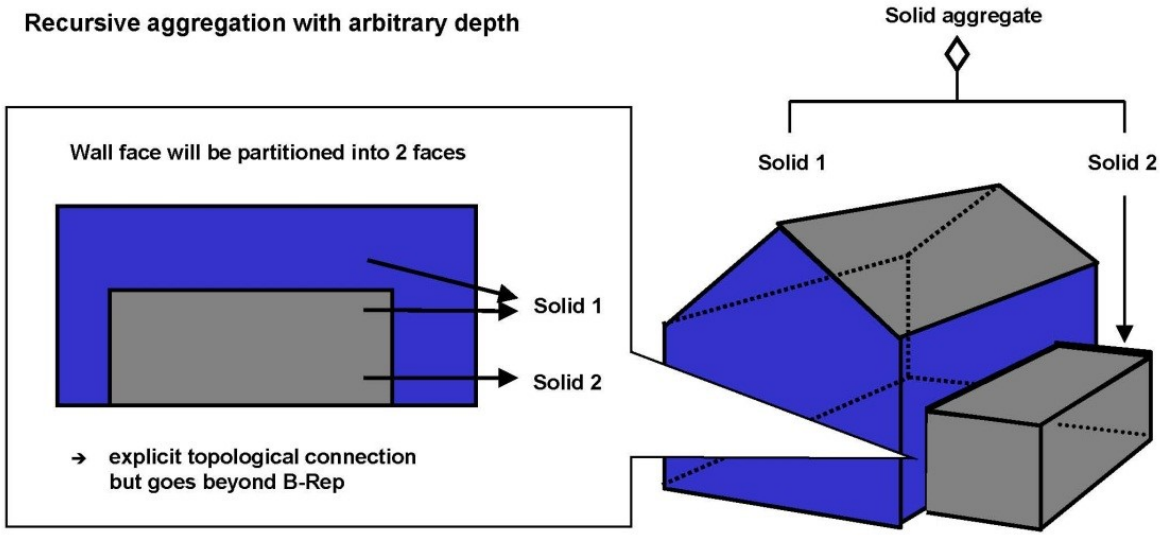


Figure 12. Recursive aggregation of objects and geometries in CityGML (graphic: IGG Uni Bonn).

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

NOTE replace GML? Or retain it as an example.

9.2. Spatial reference system

When dealing with geoinformation and virtual 3D city models in particular, the exact spatial reference is of utmost importance and a key requirement for the integration of different spatial datasets in a single 3D city model. CityGML inherits GML3's spatial capabilities of handling Coordinate Reference Systems (CRS) which is the usual way of denoting the spatial reference in GML 3.1.1. As CityGML is a true 3D standard, geometry elements are associated with a 3D CRS. There are only few exceptions to this rule where CityGML allows a 2D geometry element (for example, the referencePoint of a GeoreferencedTexture defined in CityGML's Appear-ance module must be given with 2D coordinate values, cf. chapter 9.4).

In general, a geometry may point to the CRS definition used by this geometry through the attribute srsName which is inherited from the abstract GML superclass gml:_Geometry. This may be a reference to a well-known CRS definition provided by an authority organization such as the European Petroleum Survey Group (EPSG), but may also be a pointer to a CRS that is locally defined within the same CityGML instance document. The OGC document "Definition identifier URNs in OGC namespace" (cf. Whiteside 2009; OGC Doc. No. 07-092r3) provides best practices for the URN

encoding of CRS references. Amongst others, it describes how to reference a single well-known 3D CRS definition (such as a 3D geographic CRS) as well as a compound CRS which combines two or more well-known CRS definitions (e.g., a projected CRS for the planimetry with a vertical CRS for the height reference). Examples for denoting a compound CRS for a CityGML instance document are given in Annex G.

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

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

It is strongly recommended that any CityGML instance document explicitly specifies the CRS for all contained geometry elements. This is especially important if the instance document is to be exchanged externally with third parties or is to be integrated with other spatial datasets. A mixed usage of different CRSs within the same dataset is possible and conformant with GML 3.1.1, whereas a single CRS reference given on the embracing CityModel feature collection (cf. chapter 10.1) simplifies the processing of the dataset by software systems. As for CityGML 2.0, this recommendation is non-normative and thus not accompanied by a conformance class. The main reason for this is to maintain backwards compatibility with CityGML 1.0.

9.3. Implicit geometries, prototypic objects, scene graph concepts

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

implicit geometries are part of CityGML's spatial model. The UML diagram is depicted in Fig. 13. The corresponding XML schema definition is provided in annex A.1.

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

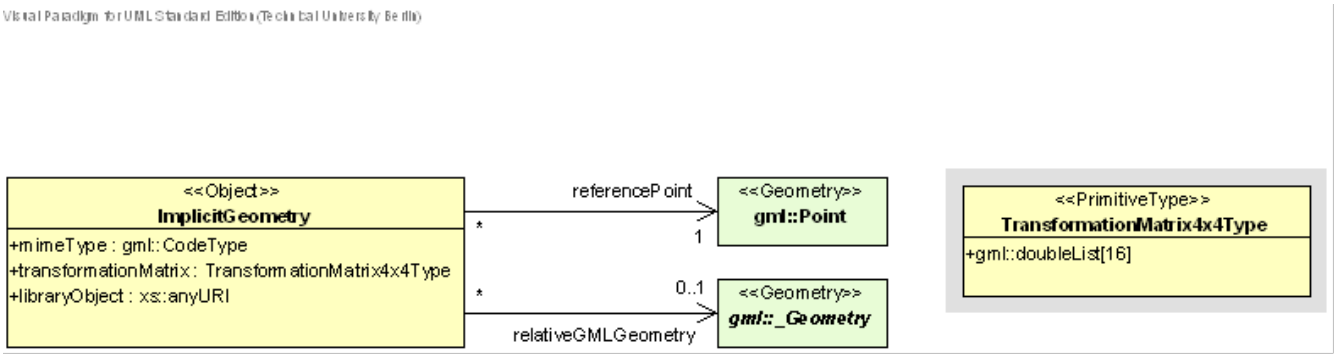


Figure 13. UML diagram of ImplicitGeometries. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Core module.

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

The term implicit geometry refers to the principle that a geometry object with a complex shape can be simply represented by a base point and a transformation, implicitly unfolding the object's shape at a specific location in the world coordinate system.

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

The implicit representation of 3D object geometry has some advantages compared to the explicit modelling, which represents the objects using absolute world coordinates. It is more space-efficient,

and thus more exten-sive scenes can be stored or handled by a system. The visualisation is accelerated since 3D graphics cards sup-port the scene graph concept. Furthermore, the usage of different shape versions of objects is facilitated, e.g. different seasons, since only the library objects have to be exchanged (see example in Fig. 65).

XML namespace

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

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

9.3.1. Code lists

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

9.3.2. Example CityGML datasets

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

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

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

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

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

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

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

Chapter 10. Appearance Model

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

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

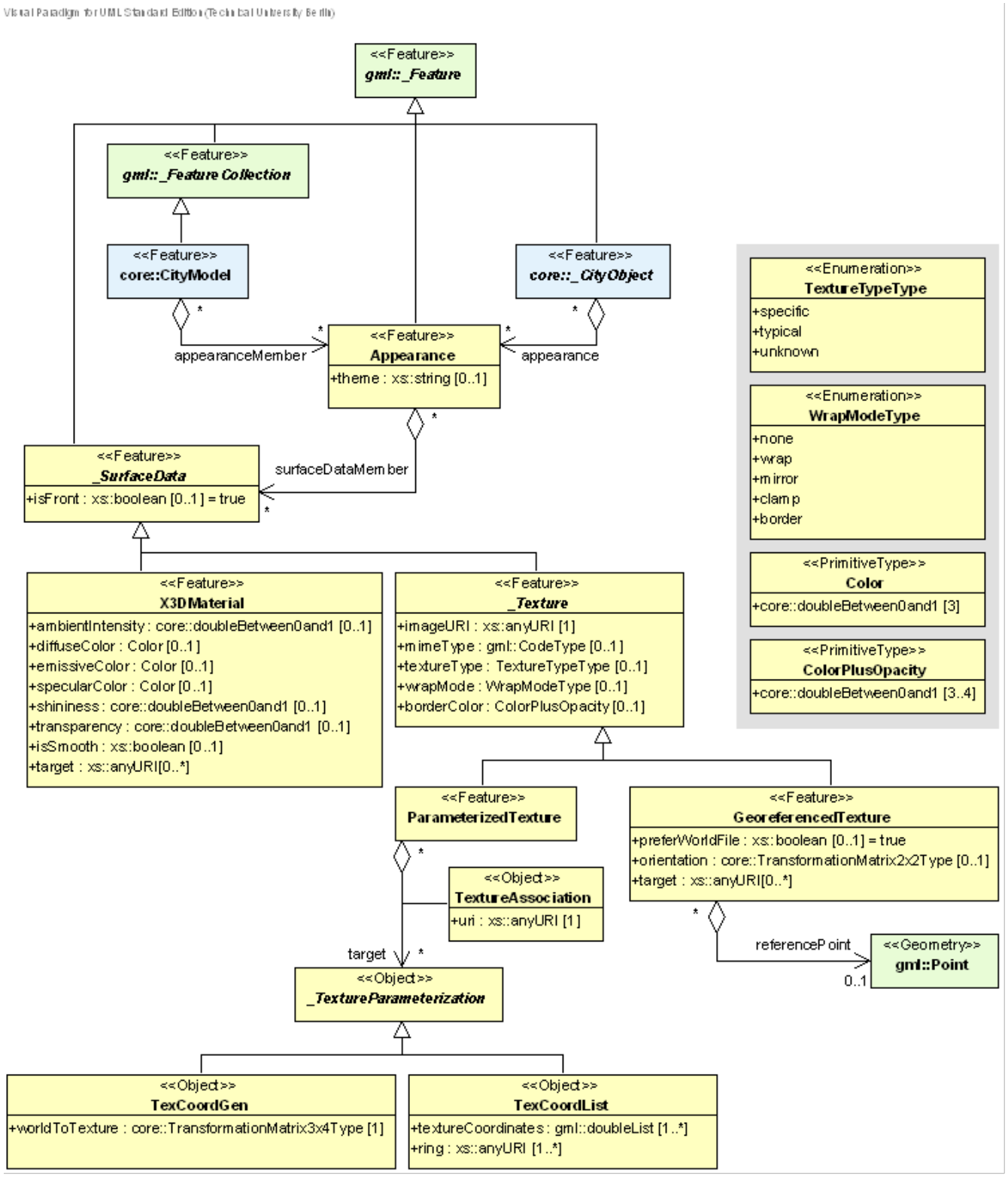


Figure 14. UML diagram of CityGML's appearance model. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Appearance module.

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

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

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

XML namespace

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

10.1. Relation between appearances, features and geometry

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

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

Each surface geometry object can have per theme at most one active front-facing material, one active back-facing material, one active front-facing texture, and one active back-facing texture. If

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

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

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

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

```
  24:               frontMaterial := surfaceData
  25:             }
  26:             if (surfaceData is backside texture AND frontTexture is empty) {
  27:               frontTexture := surfaceData
  28:             }
  29:           } else {
  30:             // only pick the first surfaceData for a particular category
  31:             if (surfaceData is frontside material AND frontMaterial is empty) {
  32:               frontMaterial := surfaceData
  33:             }
  34:             if (surfaceData is frontside texture AND frontTexture is empty) {
  35:               frontTexture := surfaceData
  36:             }
  37:             if (surfaceData is backside material AND backMaterial is empty) {
  38:               backMaterial := surfaceData
  39:             }
  40:             if (surfaceData is backside texture AND backTexture is empty) {
  41:               backTexture := surfaceData
  42:             }
  43:           }
  44:
  45:           // shortcut: could stop here if all 4 categories have been found
  46:         }
  47:       }
  48:     }
  49:     cObj := cObj.parent // this also includes the global CityModel
  50:   }
  51:   gmlSurface := gmlSurface.parent // this also includes a root gml:MultiSurface
  52:   if (gmlSurface isA gml:OrientableSurface AND gmlSurface.orientation is
negative) {
  53:     negate flip
  54:   }
  55: }
```

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

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

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

geometry) of a given ImplicitGeometry can differ between its occurrences. A consistent appearance results if all required surface data objects are placed in Appearance objects and the latter are stored either

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

10.2. Appearance and SurfaceData

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

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

10.2.1. AppearanceType, Appearance, AppearancePropertyType

NOTE insert UML

10.2.2. appearanceMember, appearance

NOTE insert UML

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

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

10.2.3. AbstractSurfaceDataType, _SurfaceData, SurfaceDataPropertyType

NOTE insert UML

10.3. Material

Materials define light reflection properties being constant for a whole surface geometry object. The definition of the class X3DMaterial is adopted from the X3D and COLLADA specification (cf. X3D, COLLADA specifica-tion). diffuseColor defines the color of diffusely reflected light. specularColor defines the color of a directed reflection. emissiveColor is the color of light generated by the surface. All colors use RGB values with red, green, and blue between 0 and 1. Transparency is defined separately using the transparency element where 0 stands for fully opaque and 1 for fully transparent. ambientIntensity defines the minimum percentage of dif-fuseColor that is visible regardless of light sources. shininess controls the sharpness of the specular highlight. 0 produces a soft glow while 1 results in a sharp highlight. isSmooth gives a hint for normal interpolation. If this boolean flag is set to true, vertex normals should be used for shading (Gouraud shading). Otherwise, normals should be constant for a surface patch (flat shading).

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

10.3.1. X3DMaterialType, X3DMaterial

NOTE	insert UML

10.4. Texture and texture mapping

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

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

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

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

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

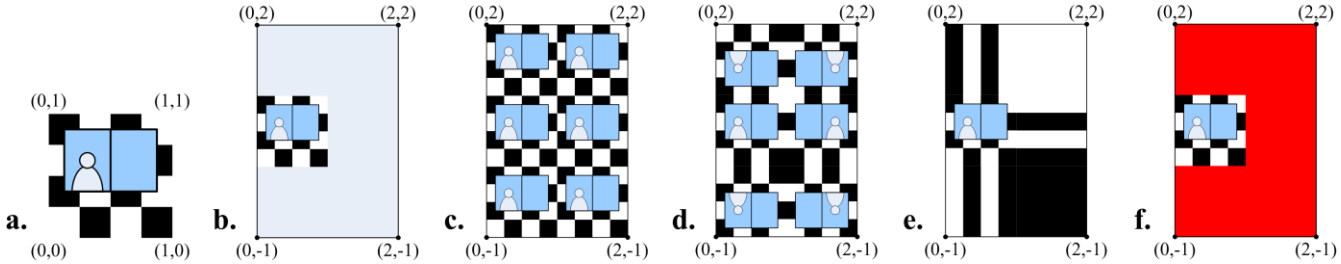


Figure 15. A texture (a) applied to a facade using different wrap modes: (b) none, (c) wrap, (d) mirror, (e) clamp and (f) border. The border color is red. The numbers denote texture coordinates (image: Hasso-Plattner-Institute).

10.4.1. AbstractTextureType, _Texture, WrapModeType, TextureTypeType

NOTE	insert UML

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

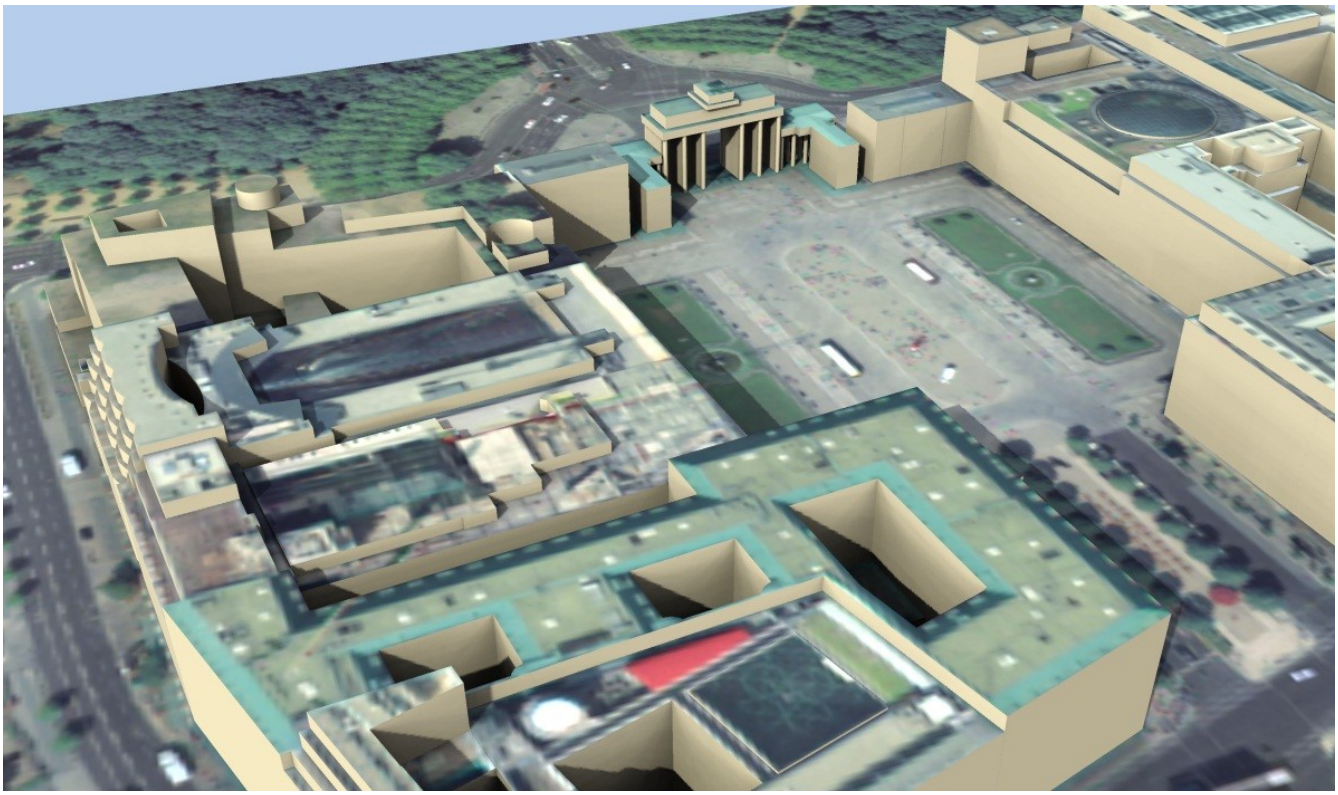


Figure 16. A georeferenced texture applied to ground and roof surfaces (source: Senate of Berlin, Hasso-Plattner-Institute).

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

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

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

NOTE insert equation

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

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

10.4.2. GeoreferencedTextureType, GeoreferencedTexture

NOTE insert UML

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

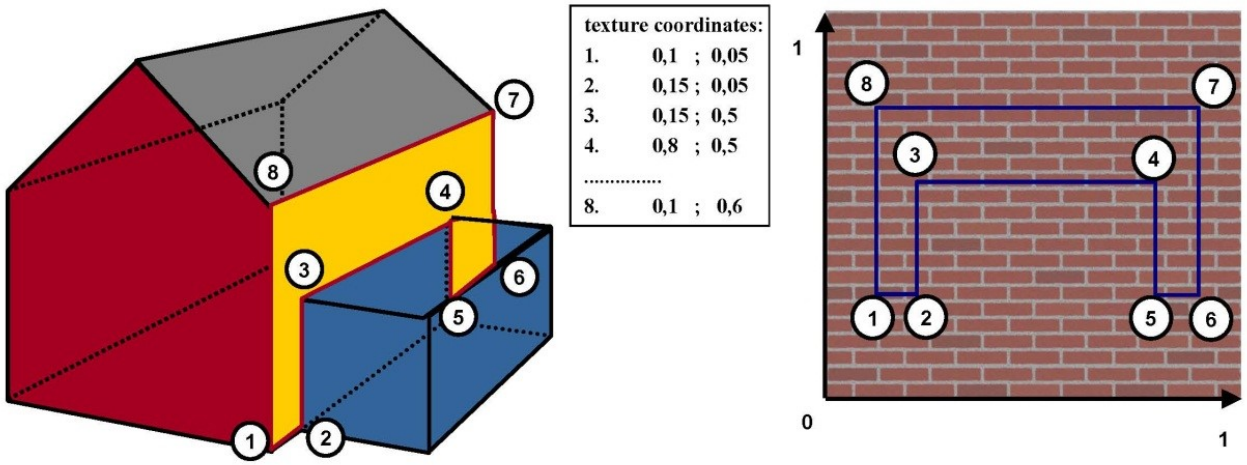


Figure 17. Positioning of textures using texture coordinates (image: IGG Uni Bonn).

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

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

NOTE combine figures 18

Figure 18. Projecting a photograph (a) onto multiple facades (b) using the worldToTexture transformation. The photograph does not cover the left facade completely. Thus, the texture appears to be clipped. Texture wrapping is set to "none" (source: Senate of Berlin, Hasso-Plattner-Institute).

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

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

NOTE insert transformation matrix

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

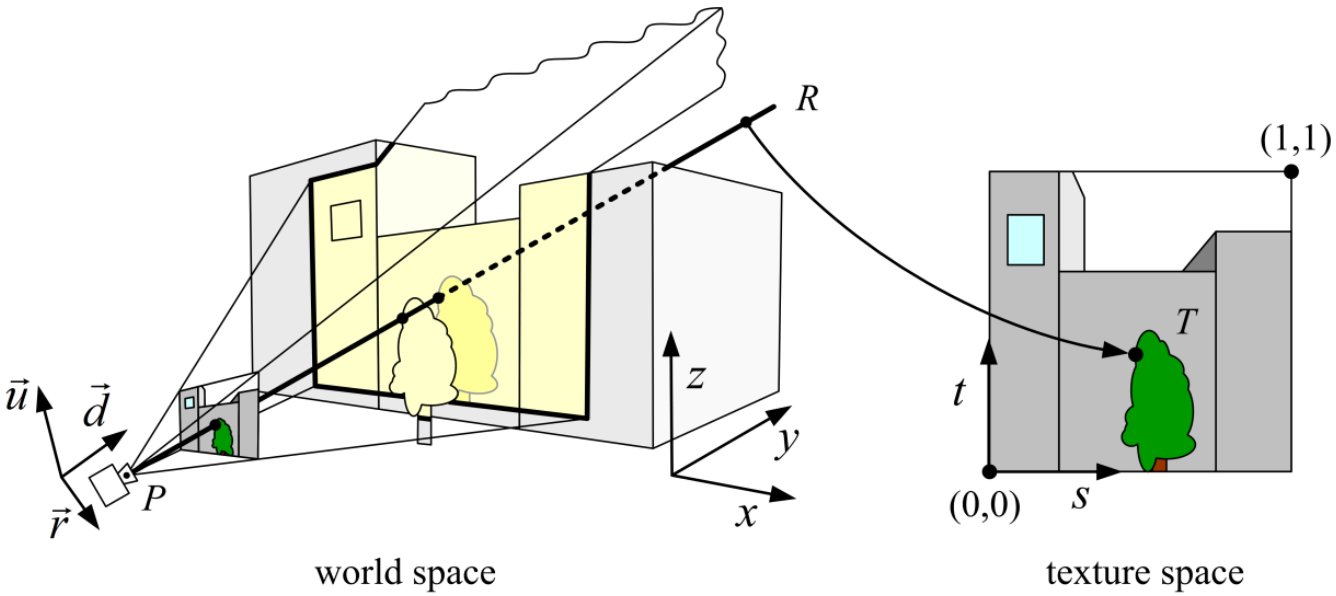


Figure 19. Projective texture mapping. All points on a ray R starting from the projection center P are mapped to the same point T in texture space (image: Hasso-Plattner-Institute, IGG TU Berlin).

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

NOTE insert formula

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

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

NOTE insert formula

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

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

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

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

NOTE insert formula

10.4.3. ParameterizedTextureType, ParameterizedTexture, TextureAssociationType

NOTE insert UML

10.4.4. AbstractTextureParameterizationType, TexCoordListType, TexCoordGenType

NOTE insert UML

10.5. Related concepts

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

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

10.6. Code lists

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

Chapter 11. CityGML Conceptual Model

This section provides a detailed discussion of the CityGML Conceptual Model.

11.1. Core

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

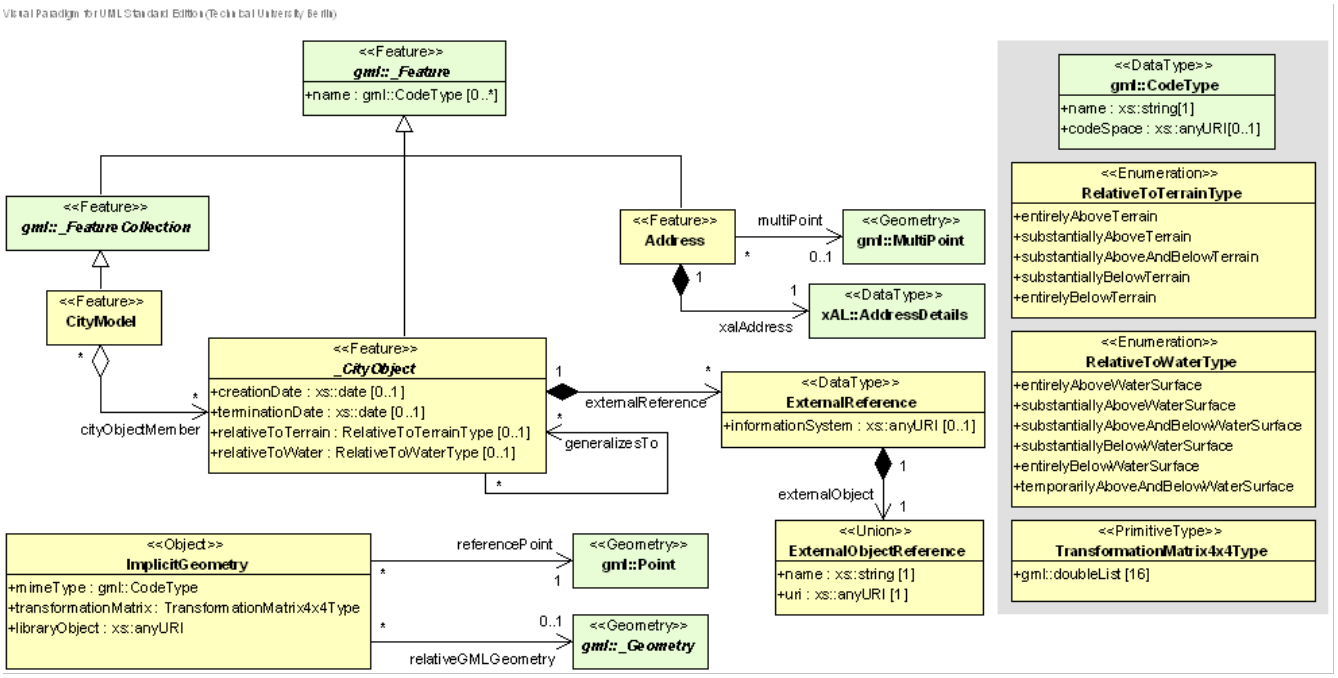


Figure 20. UML diagram of CityGML's core module. The bracketed numbers following the attribute names denote the attribute's multiplicity: the minimal and maximal number of occurrences of the attribute per object. For example, a name is optional (0) in the class _Feature or may occur multiple times (star symbol), while a _CityObject has none or at most one creationDate. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Core module.

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

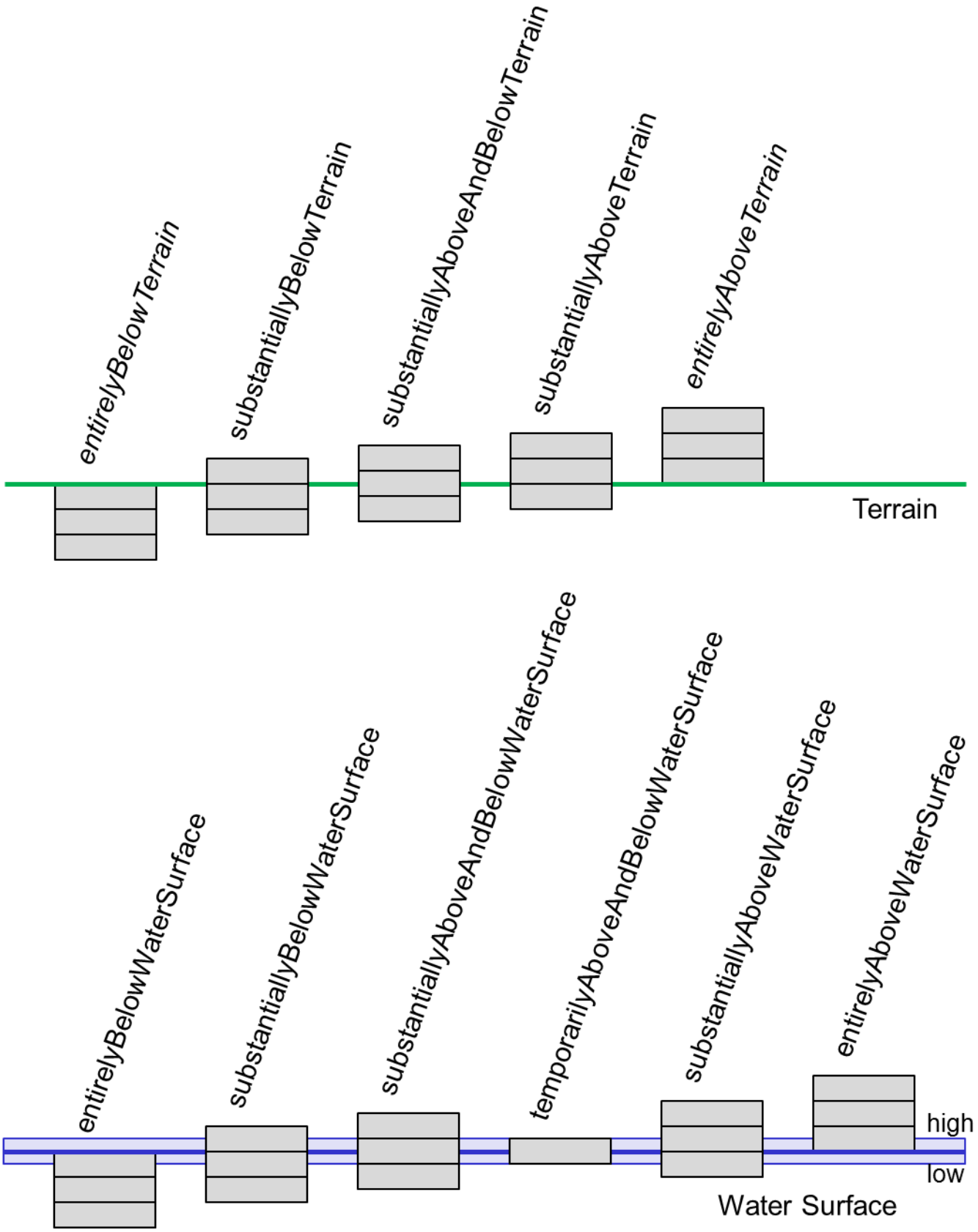


Figure 21. Topological relations of a CityGML object with respect to a) the terrain and b) the water surface.

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

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

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

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

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

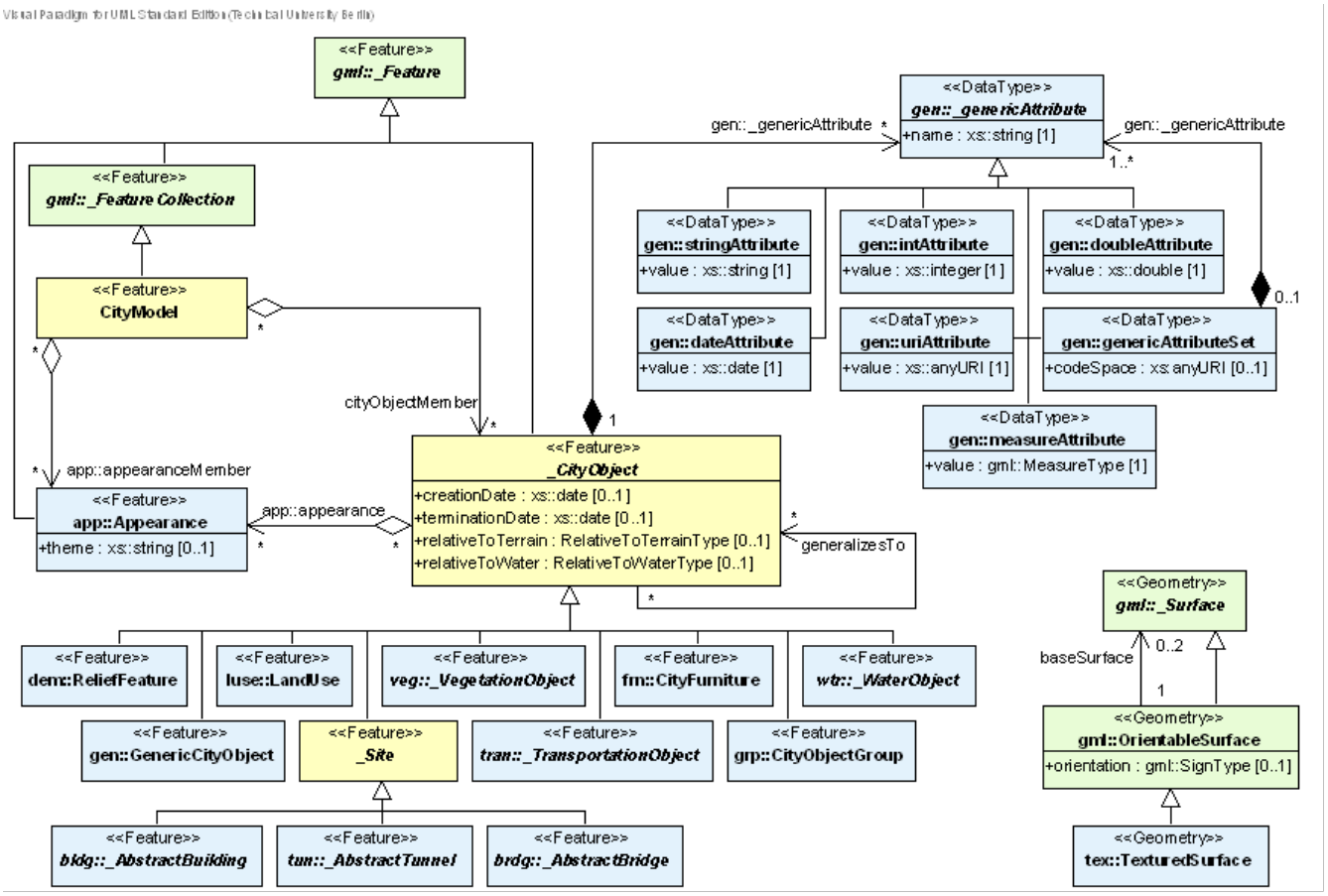


Figure 22. CityGML's top level class hierarchy. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Core module.

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

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

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

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

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

11.1.1. Base elements

AbstractCityObjectType, _CityObject

AbstractCityObjectType, _CityObject

NOTE	insert AbstractCityObjectType, _CityObject UML

CityModelType, CityModel

NOTE	insert CityModelType, CityModel UML

cityObjectMember

NOTE	insert cityObjectMember UML

AbstractSiteType, _Site

NOTE	insert AbstractSiteType, _Site UML

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

11.1.2. Generalisation relation, RelativeToTerrainType and RelativeToWaterType

GeneralizationRelationType

NOTE	insert GeneralizationRelationType UML

RelativeToTerrainType, RelativeToWaterType

NOTE	insert RelativeToTerrainType, RelativeToWaterType UML

11.1.3. External references

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

ExternalReferenceType, ExternalObjectReferenceType

NOTE	insert ExternalReferenceType, ExternalObjectReferenceType UML

11.1.4. Address information

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

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

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

AddressPropertyType, AddressType, Address

NOTE	insert AddressPropertyType, AddressType, Address UML

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

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

NOTE insert examples here if appropriate.

11.2. Digital Terrain Model

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

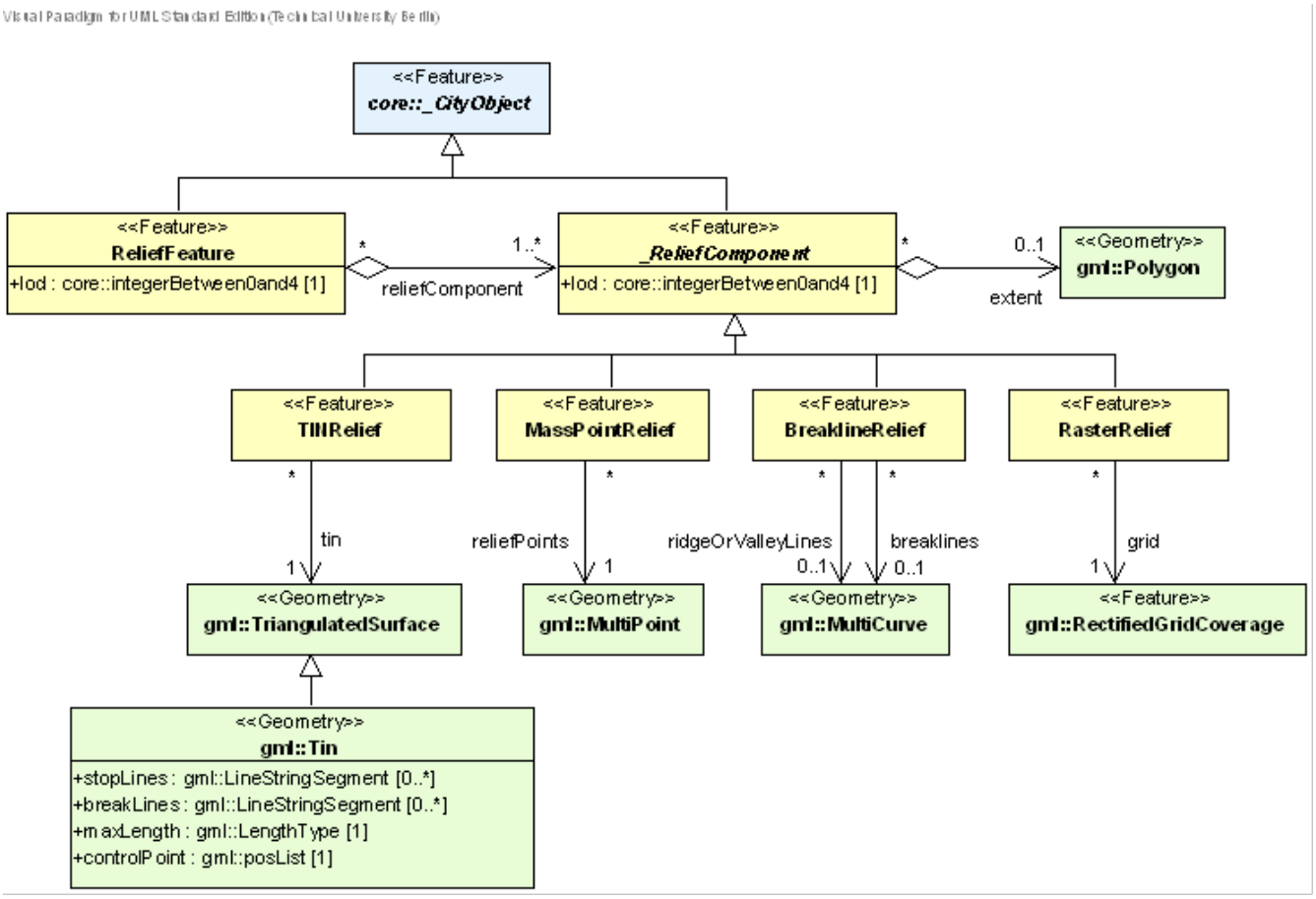


Figure 23. UML diagram of the Digital Terrain Model in CityGML. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Relief module.

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

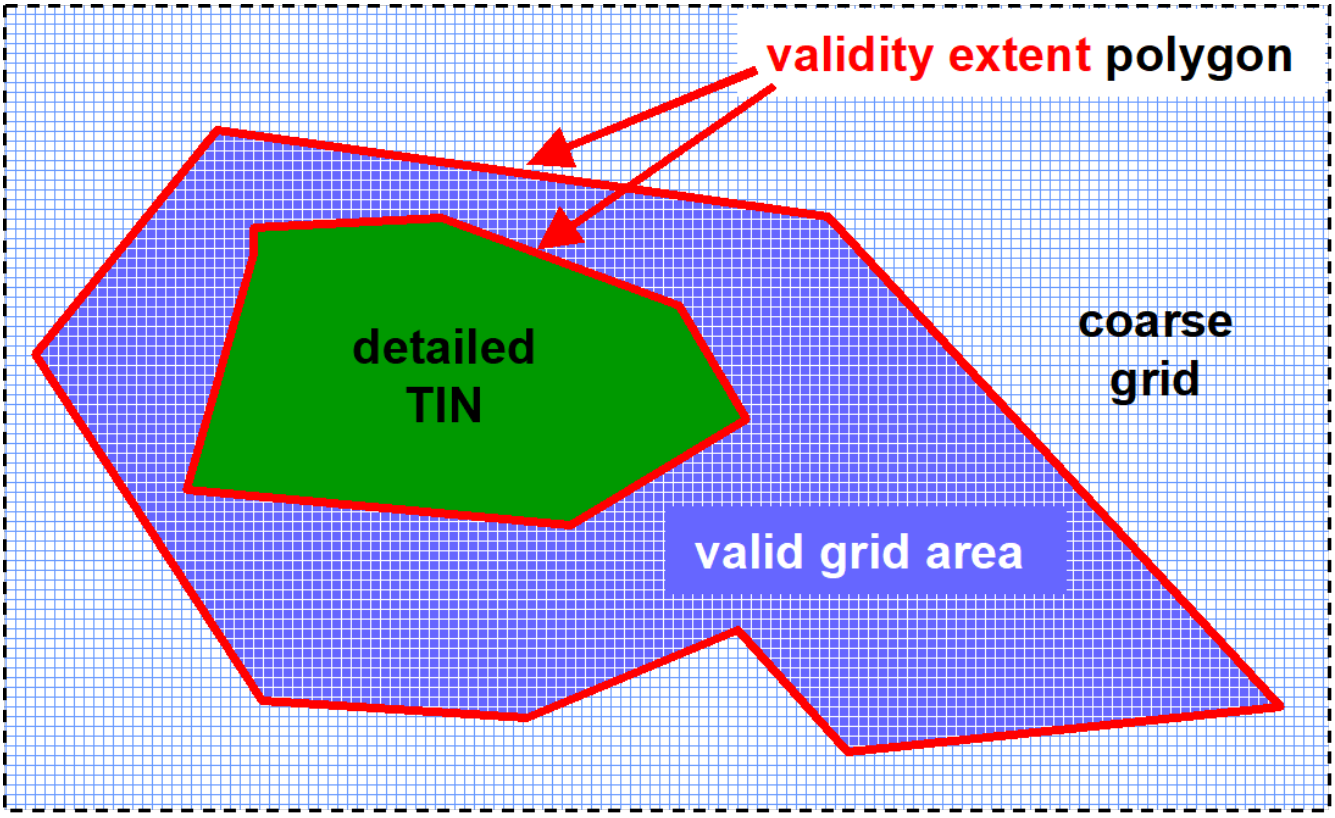


Figure 24. Nested DTMs in CityGML using validity extent polygons (graphic: IGG Uni Bonn).

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

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

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

11.2.1. Relief feature and relief component

ReliefFeatureType, ReliefFeature

NOTE	insert ReliefFeatureType, ReliefFeature UML

AbstractReliefComponentType, _ReliefComponent

NOTE	insert AbstractReliefComponentType, _ReliefComponent UML

11.2.2. TIN relief

TINReliefType, TINRelief

NOTE	insert TINReliefType, TINRelief UML

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

11.2.3. Raster relief

RasterReliefType, RasterRelief, Elevation

NOTE	insert RasterReliefType, RasterRelief, Elevation UML

11.2.4. Mass point relief

MassPointReliefType, MassPointRelief

NOTE	insert MassPointReliefType, MassPointRelief UML

11.2.5. Breakline relief

BreaklineReliefType, BreaklineRelief

NOTE	insert BreaklineReliefType, BreaklineRelief UML

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

11.3. Building Model

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

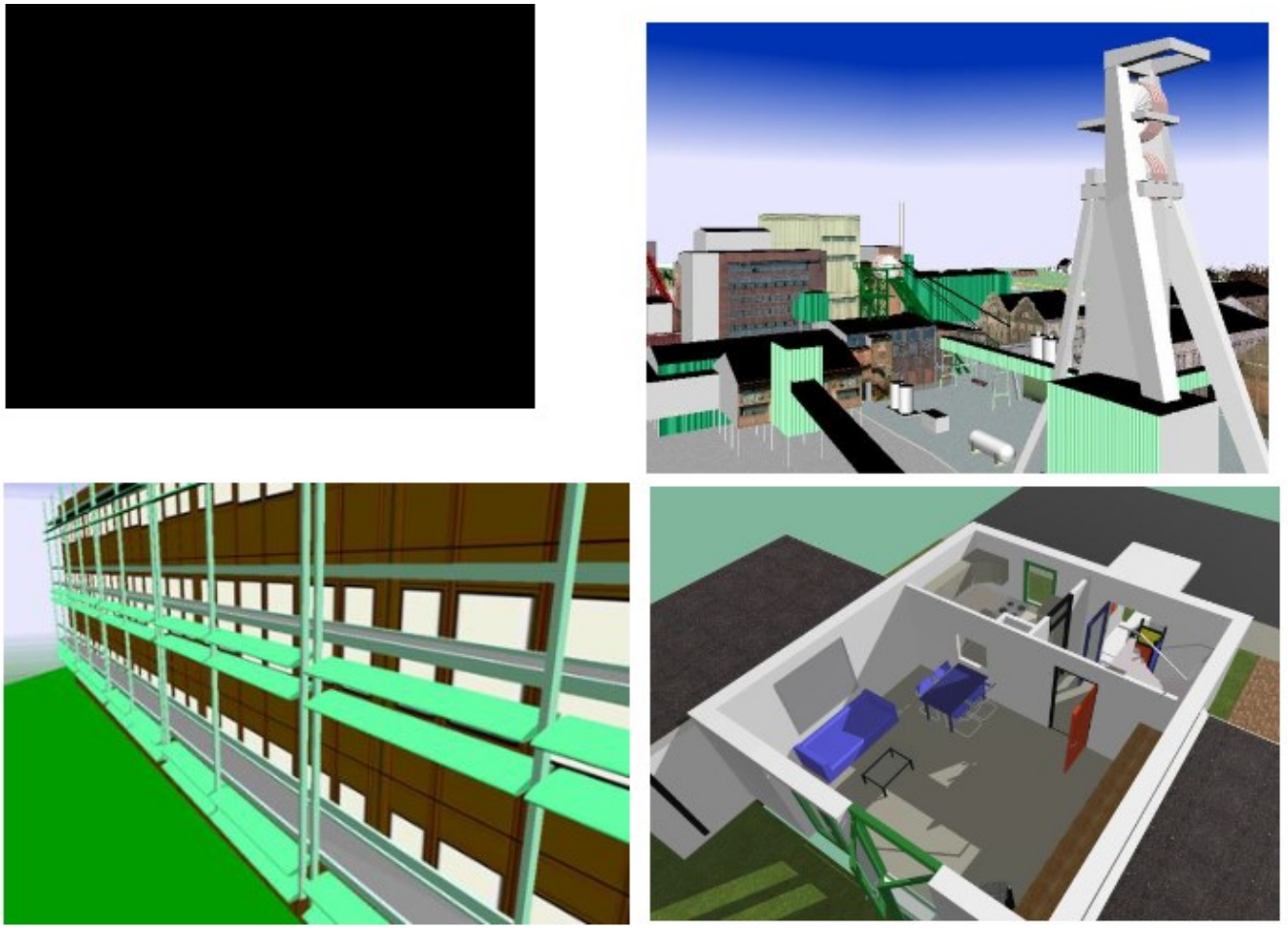


Figure 25. Examples for city or building models in LOD1 (upper left), LOD2 (upper right), LOD3 (lower left), and LOD4 (lower right) (source: District of Recklinghausen, m-g-h ingenieure+architekten GmbH).

NOTE this figure needs to be re-built.

The UML diagram of the building model is depicted in Fig. 27, for the XML schema definition see annex A.4 and below. The pivotal class of the model is _AbstractBuilding, which is a subclass of the thematic class _Site (and transitively of the root class _CityObject). _AbstractBuilding is specialised either to a Building or to a BuildingPart. Since an _AbstractBuilding consists of BuildingParts, which again are _AbstractBuildings, an aggregation hierarchy of arbitrary depth may be realised. As subclass of the root class _CityObject, an _AbstractBuilding inherits all properties from _CityObject like the GML3 standard feature properties (gml:name etc.) and the CityGML specific properties like ExternalReferences (cf. chapter 6.7). Further properties not explicitly covered by _AbstractBuilding may be modelled as generic attributes provided by the CityGML Generics module (cf. chapter 10.12) or using the CityGML Application Domain Extension mechanism (cf. chapter 10.13).

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

Both classes Building and BuildingPart inherit the attributes of _AbstractBuilding: the class of the building, the function (e.g. residential, public, or industry), the usage, the year of construction, the year of demolition, the roof type, the measured height, and the number and individual heights of the storeys above and below ground. This set of parameters is suited for roughly reconstructing the

three-dimensional shape of a building and can be provided by cadastral systems. Furthermore, Address features can be assigned to Buildings or BuildingParts.

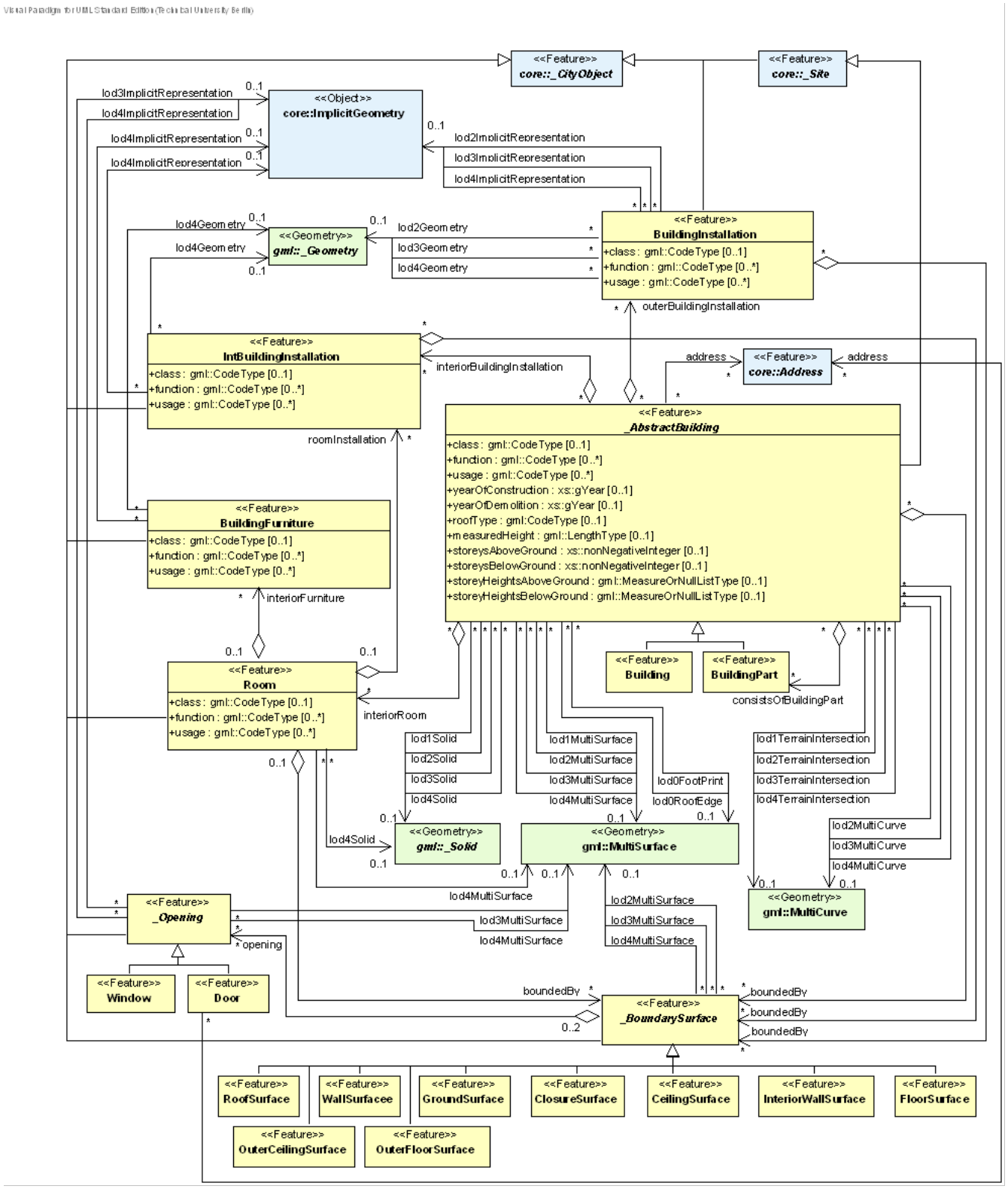


Figure 26. UML diagram of CityGML's building model. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Building module.

The geometric representation and semantic structure of an _AbstractBuilding is shown in Fig. 27. The model is successively refined from LOD0 to LOD4. Therefore, not all components of a building model are represented equally in each LOD and not all aggregation levels are allowed in each LOD. In CityGML, all object classes are associated to the LODs with respect to the proposed minimum

acquisition criteria for each LOD (cf. chapter 6.2). An object can be represented simultaneously in different LODs by providing distinct geometries for the corresponding LODs.

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

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

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

The visible surface of a room is represented geometrically as a Solid or MultiSurface. Semantically,

the surface can be structured into specialised _BoundarySurfaces, representing floor (FloorSurface), ceiling (CeilingSur-face), and interior walls (InteriorWallSurface). Room furniture, like tables and chairs, can be represented in the CityGML building model with the class BuildingFurniture. A BuildingFurniture may have the attributes class, function and usage. Annexes G.1 to G.6 provide example CityGML documents containing a single building model which is subsequently refined from a coarse LOD0 representation up to a semantically rich and geomet-ric-topologically sound LOD4 model including the building interior.

11.3.1. Building and Building Part

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

BuildingType, Building

NOTE	Insert BuildingType, Building UML

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

BuildingType, Building Part

NOTE	Insert BuildingType, Building Part UML

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

Figure 27. Examples of buildings consisting of one and two building parts (source: City of Coburg)

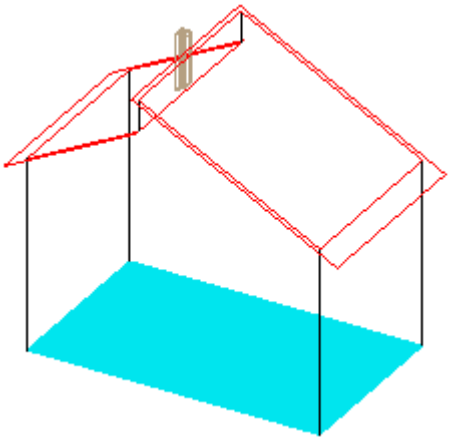
AbstractBuildingType, _AbstractBuilding

NOTE	AbstractBuildingType, _AbstractBuilding UML

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

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

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



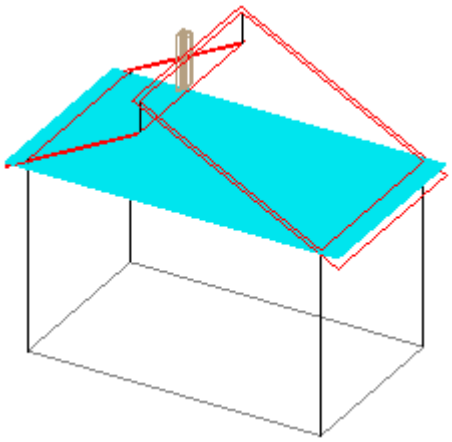


Figure 28. The two possibilities of modeling a building in LOD0 using horizontal 3D surfaces. On the left, the building footprint (lod0FootPrint) is shown (cyan) which denotes the shape of the building on the ground. The corresponding surface representation is located at ground level. On the right, the lod0RoofEgde representation is illustrated (cyan) which results from a horizontal projection of the building's roof and which is located at the eaves height (source: Karlsruhe Institute of Technology (KIT), courtesy of Franz-Josef Kaiser).

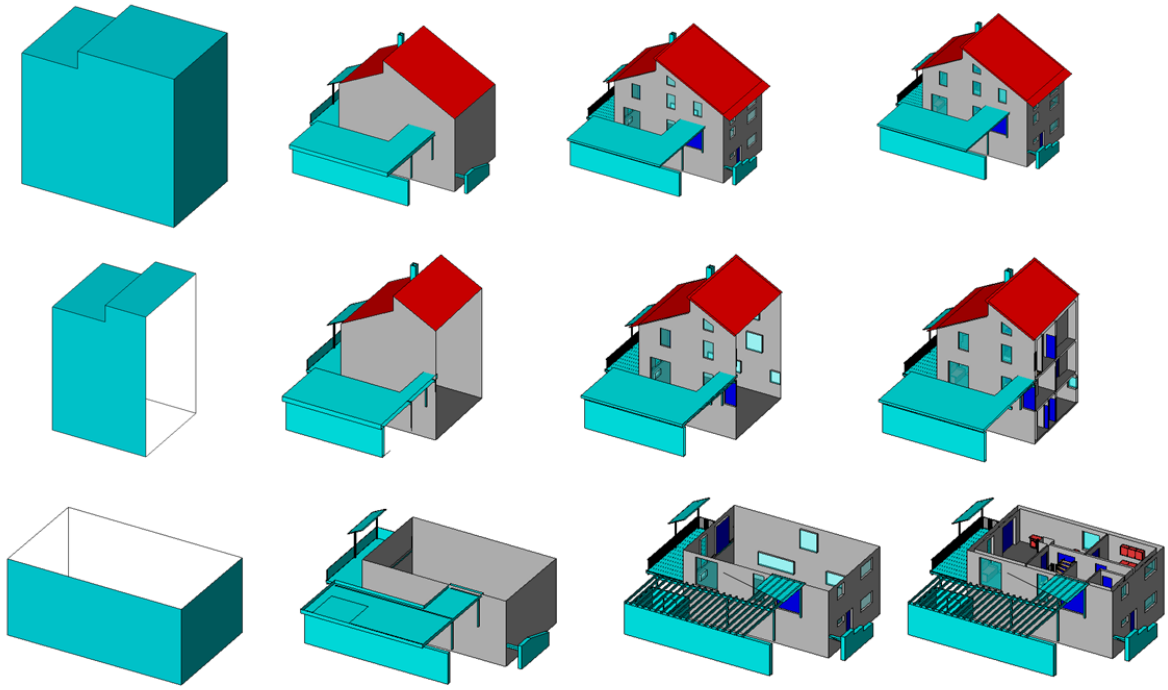


Figure 29. Building model in LOD1 – LOD4 (source: Karlsruhe Institute of Technology (KIT), courtesy of Franz-Josef Kaiser).<o:p></o:p>

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

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

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

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

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

Table 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).

11.3.2. Outer building installations

BuildingInstallationType, BuildingInstallation

Note: insert BuildingInstallation UML

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

11.3.3. Boundary surfaces

AbstractBoundarySurfaceType, _BoundarySurface

NOTE	Insert AbstractBoundarySurfaceType, _BoundarySurface UML

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

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

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

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

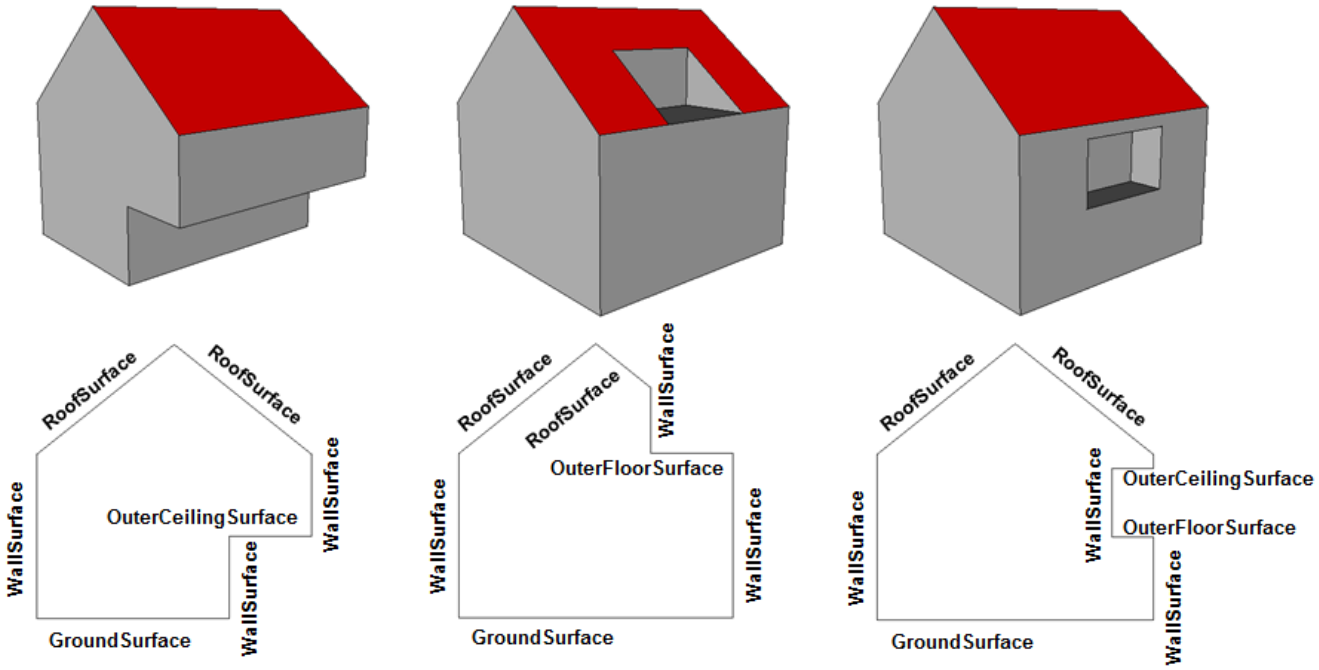
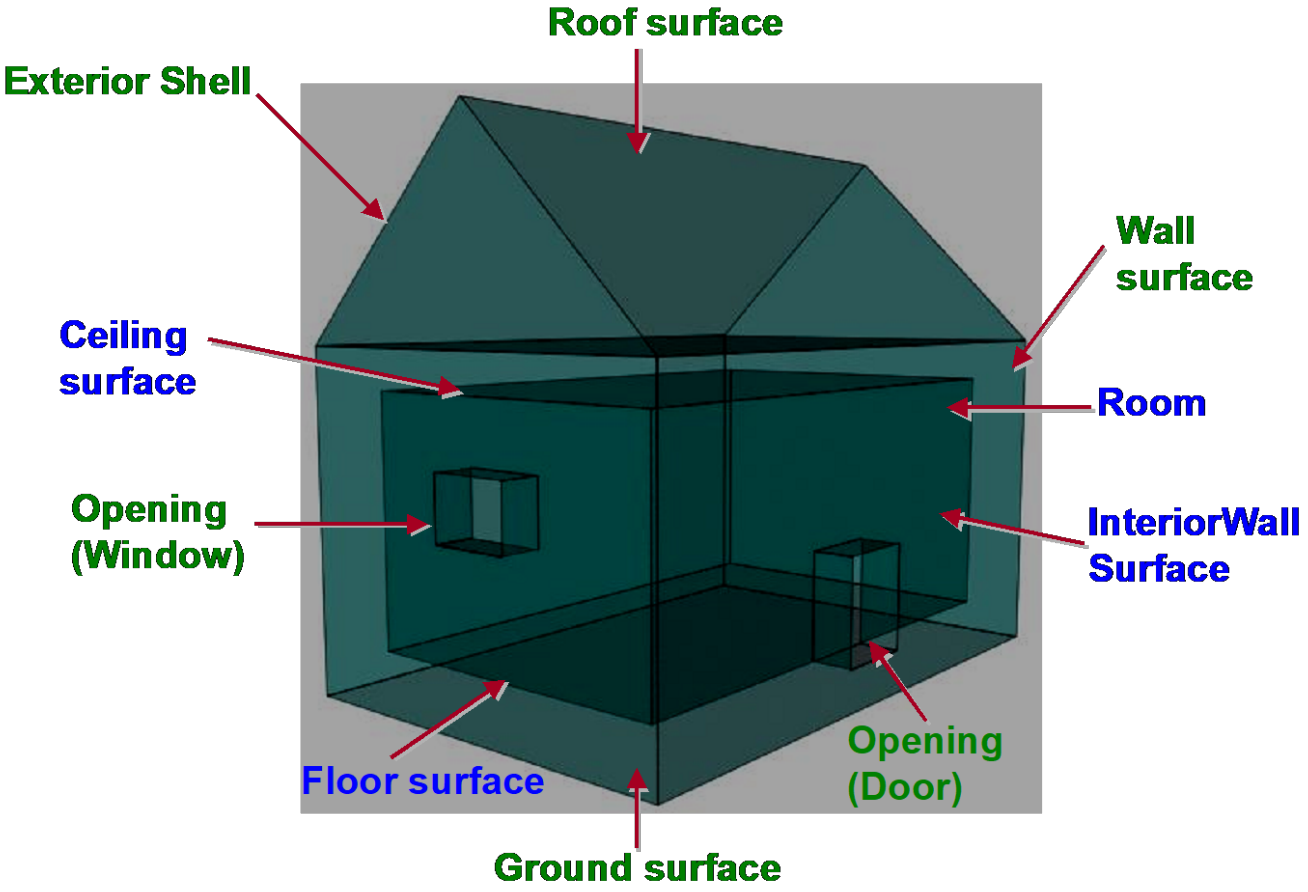


Figure 30. Examples of the classification of _BoundarySurfaces of the outer building shell (source: Karlsruhe Institute of Technology (KIT))


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

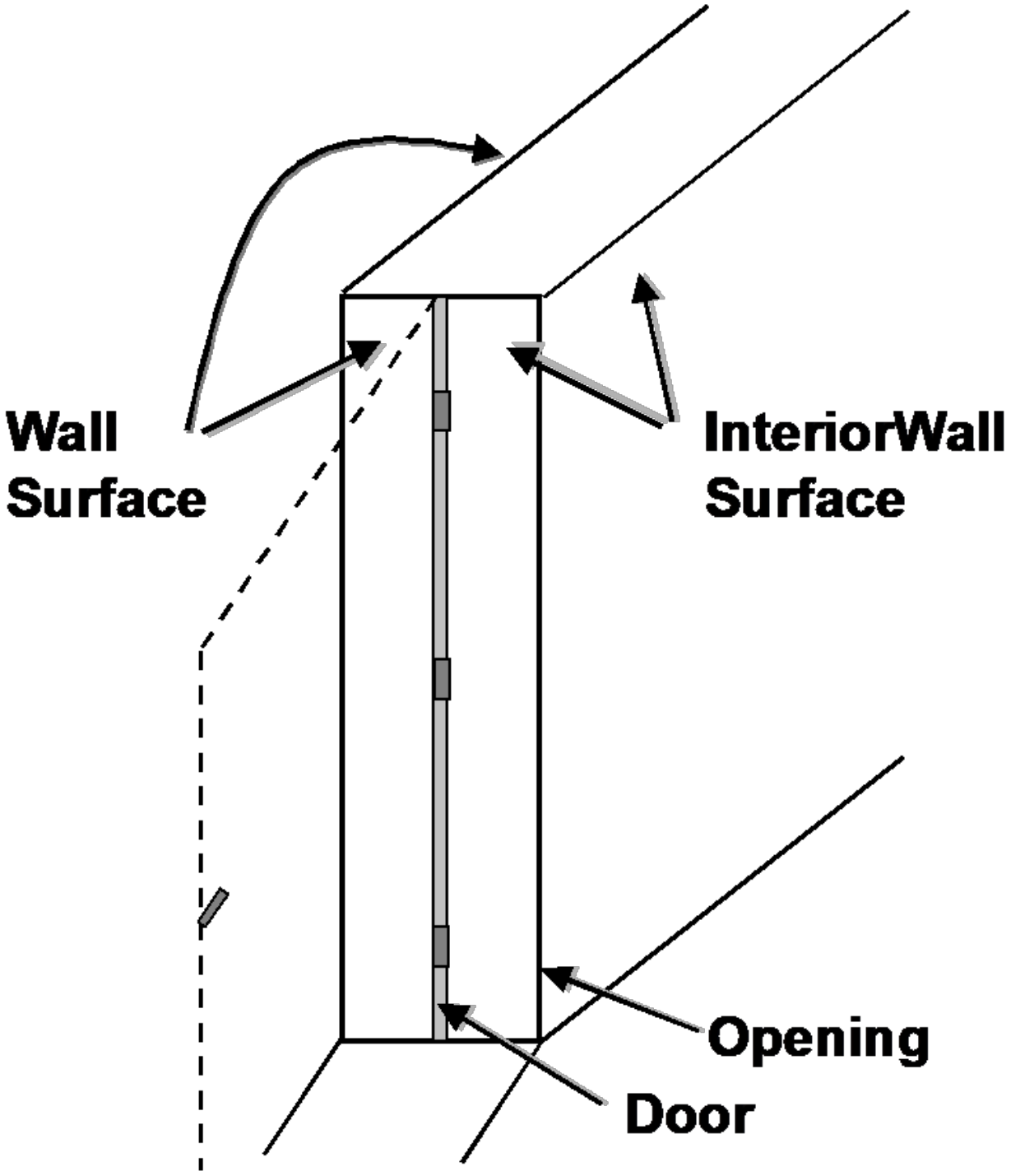


Figure 31. Classification of BoundarySurfaces (left), in particular for Openings (right) (graphic: IGG Uni Bonn).

GroundSurfaceType, GroundSurface

NOTE	insert GroundSurfaceType, GroundSurface uml

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

OuterCeilingSurfaceType, OuterCeilingSurface

NOTE insert OuterCeilingSurfaceType, OuterCeilingSurface UML

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

WallSurfaceType, WallSurface

NOTE insert WallSurfaceType, WallSurface UML

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

OuterFloorSurfaceType, OuterFloorSurface

NOTE insert OuterFloorSurfaceType, OuterFloorSurface UML

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

RoofSurfaceType, RoofSurface

NOTE insert RoofSurfaceType, RoofSurface UML

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

ClosureSurfaceType, ClosureSurface

NOTE insert ClosureSurfaceType, ClosureSurface UML

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

FloorSurfaceType, FloorSurface

NOTE insert FloorSurfaceType, FloorSurface UML

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

InteriorWallSurfaceType, InteriorWallSurface

NOTE	insert InteriorWallSurfaceType, InteriorWallSurface UML

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

CeilingSurfaceType, CeilingSurface

NOTE	Insert CeilingSurfaceType, CeilingSurface UML

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

11.3.4. Openings

AbstractOpeningType, _Opening

NOTE	insert AbstractOpeningType, _Opening UML

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

WindowType, Window

NOTE	insert WindowType, Window UML

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

DoorType, Door

NOTE	insert DoorType, Door UML

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

11.3.5. Building Interior

RoomType, Room

NOTE	insert RoomType, Room UML

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

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

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

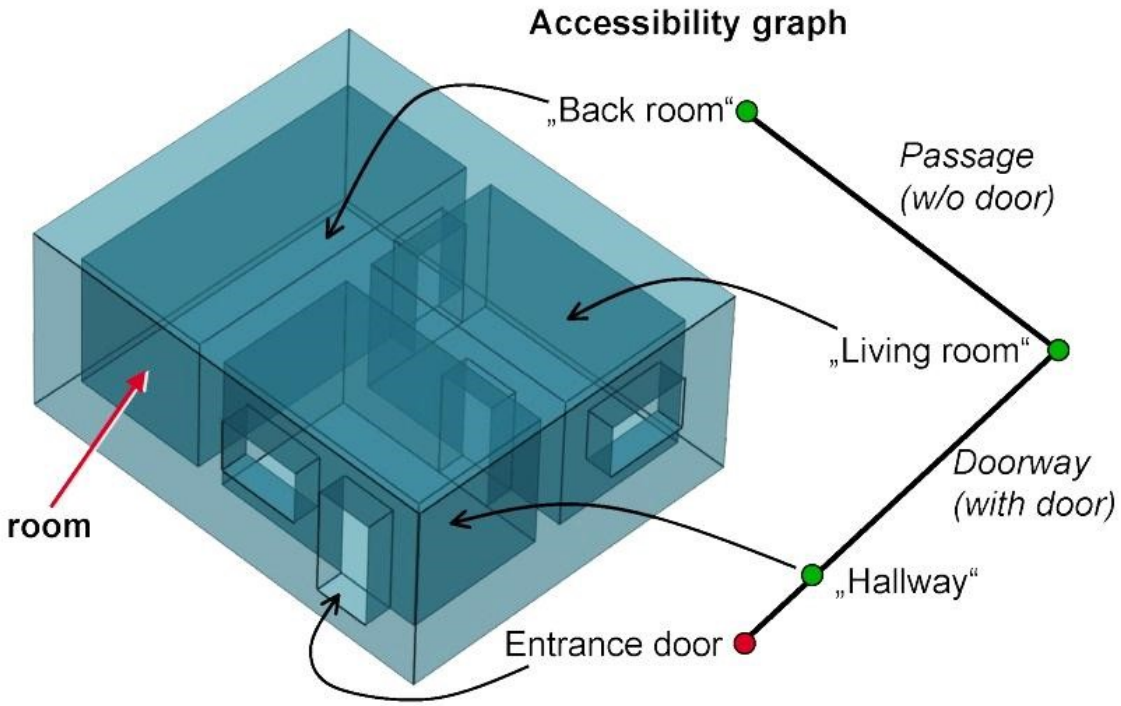


Figure 32. Accessibility graph derived from topological adjacencies of room surfaces (graphic: IGG Uni Bonn).

BuildingFurnitureType, BuildingFurniture

NOTE	insert BuildingFurnitureType, BuildingFurniture UML

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

IntBuildingInstallationType, IntBuildingInstallation

NOTE	insert IntBuildingInstallationType, IntBuildingInstallation UML

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

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

11.3.6. Modelling building storeys using CityObjectGroups

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

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

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

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

11.3.7. Examples

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

Figure 33. 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 34. Textured LOD2 model of the Bernhardus church in Karlsruhe (source: Karlsruhe Institute of Technology (KIT), courtesy of City of Karlsruhe).

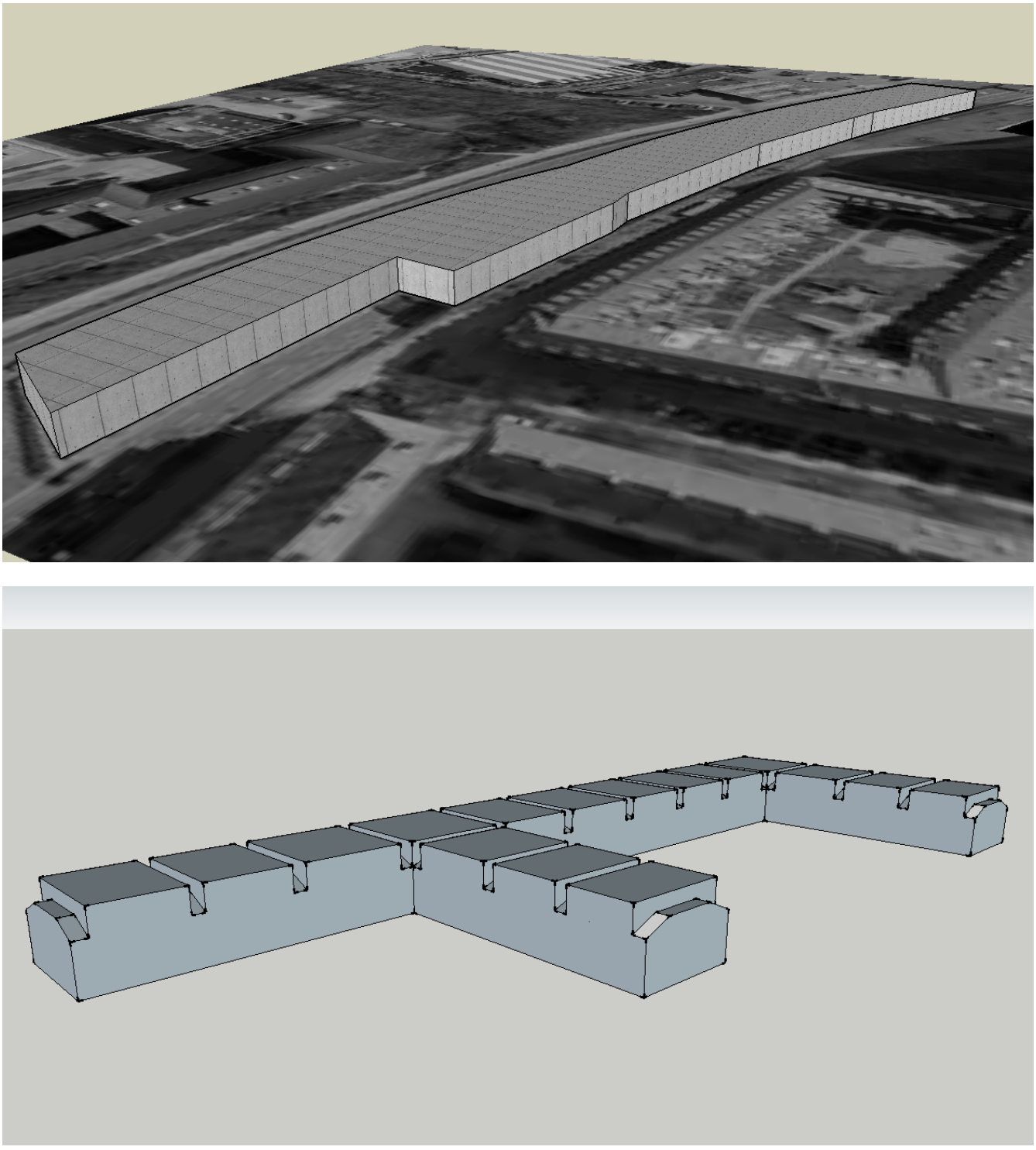
The model shown in Fig. 36 was derived from a 3D CAD model generated during the planning phase of the building. On the left side of Fig. 36, the building is shown whereas on the right side the LOD3 model is present-ed. The building itself is bounded by wall surfaces, roof surfaces and a ground surface. 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 35. 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).

11.4. Tunnel Model

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



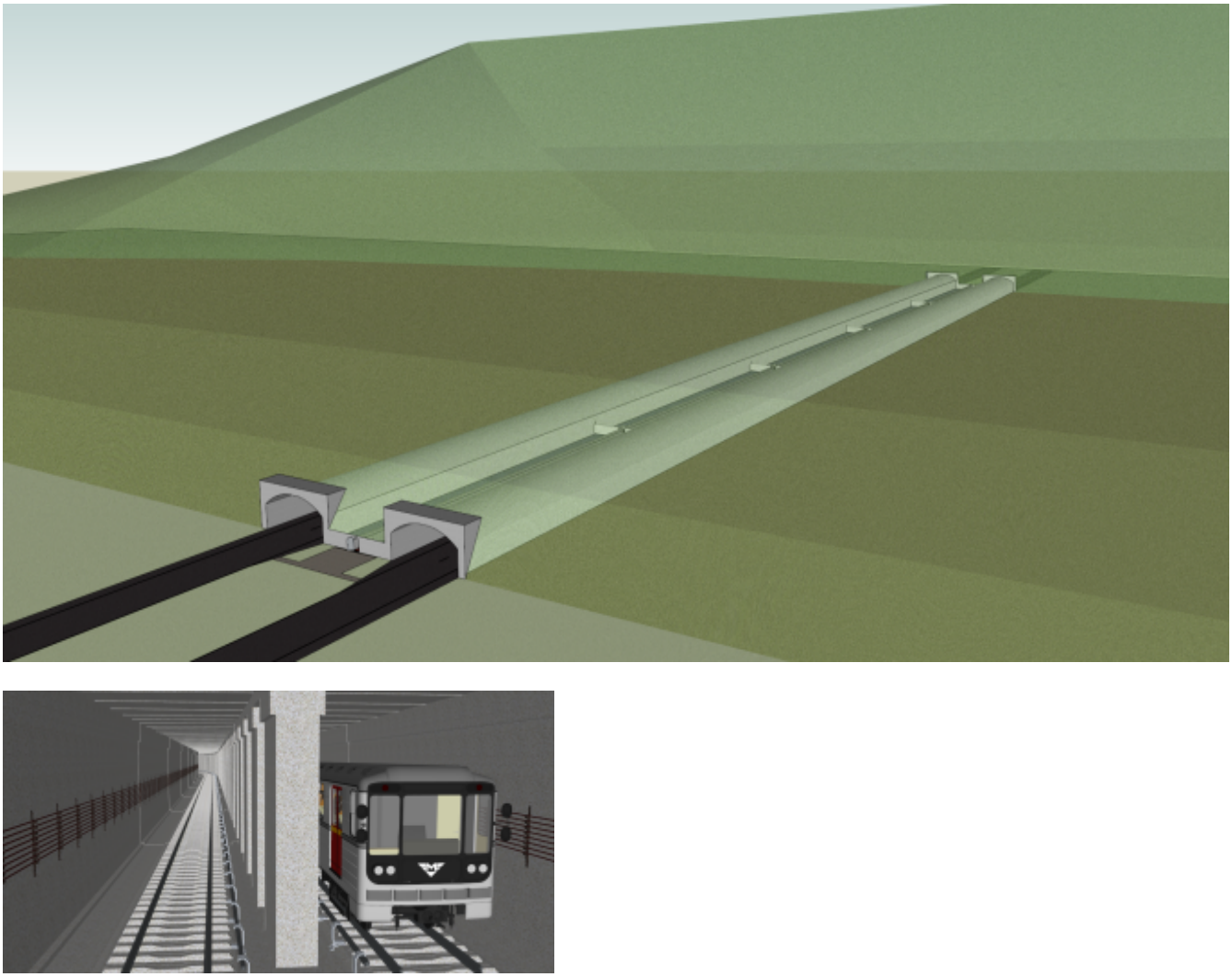


Figure 36. Examples for tunnel models in LOD1 (upper left), LOD2 (upper right), LOD3 (lower left) and LOD4 (lower right) (source: Google 3D warehouse).

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

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

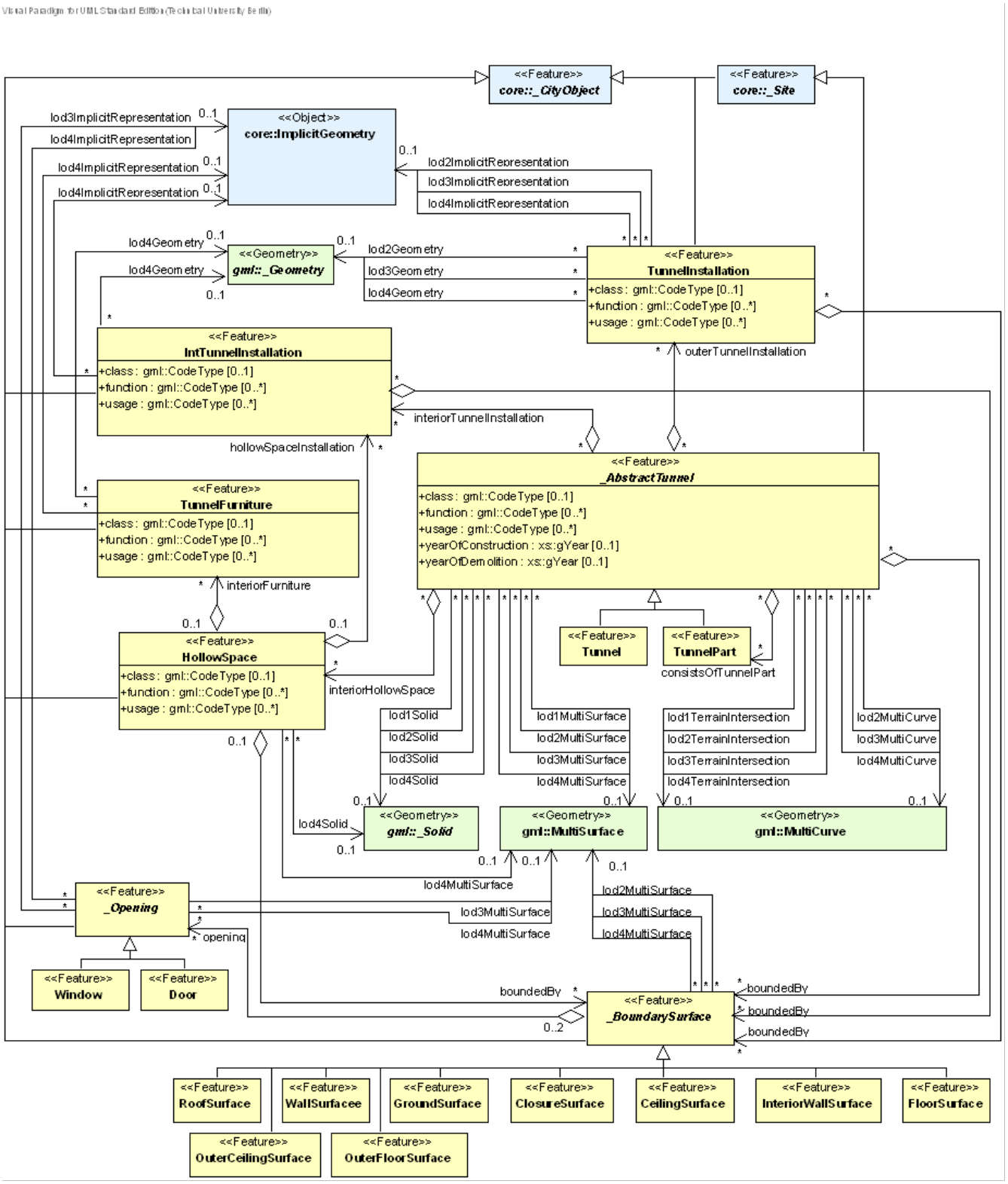


Figure 37. UML diagram of CityGML's tunnel model. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Tunnel module.

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

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

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

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

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

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

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

11.4.1. Tunnel and Tunnel Part

TunnelType, Tunnel

NOTE	insert TunnelType, Tunnel UML

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

TunnelPartType, TunnelPart

NOTE	insert TunnelPartType, TunnelPart UML

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

Figure 38. Example of a tunnel modeled with two tunnel parts (source: Helmut Stracke).

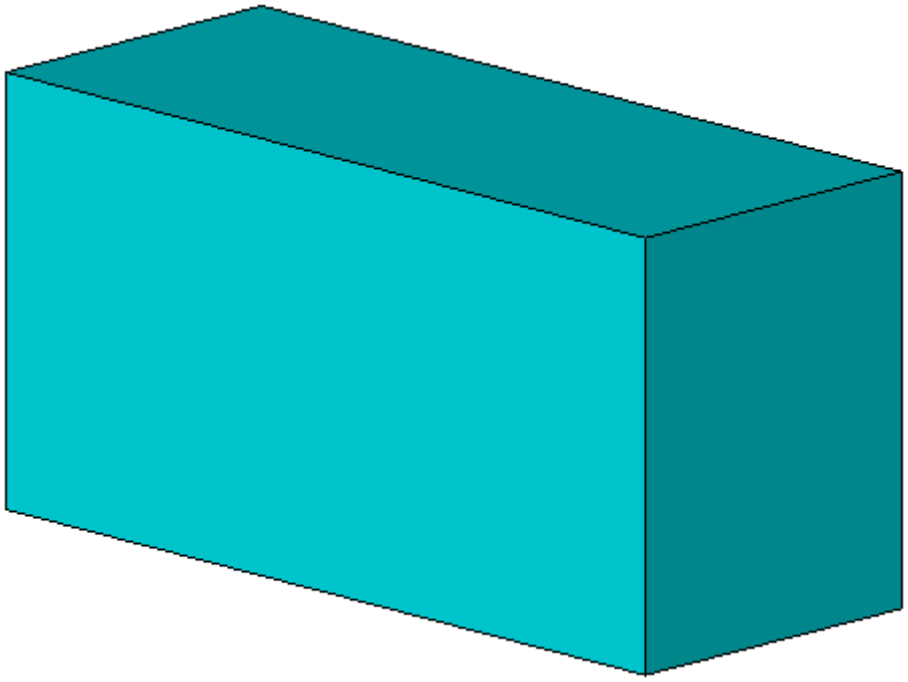
AbstractTunnelType, _AbstractTunnel

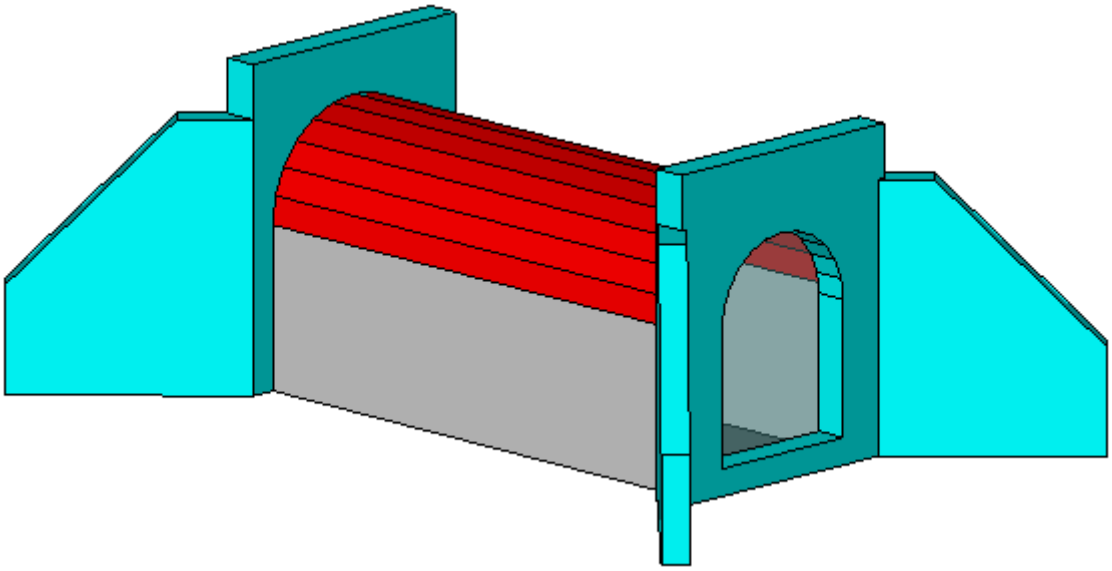
NOTE	insert AbstractTunnelType, _AbstractTunnel UML

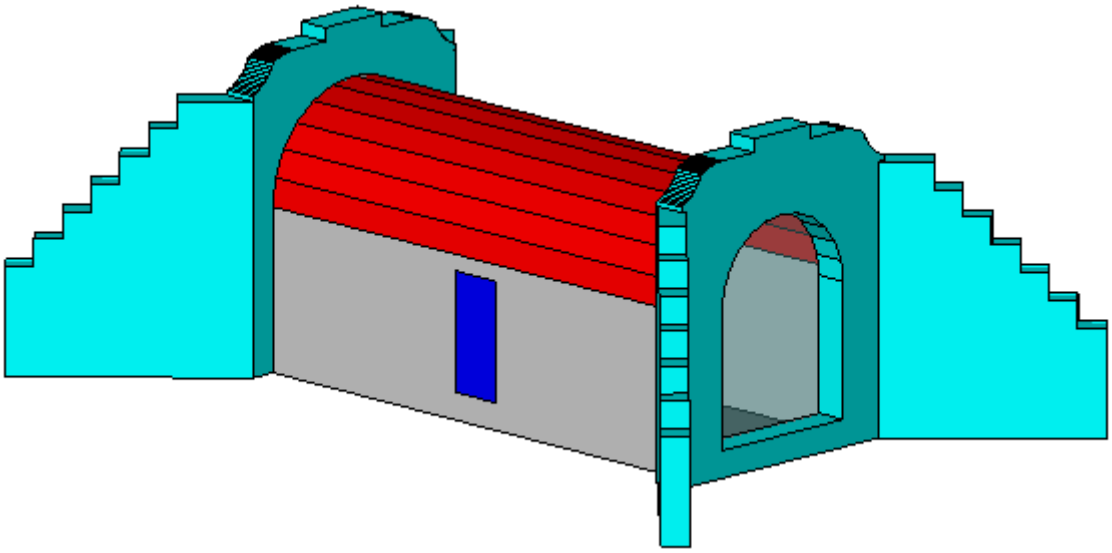
The abstract class _AbstractTunnel contains properties for tunnel attributes, purely geometric representations, and geometric/semantic representations of the tunnel or tunnel part in different levels of detail. The attributes describe:

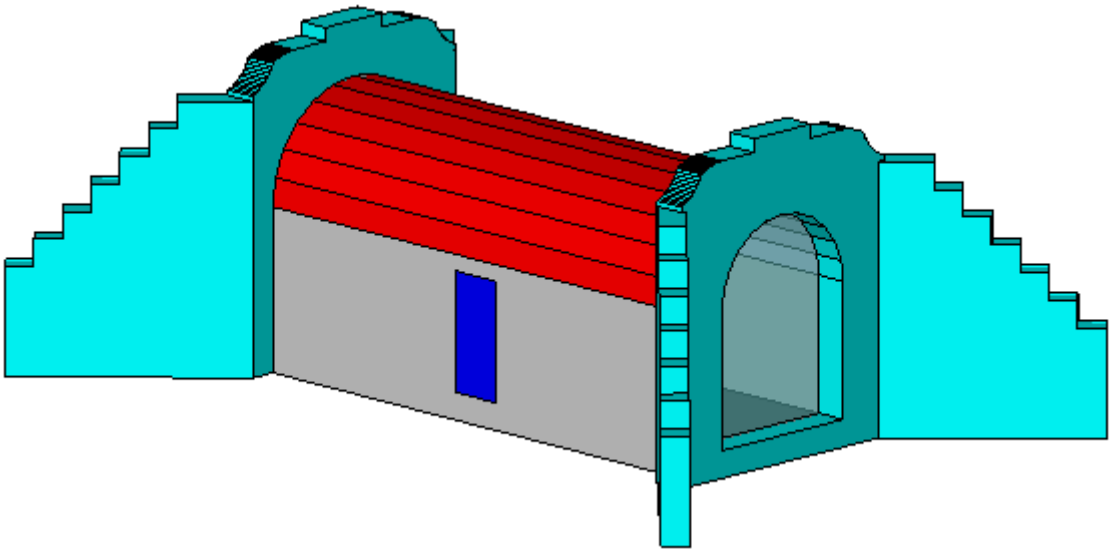
1. The classification of the tunnel or tunnel part (class), the different functions (function), and the usage (us-age). The type of these attributes is gml:CodeType and the values can be specified in separate code lists.
2. The year of construction (yearOfConstruction) and the year of demolition (yearOfDemolition)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

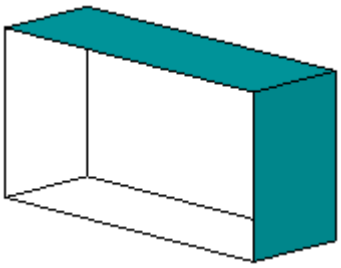
Spanning the different levels of detail, the tunnel model differs in the complexity and granularity of the geomet-ric representation and the thematic structuring of the model into components with a special semantic meaning. This is illustrated in Fig. 40, showing the same tunnel in four different LODs.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

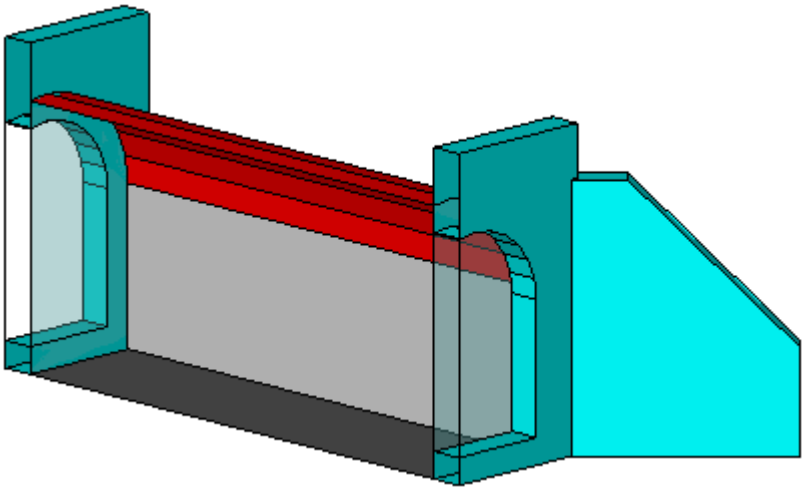


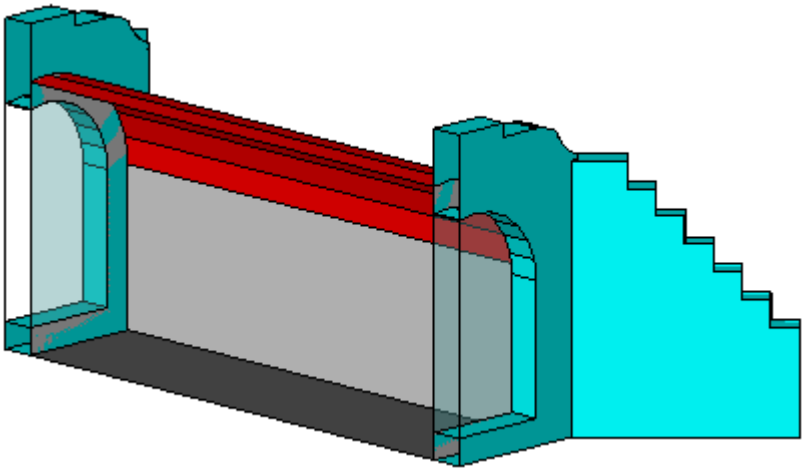


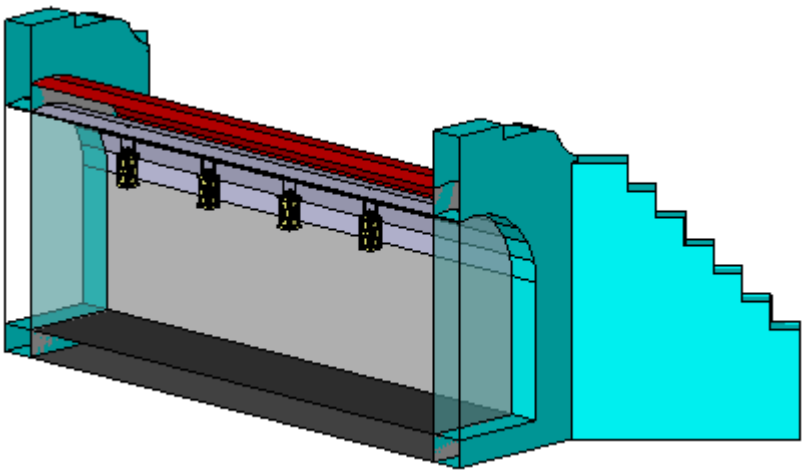












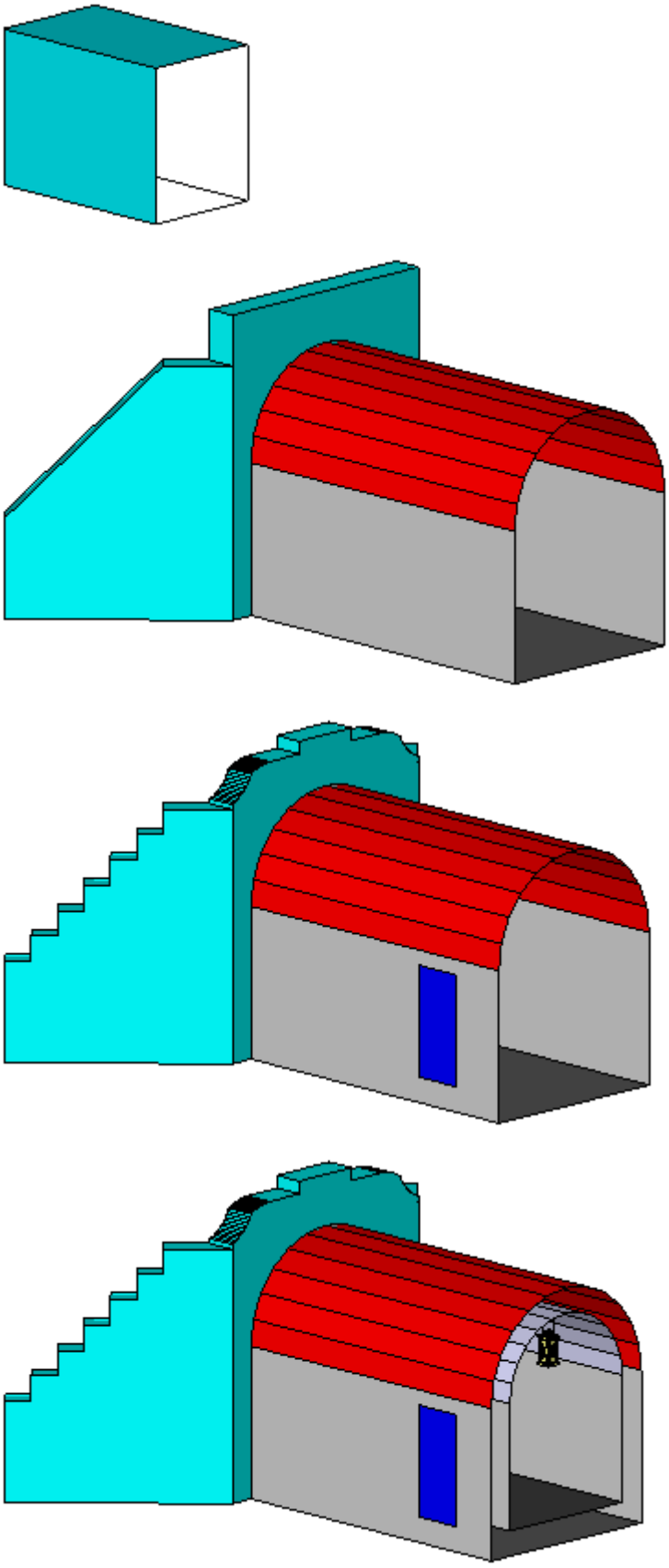


Figure 39. Tunnel model in LOD1 – LOD4 (source: Karlsruhe Institute of Technology (KIT)).

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

Table 7. Semantic themes of the class _AbstractTunnel

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

11.4.2. Outer Tunnel Installations

TunnelInstallationType, TunnelInstallation

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

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

11.4.3. Boundary surfaces

NOTE	insert Boundary surfaces UML

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

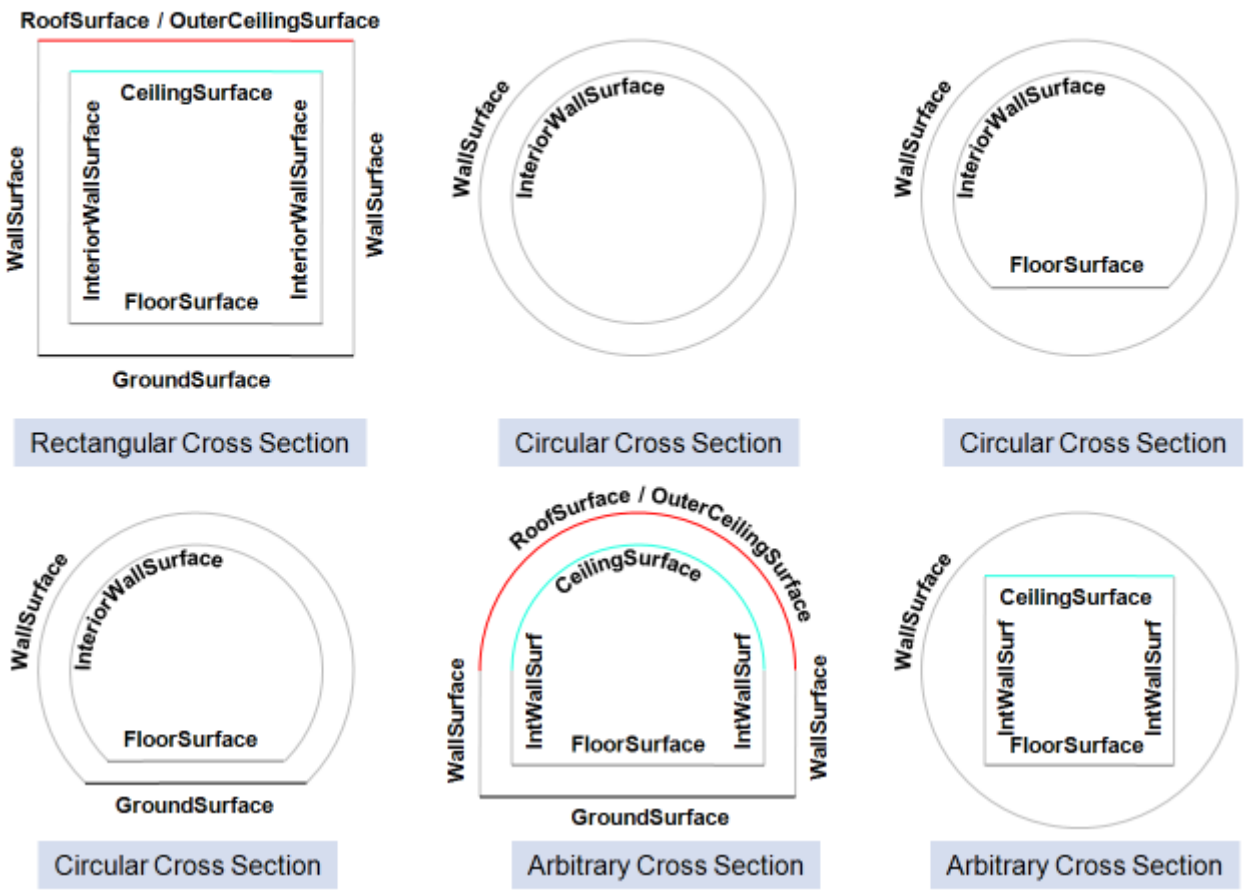


Figure 40. Examples for the use of boundary surfaces for tunnels with different cross sections. WallSurface, RoofSurface, GroundSurface, OuterCeilingSurface and OuterFloorSurface are available in LOD2– 4, whereas InteriorWallSurface, FloorSurface and CeilingSurface may only be used in LOD4 to model the interior boundary surfaces of a hollow space.

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

GroundSurfaceType, GroundSurface

NOTE	insert GroundSurfaceType, GroundSurface UML

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

OuterCeilingSurfaceType, OuterCeilingSurface

NOTE	insert OuterCeilingSurfaceType, OuterCeilingSurface UML

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

WallSurfaceType, WallSurface

NOTE	insert WallSurfaceType, WallSurface UML

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

OuterFloorSurfaceType, OuterFloorSurface

NOTE	insert OuterFloorSurfaceType, OuterFloorSurface UML

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

RoofSurfaceType, RoofSurface

NOTE	insert RoofSurfaceType, RoofSurface UML

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

ClosureSurfaceType, ClosureSurface

NOTE	insert ClosureSurfaceType, ClosureSurface UML

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

FloorSurfaceType, FloorSurface

NOTE	insert FloorSurfaceType, FloorSurface UML

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

InteriorWallSurfaceType, InteriorWallSurface

NOTE	insert InteriorWallSurfaceType, InteriorWallSurface UML

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

CeilingSurfaceType, CeilingSurface

NOTE	insert CeilingSurfaceType, CeilingSurface UML

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

11.4.4. Openings

AbstractOpeningType, _Opening

NOTE	insert AbstractOpeningType, _Opening UML

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

WindowType, Window

NOTE	insert WindowType, Window UML

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

DoorType, Door

NOTE	insert DoorType, Door UML

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

11.4.5. Tunnel Interior

HollowSpaceType, HollowSpace

NOTE	insert HollowSpaceType, HollowSpace UML

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

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

TunnelFurnitureType, TunnelFurniture

NOTE	insert TunnelFurnitureType, TunnelFurniture UML

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

IntTunnelInstallationType, IntTunnelInstallation

NOTE	insert IntTunnelInstallationType, IntTunnelInstallation UML

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

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

11.4.6. Examples

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

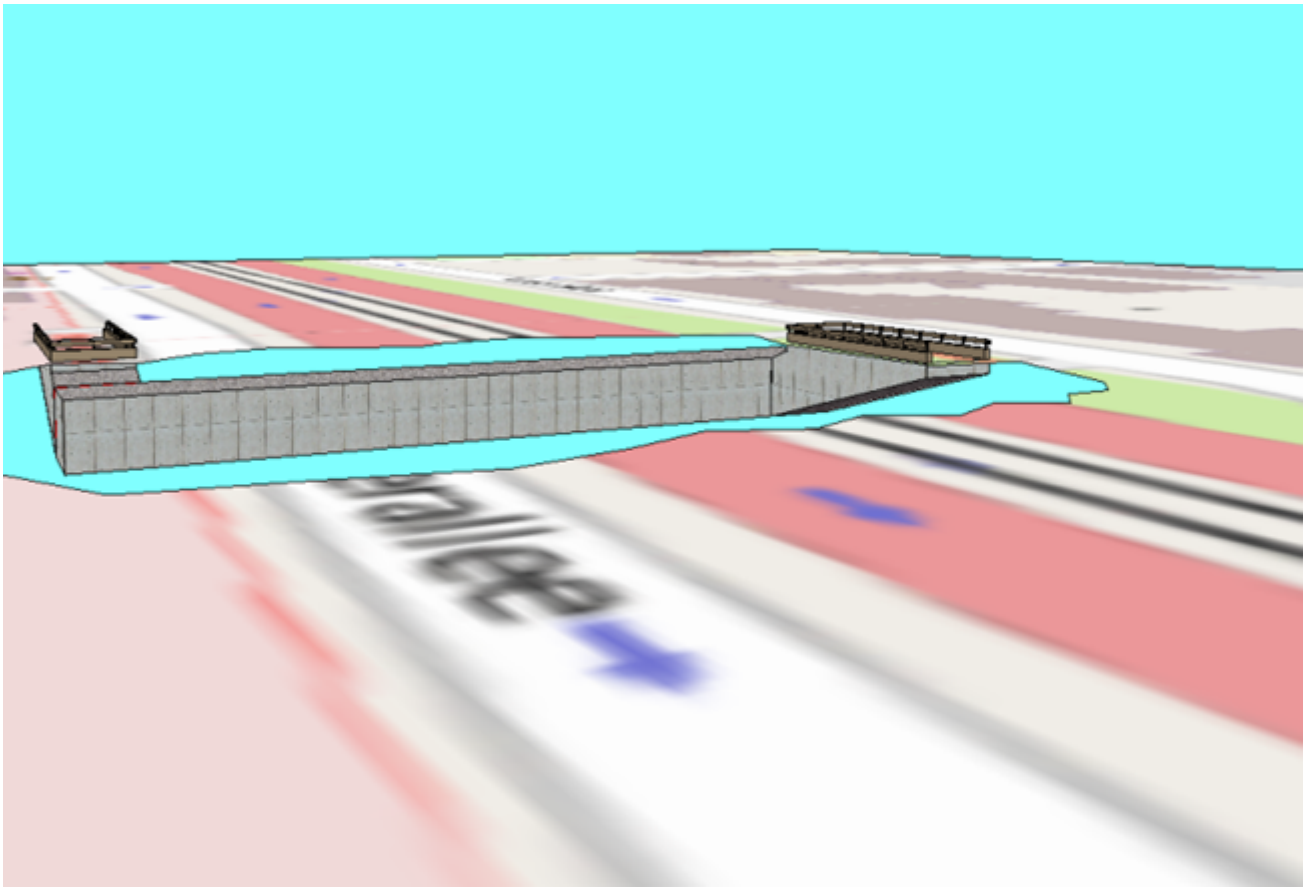


Figure 41. Example of a tunnel modeled in LOD3 (real situation on the left side; CityGML model on the right side) (source: Karlsruhe Institute of Technology (KIT), courtesy of City of Karlsruhe).

NOTE insert Fig 43

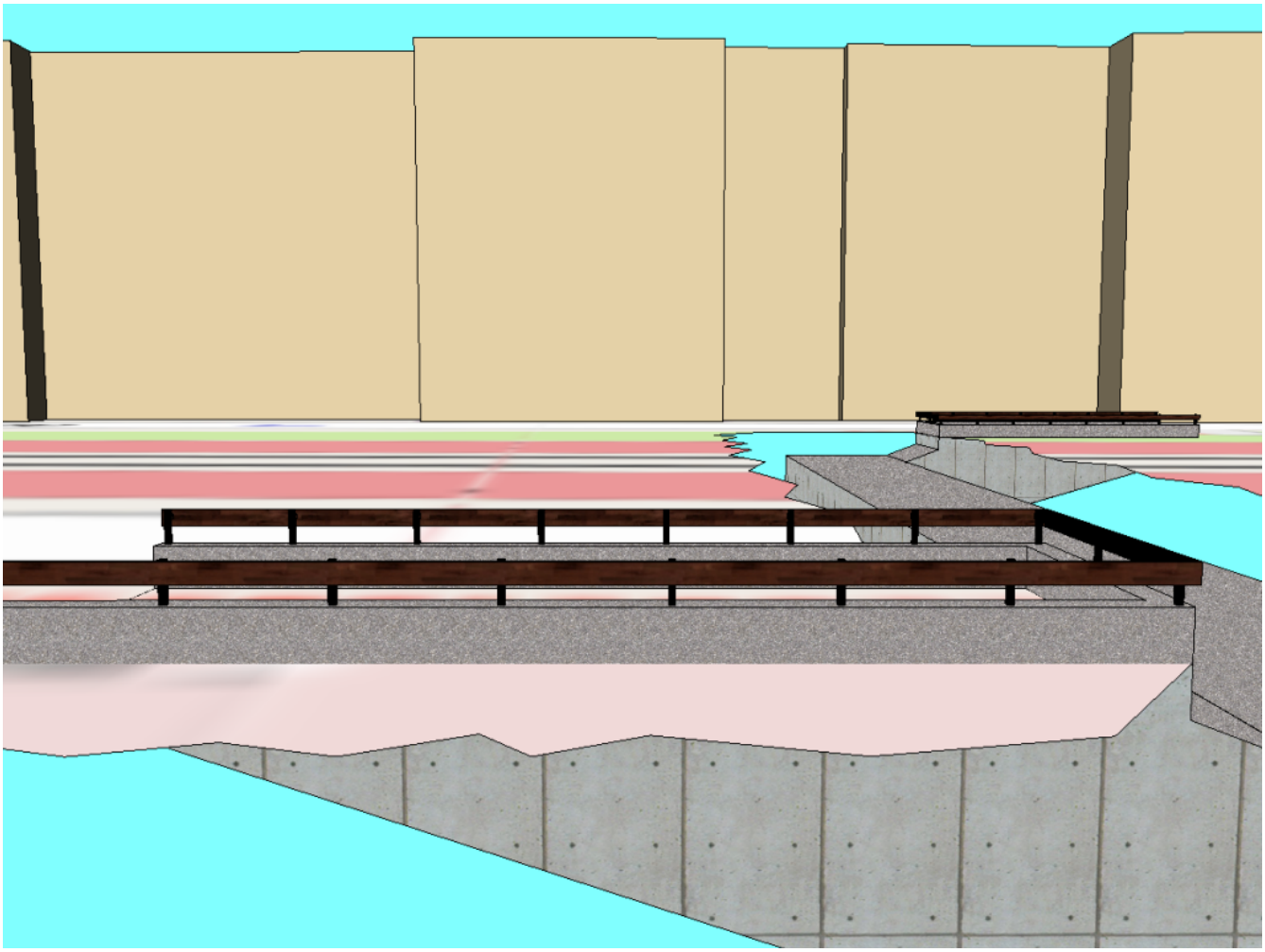
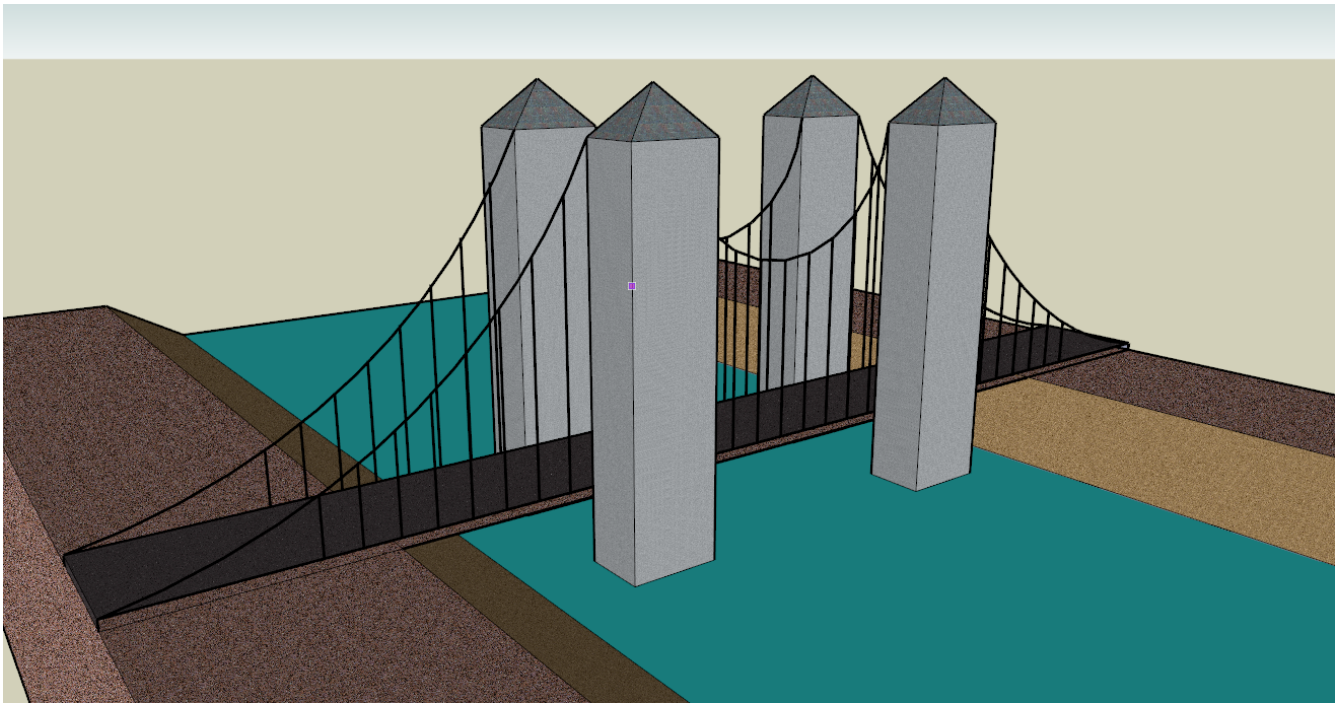


Figure 42. The same LOD3 tunnel shown from a different perspective. The camera is positioned in front of the left entrance and pointing in the direction of the right entrance. (real situation on the left side; CityGML model on the right side). The model on the right also includes an LOD1 representation 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.

11.5. Bridge Package

The bridge model allows for the representation of the thematic, spatial and visual aspects of bridges and bridge parts in four levels of detail, LOD 1 – 4. The bridge model of CityGML is defined by the thematic extension module Bridge (cf. chapter 7). Fig. 44 illustrates examples of bridge models in all LODs.



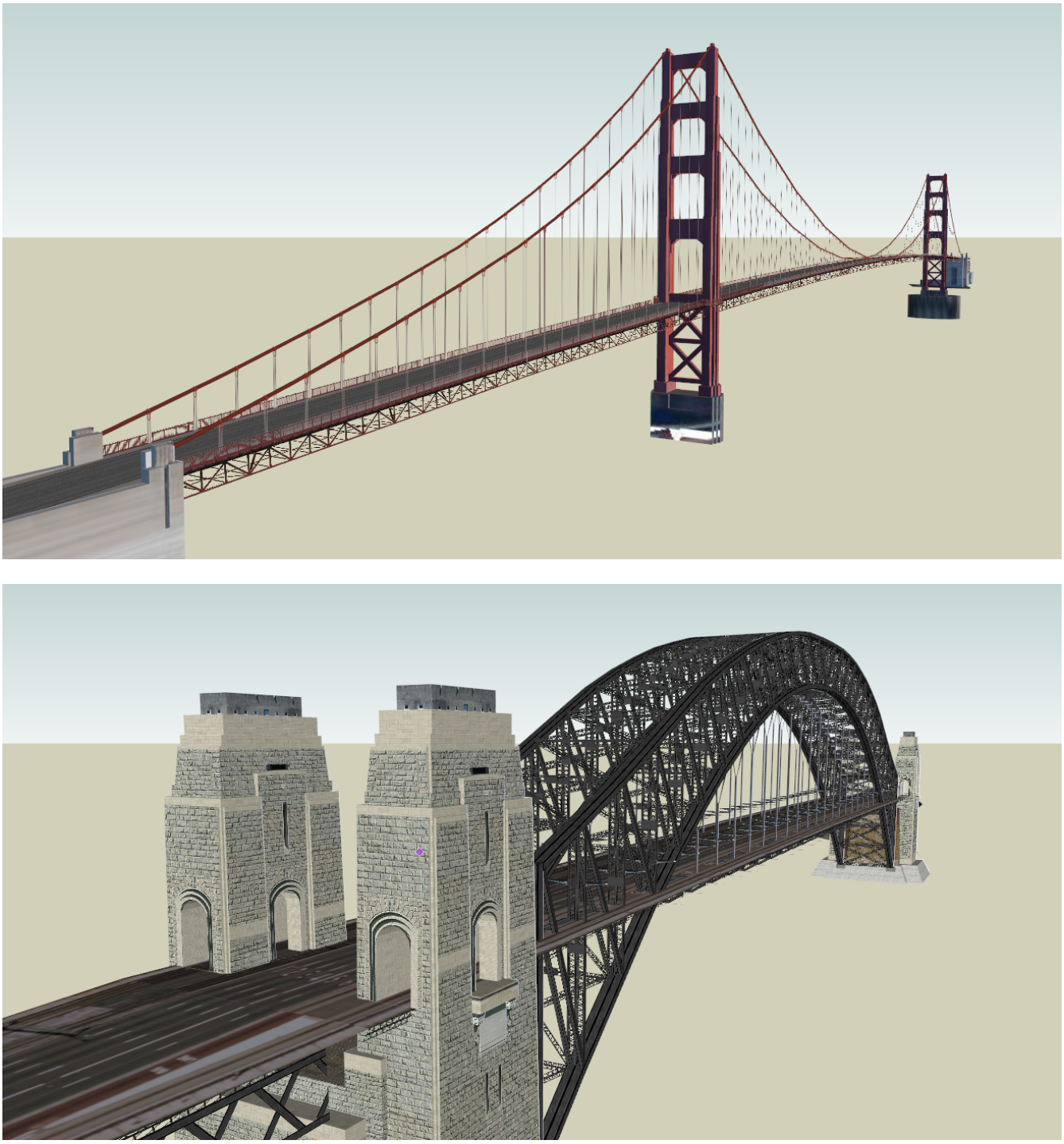


Figure 43. Examples for bridge models in LOD1 (upper left), LOD2 (upper right), LOD3 (lower left) and LOD4 (lower right) (source: Google 3D warehouse)

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

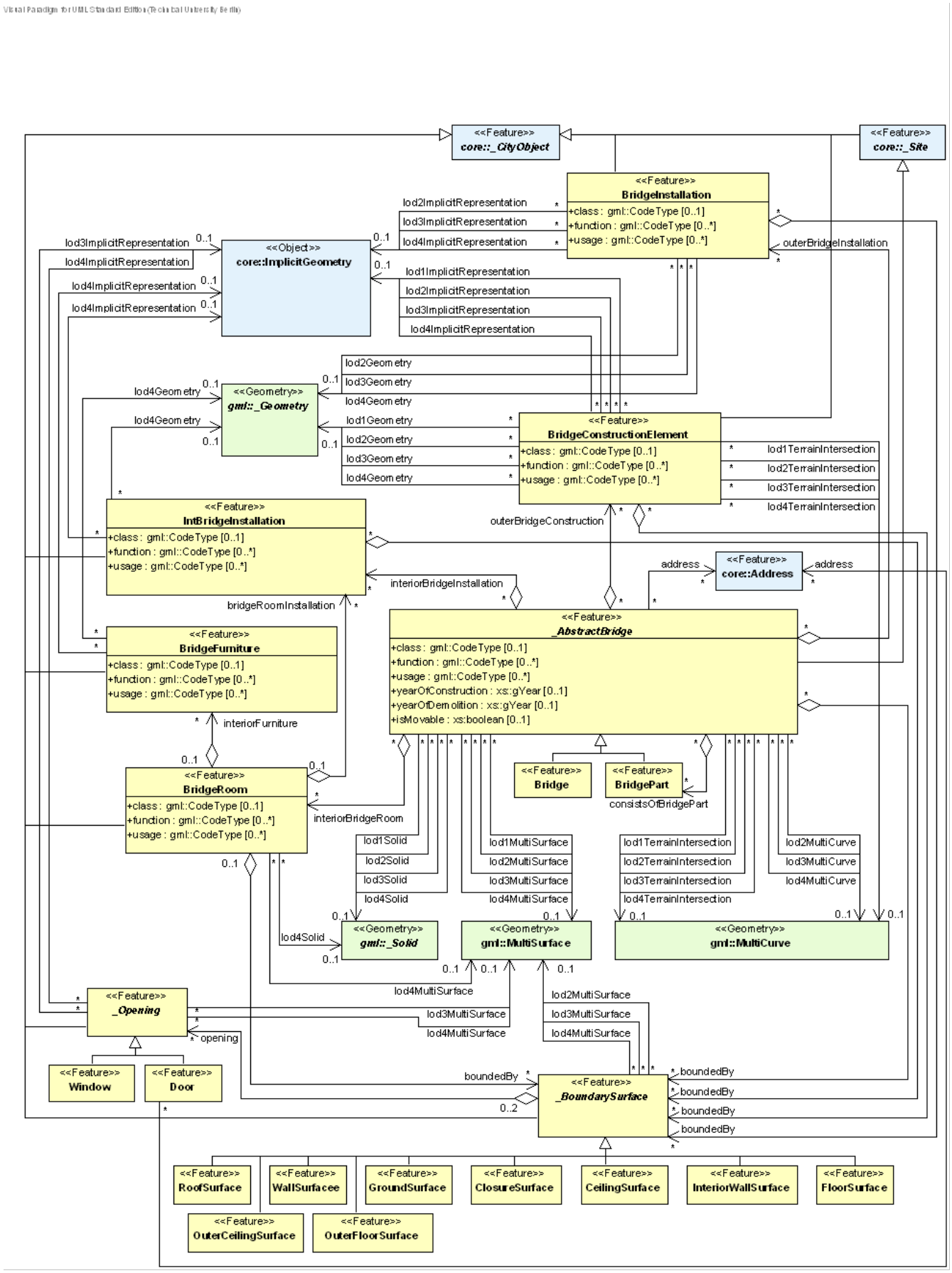
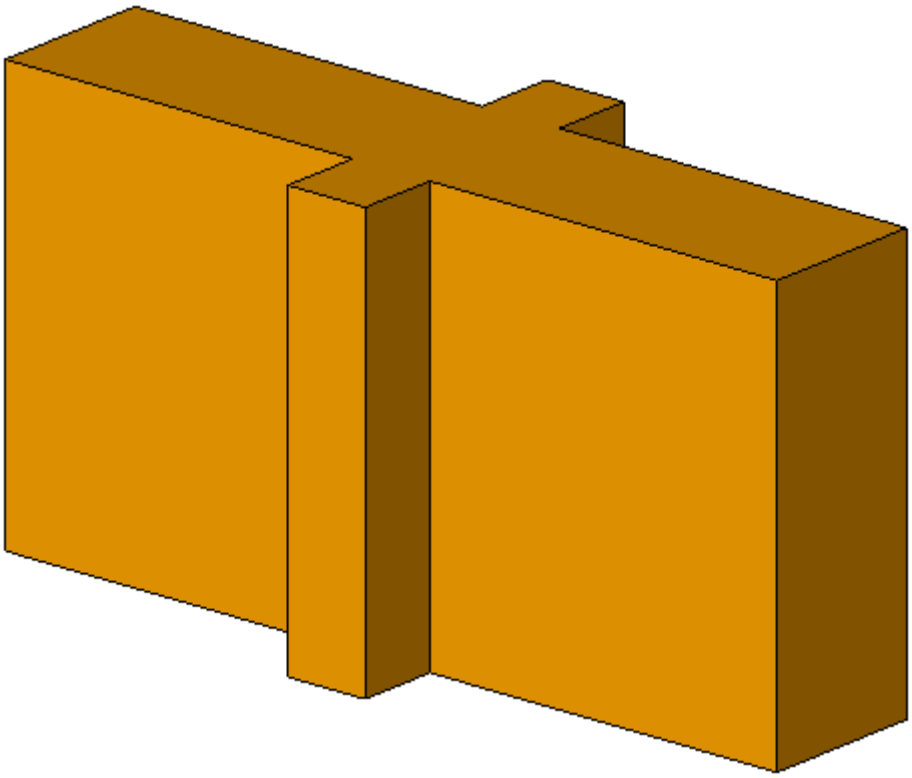
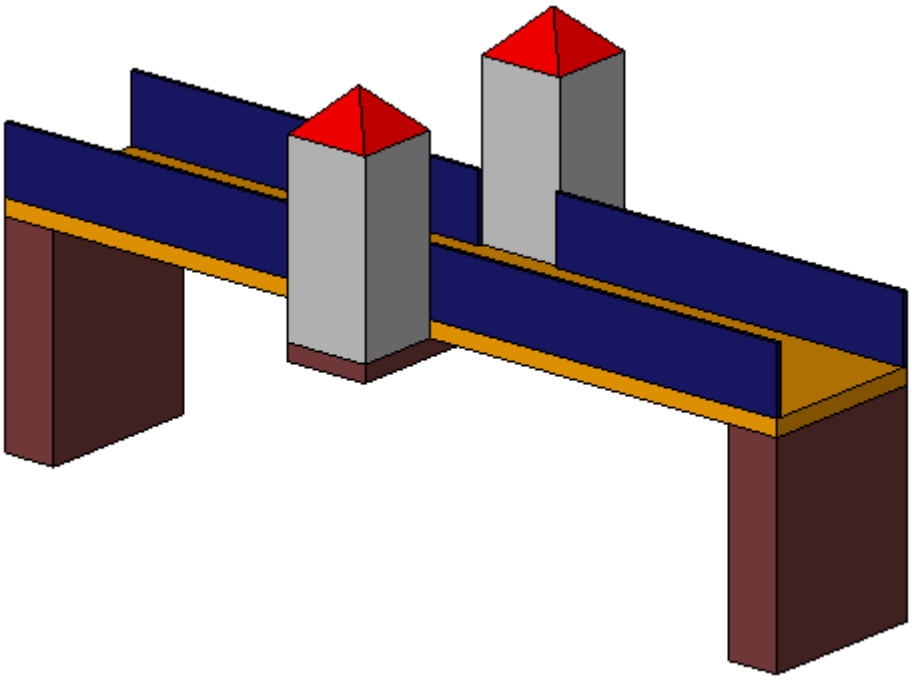


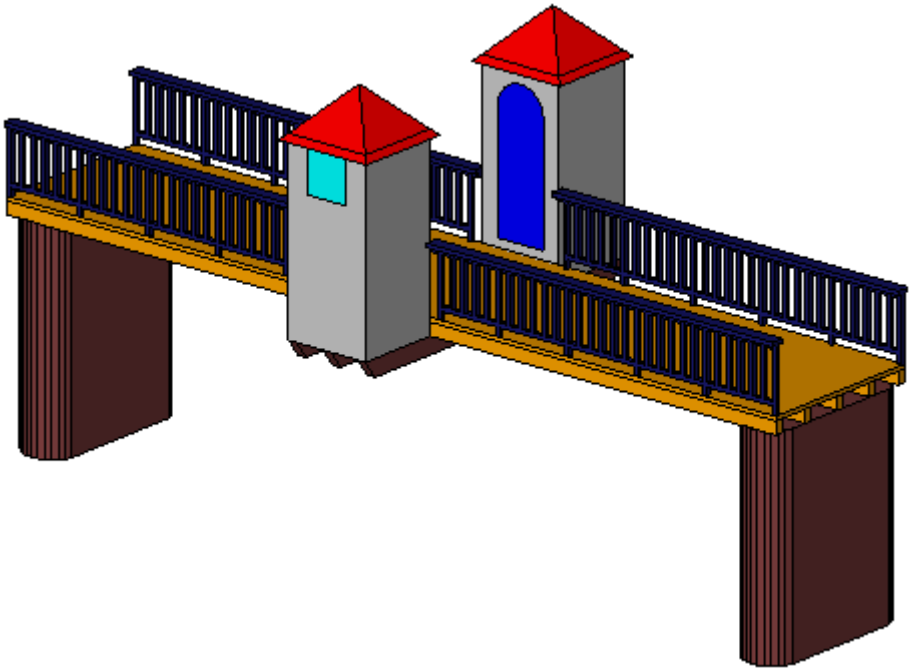
Figure 44. UML diagram of the bridge model, part one

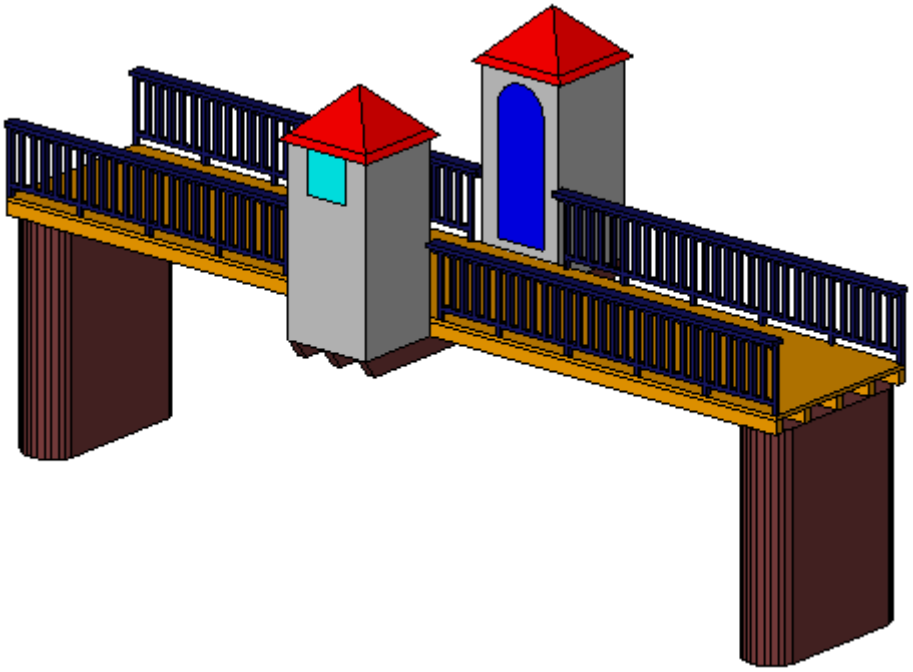
A (movable or unmovable) bridge is represented by an object of the class Bridge. This class inherits its attributes and relations from the abstract base class _AbstractBridge. The spatial properties are defined by a solid for each of the four LODs (relations lod1Solid to lod4Solid). In analogy to the

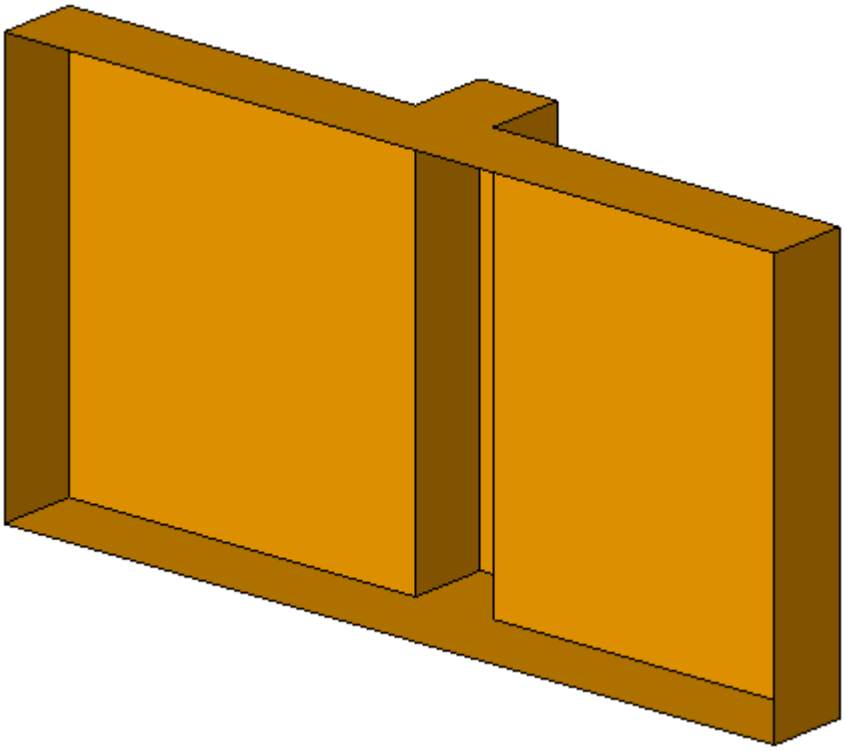
building model, the semantical as well as the geometrical richness increases from LOD1 (blocks model) to LOD3 (architectural model). Simple examples of bridges in each of those LODs are depicted in Fig. 46. Interior structures like rooms are dedicated to LOD4. To cover the case of bridge models where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

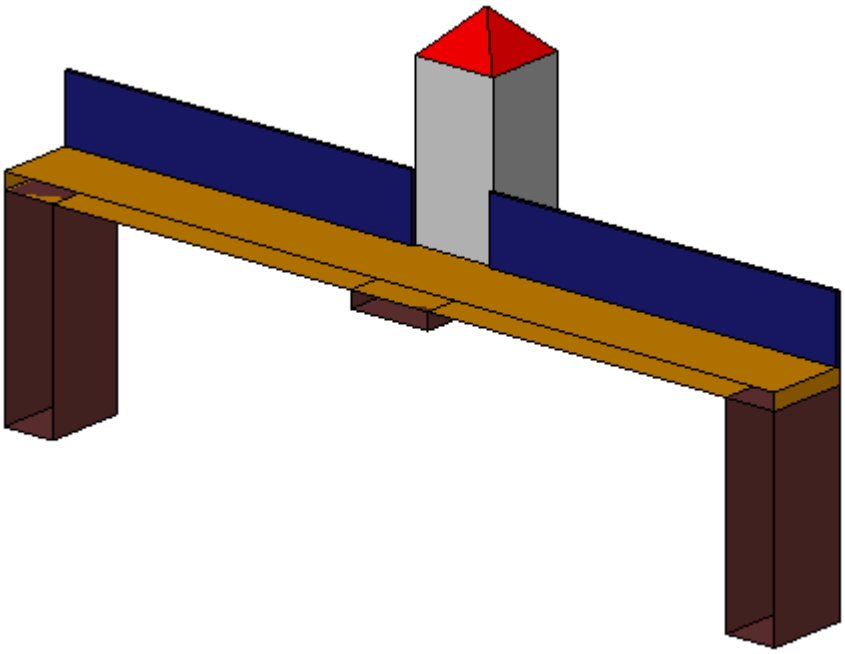


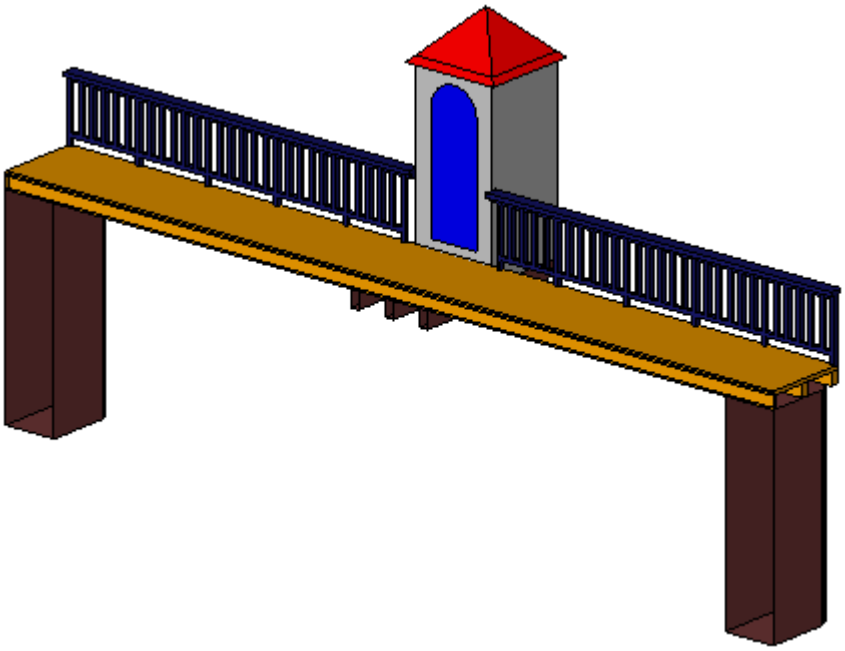


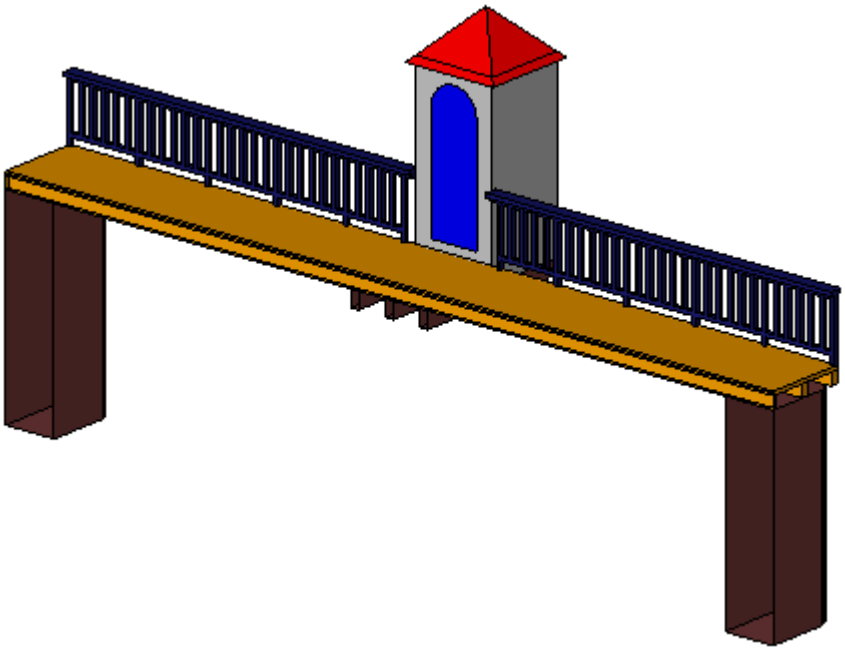


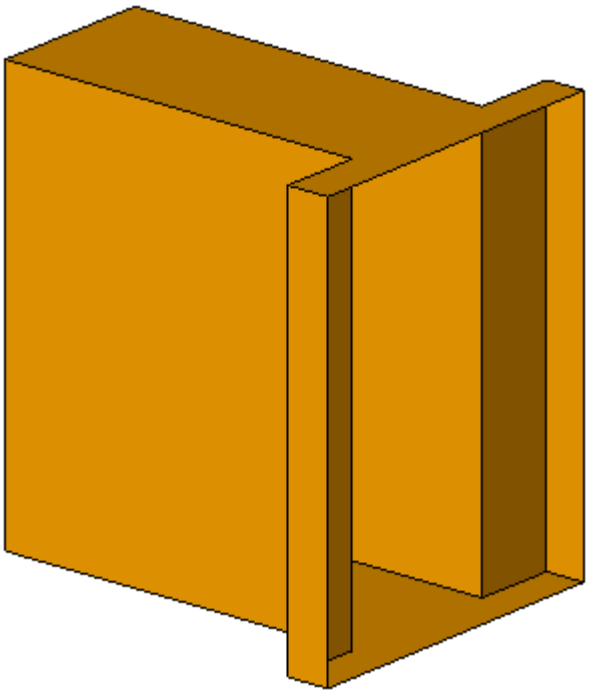


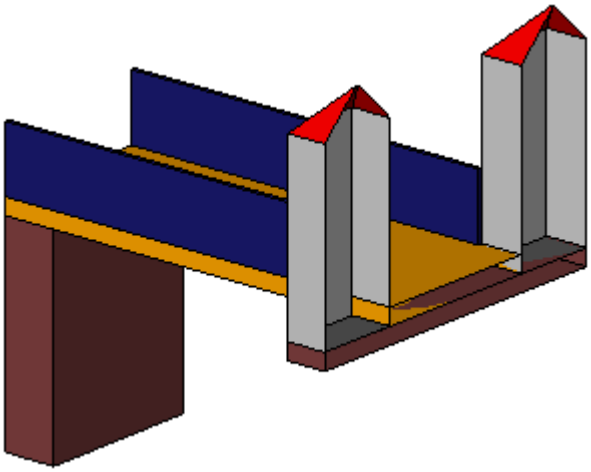


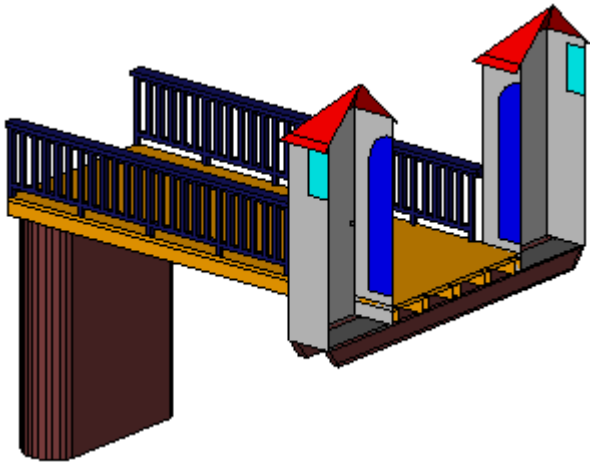












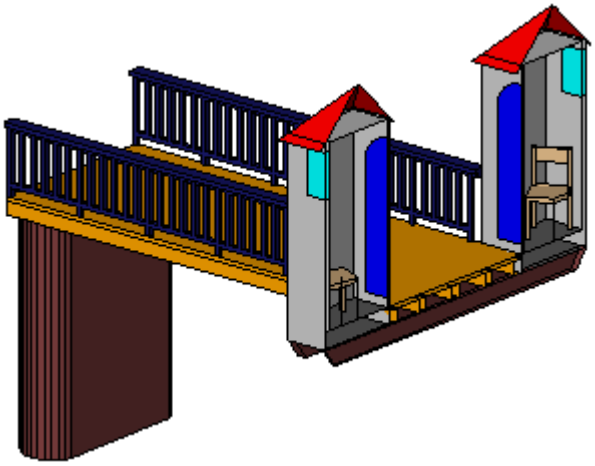


Figure 45. Bridge model in LOD1 – LOD4. (source: Karlsruhe Institute of Technology (KIT))

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 8. Semantic themes of the class _AbstractBridge

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

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

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

NOTE	the following are annimated GIFs

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

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

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

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

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

Figure 46. Examples for movable bridges (source: ISO 6707).

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

11.5.1. Bridge and bridge part

BridgeType, Bridge

NOTE	insert BridgeType, Bridge UML

BridgePartType, BridgePart

NOTE	insert BridgePartType, BridgePart UML

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

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

AbstractBridgeType, _AbstractBridge

NOTE	insert AbstractBridgeType, _AbstractBridge UML

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

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

11.5.2. Bridge construction elements and bridge installations

BridgeConstructionElementType, BridgeConstructionElement

NOTE	insert BridgeConstructionElementType, BridgeConstructionElement UML

BridgeInstallationType, BridgeInstallation

NOTE	insert BridgeInstallationType, BridgeInstallation UML

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

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

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

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

11.5.3. Boundary surfaces

AbstractBoundarySurfaceType, _BoundarySurface

NOTE	insert AbstractBoundarySurfaceType, _BoundarySurface UML

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

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

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

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

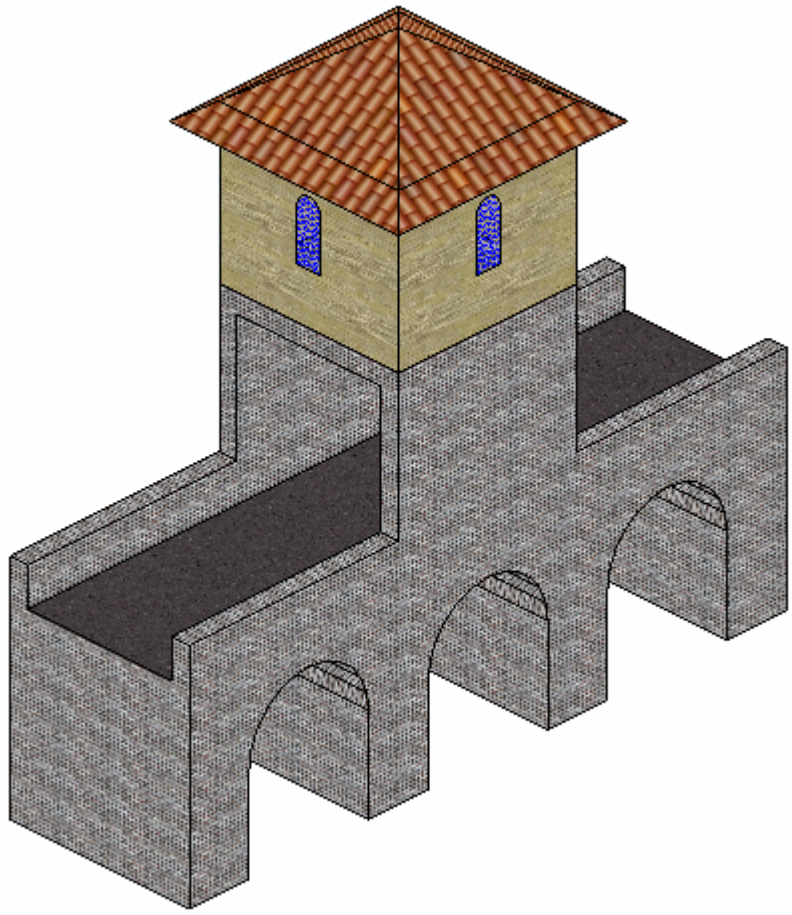


Figure 47. Different BoundarySurfaces of a bridge.

NOTE	need to add annotations to Figure 50

GroundSurfaceType, GroundSurface

NOTE	insert GroundSurfaceType, GroundSurface UML

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

OuterCeilingSurfaceType, OuterCeilingSurface

Note: insert OuterCeilingSurfaceType, OuterCeilingSurface UML

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

WallSurfaceType, WallSurface

NOTE	insert WallSurfaceType, WallSurface UML

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

OuterFloorSurfaceType, OuterFloorSurface

NOTE	insert OuterFloorSurfaceType, OuterFloorSurface UML

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

RoofSurfaceType, RoofSurface

NOTE	insert RoofSurfaceType, RoofSurface UML

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

ClosureSurfaceType, ClosureSurface

NOTE	insert ClosureSurfaceType, ClosureSurface UML

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

FloorSurfaceType, FloorSurface

NOTE	insert FloorSurfaceType, FloorSurface UML

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

InteriorWallSurfaceType, InteriorWallSurface

NOTE	insert InteriorWallSurfaceType, InteriorWallSurface UML

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

CeilingSurfaceType, CeilingSurface

NOTE	insert CeilingSurfaceType, CeilingSurface UML

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

11.5.4. Openings

AbstractOpeningType, _Opening

NOTE	insert AbstractOpeningType, _Opening UML

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

WindowType, Window

NOTE	insert WindowType, Window UML

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

DoorType, Door

NOTE	insert DoorType, Door UML

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

11.5.5. Bridge Interior

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

BridgeRoomType, BridgeRoom

NOTE	insert BridgeRoomType, BridgeRoom UML

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

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

BridgeFurnitureType, BridgeFurniture

NOTE	insert BridgeFurnitureType, BridgeFurniture UML

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

IntBridgeInstallationType, IntBridgeInstallation

NOTE	insert IntBridgeInstallationType, IntBridgeInstallation UML

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

11.5.6. 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).

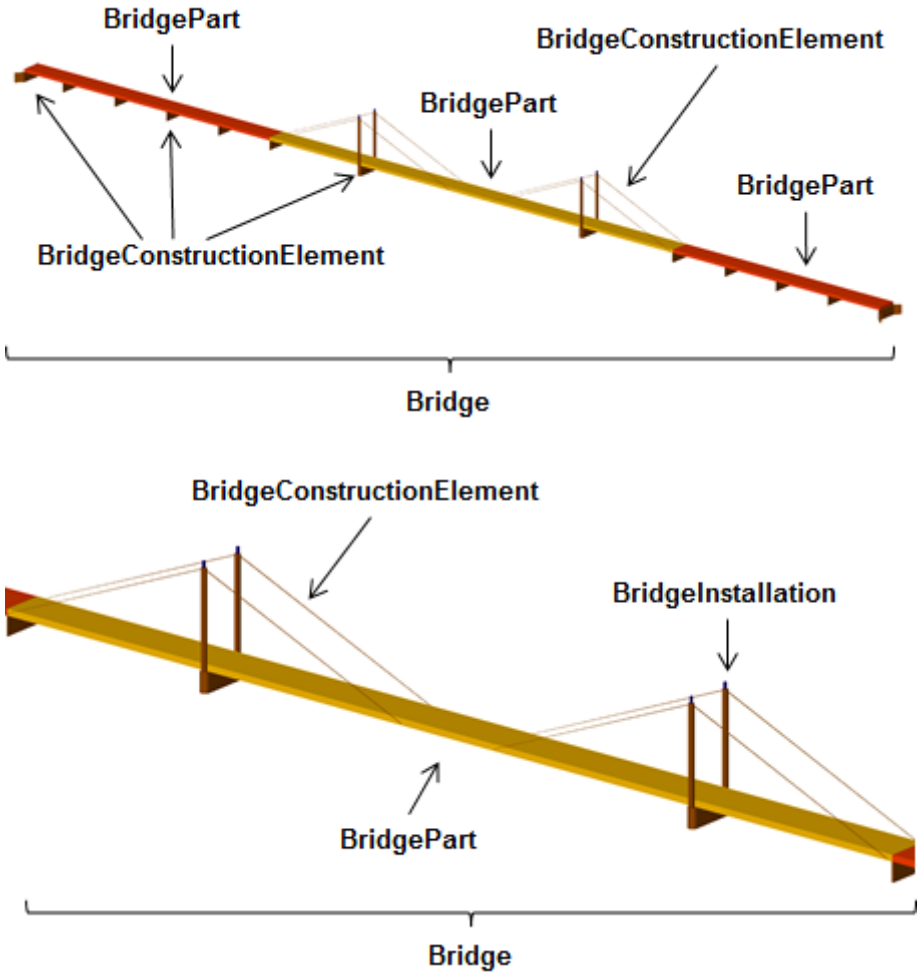


Figure 48. The bridge of Rees, consisting of a Bridge feature and three BridgePart features (left). The bridge contains BridgeConstructionElement and BridgeInstallation features (right).

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

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

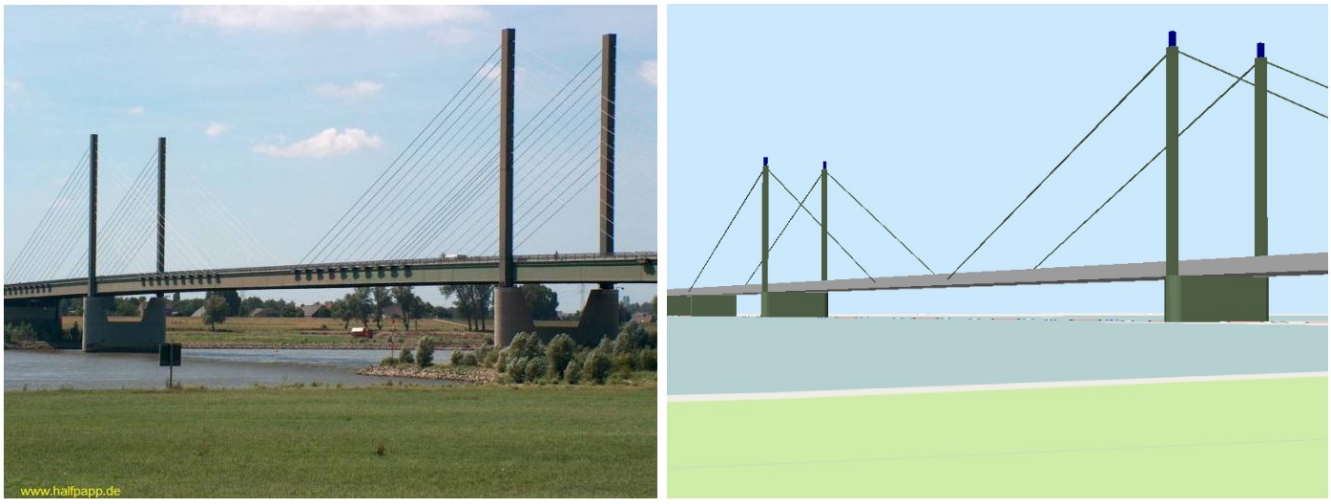


Figure 49. The bridge of Rees (left photo (source: Harald Halfpapp); right LOD2 model (source: District of Recklinghausen / KIT)).

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).

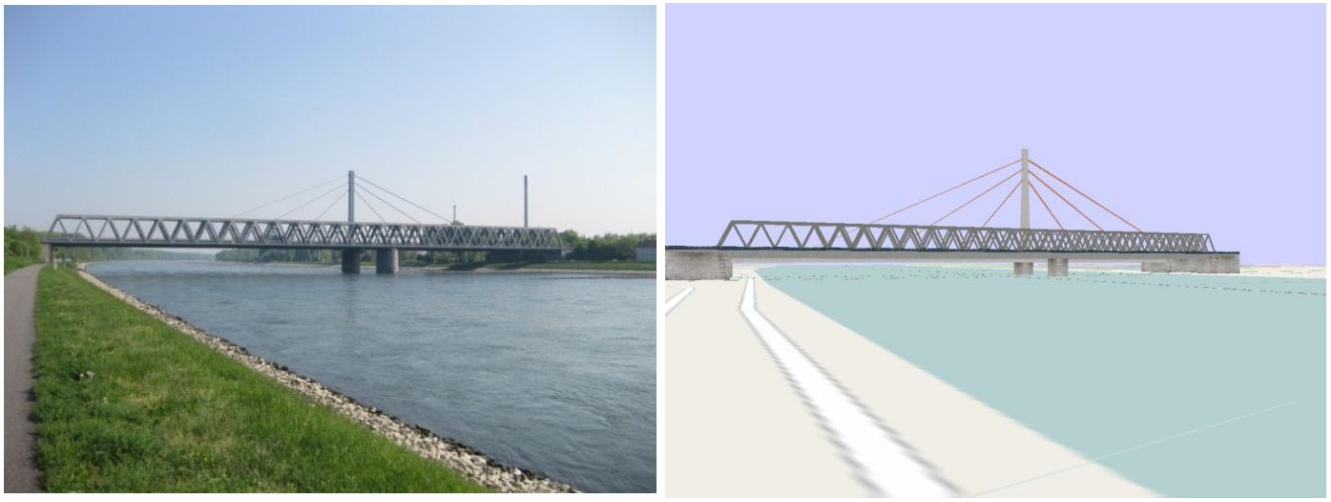


Figure 50. Bridge over the river Rhine at Karlsruhe (left a photo, right the 3D CityGML model) (source: Karlsruhe Institute of Technology (KIT), courtesy of City of Karlsruhe).

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

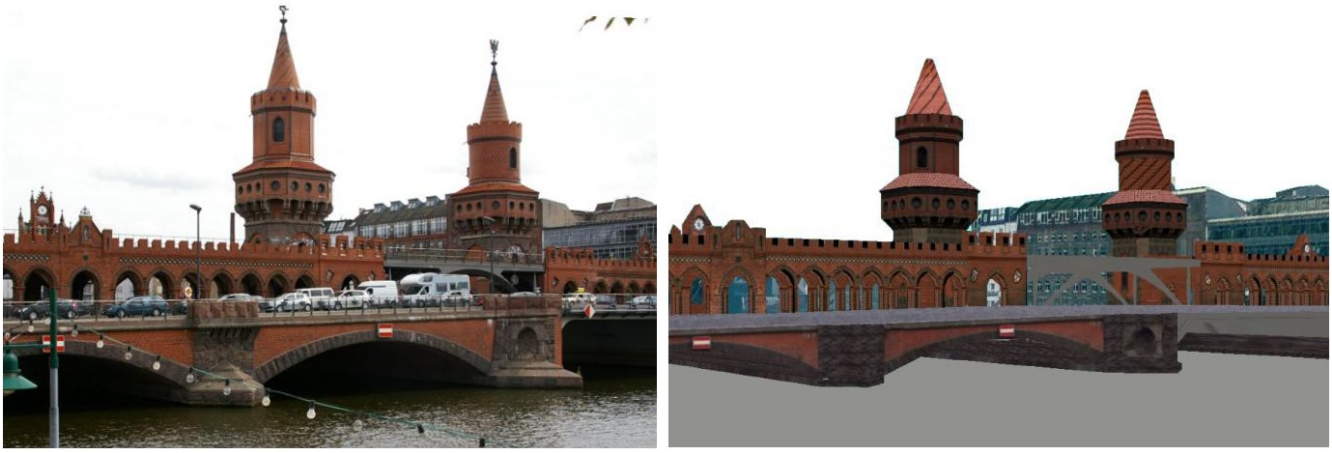


Figure 51. The bridge“Oberbaumbrücke” in Berlin represented as bridge model in LOD4 (left a photo, right the 3D CityGML model) (source: Berlin Senate of Business, Technology and Women; Business Location Center, Berlin; Technische Universität Berlin; Karlsruhe Institute of Technology (KIT)).

11.6. Water Bodies

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 sanita-tion. 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 (cf. chapter 7).

The water bodies model represents the thematic aspects and three-dimensional geometry of rivers, canals, lakes, and basins. In the LOD 2-4 water bodies are bounded by distinct thematic surfaces. These surfaces are the obligatory WaterSurface, defined as the boundary between water and air, the optional WaterGroundSurface, defined as the boundary between water and underground (e.g. DTM or floor of a 3D basin object), and zero or more WaterClosureSurfaces, defined as virtual boundaries between different water bodies or between water and the end of a modelled region (see Fig. 55). A dynamic element may be the WaterSurface to represent temporarily changing situations of tidal flats.

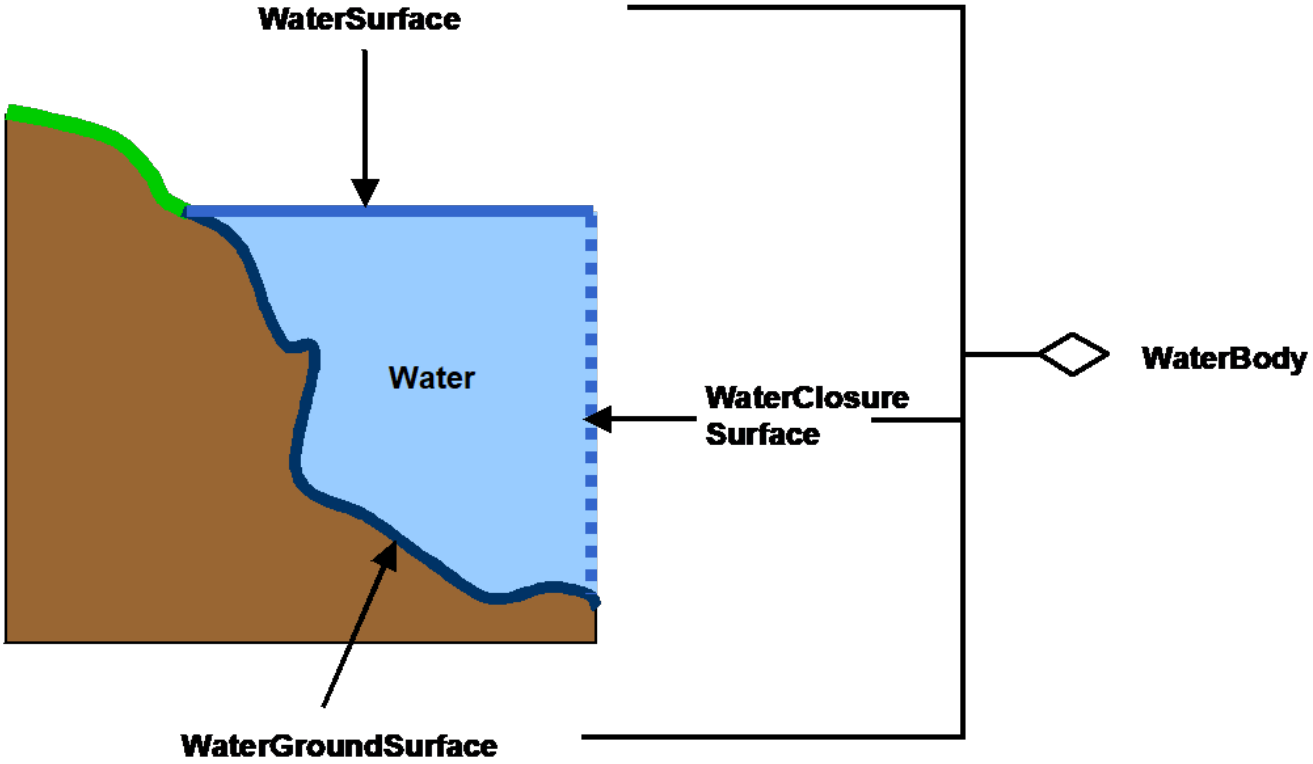


Figure 52. Illustration of a water body defined in CityGML (graphic: IGG Uni Bonn).

The UML diagram of the water body model is depicted in Fig. 56, for the XML schema definition see below and annex A.13. Each WaterBody object may have the attributes class, function and usage whose possible values can be enumerated in code lists (cf. chapter 10.6.3 and annex C.9). 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.

WaterBody is a subclass of _WaterObject and transitively of the root class _CityObject. The class _WaterObject may be differentiated in further subclasses of water objects in the future. The geometrical representation of the WaterBody varies through the different levels of detail. Since WaterBody is a subclass of _CityObject and hence a feature, it inherits the attribute gml:name. The WaterBody can be differentiated semantically by the class _WaterBoundarySurface. A _WaterBoundarySurface is a part of the water body's exterior shell with a special function like WaterSurface, WaterGroundSurface or WaterClosureSurface. As with any _CityObject, WaterBody objects as well as WaterSurface, WaterGroundSurface, and WaterClosureSurface may be assigned ExternalRef-erences (cf. chapter 6.7) and may be augmented by generic attributes using CityGML's Generics module (cf. chapter 10.12).

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.

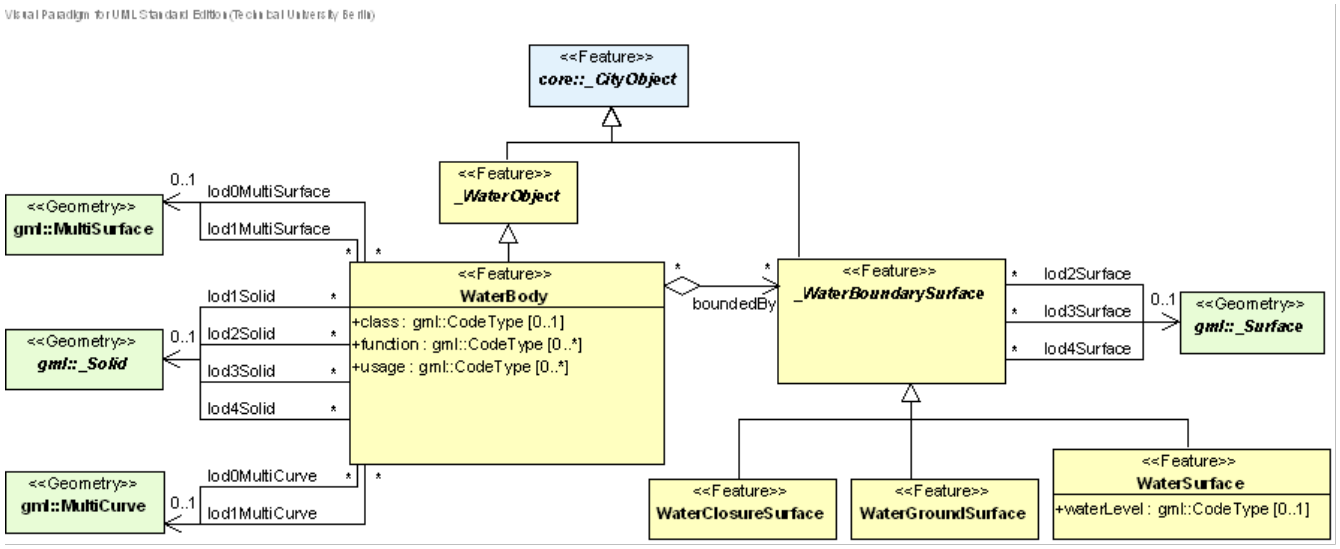


Figure 53. UML diagram of the water body model in CityGML. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML WaterBody module.

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

Starting from LOD1 water bodies may also be modelled as water filled volumes represented by Solids. If a water body is represented by a gml:Solid in LOD2 or higher, the surface geometries of the corresponding thematic WaterClosureSurface, WaterGroundSurface, and WaterSurface objects must coincide with the exterior shell of the gml:Solid. This can be ensured, if for each LOD X the respective lodXSolid representation (where X is between 2 and 4) does not redundantly define the geometry, but instead references the corresponding polygons (using GML3's XLink mechanism) of the lodXSurface elements (where X is between 2 and 4) of Water-ClosureSurface, WaterGroundSurface, and WaterSurface.

LOD2 to LOD4 demand a higher grade of detail and therefore any WaterBody can be outlined by thematic surfaces or a solid composed of the surrounding thematic surfaces. Every object of the class WaterSurface, WaterClosureSurface, and WaterGroundSurface must have at least one associated surface geometry. This means, that every WaterSurface, WaterClosureSurface, and WaterGroundSur-face feature within a CityGML instance document must contain at least one of the following properties: lod2Surface, lod3Surface, lod4Surface.

The water body model implicitly includes the concept of TerrainIntersectionCurves (TIC), e.g. to specify the exact intersection of the DTM with the 3D geometry of a WaterBody or to adjust a WaterBody or WaterSurface to the surrounding DTM (see chapter 6.5). The rings defining the WaterSurface polygons implicitly delineate the intersection of the water body with the terrain or basin.

11.6.1. Water Body

AbstractWaterObjectType, _WaterObject

NOTE	insert AbstractWaterObjectType, _WaterObject UML

WaterBodyType, WaterBody

NOTE	insert WaterBodyType, WaterBody UML

11.6.2. Boundary surfaces

With respect to different functions and characteristics three boundary classes for water are defined to build a solid or composite surface geometry (Fig. 55).

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 chang-ing water levels, which can be reflected by including different static water surfaces having different wa-terLevels (gml:CodeType), as for example highest flooding event, mean sea level, or minimum water level. This offers the opportunity to describe significant water surfaces due to levels that are important for certain representations e.g. in tidal zones.
2. Boundary class “Water to Ground”. The WaterGroundSurface may be known by sonar exploration or other depth measurements. Also part of the ground surface is the boundary “Water to Construction”. The ground surface might be identical to the underwater terrain model, but also describes the contour to other underwa-ter objects. The usefulness of this concept arises from the existence of water defence constructions like sluices, sills, flood barrage or tidal power stations. The use of WaterGroundSurface as boundary layer to man-made constructions is relevant in urban situations, where such objects may enclose the modeled water body completely, for example fountains and swimming pools. The WaterSurface objects together with the WaterGroundSurface objects enclose the WaterBody as a volume.
3. Boundary class “Water to Water”. The WaterClosureSurface is an optional feature that comes in use when the union of the WaterSurfaces and WaterGroundSurfaces of a water body does not define a closed volume. The WaterClosureSurface is then used to complete the enclosure of water volumes and to separate water volumes from those where only the surface is known. This might occur, where the cross section and ground surface of rivers is partly available during its course.

_WaterBoundarySurfaces should only be included as parts of corresponding WaterBody objects and should not be used as stand-alone objects within a CityGML model.

AbstractWaterBoundarySurfaceType, _WaterBoundarySurface

NOTE	insert AbstractWaterBoundarySurfaceType, _WaterBoundarySurface UML

WaterSurfaceType, WaterSurface

NOTE	insert WaterSurfaceType, WaterSurface UML

WaterGroundSurfaceType, WaterGroundSurface

NOTE	insert WaterGroundSurfaceType, WaterGroundSurface UML

WaterClosureSurfaceType, WaterClosureSurface

NOTE	insert WaterClosureSurfaceType, WaterClosureSurface UML

11.7. Transportation

The transportation model of CityGML is a multi-functional, multi-scale model focusing on thematic and func-tional 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 transportation model of CityGML is provided by the thematic extension module Transportation (cf. chapter 7).

The main class is TransportationComplex, which represents, for example, a road, a track, a railway, or a square. Fig. 57 illustrates the four different thematic classes.

NOTE	Four images in the .doc may all be the same.

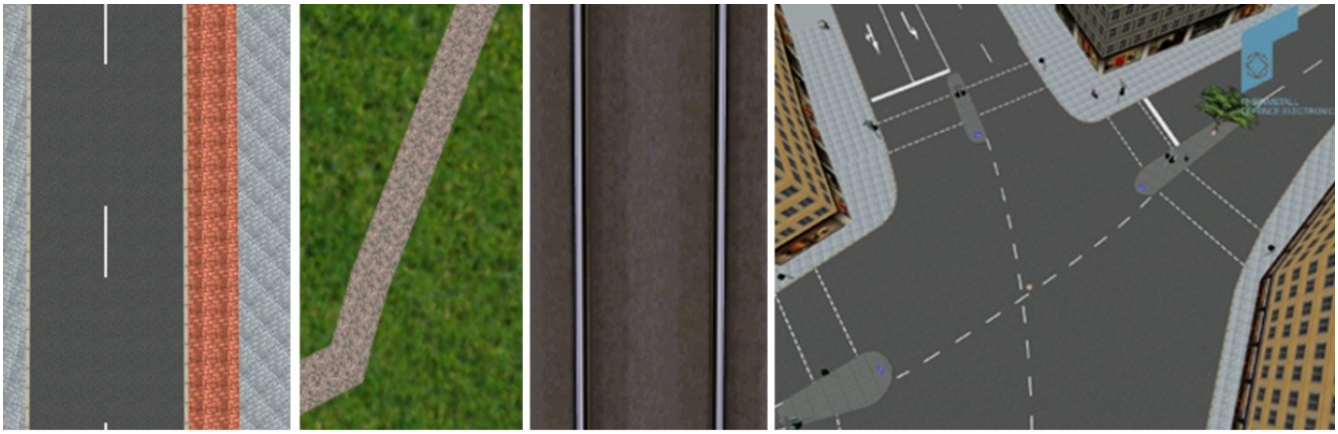


Figure 54. Representations of TransportationComplex (from left to right: examples of road, track, rail, and square) (source: Rheinmetall Defence Electronics).

A TransportationComplex is composed of the parts TrafficArea and AuxiliaryTrafficArea. Fig. 58 depicts an example for a LOD2 TransportationComplex configuration within a virtual 3D city model. The Road consists of several TrafficAreas for the sidewalks, road lanes, parking lots, and of AuxiliaryTrafficAreas below the raised flower beds.

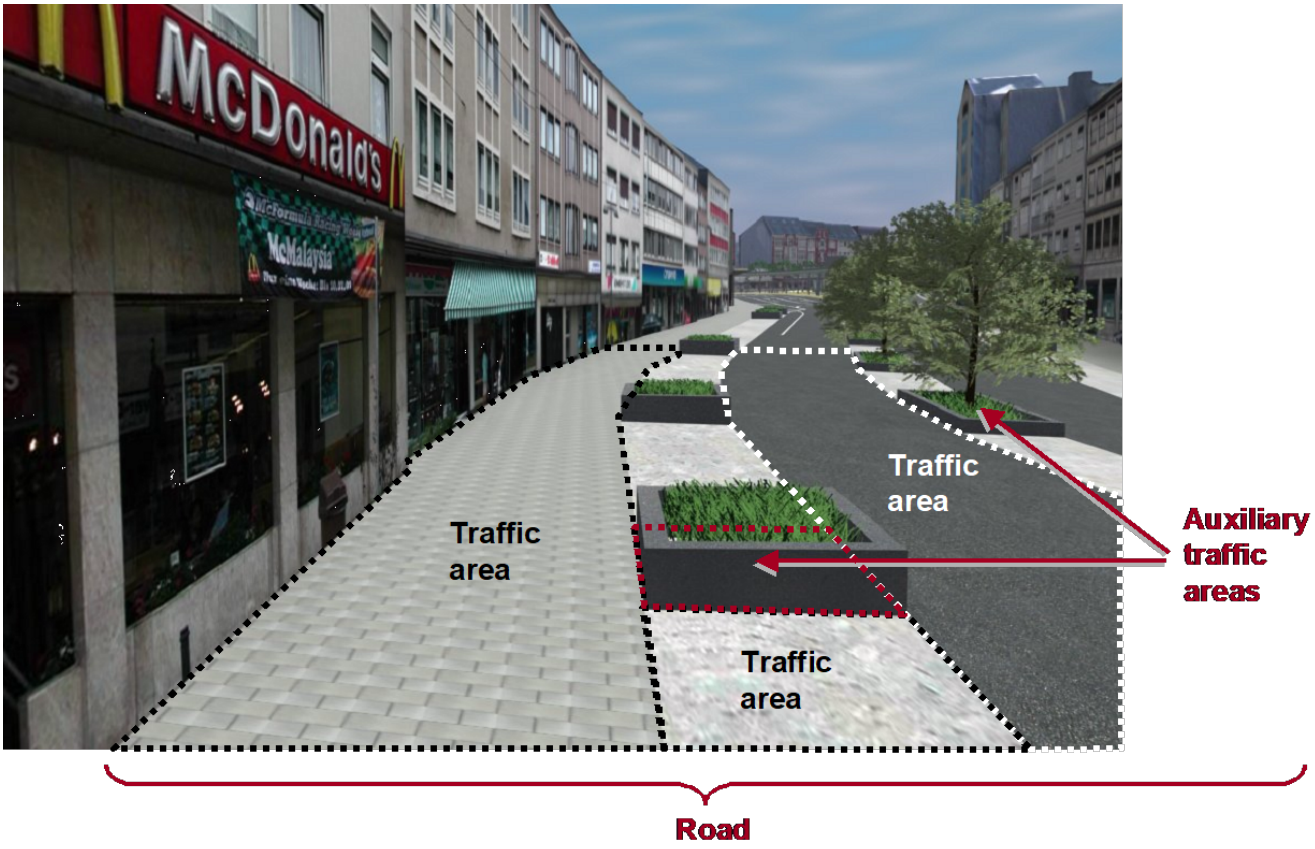


Figure 55. Example for the representation of a TransportationComplex in LOD2 in CityGML: a road, which is the aggregation of TrafficAreas and AuxiliaryTrafficAreas (source: City of Solingen, IGG Uni Bonn).

Fig. 59 depicts the UML diagram of the transportation model, for the XML schema definition see annex A.10.

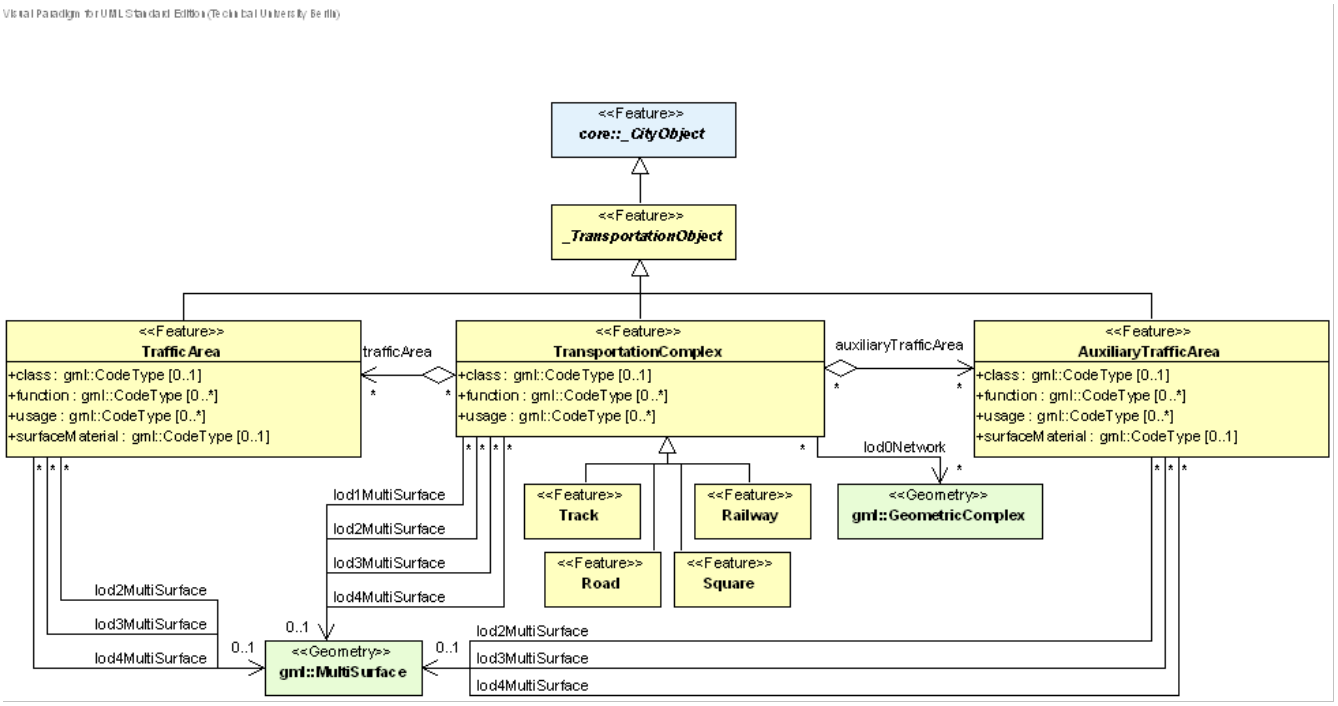


Figure 56. UML diagram of the transportation model in CityGML. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Transportation module.

The road itself is represented as a TransportationComplex, which is further subdivided into

TrafficAreas and AuxiliaryTrafficAreas. The TrafficAreas are those elements, which are important in terms of traffic usage, like car driving lanes, pedestrian zones and cycle lanes. The AuxiliaryTrafficAreas are describing further elements of the road, like kerbstones, middle lanes, and green areas.

TransportationComplex objects can be thematically differentiated using the subclasses Track, Road, Railway, and Square. Every TransportationComplex has the attributes class, function and usage whose possible values can be enumerated in code lists (chapter 10.7.4 and annex C.8). The attribute class describes the classification of the object, function describes the purpose of the object, for example national motorway, country road, or airport, while the attribute usage can be used, if the actual usage differs from the function. The attributes function and usage can occur multiple times.

In addition both TrafficArea and AuxiliaryTrafficArea may have the attributes class, function, usage, and sur-faceMaterial. The attribute class describes the classification of the object. For TrafficArea, function describes, if the object for example may be a car driving lane, a pedestrian zones, or a cycle lane, while the usage attribute indicates which modes of transportation can use it (e.g. pedestrian, car, tram, roller skates). The attribute sur-faceMaterial specifies the type of pavement and may also be used for AuxiliaryTrafficAreas (e.g. asphalt, con-crete, gravel, soil, rail, grass). The function attribute of the AuxiliaryTrafficArea defines, for example kerbstones, middle lanes, or green areas. The possible values can also be specified in code lists.

The shape of each traffic area is defined by a surface geometry. Additional metadata may be defined by using attributes from pre-defined catalogues. This affects the class, function and usage of each traffic area as well as its surface material. The attribute catalogues may be customer- or country-specific. The following tables show examples for various kinds of TrafficArea:

Table 9. Examples of TrafficArea

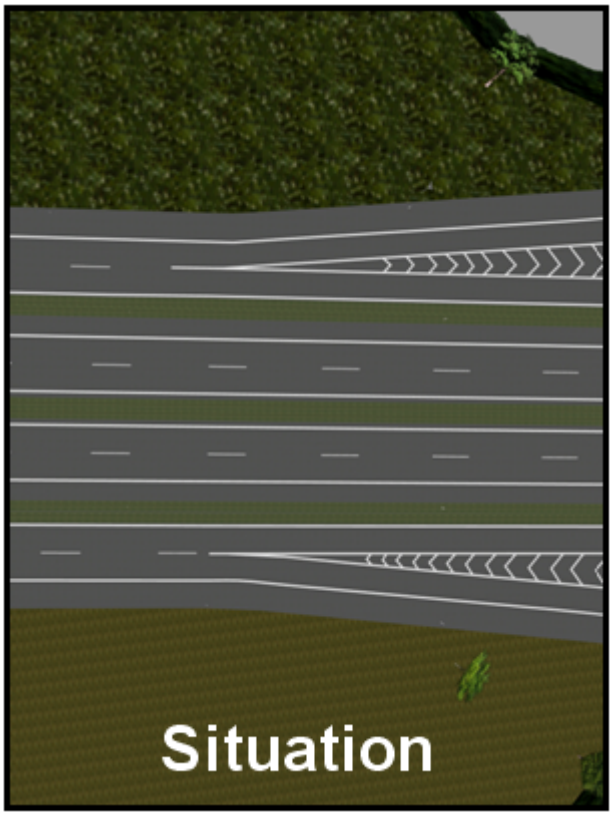
Example	Country Road	Motorway Entry
TransportationComplex – Function	road	road
TrafficArea – Usage	car, truck, bus, taxi, motorcycle	car, truck, bus, taxi, motorcycle
TrafficArea – Function	driving lane	motorway_entry
TrafficArea – SurfaceMaterial	asphalt	concrete

TransportationComplex is a subclass of _TransportationObject and of the root class _CityObject. The geomet-rical representation of the TransportationComplex varies through the different levels of detail. Since Transporta-tionComplex is a subclass of _CityObject and hence a feature, it inherits the attribute gml:name. The street name is also stored within the gml:name property of the Road feature.

In the coarsest LOD0 the transportation complexes 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 applica-tion 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 TransportationComplex provides an explicit surface geometry, reflecting the actual shape of the object, not just its centerline. In LOD2 to LOD4, it is further subdivided thematically into TrafficAreas, which are used by transportation, such as cars, trains, public transport, airplanes, bicycles or pedestrians and in AuxiliaryTrafficAreas, which are of minor importance for transportation purposes, for example road markings, green spaces or flower tubs. The different representations of a TransportationComplex for each LOD are illus-trated in Fig. 60.



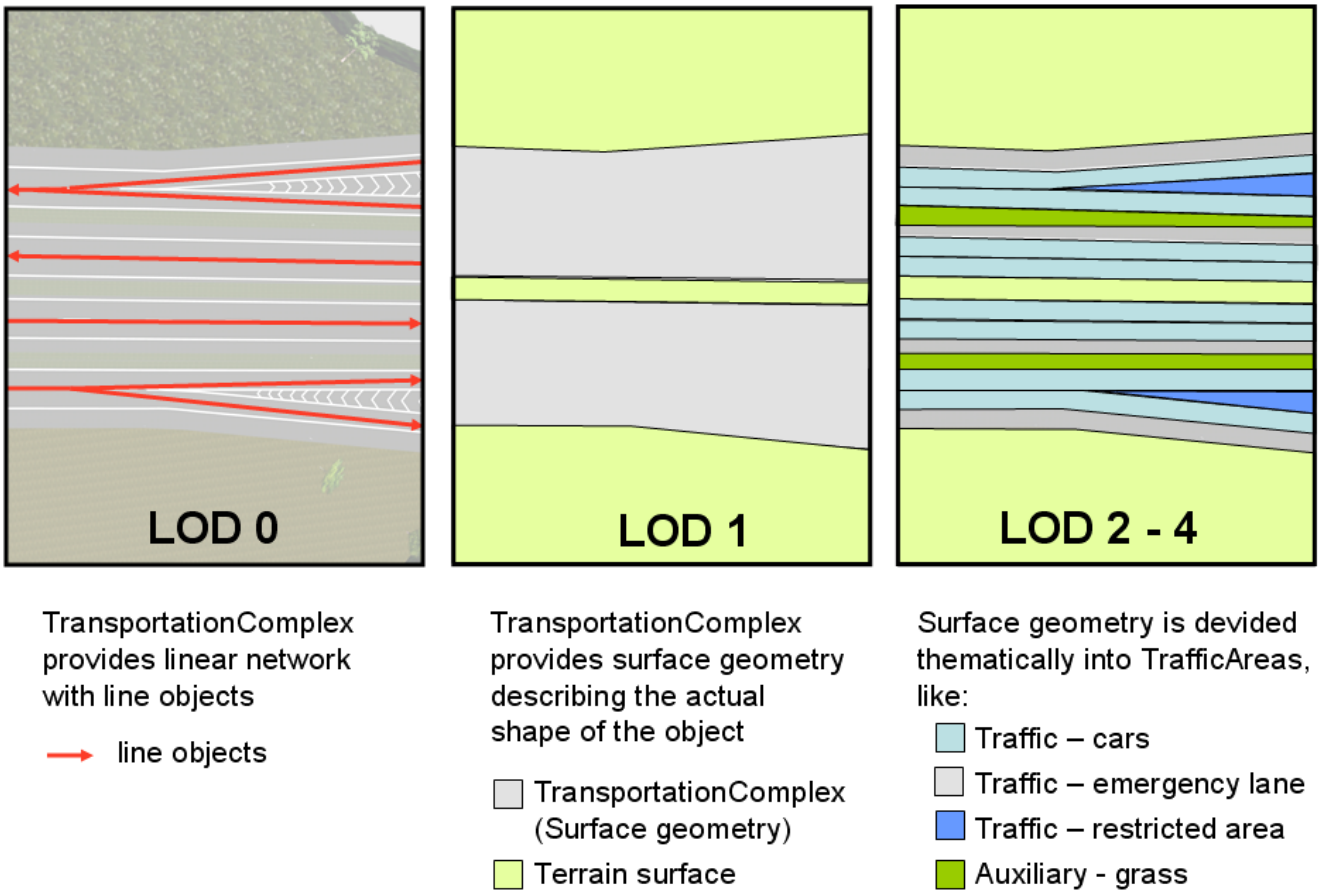


Figure 57. TransportationComplex in LOD0, 1, and 2-4 (example shows part of a motorway) (source: Rheinmetall Defence Electronics).

In LOD0 areal transportation objects like squares should be modelled in the same way as in GDF, the ISO standard for transportation networks, which 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 (cf. chapter 10.12) can be used. Moreover, further objects of interest can be added from other information systems by the use of ExternalReferences (see chapter 6.11). 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 using the concept of dictionaries (see chapter 6.6). These directories may be country- or user-specific (especially for country-specific road signs and signals).

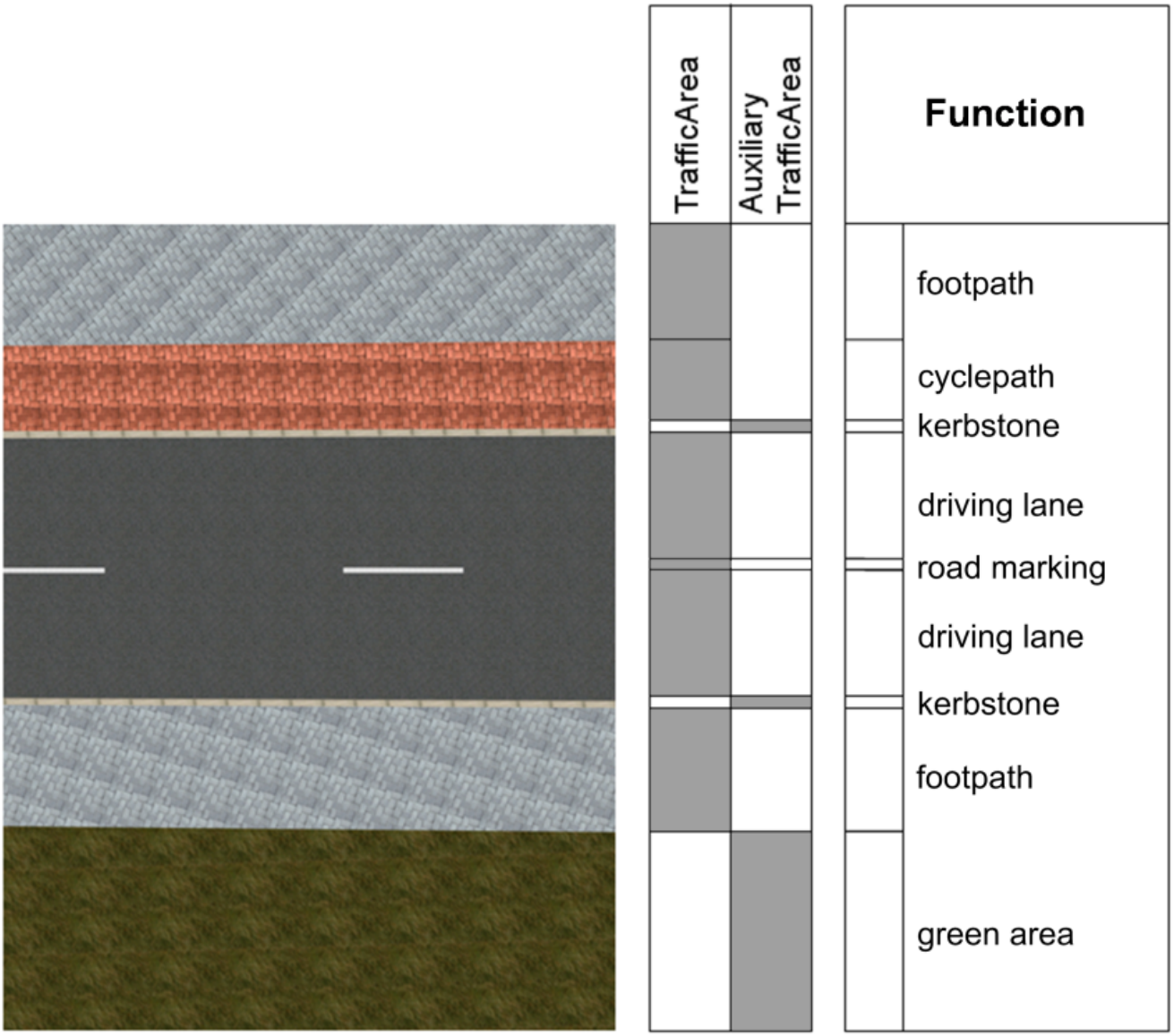


Figure 58. TransportationComplex in LOD 2-4: representation of a road with a complex cross-section profile (example shows urban road) (source: Rheinmetall Defence Electronics).

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.

NOTE insert fig62

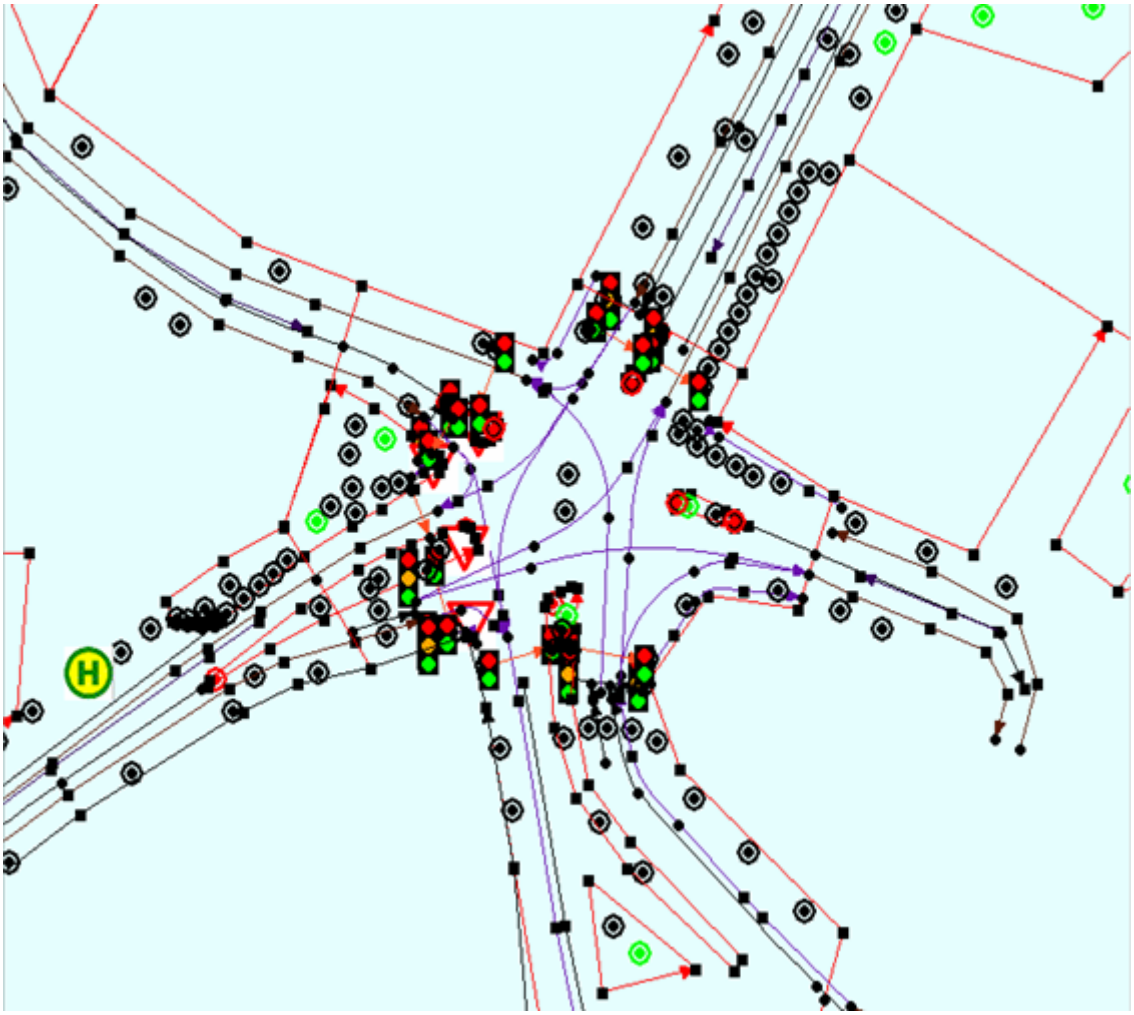


Figure 59. Complex urban intersection (left: linear transportation network with surface descriptions and external references, right: generated scene) (source: Rheinmetall Defence Electronics).

11.7.1. Transporatation Complex

AbstractTransportationObjectType, _TransportationObject

NOTE	insert AbstractTransportationObjectType, _TransportationObject UML

_TransportationObject represents the abstract superclass for transportation objects. Future extensions of the CityGML transportation model shall be modelled as subclasses of this class.

TransportationComplexType, TransportationComplex

NOTE	insert TransportationComplexType, TransportationComplex UML

This type and element describe transportation complexes like roads or railways which may be aggregated from different thematic components (traffic areas, e.g. pedestrian path, and auxiliary traffic areas). As a subclass of _CityObject, TransportationComplex inherits all attributes and relations, in particular an id, names, external references, and generalisation relations. Furthermore, it represents the superclass for thematically distinct types of transportation complexes.

11.7.2. Subclasses of Transportation Complexes

TrackType, Track

NOTE	insert TrackType, Track UML

A Track is a small path mainly used by pedestrians. It is a subclass of TransportationComplex and thus inherits all its attributes and relations.

RoadType, Road

NOTE	insert RoadType, Road UML

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. It is a subclass of TransportationComplex and thus inherits all its attributes and relations.

RailwayType, Railway

NOTE	insert RailwayType, Railway UML

Railway represents routes that are utilised by rail vehicles like trams or trains. It is a subclass of Transportation-Complex and thus inherits all its attributes and relations.

11.7.3. SquareType, Square

NOTE	insert SquareType, Square UML

A Square is an open area commonly found in cities (e.g. a plaza, market square). It is a subclass of Transporta-tionComplex and thus inherits all its attributes and relations.

11.7.4. Subdivisions of Transportation Complexes

TrafficAreaType, TrafficArea

NOTE	insert TrafficAreaType, TrafficArea UML

AuxiliaryTrafficAreaType, AuxiliaryTrafficArea

NOTE	insert AuxiliaryTrafficAreaType, AuxiliaryTrafficArea UML

11.8. Vegetation Objects

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 distribu-tion, structure and diversification can be made. Habitats can be analysed and impacts on the fauna can be de-rived. The vegetation model may be used as a basis

for simulations of, for example forest fire, urban aeration or micro climate. The model could be used, for example to examine forest damage, to detect obstacles (e.g. con-cerning air traffic) or to perform analysis tasks in the field of environmental protection. The vegetation model of CityGML is defined by the thematic extension module Vegetation (cf. chapter 7).

The vegetation model of CityGML distinguishes between solitary vegetation objects like trees and vegetation areas, which represent biotopes like forests or other plant communities (Fig. 63). 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. The UML diagram of the vegetation model is depicted in Fig. 64, for the XML schema definition see below and annex A.12.

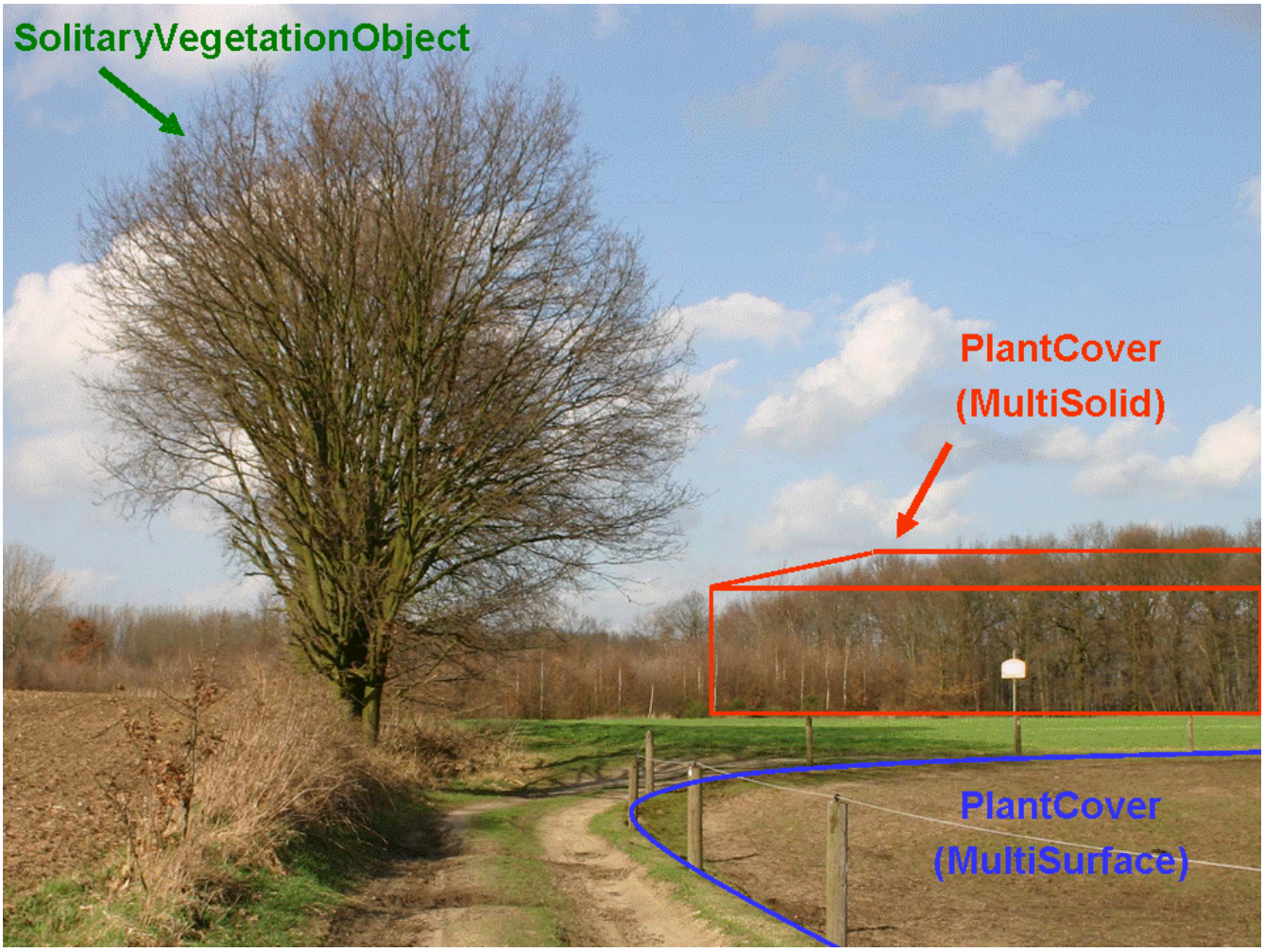


Figure 60. Example for vegetation objects of the classes SolitaryVegetationObject and PlantCover (graphic: District of Recklinghausen).

A SolitaryVegetationObject may have the attributes class, function, usage, species height, trunkDiameter and crownDiameter. The attribute class contains the classification of the object or plant habit, e.g. tree, bush, grass, and can occur only once (see chapter 10.8.4 and annex C.7). The attribute species defines the species' name, for example "Abies alba", and can occur at most once (see chapter 10.8.4 and annex C.7). The optional attributes function and usage denotes the intended respectively real purpose of the object, for example botanical monu-ment, 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 crownDiame-ter 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).

A PlantCover feature may have the attributes class, function, usage and averageHeight. The plant community of a PlantCover is represented by the attribute class. The values of this attribute can be specified in a code list (cf. chapter 10.8.4 and annex C.7) 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 corre-sponding 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 attribute averageHeight denotes the average relative vegetation height.

Since both SolitaryVegetationObject and PlantCover are derived fom _CityObject, they inherit all attributes of a city object, in particular a name (gml:name) and an ExternalReference to a corresponding object in an external information system, which may contain botanical information from public environmental agencies (see chapter 6.7).

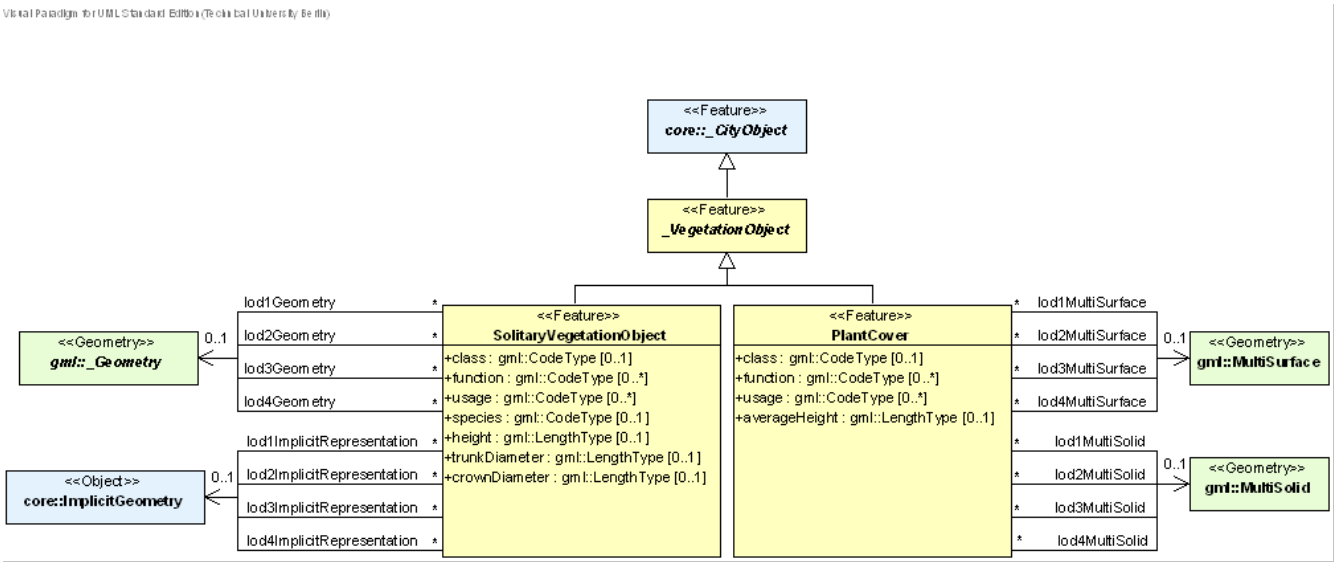


Figure 61. UML diagram of vegetation objects in CityGML. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Vegetation module.

The geometry of a SolitaryVegetationObject may be defined in LOD 1-4 explicitly by a GML geometry having absolute coordinates, or prototypically by an ImplicitGeometry (cf. chapter 8.2). Solitary vegetation objects probably are one of the most important features where implicit geometries are appropriate, since the shape of most types of vegetation objects, such as trees of the same species, can be treated as identical in most cases. Furthermore, season dependent appearances may by mapped using ImplicitGeometry. For visualisation purposes, only the content of the library object defining the object's shape and appearance has to be swapped (cf. Fig. 65).

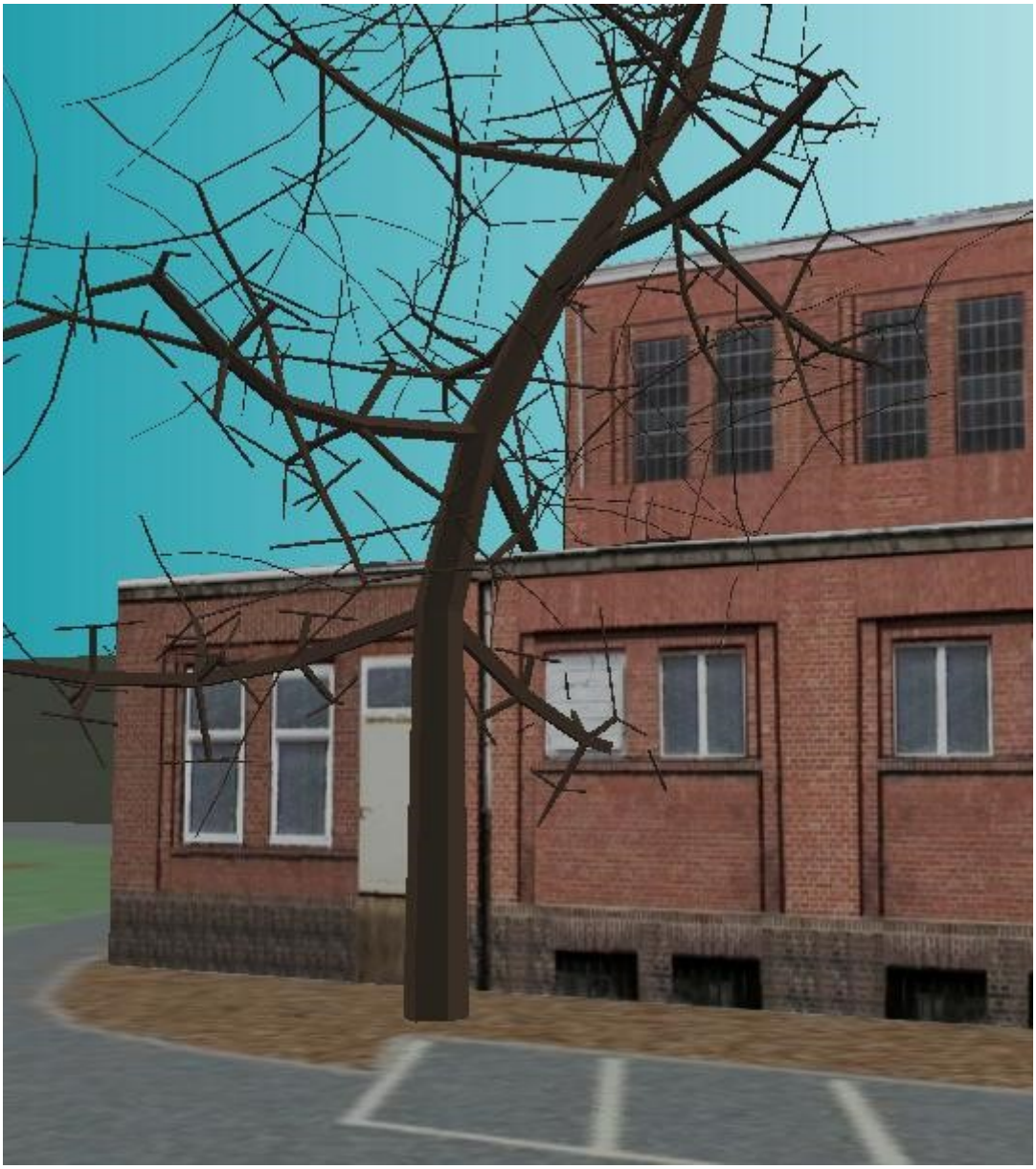


Figure 62. Visualisation of a vegetation object in different seasons (source: District of Recklinghausen).

A SolitaryVegetationObject or a PlantCover may have a different geometry in each LOD. Whereas a SolitaryVegetationObject is associated with the gml:_Geometry class representing an arbitrary GML geometry (by the relation lodXGeometry, X [1..4]), a PlantCover is restricted to be either a gml:MultiSolid or a gml:MultiSurface. An example of a PlantCover modelled as gml:MultiSolid is a ‘solid forest model’, see Fig. 66.

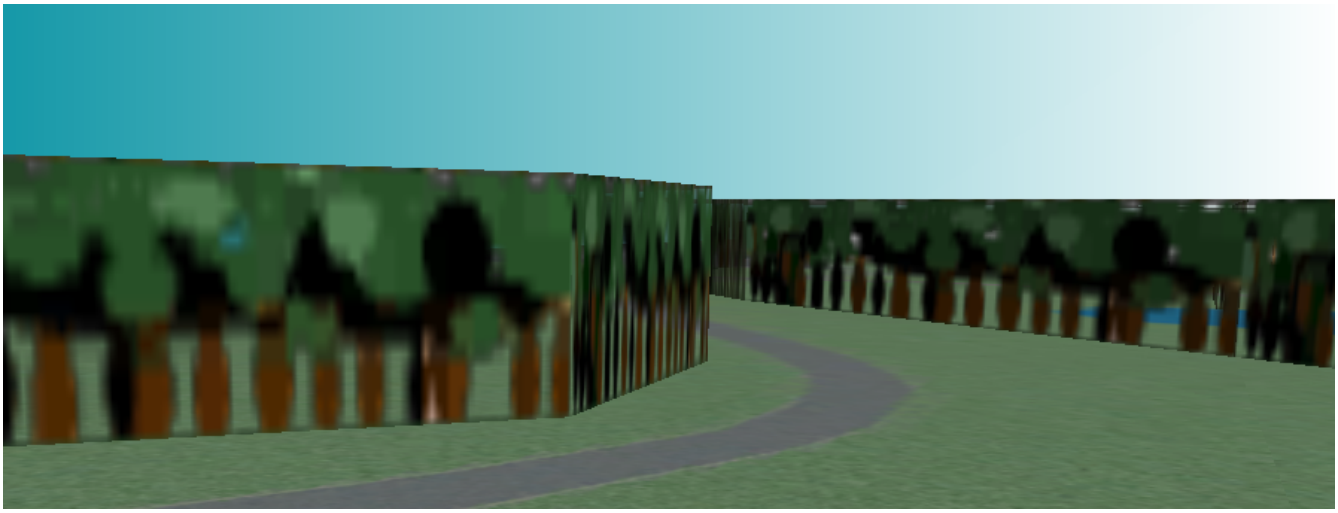


Figure 63. Example for the visualisation/modelling of a solid forest (source: District of Recklinghausen).

XML namespace

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

11.8.1. Vegetation Object

AbstractVegetationObjectType, _VegetationObject

NOTE	insert AbstractVegetationObjectType, _VegetationObject UML

11.8.2. Solitary Vegetation Objects

SolitaryVegetationObjectType, SolitaryVegetationObject

NOTE	insert SolitaryVegetationObjectType, SolitaryVegetationObject UML

11.8.3. Plant cover objects

PlantCoverType, PlantCover

NOTE	insert PlantCoverType, PlantCover UML

11.8.4. Code lists

The attributes class, function, and usage of the features PlantCover and SolitaryVegetationObject as well as the attribute species of the feature SolitaryVegetationObject are specified as gml:CodeType. The values of these properties can be enumerated in code lists. Proposals for corresponding code lists can be found in annex C.7.

11.8.5. Example CityGML dataset

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. The attribute values of the class and species attributes are taken from code lists proposed by the SIG 3D which are presented in annex C.7.

NOTE include examples, GML or other?

11.9. City Furniture

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

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

Figure 64. 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 65. Real situation showing lanterns and delimitation stakes (left). In the right image they are modelled as CityFurniture ob-jects with ImplicitGeometry representations (source: 3D city model of Barkenberg).

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

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

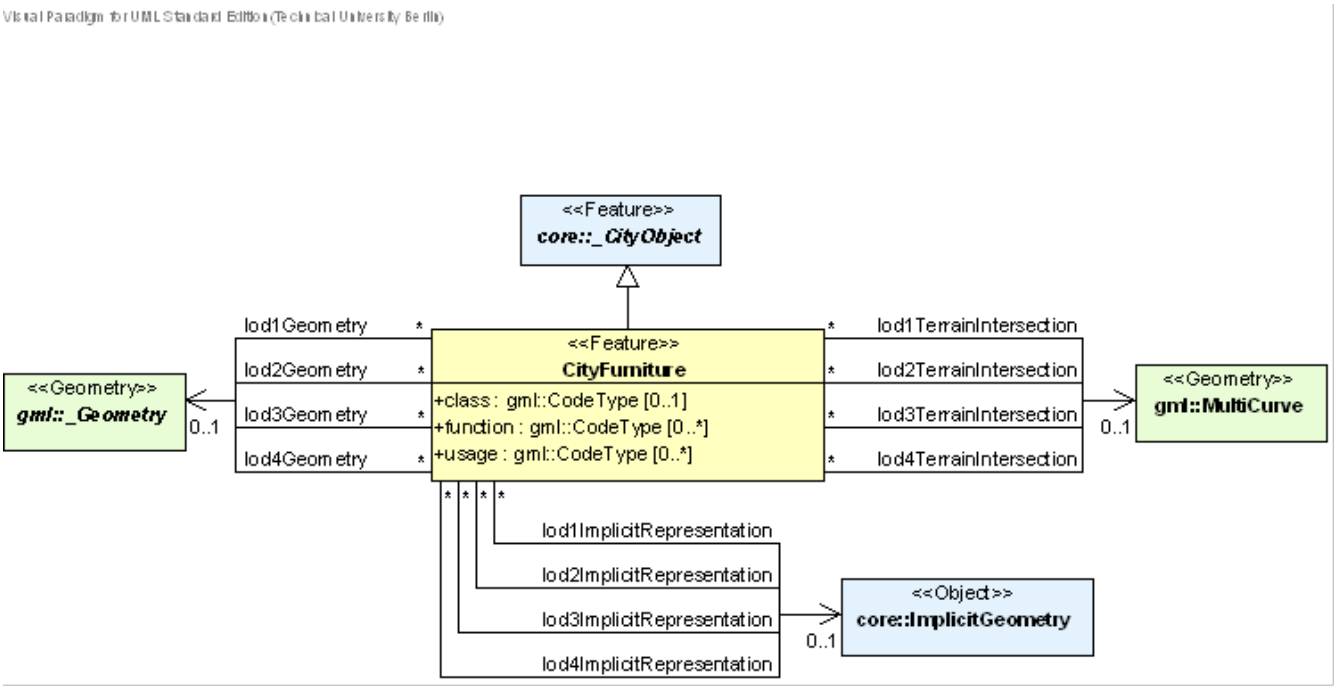


Figure 66. UML diagram of city furniture objects in CityGML. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML CityFurniture module.

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

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

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

XML namespace

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

11.9.1. City furniture object

CityFurnitureType, CityFurniture

NOTE	insert CityFurnitureType, CityFurniture UML

11.9.2. Code lists

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

11.9.3. Example CityGML dataset

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

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

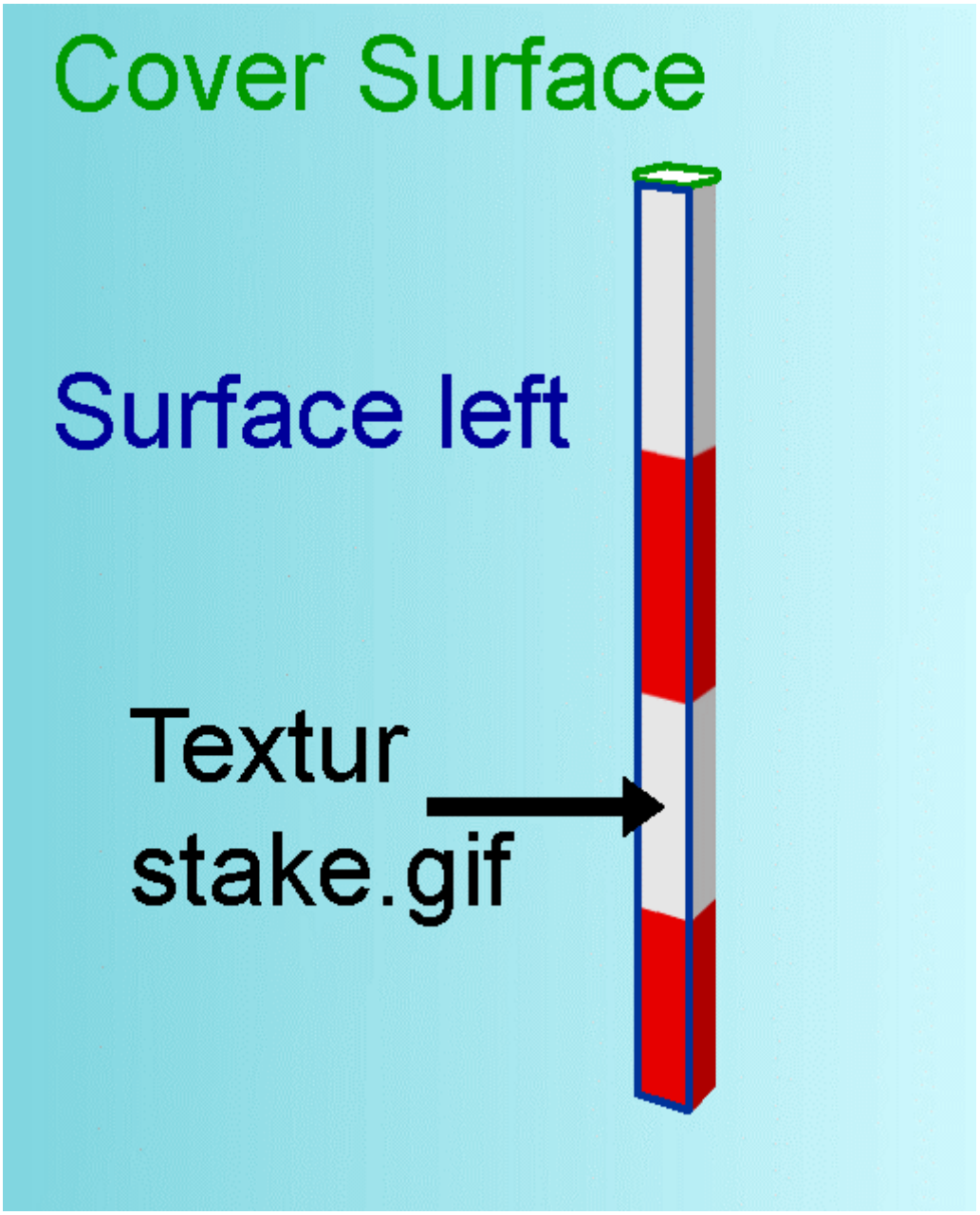


Figure 67. Example of a simple city furniture object (source: District of Recklinghausen).

NOTE insert example - GML?

11.10. Land Use

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

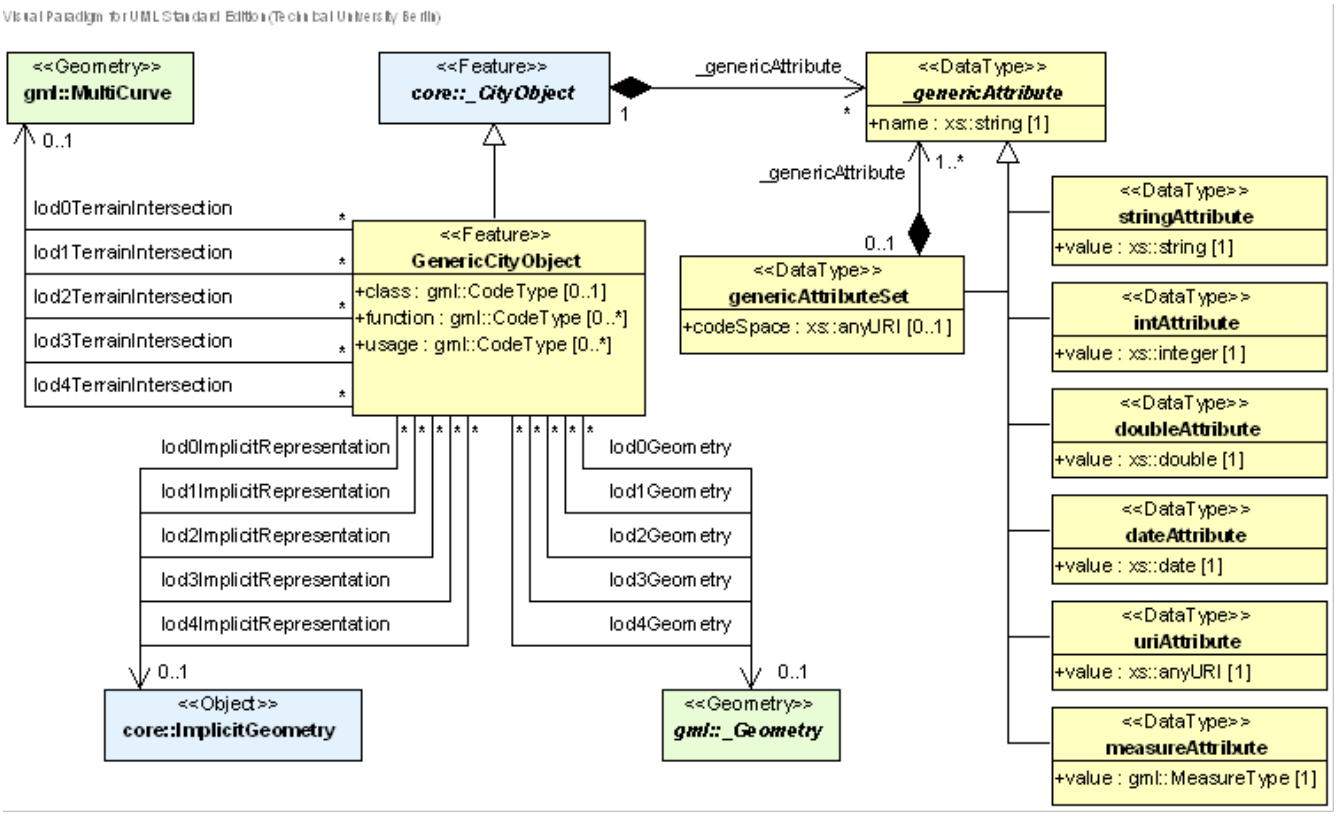


Figure 68. UML diagram of land use objects in CityGML. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML LandUse module.

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

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

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

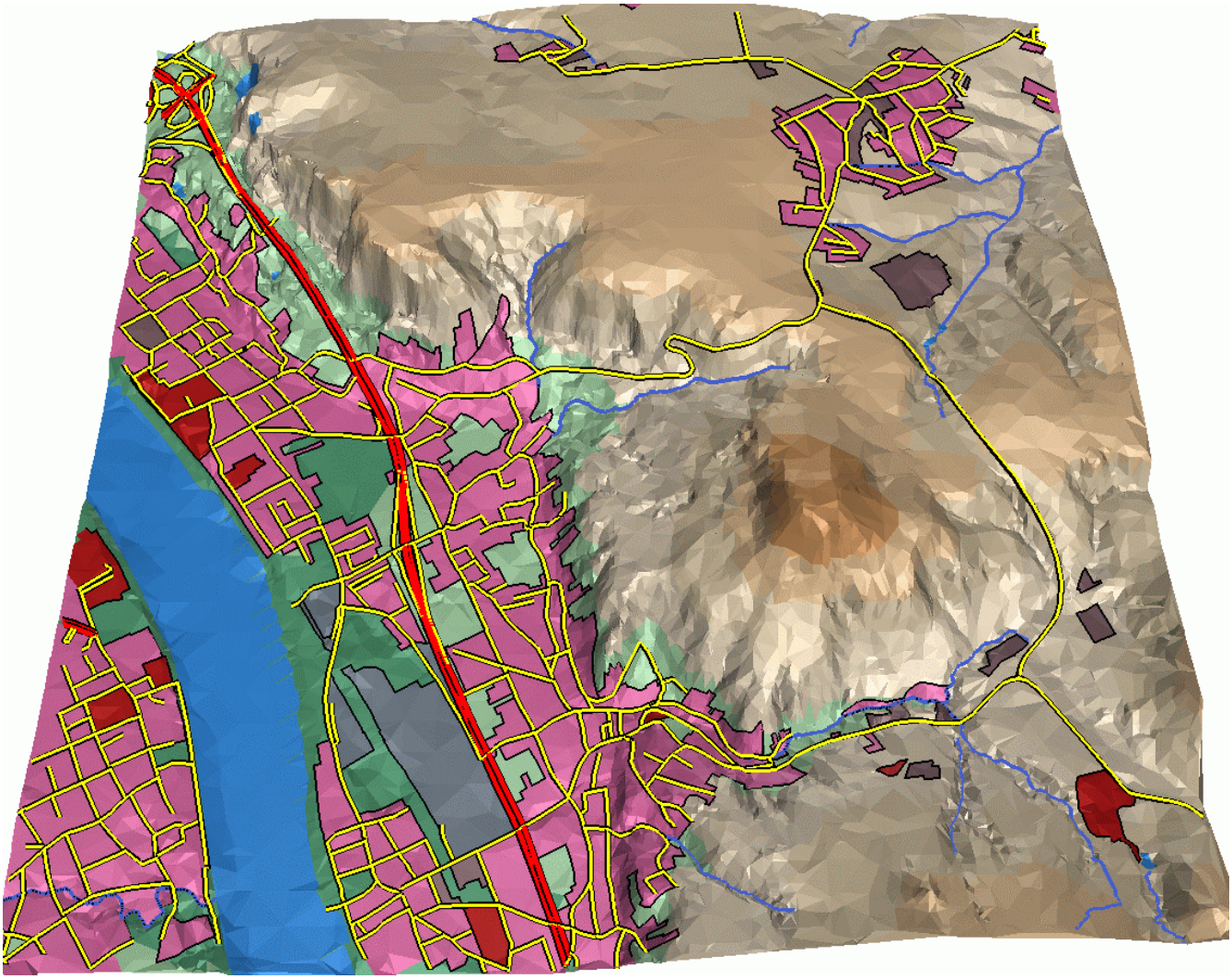


Figure 69. LOD0 regional model consisting of land use objects in CityGML (source: IGG Uni Bonn).

XML namespace

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

11.10.1. Land use object

LandUseType, LandUse

NOTE	insert LandUseType, LandUse UML

11.10.2. Code lists

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

11.11. City Object Groups

The CityGML grouping concept has been introduced in chapter 6.8. CityObjectGroups are modelled using the Composite Design Pattern from software engineering (cf. Gamma et al. 1995): CityObjectGroups aggregate CityObjects and furthermore are defined as special CityObjects. This implies that a group may become a member of another group realizing a recursive aggregation schema. However, in a CityGML instance document it has to be ensured (by the generating application) that no cyclic groupings are included. Fig. 73 shows the UML dia-gram for the class CityObjectGroup, for the XML schema see annex A.6. The grouping concept of CityGML is defined by the thematic extension module CityObjectGroup (cf. chapter 7).

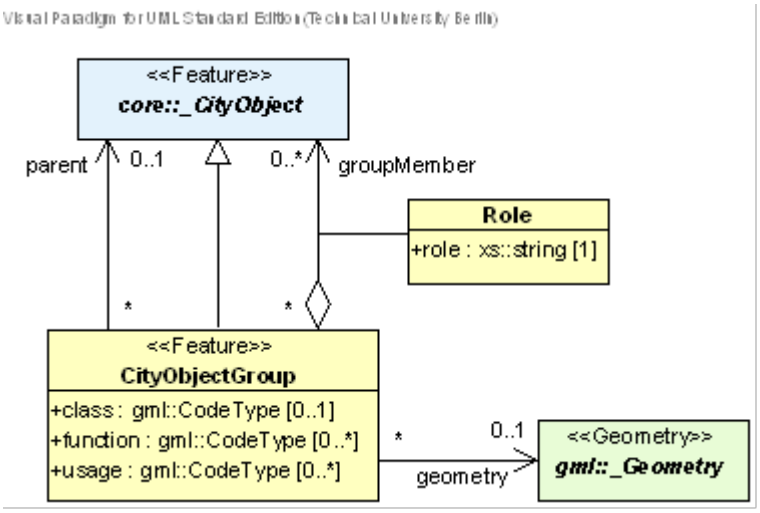


Figure 70. UML diagram of city object groups in CityGML. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML CityObjectGroup module.

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

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

XML namespace

The XML namespace of the CityGML CityObjectGroup module is identified by the Uniform Resource Identifier (URI) http://www.opengis.net/citygml/cityobjectgroup/2.0. Within the XML Schema

definition of the CityOb-jectGroup module, this URI is also used to identify the default namespace.

11.11.1. City object group

CityObjectGroupType, CityObjectGroup

NOTE	insert CityObjectGroupType, CityObjectGroup UML

11.11.2. Code lists

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

11.12. Generic city objects and attributes

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

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

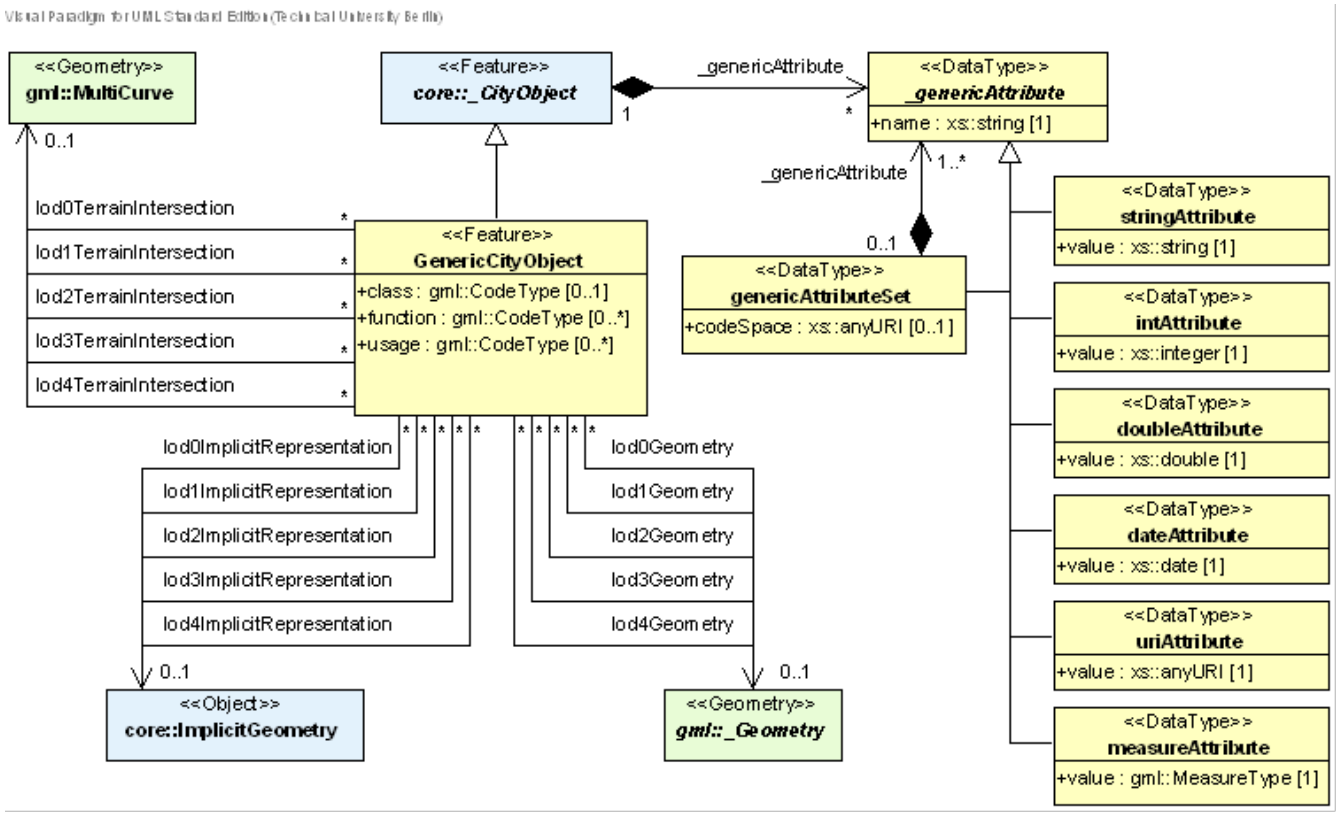


Figure 71. UML diagram of generic objects and attributes in CityGML. Prefixes are used to indicate XML namespaces associated with model elements. Element names without a prefix are defined within the CityGML Generics module.

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

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

In order to model generic attributes, the abstract base class _CityObject defined within the CityGML Core module is augmented by the additional property element _genericAttribute using CityGML's Application Do-main Extension mechanism (cf. chapter 6.12). By this means, each thematic subclass of _CityObject inherits this property and, thus, may be assigned an arbitrary number of generic attributes in order to represent additional properties of features not represented by the explicitly modelled thematic classes of the CityGML data model.

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

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

The geometry of a GenericCityObject can either be an explicit GML3 geometry or an ImplicitGeometry (see chapter 8.2). In the case of an explicit geometry the object can have only one geometry for each LOD, which may be an arbitrary 3D GML geometry object (class gml:_Geometry, which is the base class of all GML geome-tries, lodXGeometry, X [0..4]). Absolute coordinates according to the reference system of the city model must be given for the explicit geometry. In the case of an ImplicitGeometry, a reference point (anchor point) of the object and optionally a transformation matrix must be given. In order to compute the actual location of the object, the transformation of the local coordinates into the reference system of the city model must be processed and the anchor point coordinates must be added. The shape of an ImplicitGeometry can be given as an external resource with a proprietary format, e.g. a VRML or DXF file from a local file system or an external web service. Alternatively the shape can be specified as a 3D GML3 geometry with local cartesian coordinates using the property relativeGeometry (further details are given in chapter 8.2).

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

XML namespace

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

11.12.1. Generic city object

GenericCityObjectType, GenericCityObject

NOTE	insert GenericCityObjectType, GenericCityObject UML

11.12.2. Generic attributes

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

GenericAttributeSetType, genericAttributeSet

NOTE	insert GenericAttributeSetType, genericAttributeSet UML

11.12.3. Code lists

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

11.13. Application Domain Extensions (ADE)

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 mod-elled 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 be-tween 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 appli-cation 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 de-fined by the CityGML schema. This is the case, if the application data could be represented by addi-tional 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 this specifica-tion. Generic attributes and objects have already been introduced as a first possibility to support the exchange of application specific data (see section 10.12). Whereas they allow to extend CityGML without changing its XML schema definition, 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 appli-cation 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 occur-rence of generic objects and attributes cannot be validated by an XML parser. This may reduce seman-tic 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 XML 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 interopera-bility 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 (see below). 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).

11.13.1. Technical principle of ADEs

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

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

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

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

   In this case, extension of the CityGML feature type is not being realised by the inheritance mechanism of XML schema. Instead, every CityGML feature type provides a "hook" in its XML schema definition, that allows to attach additional properties to it by ADEs. This "hook" is implemented as a GML proper-ty of the form

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

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

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

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

11.13.2. Example ADE

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

The XSD schema which has to be defined to implement this model declares a new namespace for the noise extension (http://www.citygml.org/ade/noise_de/2.0). Additionally, the namespaces of the extended CityGML modules are declared (for corresponding prefixes see chapter 4.3 and chapter 7), and the respective schema definition files are imported. The XML schema adds the elements buildingReflectionCorrection and buildingHabitants, both being members of the substitution group

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

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

NOTE insert example here (GML?)

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

NOTE insert example here (GML?)

Code lists

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

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

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

then the attribute value can only be interpreted as a simple text token and validation requires additional knowledge.

The governance of code lists is decoupled from the governance of the CityGML schema and specification. Accordingly, code lists can be specified by any organisation or information community according to their infor-mation needs. There shall be one authority per codeSpace and hence per code list who is in charge of the con-tents and the maintenance of the code list. As a result, the code list values are managed outside the CityGML schema. Thus, in contrast to a fixed enumeration enforced by the CityGML schema, changes to a code list do not require a revision of the CityGML schema and specification.

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

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

NOTE Issue - should we use GML in the examples?

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

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

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

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

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

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

Annex A: Conformance Class Abstract Test Suite (Normative)

NOTE	Ensure that there is a conformance class for each requirements class and a test for each requirement (identified by requirement name and number)

A.1. Conformance Class A

A.1.1. Requirement 1

Test id:	/conf/conf-class-a/req-name-1
Requirement:	/req/req-class-a/req-name-1
Test purpose:	Verify that…
Test method:	Inspect…

A.1.2. Requirement 2

Annex B: Title ({Normative/Informative})

NOTE	Place other Annex material in sequential annexes beginning with "B" and leave final two annexes for the Revision History and Bibliography

Annex C: Revision History

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

Annex D: Bibliography

NOTE

Example Bibliography (Delete this note).

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

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

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

– Actual References:

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

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

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