PFG (2020) 88:43–61
https://doi.org/10.1007/s41064-020-00095-z

ORIGINAL ARTICLE

CityGML 3.0: New Functions Open Up New Applications

Tatjana Kutzner[1] · Kanishk Chaturvedi[1] · Thomas H. Kolbe[1]

Received: 18 September 2019 / Accepted: 24 January 2020 / Published online: 26 February 2020

Abstract
The development of the next major version 3.0 of the international OGC standard CityGML is nearing its end. CityGML 3.0 will come up with a variety of new features and revisions of existing modules that will increase the usability of CityGML for more user groups and areas of application. This includes a new space concept, a revised level-of-detail (LOD) concept, the representation of time-dependent properties, the possibility to manage multiple versions of cities, the representation of city objects by point clouds, an improved modelling of constructions, the representation of building units and storeys, an improved representation of traffic infrastructure as well as a clear separation of the conceptual model and the data encodings that allow for providing further encoding specifications besides GML. This paper gives an overview of these new and revised concepts, and illustrates their application through selected use cases.

Keywords CityGML 3.0 · 3D city models · Space concept

Zusammenfassung
CityGML 3.0: Neue Funktionen eröffnen neue Anwendungen. Die Entwicklung der nächsten Hauptversion 3.0 des internationalen OGC-Standards CityGML nähert sich dem Ende. CityGML 3.0 wird mit einer Vielzahl an neuen Funktionen und der Überarbeitung bestehender Module aufwarten, die die Benutzerfreundlichkeit von CityGML für weitere Benutzergruppen und Anwendungsbereiche verbessern. Dazu gehören ein neues Space-Konzept, ein überarbeitetes Level-of-Detail (LOD)-Konzept, die Darstellung von zeitabhängigen Eigenschaften, die Möglichkeit, mehrere Versionen von Stadtmodellen gleichzeitig zu verwalten, die Darstellung von Stadtobjekten durch Punktwolken, eine verbesserte Modellierung von sonstigen Bauwerken, die Darstellung von Gebäudeeinheiten und Etagen, eine verbesserte Darstellung der Verkehrsinfrastruktur sowie eine klare Trennung des konzeptuellen Modells von der Datenhaltung, die es erlaubt, neben GML weitere Datenformate bereitzustellen. Dieser Artikel gibt einen Überblick über die neuen und überarbeiteten Konzepte und veranschaulicht ihre Anwendung anhand ausgewählter Beispiele.

1 Overview and Development of CityGML 3.0

Semantic 3D city models are nowadays commonly used for representing the real-world entities of cities and landscapes, such as buildings, bridges, tunnels, transportation objects, city furniture, water bodies, and vegetation. One well-known standard for modelling, storing, and exchanging semantic 3D city models is the international standard CityGML issued by the Open Geospatial Consortium (OGC) (Gröger et al. 2012). The current version 2.0 of the standard was adopted by OGC in March 2012. To increase the usability of CityGML for more user groups and areas of application, the OGC CityGML SWG and the Special Interest Group 3D (SIG 3D) of the initiative Geodata Infrastructure Germany (GDI-DE) started in 2013 to work on the next major version CityGML 3.0.

Since the release of CityGML 2.0, several change requests have been submitted to OGC. Further requirements and ideas were gathered and discussed at an international workshop jointly hosted by OGC and SIG 3D in June

✉ Tatjana Kutzner
kutzner@tum.de

Kanishk Chaturvedi
kanishk.chaturvedi@tum.de

Thomas H. Kolbe
thomas.kolbe@tum.de

[1] Chair of Geoinformatics, Technical University of Munich, 80333 Munich, Germany

2013. Both, the outcomes of this discussion and the change requests, were organised into 14 Work Packages (WPs) that defined the overall work scope for CityGML 3.0. Persons interested in participating could do so as either work package lead, editor, co-editor, contributor, or reviewer. Some WPs consistently progressed and produced useful results, whereas other WPs were not active at all. By the end of 2017, the results of the active WPs were integrated into a consolidated version of the CityGML 3.0 Conceptual Model (Kutzner and Kolbe 2018). Afterwards, a process of refining and resolving open issues took place until the end of 2019. Thus, it is a good point in time now to take a closer look at the new concepts and improvements CityGML 3.0 will offer. The current version of the conceptual UML model as well as the GML encoding (XML schemas) can be freely accessed in the github repositories OGC (2019a) and OGC (2019b), respectively.

The CityGML model has been fully revised to reflect the increasing need for better interoperability with other relevant standards in the field like Industry Foundation Classes (IFC) (ISO 16739 2013), IndoorGML (Lee et al. 2016), Land Administration Domain Model (LADM) (ISO 19152 2012), INSPIRE (European Parliament and Council 2007), as well as with linked data and Semantic Web Technologies like the Resource Description Framework (RDF) (W3C 2014). External object references have been rephrased and are now better aligned to an RDF representation. Like before, each city object can have an arbitrary number of references to other objects in other datasets or databases, but these can now be additionally qualified by a relation type (that can point to a definition from an external ontology, e.g. the sameAs relation from OWL) given by an additional URI and allow for mapping to RDF triples.

The CityGML 3.0 standard will consist of two parts: the CityGML 3.0 Conceptual Model specification, which is planned to be released early 2020, and the CityGML 3.0 GML Encoding specification, which is to be published a couple of months after. Further encoding specifications (e.g. relational database schema, JSON-based representation) may follow in the future. The CityGML 3.0 Conceptual Model defines 17 modules as shown in Fig. 1. All modules from CityGML 2.0 are part of CityGML 3.0. In addition, the new modules *Dynamizer*, *Versioning*, *PointCloud*, and *Construction* were introduced, and the modules *Core*, *Generics*, *Building*, and *Transportation* have been revised. The other modules have been updated to work with the new Core module.

CityGML 3.0 applies a model-driven approach in the creation of the data model and exchange formats. Over the last decade, this approach has become the standard procedure for defining geospatial application schemas. The model-driven approach involves two steps: (1) the definition of data models at the conceptual level, which is commonly done using the modelling language UML (ISO 19505-2 2012), and (2) the automatic derivation of transfer formats from these UML models by applying predefined transformation rules. This approach has been applied, e.g. in the development of the European INSPIRE Data Specifications (JRC 2014) and the German AFIS–ALKIS–ATKIS (AAA) Reference Model (AdV 2009). The following tools are used to implement the model-driven approach: Enterprise Architect (Sparx Systems 2015) for defining the CityGML 3.0 UML model and ShapeChange (Interactive Instruments 2019) for deriving the GML application schemas. Also, the feature catalogue with a detailed overview and explanation of all classes, attributes, and relationships is derived automatically.

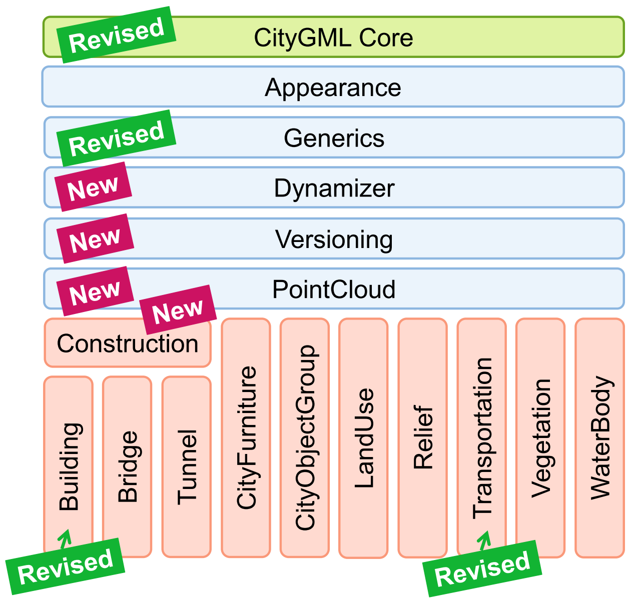


Fig. 1 CityGML 3.0 module overview. The vertical boxes show the different thematic modules. Horizontal modules specify concepts that are applicable to all thematic modules

UML data models developed for the GI domain are usually based on relevant standards from the ISO 191xx series of geographic information standards. The data model of CityGML 3.0 is now based on these standards as well. This means that the geometry types from ISO 19107 as well as the data types from ISO 19103 are used and that the rules for defining application schemas in UML from ISO 19109 are applied. In addition, the transformation rules for converting UML models to GML application schemas from ISO 19136 are applied in the GML encoding.

The application of the ISO-compliant transformation rules inevitably leads to some changes in the XML encoding compared to CityGML 2.0 and 1.0. Even if CityGML 3.0 would not change anything on the conceptual model with regard to CityGML 2.0, the encoding would be slightly different and datasets would have to be converted and software for CityGML 2.0 would have to be adapted for CityGML 3.0. However, all modifications to the new CityGML 3.0 model are carried out in a way to ensure backwards compatibility with CityGML 1.0 and 2.0, i.e. it is possible to transform all CityGML 1.0 and 2.0 datasets into the new model by applying syntactical transformations only. Backwards compatibility is a major requirement for CityGML 3.0 to preserve the investments by anybody providing CityGML tools, datasets, and extensions.

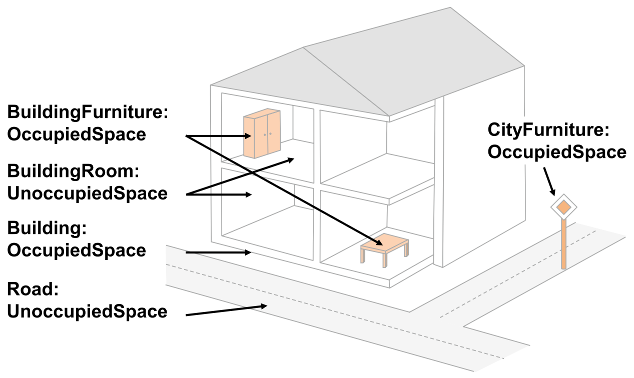

BuildingFurniture: OccupiedSpace
BuildingRoom: UnoccupiedSpace
Building: OccupiedSpace
Road: UnoccupiedSpace
CityFurniture: OccupiedSpace

Fig. 2 Occupied and unoccupied spaces

2 The New CityGML 3.0 Core Module

2.1 The CityGML 3.0 Space Concept

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

To obtain a more precise definition of spaces, they are further subdivided into *physical spaces* and *logical spaces*. Physical spaces are spaces that are fully or partially bounded by physical objects. Buildings and rooms, for instance, are physical spaces as they are bounded by walls and slabs. Traffic spaces of roads are physical spaces as they are bounded by road surfaces against the ground. Logical spaces, in contrast, are spaces that are not necessarily bounded by physical objects, but are defined according to thematic considerations. Depending on the application, logical spaces can also be bounded by non-physical, i.e. virtual boundaries and they can represent aggregations of physical spaces. A building unit, for instance, is a logical space as it aggregates specific rooms to flats, the rooms being the physical spaces that are bounded by wall surfaces, whereas the aggregation as a whole is being delimited by a virtual boundary. Other examples are city districts which are bounded by virtual vertically extruded administrative boundaries; public spaces vs. security zones in airports; or city zones with specific regulations stemming from urban planning. The definition of physical and logical spaces and of corresponding physical and virtual boundaries is in line with the discussion in Smith and Varzi (2000) on the difference between bona fide and fiat boundaries to bound objects. Bona fide boundaries are physical boundaries; they correspond to the physical boundaries of physical spaces in CityGML 3.0. In contrast, fiat boundaries are man-made boundaries; they are equivalent to the virtual boundaries of logical spaces.

Physical spaces, in turn, are further classified into *occupied spaces* and *unoccupied spaces*. Occupied spaces represent physical volumetric objects that occupy space in the urban environment. Examples for occupied spaces are buildings, bridges, trees, city furniture, and water bodies. Occupying space means that some space is blocked by these volumetric objects; for instance, the space blocked by the building in Fig. 2 cannot be used any more for driving through this space or placing a tree on that space. In contrast, unoccupied spaces represent physical volumetric entities that do not occupy space in the urban environment, i.e. no space is blocked by these volumetric objects. Examples for unoccupied spaces are building rooms and traffic spaces. There is a risk of misunderstanding the term OccupiedSpace. However, we decided to use the term anyway, as it is established in the field of robotics for over three decades (Elfes 1989). The navigation of mobile robots makes use of a so-called occupancy map that marks areas that are occupied by matter and, thus, are not navigable for robots.

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

(1) *Spatial partitioning* Semantic objects of either the space type OccupiedSpace or UnoccupiedSpace are subdivided into different parts that are of the same space type as the parent object. Examples are Buildings that can be subdivided into BuildingParts, or Buildings that are partitioned into ConstructiveElements. Buildings as well as BuildingParts and ConstructiveElements

represent OccupiedSpaces. Similarly, Roads can be subdivided into TrafficSpaces and AuxiliaryTrafficSpaces, all objects being UnoccupiedSpaces.

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

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

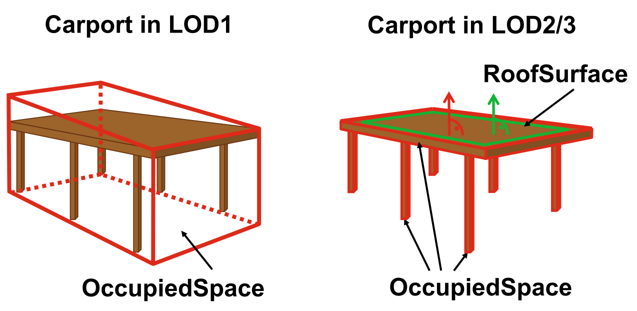


Fig. 3 Representation of a carport as OccupiedSpace in different LODs. The red boxes represent solids, the green area represents a surface. In addition, the normal vectors of the roof solid (in red) and the roof surface (in green) are shown

The classification into OccupiedSpace and UnoccupiedSpace might not always be apparent at first sight. Carports, for instance, represent an OccupiedSpace, although they are not closed and most of the space is free of matter, see Fig. 3. Since a carport is a roofed, immovable structure with the purpose of providing shelter to objects (i.e. cars), carports are frequently represented as buildings in cadastres. Thus, also in CityGML, a carport should be modelled as an instance of the class Building. Since Building is transitively a subclass of OccupiedSpace, a carport is an OccupiedSpace as well. However, only in LOD1, the entire volumetric region covered by the carport would be considered as physically occupied. In LOD1, the occupied space is defined by the entire carport solid (unless a room would be defined in LOD1 that would model the unoccupied part below the roof); whereas in LOD2 and LOD3, the solids represent more realistically the really physically occupied space of the carport. In addition, for all OccupiedSpaces, the normal vectors of the thematic surfaces like the RoofSurface need to point away from the solids, i.e. consistent with the solid geometry.

In contrast, a room is a physically unoccupied space. In CityGML, a room is represented by the class BuildingRoom that is a subclass of UnoccupiedSpace. In LOD1, the entire room solid would be considered as unoccupied space, which can contain furniture and installations, though, as is shown in Fig. 4. In LOD2 and 3, the solid represents more realistically the really physically unoccupied space of the room (possibly somewhat generalised as indicated in the figure). For all UnoccupiedSpaces, the normal vectors of the bounding thematic surfaces like the InteriorWallSurface need to point inside the object, i.e. opposite to the solid geometry.

The concepts of Spaces and Space Boundaries are represented in the UML model of the CityGML 3.0 Core module by introducing the two pivotal abstract classes *AbstractSpace* and *AbstractSpaceBoundary* as shown in Fig. 5.

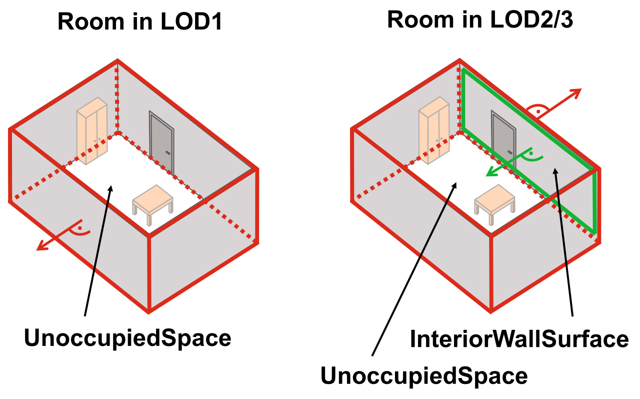


Fig. 4 Representation of a room as UnoccupiedSpace in different LODs. The red boxes represent solids, the green area represents a surface. In addition, the normal vectors of the room solid (in red) and the wall surface (in green) are shown

From the class *AbstractSpace*, the following subclasses are derived: the classes *AbstractPhysicalSpace* and *AbstractLogicalSpace* to classify spaces into physical and logical spaces as well as the classes *AbstractOccupiedSpace* and *AbstractUnoccupiedSpace* to categorise physical spaces into physically occupied and unoccupied spaces. The concrete classes like *Building*, *BuildingRoom*, or *TrafficSpace* are then defined as subclasses of these abstract classes. From the class *AbstractSpaceBoundary*, the class *AbstractThematicSurface* is derived. It is the superclass for all concrete surface classes like *WallSurface*, *ClosureSurface*, *WaterSurface* or *LandUse*. The relation between *AbstractSpace* and *AbstractSpaceBoundary* is represented in the UML model through an association between both classes.

The classification of real-world objects into spaces and space boundaries is solely based on the semantics of these objects and not on their used geometry type, as CityGML

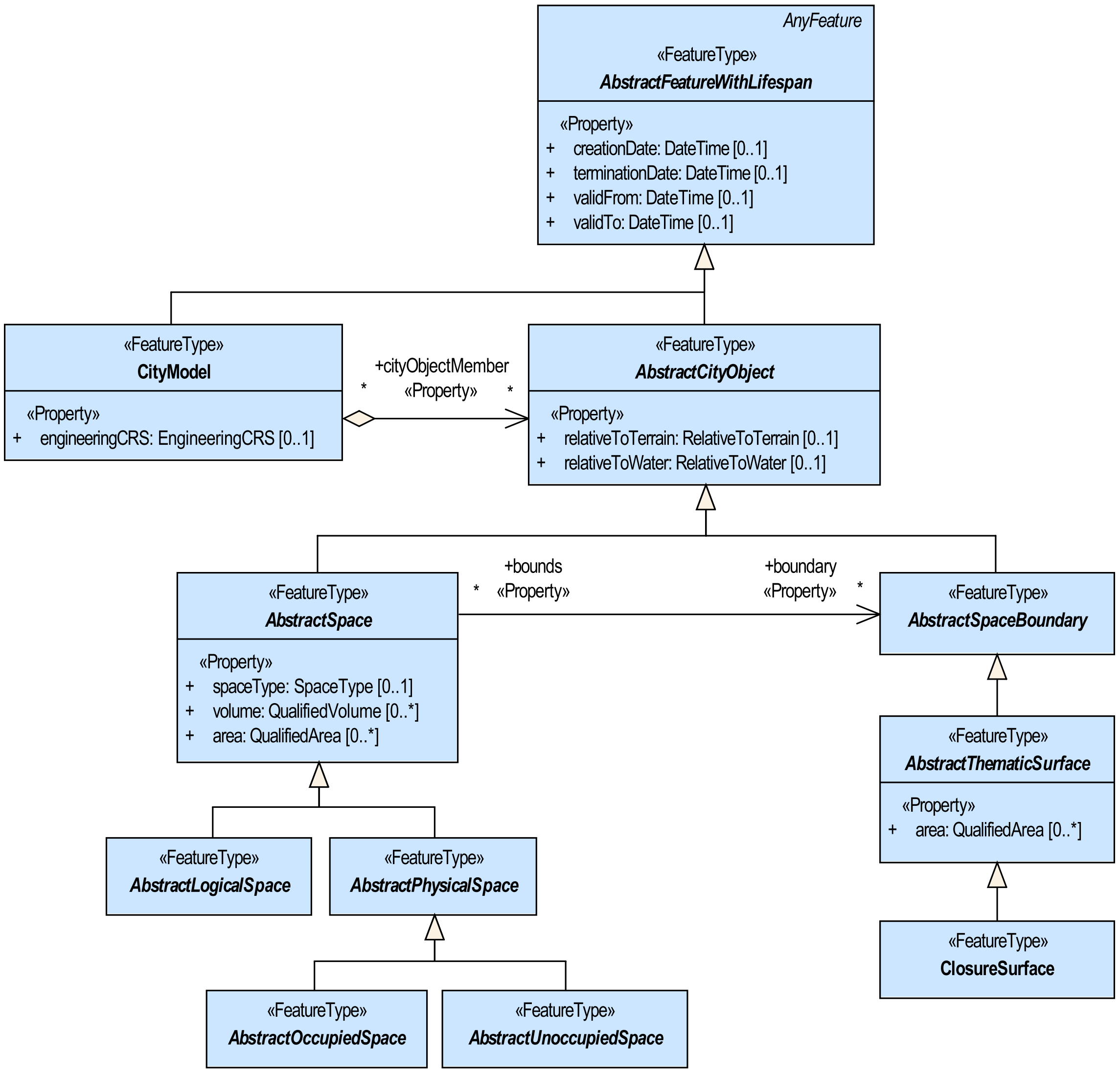


Fig. 5 Excerpt from the CityGML 3.0 Core UML model defining the space concept

3.0 allows various geometrical representations for objects. A building, for instance, can be spatially represented by a 3D solid (e.g. in LOD1), but at the same time, the real-world geometry can also be abstracted by a single point (LOD0), by a 3D point cloud, or by a 3D mesh (LOD3).

The space concept was strongly motivated from urban planning, the work on IndoorGML and indoor navigation as well as the Land Administration Domain Model. By introducing spaces in CityGML, it will become easier to link CityGML with IndoorGML and to apply analytical frameworks from the urban geography domain like Space Syntax (Hillier and Hanson 1984) or from the analyses of urban activities like living, working, or traffic (which are all taking place in or affecting spaces). In addition, the space concept was inspired by the work of Billen et al. (2012) on defining a generic ontology for the urban space. This ontology partitions the universe into spaces that are classified into physical spaces and fictional spaces. Physical spaces can have boundaries and can be divided into sub-spaces. Physical sub-spaces, in turn, are classified into penetrable and non-penetrable physical sub-spaces, depending on whether they can be penetrated by an agent such as a human, an electromagnetic, or a specific behaviour. Real-world objects can then be linked to the physical sub-spaces. Buildings, for instance, are considered to have a non-permeable space, whereas rooms are permeable spaces. The concepts defined in the ontology can be found again in the space concept of CityGML 3.0. Physical and fictional spaces are represented in CityGML 3.0 by the classes *AbstractPhysicalSpace* and *AbstractLogicalSpace*. The non-penetrable and penetrable physical sub-spaces are modelled in CityGML 3.0 by the classes *AbstractOccupiedSpace* and *AbstractUnoccupiedSpace*; thus, buildings, which are considered by Billen et al. (2012) as non-penetrable, are defined as occupied spaces and rooms, which are considered as penetrable, are defined as unoccupied spaces. In contrast to the ontology, where only physical spaces can have boundaries, all spaces can be bounded by thematic surfaces in CityGML 3.0. The space concept corresponds also with the observations made in Yan et al. (2019) on the categorization of spaces into indoor and outdoor spaces to improve indoor/outdoor navigation. The classification in CityGML 3.0 does not go as far as differentiating semi-indoor and semi-outdoor spaces as well, however, CityGML 3.0 can be used to derive this classification. The modelling in CityGML 3.0 does not focus on modelling the navigable space, but on an exact representation of spatial objects such as the carport in LOD2/3 as shown in Fig. 3. Nevertheless, CityGML 3.0 also allows for deriving the navigable space of the carport. This distinguishes the modelling with CityGML 3.0 from the concepts defined in Yan et al. (2019).

The new space concept offers several advantages:

- In CityGML 3.0 all geometric representations are defined in the Core module only. This makes (a) data models of the thematic modules simpler as they no longer need to be associated directly with the geometry classes, and (b) implementation easier as all spatial concepts have only to be implemented once in the Core module and all thematic modules like Building, Relief, WaterBody, etc. are inheriting them.
- The space concept supports the expression of explicit topological, geometrical, and thematic relations between spaces and spaces, spaces and space boundaries, and space boundaries and space boundaries. Thus, implementing the checking of geometric-topological consistency will become easier, because most checks can be expressed and performed on the CityGML Core module and then automatically apply to all thematic modules. When a new thematic module [or an Application Domain Extension (ADE)] will be added and its spatial representations will be expressed using space and space boundary classes, the validity checks automatically hold also for the new thematic module (or ADE).
- Some new application areas could be opened up. For example, urban planning, urban analytics, autonomous driving and driver assistance systems, and navigation in general will benefit from the space concept. Also the modelling of the underground (hollow spaces, geological rock layers and their separating surfaces) could be very elegantly done in the future using spaces and space boundaries.
- For the analysis of navigable spaces (e.g. to generate IndoorGML data from CityGML) algorithms can be defined on the level of the Core module. These algorithms will then work with all CityGML feature classes and also ADEs as they are derived from the Core. The same is true for other applications of 3D city models listed in Biljecki et al. (2015) such as visibility analyses including shadow casting or solar irradiation analyses.
- Practitioners and developers do not see much of the space concept, because the space and space boundary classes are just abstract classes. Only elements representing objects from concrete subclasses such as Building, BuildingRoom, or TrafficSpace will appear in CityGML files.

2.2 The CityGML 3.0 LOD Concept

CityGML 3.0 will include a revised Level of Detail (LOD) concept which comprises a central definition of all geometries in the Core module and the representation of the interior of city objects at any level of detail.

In CityGML 2.0, each module defines the required geometries itself, leading to redundancy, for instance, in

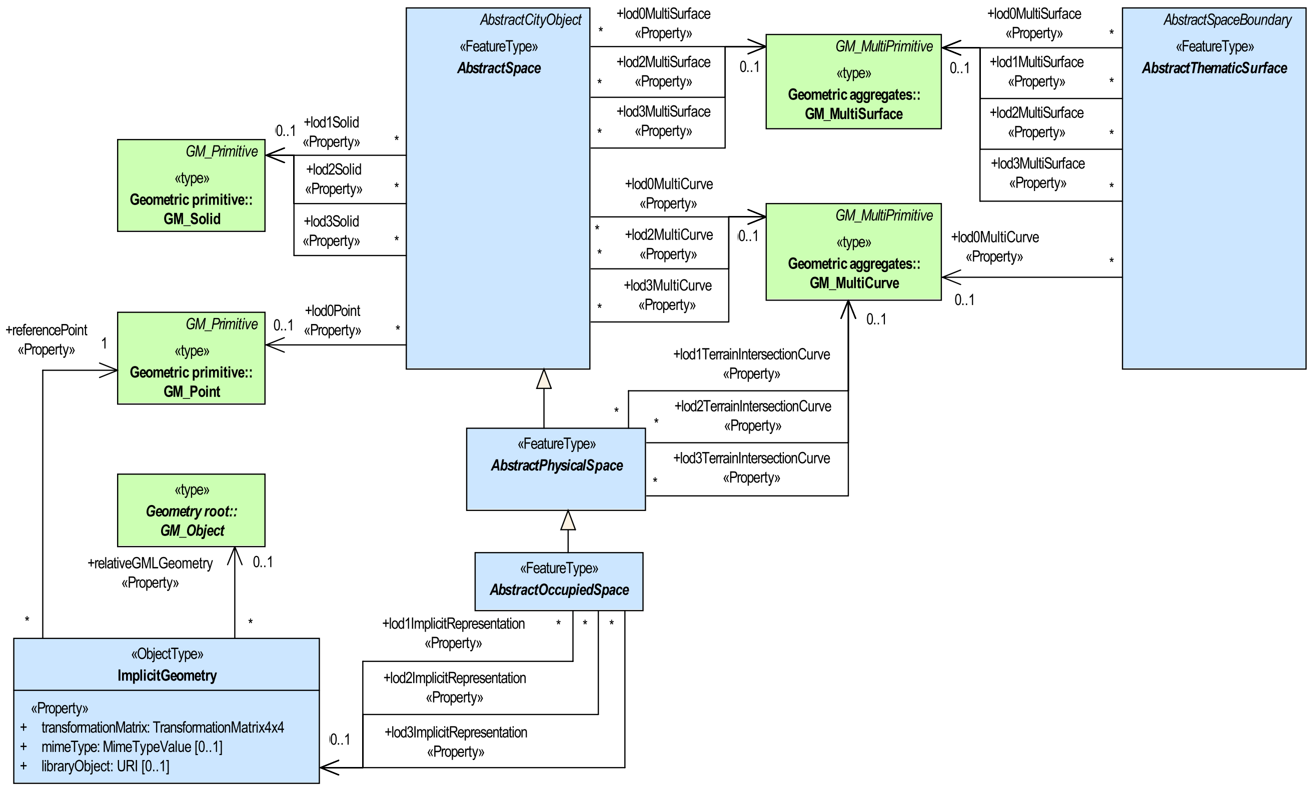

AbstractCityObject
«FeatureType»
AbstractSpace
+lod0MultiSurface «Property»
+lod2MultiSurface «Property»
+lod3MultiSurface «Property»
GM_MultiPrimitive
«type»
Geometric aggregates:: GM_MultiSurface
+lod1Solid «Property»
+lod2Solid «Property»
+lod3Solid «Property»
GM_Primitive
«type»
Geometric primitive:: GM_Solid
+lod0MultiCurve «Property»
+lod2MultiCurve «Property»
+lod3MultiCurve «Property»
GM_MultiPrimitive
«type»
Geometric aggregates:: GM_MultiCurve
AbstractSpaceBoundary
«FeatureType»
AbstractThematicSurface
+lod1MultiSurface «Property»
+lod0Point «Property»
GM_Primitive
«type»
Geometric primitive:: GM_Point
+referencePoint «Property»
+lod1TerrainIntersectionCurve «Property»
+lod2TerrainIntersectionCurve «Property»
+lod3TerrainIntersectionCurve «Property»
«FeatureType»
AbstractPhysicalSpace
«type»
Geometry root:: GM_Object
+relativeGMLGeometry «Property»
«FeatureType»
AbstractOccupiedSpace
+lod1ImplicitRepresentation «Property»
+lod2ImplicitRepresentation «Property»
+lod3ImplicitRepresentation «Property»
«ObjectType»
ImplicitGeometry
«Property»
+ transformationMatrix: TransformationMatrix4x4
+ mimeType: MimeTypeValue [0..1]
+ libraryObject: URI [0..1]

Fig. 6 Excerpt from the CityGML 3.0 Core UML model defining the LOD concept. The geometries in CityGML 3.0 are now associated with the classes in the Core model and no longer need to be specified in each thematic module separately

the Building, Tunnel, and Bridge modules. To overcome these redundancies, nearly all geometry representations are moved from the thematic modules to the Core module and are associated with the semantic concepts of spaces and space boundaries, see Fig. 6. The geometry classes from ISO 19107 are used for defining the various geometric representations. Since all feature types in the thematic modules are defined as subclasses of the space and space boundary classes, they automatically inherit the geometry classes and, thus, no longer require direct associations with them.

Spaces and all its subclasses like Building, Room, and TrafficSpace can now be spatially represented as single points in LOD0, multi-surfaces in LOD0/2/3, solids in LOD1/2/3, and multi-curves in LOD2/3. Space boundaries and all its subclasses such as WallSurface, LandUse, or Relief can now be represented as multi-surfaces in LOD0/2/3 and as multi-curves in LOD2/3.

LOD4, which is used for representing the interior of objects in CityGML 2.0 (like indoor modelling for buildings and tunnels), has been removed; only the LODs 0/1/2/3 will remain. Instead, the interior of objects can be expressed now integrated with the LODs 0/1/2/3. This allows, for instance, the representation of floor plans in LOD0 (Konde et al. 2018). It will even be possible to model the outside shell of a building in LOD1, while representing the interior structure in LOD2 or 3. Details on the changes to the CityGML LOD concept are provided in Löwner et al. (2016).

In addition, the new PointCloud module adds the possibility to use 3D point clouds to represent the geometries of physical spaces and space boundaries.

3 Refinement of Constructions and Buildings

CityGML 3.0 will contain a new Construction module that defines concepts common to all kinds of man-made constructions like buildings, bridges, and tunnels. This means that the module integrates all classes that are similar over different types of constructions and that are defined separately in the Building, Bridge, and Tunnel modules in CityGML 2.0. These are, in particular, the various thematic surfaces like RoofSurface, GroundSurface, or WallSurface, and the openings Door and Window. Figure 7 shows the Construction module. The module defines a class *AbstractConstruction* as subclass of *AbstractOccupiedSpace* which is associated with the different thematic surfaces. Buildings, bridges, and tunnels, in turn, are defined as subclasses of the class *AbstractConstruction*, inheriting in this way automatically the associations with all thematic surfaces. This

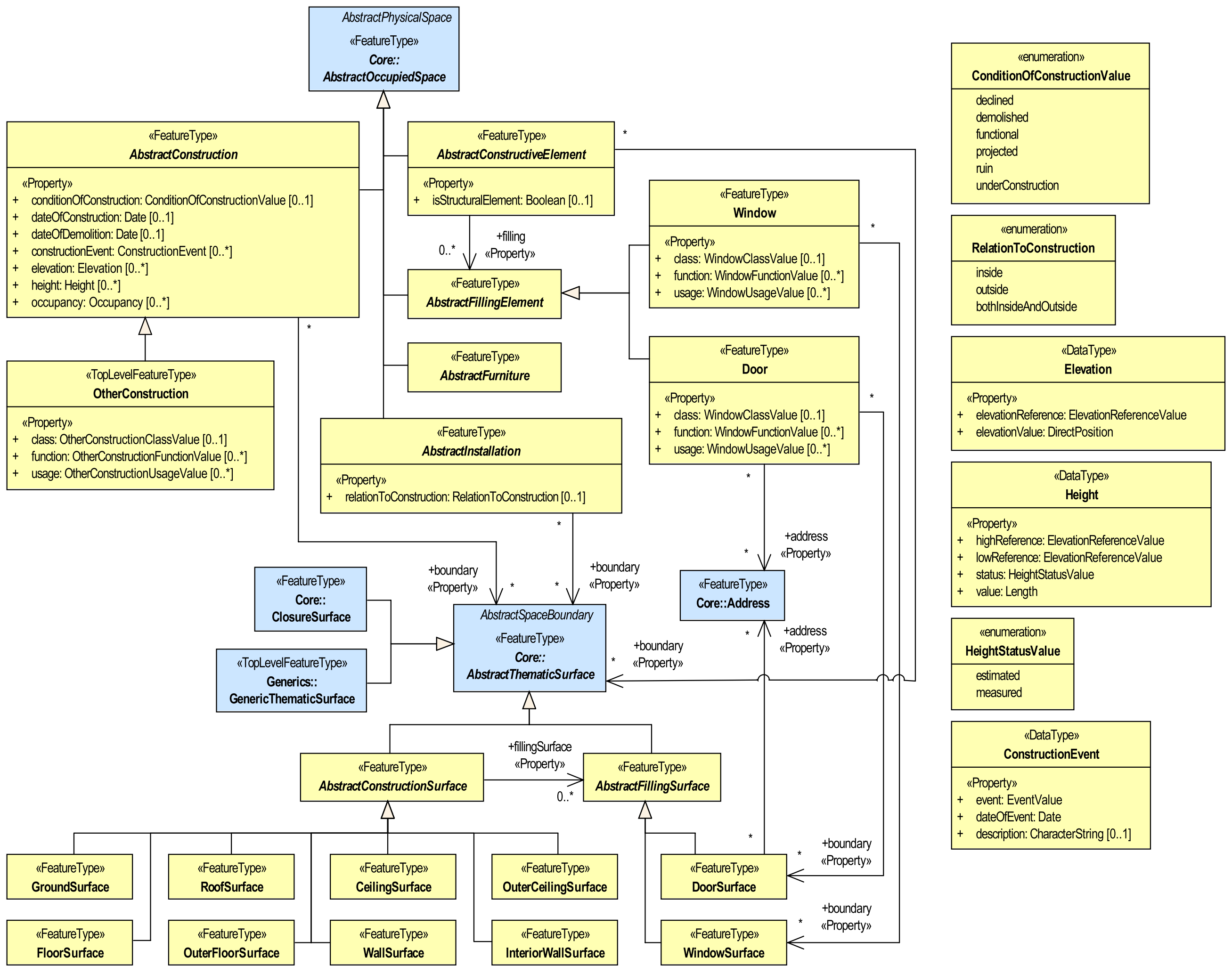


Fig. 7 CityGML 3.0 Construction module

leads to a substantial simplification of the UML models of the Building, Bridge, and Tunnel modules. In addition, for representing man-made structures that are neither buildings, nor tunnels, nor bridges (e.g. large chimneys or city walls), the new class *OtherConstruction* has been introduced as subclass of *AbstractOccupiedSpace*.

To facilitate a more direct mapping of IFC onto CityGML, a new feature type *AbstractConstructiveElement* is introduced and corresponding subclasses *BuildingConstructiveElement*, *BridgeConstructiveElement*, and *TunnelConstructiveElement* are defined in the modules Building (see Fig. 8), Bridge, and Tunnel. These feature types allow for mapping constructive elements from BIM data sets given in the IFC standard (e.g. the IFC classes IfcWall, IfcRoof, IfcBeam, IfcSlab, etc.) onto CityGML. These volumetric components are bounded by the various thematic surfaces RoofSurface, WallSurface, etc. as well. Thus, space and space boundary establish the explicit connection between (volumetric) constructive elements and their thematic boundary surfaces.

The interoperability of CityGML with INSPIRE is improved by adopting attributes from the INSPIRE Building data theme (JRC 2013) which allow for specifying multiple elevation levels and measured heights. In addition, various events and their dates can be specified, such as the date a building permit was issued, the start of renovation, or the end of renovation (see corresponding attributes in the class *AbstractConstruction* in Fig. 7).

Doors and windows have a clearer semantics now. The classes *Window* and *Door* represent now filling elements; in

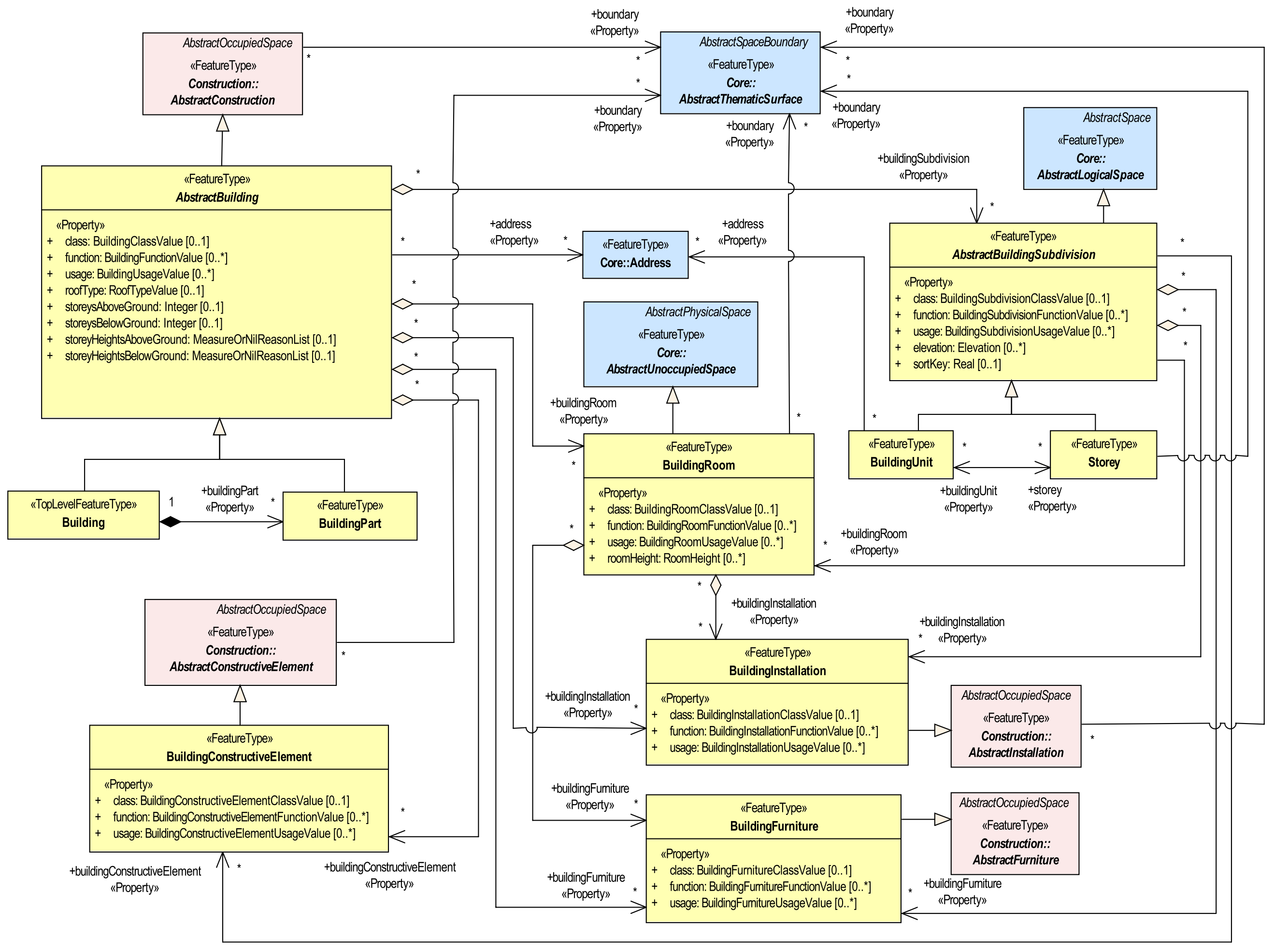

AbstractOccupiedSpace
«FeatureType»
Construction::AbstractConstruction
AbstractSpaceBoundary
«FeatureType»
Core::AbstractThematicSurface
+boundary «Property»
AbstractSpace
«FeatureType»
Core::AbstractLogicalSpace
«FeatureType»
AbstractBuilding
«Property»
+ class: BuildingClassValue [0..1]
+ function: BuildingFunctionValue [0..*]
+ usage: BuildingUsageValue [0..*]
+ roofType: RoofTypeValue [0..1]
+ storeysAboveGround: Integer [0..1]
+ storeysBelowGround: Integer [0..1]
+ storeyHeightsAboveGround: MeasureOrNilReasonList [0..1]
+ storeyHeightsBelowGround: MeasureOrNilReasonList [0..1]
+address «Property»
«FeatureType»
Core::Address
+buildingSubdivision «Property»
«FeatureType»
AbstractBuildingSubdivision
«Property»
+ class: BuildingSubdivisionClassValue [0..1]
+ function: BuildingSubdivisionFunctionValue [0..*]
+ usage: BuildingSubdivisionUsageValue [0..*]
+ elevation: Elevation [0..*]
+ sortKey: Real [0..1]
AbstractPhysicalSpace
«FeatureType»
Core::AbstractUnoccupiedSpace
+buildingRoom «Property»
«FeatureType»
BuildingRoom
«Property»
+ class: BuildingRoomClassValue [0..1]
+ function: BuildingRoomFunctionValue [0..*]
+ usage: BuildingRoomUsageValue [0..*]
+ roomHeight: RoomHeight [0..*]
«FeatureType»
BuildingUnit
«FeatureType»
Storey
+buildingUnit «Property»
+storey «Property»
«TopLevelFeatureType»
Building
+buildingPart «Property»
«FeatureType»
BuildingPart
AbstractOccupiedSpace
«FeatureType»
Construction::AbstractConstructiveElement
+buildingInstallation «Property»
«FeatureType»
BuildingInstallation
«Property»
+ class: BuildingInstallationClassValue [0..1]
+ function: BuildingInstallationFunctionValue [0..*]
+ usage: BuildingInstallationUsageValue [0..*]
AbstractOccupiedSpace
«FeatureType»
Construction::AbstractInstallation
«FeatureType»
BuildingConstructiveElement
«Property»
+ class: BuildingConstructiveElementClassValue [0..1]
+ function: BuildingConstructiveElementFunctionValue [0..*]
+ usage: BuildingConstructiveElementUsageValue [0..*]
+buildingConstructiveElement «Property»
+buildingFurniture «Property»
«FeatureType»
BuildingFurniture
«Property»
+ class: BuildingFurnitureClassValue [0..1]
+ function: BuildingFurnitureFunctionValue [0..*]
+ usage: BuildingFurnitureUsageValue [0..*]
AbstractOccupiedSpace
«FeatureType»
Construction::AbstractFurniture

Fig. 8 CityGML 3.0 Building module

addition, the classes *WindowSurface* and *DoorSurface* are introduced to represent filling surfaces.

The Building module introduces a new class *AbstractBuildingSubdivision*, which is modelled as a subclass of *AbstractLogicalSpace*, and the two specialisations *BuildingUnit* and *Storey* to allow for representing building units (like apartments) and storeys.

4 Dynamics of Changing Cities

In general, a city object can have properties related to its geometry, topology, semantics, and appearance and all of these properties may change w.r.t. time. For example, a construction event leads to the change in geometry of a building (i.e. addition of a new building floor or demolition of an existing door). The geometry of an object can be further classified according to its shape, location, and extent, which can also be changed over time. A moving car object involves changing only the location of the car object; however, a flood incident involves variations in the location and shape of water. There might be other properties which change w.r.t. thematic data of city objects, e.g. hourly variations in energy or gas consumption of a building or changing the building usage from residential to commercial. Some properties involve changes in appearances over the period of time, such as building textures changing over years or traffic cameras recording videos of moving traffic over definite intervals. 3D city models comprise relevant real-world entities and also represent interrelationships between objects. Such interrelationships may also change over time. Hence, it is important to consider that the representation of time-varying data is required to be associated with these different properties.

Temporal variations involve time points mapped onto the specific attributes (i.e. spatial, thematic, topology, or appearance). Such mappings are often realised by discrete recordings or by interpolation functions and involve quantitative changes which can be defined as a function of time. For example, varying energy consumption values of a building can be determined for specific points of time (1) in the past by querying a database for historic data, (2) in the present by querying a real-time sensor or IoT device, and (3) in the future by a simulation software. However, there are other scenarios where features begin or cease to exist over different time intervals, for example, addition of a new building or demolition of an old building. Such scenarios involve qualitative changes and are fundamentally different from the quantitative changes. Such changes can not be defined as a function of time on a feature's property as the state of features changes. Hence, it is also essential to consider that semantic 3D city models handle both quantitative and qualitative changes within cities/city objects. Since both quantitative and qualitative changes involve variations w.r.t. time, it is also important to determine if they can be handled by the same mechanisms. A detailed discussion on the requirements of applications regarding the support of dynamic data is given in Chaturvedi and Kolbe (2019).

In the current version of CityGML (version 2.0), such time variations are not supported. Hence, two new concepts (Versioning module and Dynamizer module) are proposed for CityGML 3.0 to manage time-dependent properties. The Versioning module manages qualitative changes that are slower in nature, e.g. (1) the history or evolution of cities such as construction or demolition of buildings, and (2) managing multiple versions of the city models. The Dynamizer module manages quantitative changes representing high frequent or dynamic variations of object properties, e.g. variations of (1) thematic attributes such as changes of physical quantities (energy demands, temperature, solar irradiation levels), (2) spatial properties such as change of a feature's geometry, with respect to shape and location (moving objects), and (3) real-time sensor observations. In this case, only some of the properties of otherwise static objects need to represent such time-varying values.

4.1 Versioning of Cities

Within the new *Versioning* module, CityGML 3.0 introduces bitemporal timestamps for all objects in alignment with the INSPIRE data specifications. Besides the attributes *creationDate* and *terminationDate* from CityGML 2.0, which refer to the time period in which a specific version of an object is an integral part of the 3D city model, all objects now can additionally have the attributes *validFrom* and *validTo*, which represent the lifespan a specific version of an object has in the real world. Furthermore, each geographic feature is being provided with two identifiers: the *identifier* property which is stable along the lifetime of the real-world object, and the *gml:id* attribute which is to mark the respective version of the object. In this way, not only the current version of a 3D city model, but also its entire history can be represented in CityGML and exchanged even within a single file. The module defines two new feature types: *Version*, which can be used to explicitly define named states of the 3D city model and denote all the specific versions of objects belonging to such states, and *VersionTransition*, which allows to explicitly link different versions of the 3D city model by describing the reason of change and the modifications applied. Details on the versioning concept are given in Chaturvedi et al. (2017). The versioning concept has already been used and further extended by a proposed proof-of-concept called UrbanCo-2Fab (Samuel et al. 2016) to represent multiple viewpoints of urban evolution.

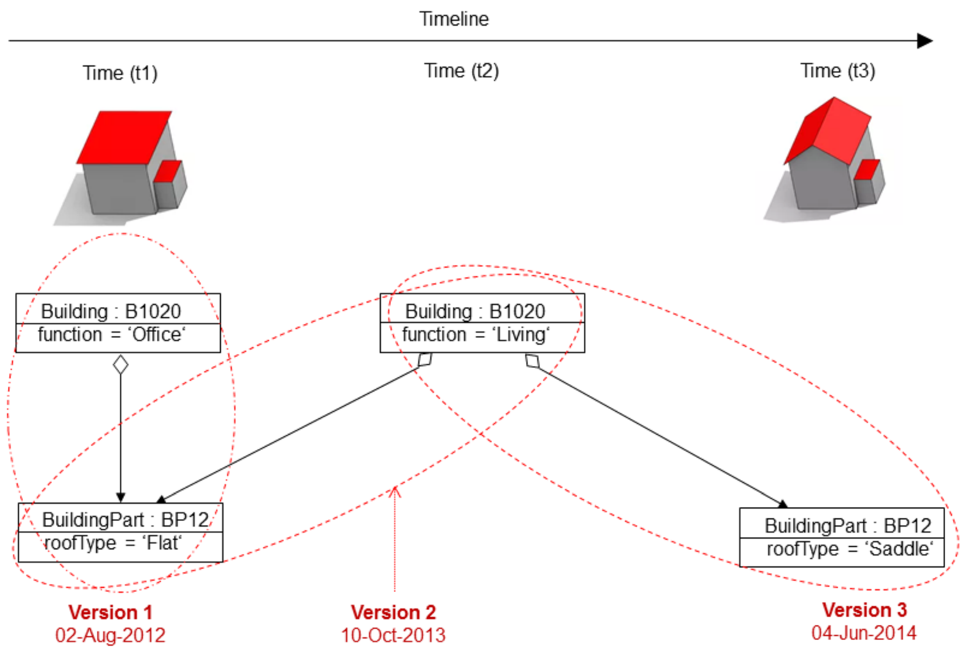


Fig. 9 An instance example of versions representing modifications of a building

The following example illustrates one such possibility of modification scenarios. As shown in Fig. 9, a building with the major ID *B1020* has a function property *Office* and one of its building parts with major ID *BP12* has a *roofType* property *Flat*. Over a period of time, the building function property is changed to *Living* which has been captured in version 2. Furthermore, at a point in time, the *roofType* property of the same building has been changed to *Saddle*. Using the Versioning module, version management can easily be supported in a single CityGML data set as shown in Fig. 10. The building object in version 1 can be denoted as *B1020_t1* at a specific point in time t1. XPath can be used with XLink to retrieve all the instances of the same building object. The instance data can also include version elements for managing different versions of an object. However, due to limited space in this paper, the example illustrates a simple version management where it is sufficient to just use the bi-temporal time attributes and the major/minor IDs.

The advantage of this approach is that it not only facilitates the data model for supporting different versions, but also allows the different versions to be used in an interoperable exchange format and the exchange of all versions of a repository within a single dataset. Such a dataset can be used by different software systems to visualise and work with all the versions. The approach not only addresses the implementation of versionable CityGML models but also considers new aspects to previous work such as managing multiple histories or multiple interpretations of the past of a city. Also, collaborative work is supported since the Versioning module provides all functionalities to represent a tree of workspaces as version control systems like git or svn. The proposed UML model handles versions and version transitions as feature types, which allows the version management to be completely handled using the OGC Web Feature Service (Vretanos 2010). No extension of other OGC standards is required.

Fig. 10 Representation of different versions of city objects within one CityGML dataset encoded in GML

```
<cityObjectMember>
    <Building gml:id="B1020_t1">
        <identifier>B1020</identifier>
        <consistsOfBuildingPart>
            <BuildingPart xlink:href="//identifier[text()='BP12']"/>
        </consistsOfBuildingPart>
        <creationDate>2012-08-02</creationDate>
        <terminationDate>2013-10-10</terminationDate>
        <function>Office</function>
    </Building>
</cityObjectMember>
<cityObjectMember>
    <Building gml:id="B1020_t2">
        <identifier>B1020</identifier>
        <consistsOfBuildingPart>
            <BuildingPart xlink:href="//identifier[text()='BP12']"/>
        </consistsOfBuildingPart>
        <creationDate>2013-10-10</creationDate>
        <function>Living</function>
    </Building>
</cityObjectMember>
<cityObjectMember>
    <BuildingPart gml:id="BP12_t1">
        <identifier>BP12</identifier>
        <creationDate>2012-08-02</creationDate>
        <terminationDate>2014-06-04</terminationDate>
        <roofType>Flat</roofType>
    </BuildingPart>
</cityObjectMember>
<cityObjectMember>
    <BuildingPart gml:id="BP12_t3">
        <identifier>BP12</identifier>
        <creationDate>2014-06-04</creationDate>
        <roofType>Saddle</roofType>
    </BuildingPart>
</cityObjectMember>
```

4.2 Representation of Time-Dependent Properties

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

As shown in Fig. 11, Dynamizers serve three main purposes:

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

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

Dynamizers have already been implemented as an Application Domain Extension (ADE) for CityGML 2.0 and were employed in the OGC Future City Pilot Phase 1. More details about the UML diagram and instance documents are given in Chaturvedi and Kolbe (2017).

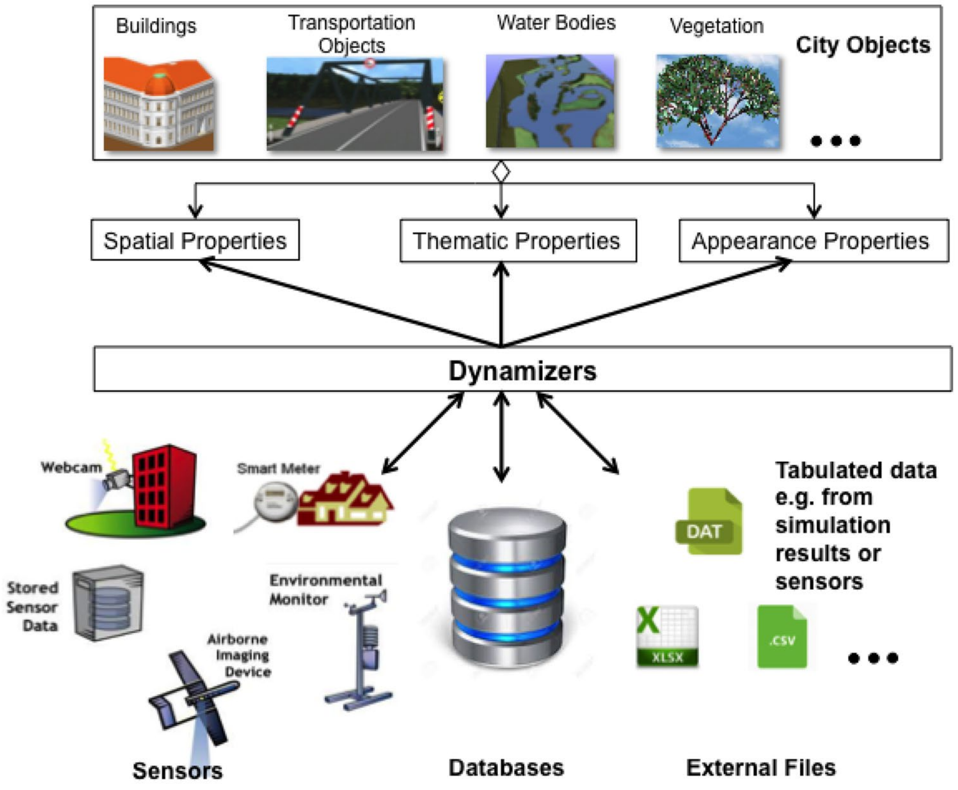


Fig. 11 Conceptual representation of Dynamizers allowing (1) enhancing the properties of city objects by overriding their static values, and (2) the representation of time-variant values from sensors, simulation specific databases, and external files. Image taken from Chaturvedi and Kolbe (2016)

5 Further New Concepts

In addition to the new and revised modules and concepts presented above, two further modules provide interesting new additions to CityGML 3.0: the Transportation module and the PointCloud module.

5.1 Transportation Module

The Transportation module defines classes for the representation of central elements of the traffic infrastructure. To improve the usability of CityGML transportation objects with traffic and driving simulations, driving assistance systems, autonomous driving, as well as with road and railway facility management systems, the data model has been substantially revised.

Figure 12 illustrates the new Transportation module. Transportation objects like roads, tracks, or railways are defined now as concrete subclasses of the abstract class *TransportationSpace*. In addition, they can be subdivided into sections, which can be regular road, track or railway legs, intersection areas, or roundabouts (class *Section* and *Intersection*). Intersection areas as well as roundabouts can belong to multiple road or track objects avoiding the redundant representation of shared spaces. As in CityGML 2.0, transportation objects can be subdivided into *TrafficArea* (e.g. driving lanes, pedestrian zones, and cycle lanes) and *AuxiliaryTrafficArea* (e.g. kerbstones, middle lanes, and green areas). To adapt the semantics of the Transportation module to the CityGML 3.0 space concept, the classes *TrafficSpace* and *AuxiliaryTrafficSpace* were introduced in addition to *TrafficArea* and *AuxiliaryTrafficArea*, the two areas representing now the bottom boundaries of the two spaces. Each traffic space can have an optional *ClearanceSpace*. This is exemplified in Fig. 13. A road consisting of a driving lane and two sidewalks is represented. The traffic spaces (in blue) define the free spaces above the driving lane and the sidewalks; whereas, the traffic areas (in green) represent

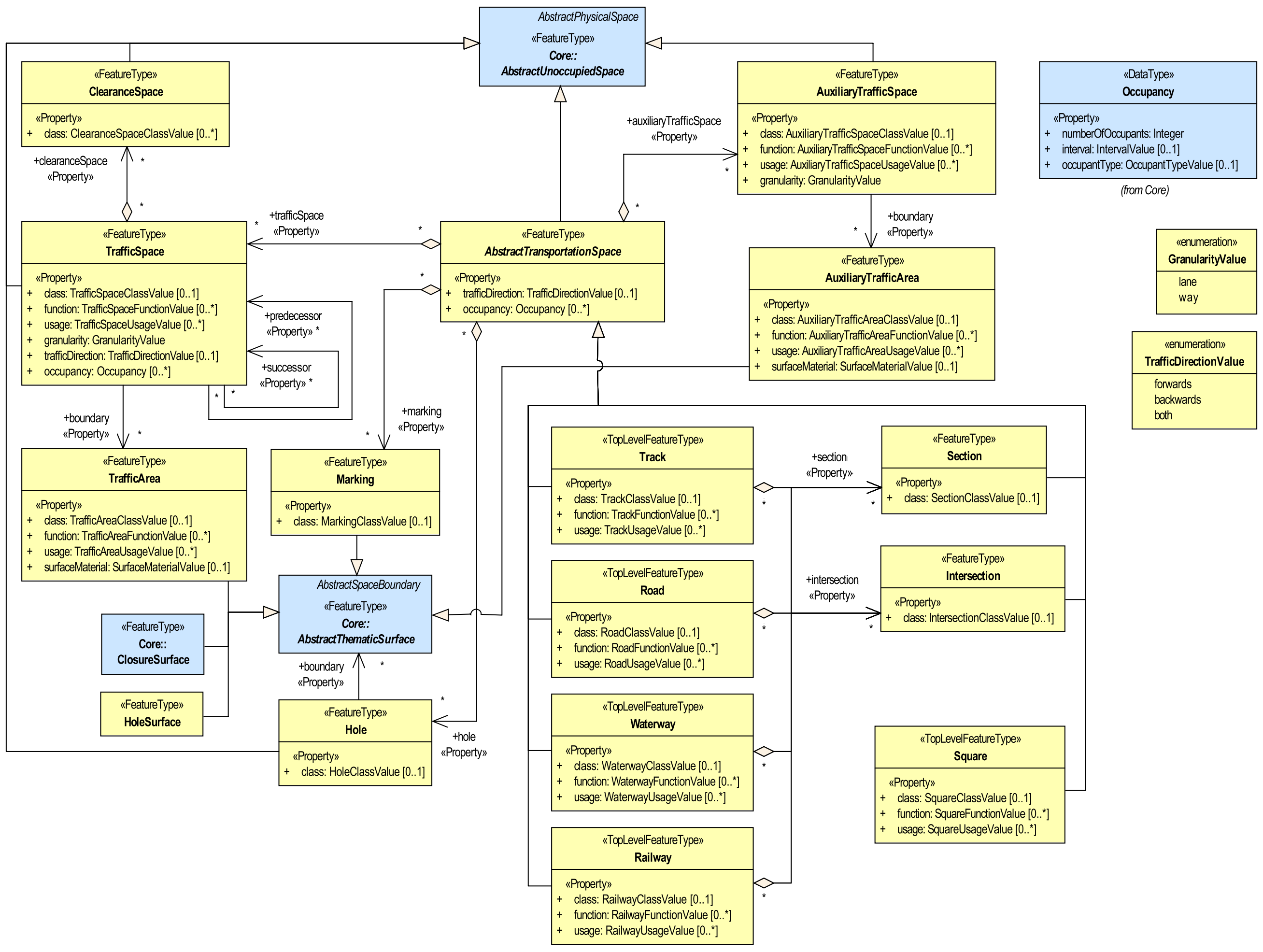

AbstractPhysicalSpace
«FeatureType»
Core::
AbstractUnoccupiedSpace
«FeatureType»
ClearanceSpace
«Property»
+ class: ClearanceSpaceClassValue [0..*]
+clearanceSpace
«Property»
«FeatureType»
AuxiliaryTrafficSpace
«Property»
+ class: AuxiliaryTrafficSpaceClassValue [0..1]
+ function: AuxiliaryTrafficSpaceFunctionValue [0..*]
+ usage: AuxiliaryTrafficSpaceUsageValue [0..*]
+ granularity: GranularityValue
+auxiliaryTrafficSpace
«Property»
«DataType»
Occupancy
«Property»
+ numberOfOccupants: Integer
+ interval: IntervalValue [0..1]
+ occupantType: OccupantTypeValue [0..1]
(from Core)
«FeatureType»
TrafficSpace
«Property»
+ class: TrafficSpaceClassValue [0..1]
+ function: TrafficSpaceFunctionValue [0..*]
+ usage: TrafficSpaceUsageValue [0..*]
+ granularity: GranularityValue
+ trafficDirection: TrafficDirectionValue [0..1]
+ occupancy: Occupancy [0..*]
+trafficSpace
«Property»
+predecessor
«Property»
+successor
«Property»
«FeatureType»
AbstractTransportationSpace
«Property»
+ trafficDirection: TrafficDirectionValue [0..1]
+ occupancy: Occupancy [0..*]
+boundary
«Property»
«FeatureType»
AuxiliaryTrafficArea
«Property»
+ class: AuxiliaryTrafficAreaClassValue [0..1]
+ function: AuxiliaryTrafficAreaFunctionValue [0..*]
+ usage: AuxiliaryTrafficAreaUsageValue [0..*]
+ surfaceMaterial: SurfaceMaterialValue [0..1]
«enumeration»
GranularityValue
lane
way
«enumeration»
TrafficDirectionValue
forwards
backwards
both
+boundary
«Property»
«FeatureType»
TrafficArea
«Property»
+ class: TrafficAreaClassValue [0..1]
+ function: TrafficAreaFunctionValue [0..*]
+ usage: TrafficAreaUsageValue [0..*]
+ surfaceMaterial: SurfaceMaterialValue [0..1]
+marking
«Property»
«FeatureType»
Marking
«Property»
+ class: MarkingClassValue [0..1]
AbstractSpaceBoundary
«FeatureType»
Core::
AbstractThematicSurface
«FeatureType»
Core::
ClosureSurface
«FeatureType»
HoleSurface
+boundary
«Property»
«FeatureType»
Hole
«Property»
+ class: HoleClassValue [0..1]
+hole
«Property»
«TopLevelFeatureType»
Track
«Property»
+ class: TrackClassValue [0..1]
+ function: TrackFunctionValue [0..*]
+ usage: TrackUsageValue [0..*]
+section
«Property»
«FeatureType»
Section
«Property»
+ class: SectionClassValue [0..1]
«TopLevelFeatureType»
Road
«Property»
+ class: RoadClassValue [0..1]
+ function: RoadFunctionValue [0..*]
+ usage: RoadUsageValue [0..*]
+intersection
«Property»
«FeatureType»
Intersection
«Property»
+ class: IntersectionClassValue [0..1]
«TopLevelFeatureType»
Waterway
«Property»
+ class: WaterwayClassValue [0..1]
+ function: WaterwayFunctionValue [0..*]
+ usage: WaterwayUsageValue [0..*]
«TopLevelFeatureType»
Square
«Property»
+ class: SquareClassValue [0..1]
+ function: SquareFunctionValue [0..*]
+ usage: SquareUsageValue [0..*]
«TopLevelFeatureType»
Railway
«Property»
+ class: RailwayClassValue [0..1]
+ function: RailwayFunctionValue [0..*]
+ usage: RailwayUsageValue [0..*]

Fig. 12 CityGML 3.0 Transportation module

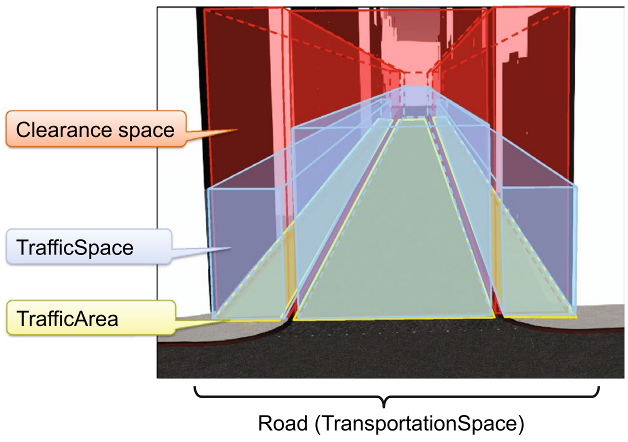


Fig. 13 Representation of a road in CityGML 3.0

the ground surfaces of the traffic spaces. In Germany, for example, roads typically have a free space height of 4.5 m and sidewalks of 2.5 m. In addition, clearance spaces (in red) are represented.

Transportation objects can now have an areal as well as a center line representation for each LOD. In the highest LOD (LOD3), each lane is represented by an individual traffic space object. Each traffic space can be linked to predecessor and successor traffic spaces. This information is typically used (and required) in navigation systems and traffic simulations (cf. Ruhdorfer et al. 2018; Labetski et al. 2018). According to Labetski et al. (2018) also *Waterway* was introduced as new subclass of TransportationSpace. In addition, the new class *Marking* allows for adding road markings to the road surface and the classes *Hole* and *HoleSurface* can be used to represent, for instance, roadway damages or manholes including their surfaces. For further information and examples on the new Transportation module please refer to Beil and Kolbe (2017).

5.2 PointCloud Module

In addition to the geometries defined in the Core module, the geometry of physical spaces and of thematic surfaces can now also be provided by 3D point clouds using MultiPoint geometry. This allows, for example, to spatially represent the building hull, a room within a building or a single wall surface just by a point cloud. All thematic feature types including transportation objects, vegetation, city furniture, etc. can be spatially represented by point clouds, too. In this way, the ClearanceSpace of a road or railway could, for instance, be modelled directly from the result of a mobile laser scanning campaign. Point clouds can either be represented inline within a CityGML file or just reference an external file of some common types such as LAS or LAZ. Figure 14 shows the new PointCloud module.

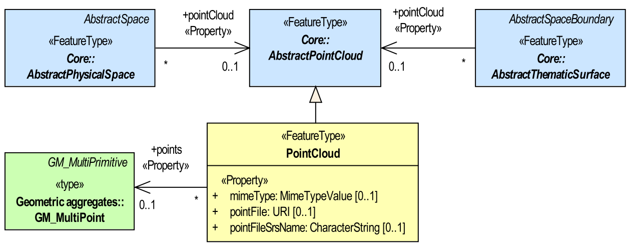


Fig. 14 CityGML 3.0 PointCloud module

6 New Applications for CityGML 3.0

6.1 Sensors and IoT in Cities

A large number of cities such as Singapore,[1] Munich,[2] Helsinki,[3] and Melbourne[4] are developing Digital Twins to improve the operational efficiency of cities. A Digital Twin (Batty 2018; Datta 2017) is a digital counterpart of a physical asset, which collects information via sensors and IoT devices, and applies advanced analytics and artificial intelligence to gain real-time insights about the physical asset's performance, operation or profitability. These sensors can be stationary such as Smart Meters and weather stations. Some of the sensors can also be non-stationary such as air-quality sensors mounted on a car measuring air pollution over different parts of a city at different time intervals or pedestrian flow analysis involving pedestrians moving into or out of a stadium (e.g. before or after a scheduled football match). Bridging the virtual and physical worlds together in this way can also help Smart City applications to improve decision-making and reduce risks by predicting issues before occurrence.

Several Smart City initiatives also highlight the importance of integrating real-time sensors and IoT devices with city objects. The Smart District Data Infrastructure (SDDI) proposed by Moshrefzadeh et al. (2017) allows integrating diverse components such as stakeholders, sensors, IoT devices and simulation tools with a virtual district model representing the physical reality of the district. To access distributed resources, the framework uses a well-defined set of OGC-based service interfaces such as Web Feature Service (Vretanos 2010), Sensor Observation Service (Bröring et al. 2012) and SensorThings API (Liang et al. 2015) as well as a Catalogue Service. The framework also facilitates privacy, security and controlled access to all stakeholders

[1] https://www.nrf.gov.sg/programmes/virtual-singapore.

[2] https://muenchen.digital/blog/digitaler-zwilling-in-muenchen-ein-leuchtturmprojekt-auf-dem-weg-zur-digitalen-metropole/.

[3] https://aec-business.com/helsinki-is-building-a-digital-twin-of-the-city/.

[4] https://www.itnews.com.au/news/vic-govt-to-build-states-first-digital-twin-528187.

and the respective components by establishing proper authorization and authentication mechanisms (Chaturvedi et al. 2019a).

Since all of the above-mentioned city initiatives consider semantic 3D city models as an integral component of their infrastructures, it is highly important that the 3D city objects support seamless integration with sensors and IoT devices. CityGML Dynamizers allow defining explicit links to these real-time sensor observations directly within city objects. In this way, the attribute of the respective city object can also be associated with time-dynamic sensor observations. For example, if a building has an indoor sensor installed for measuring real-time temperature or humidity and the indoor sensor's readings are accessible via standardised web services such as OGC Sensor Web Enablement (Bröring et al. 2011), FIWARE (FIWARE 2018), or any other proprietary API, Dynamizers can be defined for the specific building or room having explicit links to those sensor-based services, and the respective temperature/humidity attribute of the building can be overridden by the real-time sensor observations (see Fig. 15).

6.2 Improved Simulation Support

CityGML Dynamizers also enable 3D city models for an improved simulation support. The Dynamizer ADE for CityGML 2.0 (Chaturvedi and Kolbe 2017) has already been tested successfully for a solar potential simulation to assess and estimate solar energy production for the roofs and façades of 3D building objects. The simulation tool operates on 3D models structured according to the CityGML standard and generates the monthly and yearly estimates of direct, diffuse, and global irradiation values for the building surfaces. The dynamic simulation results can be represented using the international OGC TimeseriesML standard (Tomkins and Lowe 2016) within Dynamizer *AtomicTimeseries* class. The advantage of such representation of simulation results is that it allows modelling precise description of timeseries and enables cross-domain exchanging of simulation results along with city objects (Fig. 16). It is also helpful to create a snapshot of the state(s) of a city model including time-varying data for documentation and archiving. Similarly, Dynamizers are also helpful in representing moving objects in traffic simulations (Ruhdorfer 2017; Santhanavanich et al. 2018).

Furthermore, Chaturvedi et al. (2019b) proposed an implementation that allows managing and visualising static and dynamic properties of semantic 3D city models in an integrated fashion. The 3D City Database (also known as 3DCityDB[5]) (Yao et al. 2018) is an Open Source software suite, which allows storing, representing, and managing large CityGML datasets on top of spatial relational database management systems (SRDBMS) such as Oracle Spatial and PostgreSQL. It includes a Java front-end application named 3DCityDB Importer/Exporter for importing and exporting CityGML datasets with arbitrary file sizes. It also allows exporting CityGML objects in the form of 3D visualization formats (such as KML, COLLADA, and glTF) enabling them to be viewed and interactively explored in web applications such as the 3DCityDB Web Map Client or Google Earth. The implementation developed by Chaturvedi et al. (2019b) allows storing Dynamizer timeseries data (such as solar potential simulation results of a building's roof surface) along with other static properties of the same building in the 3DCityDB. It also enables CityGML Viewers such as the 3DCityDB Web Map Client to access static data using the OGC Web Feature Service (Vretanos 2010) interface and dynamic data using the OGC SWE interfaces such as the OGC Sensor Observation Service (Bröring et al. 2012) and the OGC SensorThings API (Liang et al. 2015) in an integrated fashion with the help of the newly developed InterSensor Service[6] (Chaturvedi and Kolbe 2019).

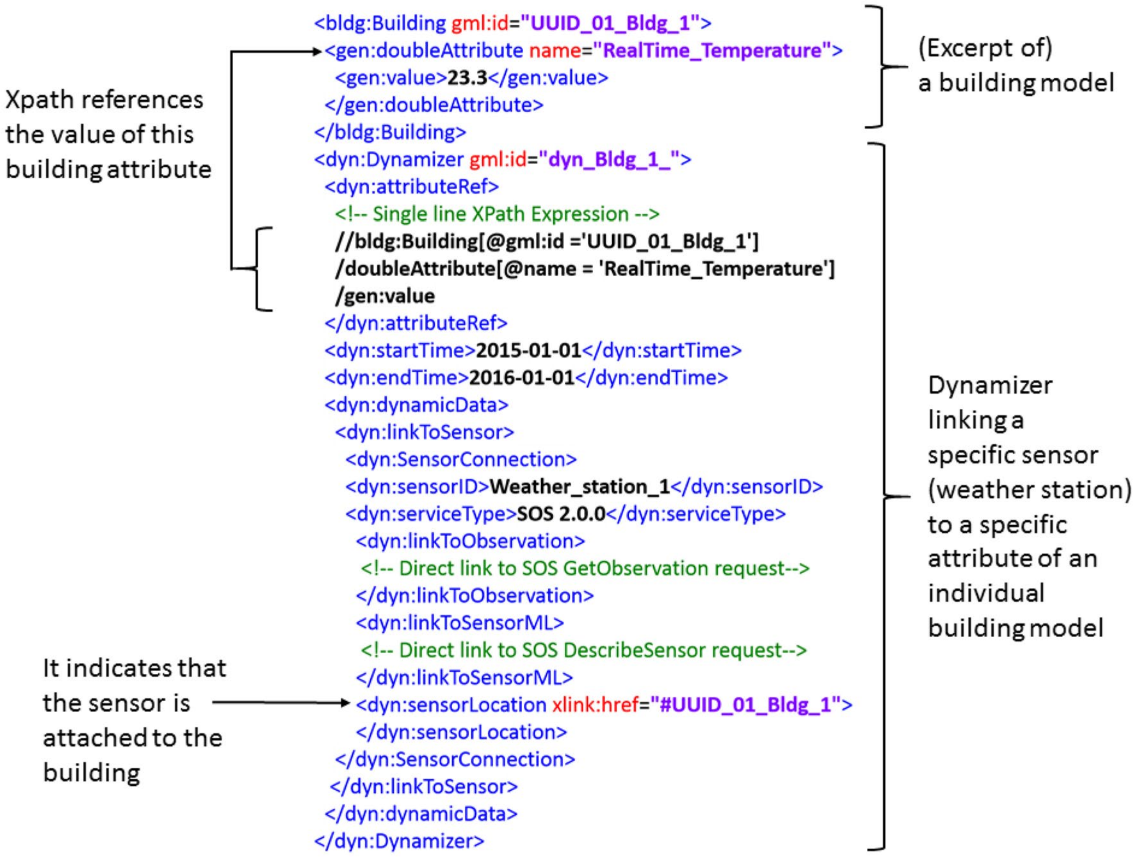
6.3 Usage of the Space Concept

The new space concept introduced in CityGML 3.0 can be useful in various applications such as enhancing the conversion of IFC into CityGML. In particular the new volumetric feature types *BuildingConstructiveElement*, *BridgeConstructiveElement*, and *TunnelConstructiveElement* that are defined as subclasses of *AbstractOccupiedSpace* facilitate the mapping of constructive elements from IFC data sets to CityGML. Walls, for instance, are represented in IFC as volumetric objects; whereas in CityGML 2.0, only the exterior and interior surfaces of walls are represented as separate features. This means that these surfaces need to be extracted when mapping IFC to CityGML 2.0. With the newly introduced constructive element classes this extraction could be avoided, because a direct mapping from IFC onto CityGML can now be achieved simply by mapping the volumetric IFC objects to volumetric CityGML objects. Figure 17 shows a building of the Technical University of Munich that was converted from IFC to CityGML 3.0. Visible are only the *BuildingConstructiveElement* objects that have been created from the IFC classes IfcWall, IfcRoof, IfcBeam, and IfcSlab. The conversion was executed using the FME-based ifc-to-citygml3 conversion tool available from TUM-GIS (2019a).

Similarly, the semantics of the IFC class *IFCSpace* is a physically unoccupied space to represent rooms. The class *BuildingRoom* represents the equivalent concept in CityGML as it is a subclass of *AbstractUnoccupiedSpace*.

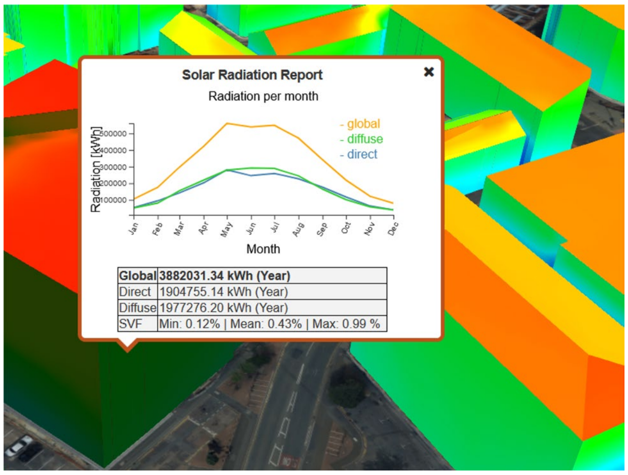
5 http://www.3dcitydb.org.

6 http://www.intersensorservice.org/.


Xpath references the value of this building attribute
(Excerpt of) a building model
Dynamizer linking a specific sensor (weather station) to a specific attribute of an individual building model
It indicates that the sensor is attached to the building

```
<bldg:Building gml:id="UUID_01_Bldg_1">
 <gen:doubleAttribute name="RealTime_Temperature">
  <gen:value>23.3</gen:value>
 </gen:doubleAttribute>
</bldg:Building>
<dyn:Dynamizer gml:id="dyn_Bldg_1_">
 <dyn:attributeRef>
  <!-- Single line XPath Expression -->
  //bldg:Building[@gml:id ='UUID_01_Bldg_1']
  /doubleAttribute[@name = 'RealTime_Temperature']
  /gen:value
 </dyn:attributeRef>
 <dyn:startTime>2015-01-01</dyn:startTime>
 <dyn:endTime>2016-01-01</dyn:endTime>
 <dyn:dynamicData>
  <dyn:linkToSensor>
   <dyn:SensorConnection>
   <dyn:sensorID>Weather_station_1</dyn:sensorID>
   <dyn:serviceType>SOS 2.0.0</dyn:serviceType>
    <dyn:linkToObservation>
     <!-- Direct link to SOS GetObservation request-->
    </dyn:linkToObservation>
    <dyn:linkToSensorML>
     <!-- Direct link to SOS DescribeSensor request-->
    </dyn:linkToSensorML>
    <dyn:sensorLocation xlink:href="#UUID_01_Bldg_1">
    </dyn:sensorLocation>
  </dyn:SensorConnection>
  </dyn:linkToSensor>
 </dyn:dynamicData>
</dyn:Dynamizer>
```

Fig. 15 Defining direct links to sensors within Dynamizer feature type. The file shows an excerpt of a CityGML dataset encoded in GML



Global	**3882031.34 kWh (Year)**
Direct	1904755.14 kWh (Year)
Diffuse	1977276.20 kWh (Year)
SVF	Min: 0.12% | Mean: 0.43% | Max: 0.99 %

Fig. 16 The standardised timeseries representation of monthly irradiation values of a building. This demo is accessible from http://manchester.virtualcitymap.de/

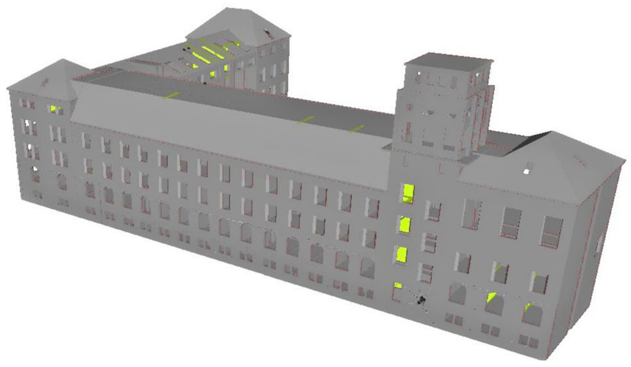


Fig. 17 BuildingConstructiveElement objects created from the IFC classes IfcWall, IfcRoof, IfcBeam, and IfcSlab

Thus, IFCSpace objects can be mapped onto *Building-Room* objects without change of semantics. This is shown in Fig. 18, where IFCSpace objects of the well-known FZK House have been converted into BuildingRoom objects in CityGML 3.0.

6.4 Conversion of CityGML 2.0 into CityGML 3.0

Section 1 mentions that it is possible to convert every City-GML 1.0 and 2.0 dataset into CityGML 3.0 by applying

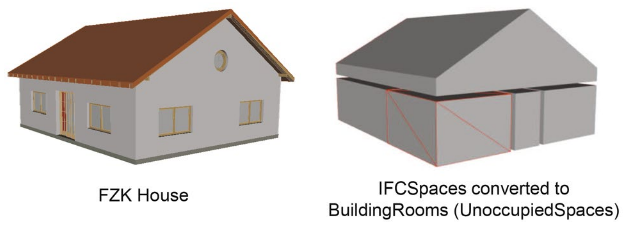


Fig. 18 BuildingRoom objects created from the IFC class IfcSpace

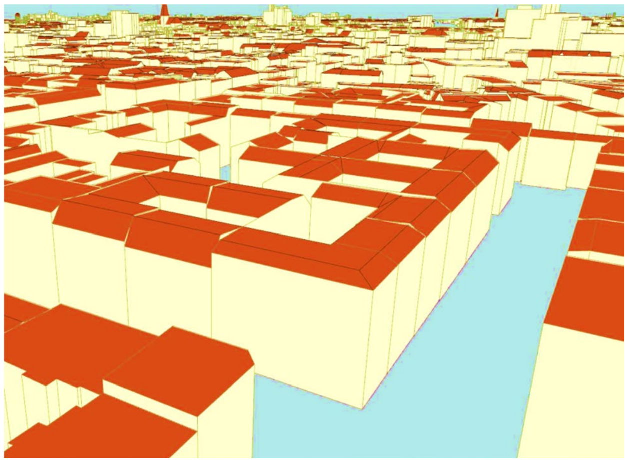


Fig. 19 The city model of Berlin Moabit represented in CityGML 3.0

syntactical transformations only. This transformation can, for instance, be carried out using an XSLT-based conversion tool, such as the citygml2-to-citygml3 conversion tool provided from TUM-GIS (2019b). This tool gives data providers and software implementers the possibility to (a) see how their existing datasets would look like in CityGML 3.0, and (b) adapt their software implementations and work with CityGML 3.0 data, and (c) reassure them on the backwards compatibility of the conceptual model of CityGML 3.0 with CityGML 2.0 by demonstrating lossless data conversion. The tool currently supports the conversion of Buildings, CityFurniture, and Appearances and is gradually being extended to support the other modules as well. Figure 19 visualises the city model of Berlin Moabit that was converted from CityGML 2.0 into CityGML 3.0 using the citygml2-to-citygml3 conversion tool.

7 Conclusions and Outlook

This paper introduces the new and revised concepts that are to become part of the next major CityGML version 3.0. Of particular importance are the new space concept and the revised LOD concept that both are defined in the Core module, the new modules Construction, Versioning, and Dynamizer, as well as the revised Building and Transportation modules. In addition, CityGML 3.0 is modelled ISO-compliant and, thus, allows for automatically deriving the GML application schemas from the UML model by applying a model-driven approach.

Several implementations already exist or are currently being created for CityGML 3.0. All software systems that can deal with generic GML3 can also read CityGML 3.0 data sets, amongst others FME, deegree, GDAL, GMLAS, and SupportGIS. The citygml2-to-citygml3 conversion tool can create CityGML 3.0 data sets from CityGML 2.0 data sets. The CityGML-3-to-Java-objects library citygml4j is currently being implemented by Claus Nagel for CityGML 3.0. An adaption of the 3DCityDB to support CityGML 3.0 will be carried out in the future. Successful application of CityGML 3.0 was also demonstrated at the OGC CityGML Hackathon in June 2019 in London as well as at the OGC CityGML Challenge in October 2019 in Manchester.

Like with CityGML 2.0, the modular specification allows that applications do not have to implement all CityGML modules. For example, not only the modules Dynamizer, Versioning, or PointCloud, but also some thematic modules can be left out when not required.

The entire CityGML 3.0 development is subject to final votings of the CityGML SWG as well as of the OGC OAB (Open Architecture Board) and the Technical Committee. The UML models provided in this paper may still undergo slight changes until the finalisation of the specification. However, we do not expect major changes anymore.

Acknowledgements Open Access funding provided by Projekt DEAL. We would like to thank the other members of the OGC CityGML SWG Modelling Subgroup Claus Nagel, Steve Smyth, Gilles Gesquière, Emmanuel Devys, Heidi Valparys, Friso Penninga, Volker Coors, Sisi Zlatanova, Helga Tauscher and Carsten Roensdorf for their valuable contributions and discussions in the development of CityGML 3.0. We also thank the modelling working group of SIG 3D of GDI-DE, in particular Gerhard Gröger, Joachim Benner, Karl-Heinz Häfele, Marc-Oliver Löwner, Jürgen Ebbinghaus, and Heinrich Geerling as well as the further contributors to the CityGML 3.0 work packages including Filip Biljecki and Linda van den Brink. Special thanks go to Son Nguyen for his work on the citygml2-to-citygml3 conversion tool and Daniel Härpfer for his work on the ifc-to-citygml3 converter.

References

AdV (2009) Documentation on the modelling of geoinformation of official surveying and mapping (GeoInfoDoc), version 6.0.1

Batty M (2018) Digital twins. Environ Plan B Urban Anal City Sci 45(5):817–820

Beil C, Kolbe TH (2017) CityGML and the streets of New York—a proposal for detailed street space modelling. ISPRS Ann Photogram Remote Sens Spat Inf Sci IV-4/W5:9–16

Biljecki F, Stoter J, Ledoux H, Zlatanova S, Çöltekin A (2015) Applications of 3d city models: state of the art review. ISPRS Int J Geo-Inf 4(4):2842–2889

Billen R, Zaki CE, Servières M, Moreau G, Hallot P (2012) Developing an ontology of space: Application to 3D city modeling. In: Leduc T, Moreau G, Billen R (eds) Usage, usability, and utility of 3D city models—European COST Action TU0801. EDP Sciences, Nantes, vol 02007. https://hal.archives-ouvertes.fr/hal-01521445

Bröring A, Echterhoff J, Jirka S, Simonis I, Everding T, Stasch C, Liang S, Lemmens R (2011) New generation sensor web enablement. Sensors 11(3):2652–2699

Bröring A, Stasch C, Echterhoff J (2012) Sensor Observation Service Interface Standard, OGC Doc. No. 12-006. http://www.opengeospatial.org/standards/sos. Accessed 04 May 2019

Chaturvedi K, Kolbe TH (2016) Integrating dynamic data and sensors with semantic 3D city models in the context of smart cities. In: Proceedings of the 11th international 3D geoinfo conference, Athens, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol IV-2/W1. https://doi.org/10.5194/isprs-annals-IV-2-W1-31-2016

Chaturvedi K, Kolbe TH (2017) Future City Pilot 1 Engineering Report, OGC Doc. No. 19-098. http://docs.opengeospatial.org/per/16-098.html. Accessed 22 Dec 2018

Chaturvedi K, Kolbe TH (2019) Towards establishing cross-platform interoperability for sensors in smart cities. Sensors 19(3)

Chaturvedi K, Smyth CS, Gesquière G, Kutzner T, Kolbe TH (2017) Managing versions and history within semantic 3D city models for the next generation of CityGML. In: Advances in 3D geoinformation. Springer, pp 191–206

Chaturvedi K, Matheus A, Nguyen SH, Kolbe TH (2019a) Securing spatial data infrastructures for distributed smart city applications and services. Future Gen Comput Syst 101:723–736

Chaturvedi K, Yao Z, Kolbe TH (2019b) Integrated management and visualization of static and dynamic properties of semantic 3D city models. In: Proceedings of the 4th international conference on smart data and smart cities, ISPRS, Kuala Lumpur, ISPRS Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences

Datta SPA (2017) Emergence of digital twins—is this the march of reason? J Innov Manag 5(3):14–33

Elfes A (1989) Using occupancy grids for mobile robot perception and navigation. Computer 22(6):46–57

European Parliament and Council (2007) Directive 2007/2/EC of the European Parliament and of the council of 14 March 2007 establishing an infrastructure for spatial information in the European Community (inspire). Off J Eur Union 50(L 108):1–14

FIWARE (2018) Open source platform for the smart digital future. https://www.fiware.org/. Accessed 16 May 2018

Gröger et al (2012) OGC City Geography Markup Language (CityGML) Encoding Standard, Version 2.0.0

Hillier B, Hanson J (1984) The social logic of space. Cambridge University Press, Cambridge. https://doi.org/10.1017/CBO9780511597237

Interactive Instruments (2019) Shapechange. https://shapechange.net/. Accessed 05 Sep 2019

ISO 16739 (2013) ISO 16739:2013 industry foundation classes (IFC) for data sharing in the construction and facility management industries

ISO 19152 (2012) ISO/TS 19152:2012 geographic information—land administration domain model

ISO 19505-2 (2012) ISO/IEC 19505-2:2012 information technology—object management group unified modeling language (OMG UML)—part 2: superstructure

JRC (2013) D2.8.III.2 Data specification on buildings—technical guidelines

JRC (2014) INSPIRE D2.5: generic conceptual model, version 3.4

Konde A, Tauscher H, Biljecki F, Crawford J (2018) Floor plans in CityGML. ISPRS Ann Photogramm Remote Sens Spat Inf Sci IV-4/W6:25–32

Kutzner T, Kolbe TH (2018) CityGML 3.0: Sneak preview. In: Kersten TP, Gülch E, Schiewe J, Kolbe TH, Stilla U (eds) PFGK18-Photogrammetrie-Fernerkundung-Geoinformatik-Kartographie, 37. Jahrestagung in München 2018, Deutsche Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V., Publikationen der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation (DGPF) e.V., vol 27, pp 835–839

Labetski A, van Gerwen S, Tamminga G, Ledoux H, Stoter J (2018) A proposal for an improved transportation model in CityGML. ISPRS—Int Arch Photogramm Remote Sens Spat Inf Sci XLII-4/W10:89–96

Lee et al (2016) OGC IndoorGML, version 1.0.2

Liang S, Huang CY, Khalafbeigi T (2015) SensorThings API Part 1: Sensing, OGC Doc. No. 15-078r6. https://www.opengeospatial.org/standards/sensorthings. Accessed 04 May 2019

Löwner MO, Gröger G, Benner J, Biljecki F, Nagel C (2016) Proposal for a new LOD and multi-representation concept for CityGML. ISPRS Ann Photogramm Remote Sens Spat Inf Sci IV-2/W1:3–12

Moshrefzadeh M, Chaturvedi K, Hijazi I, Donaubauer A, Kolbe TH (2017) Integrating and managing the information for smart sustainable districts—the smart district data infrastructure (SDDI). In: Kolbe TH, Bill R, Donaubauer A (eds) Geoinformationssysteme 2017-Beiträge zur 4. Münchner GI-Runde, Wichmann Verlag

OGC (2019a) CityGML 3.0 conceptional model. https://github.com/opengeospatial/CityGML-3.0CM. Accessed 14 Sep 2019

OGC (2019b) CityGML 3.0 encodings. https://github.com/opengeospatial/CityGML-3.0Encodings/. Accessed 14 Sep 2019

Ruhdorfer R (2017) Kopplung von Verkehrssimulation und semantischen 3D-Stadtmodellen in CityGML. Master's thesis, Technische Universität München, Chair of Geoinformatics

Ruhdorfer R, Willenborg B, Sindram M (2018) Coupling of traffic simulations and semantic 3d city models. gisScience 3:101–109

Samuel J, Périnaud C, Gay G, Servigne S, Gesquière G (2016) Representation and visualization of urban fabric through historical documents. In: Catalano CE, Luca LD (eds) Eurographics workshop on graphics and cultural heritage. The Eurographics Association. https://doi.org/10.2312/gch.20161399

Santhanavanich T, Schneider S, Rodrigues P, Coors V (2018) Integration and visualization of heterogeneous sensor data and geospatial information. ISPRS Ann Photogramm Remote Sens Spat Inf Sci IV-4/W7:115–122

Smith B, Varzi AC (2000) Fiat and bona fide boundaries. Philos Phenomenol Res 60(2):401–420. https://doi.org/10.2307/2653492

Sparx Systems (2015) Enterprise architect. http://www.sparxsystems.com/products/ea/index.html. Accessed 04 Oct 2015

Tomkins J, Lowe D (2016) Timeseries profile of observations and measurements, OGC Document No. 15-043r3. http://www.opengeospatial.org/standards/tsml. Accessed 09 Sep 2018

TUM-GIS (2019a) citygml2-to-citygml3. https://github.com/tum-gis/citygml2-to-citygml3. Accessed 05 Sep 2019

TUM-GIS (2019b) ifc-to-citygml3. https://github.com/tum-gis/ifc-to-citygml3. Accessed 05 Sep 2019

Vretanos PA (2010) Opengis web feature service 2.0 interface standard OGC document no. 09-025r1. http://www.opengeospatial.org/standards/wfs. Accessed 02 Nov 2018

W3C (2014) Rdf 1.1 specifications. https://www.w3.org/standards/techs/rdf#w3c_all. Accessed 04 Oct 2015

Yan J, Diakité AA, Zlatanova S (2019) A generic space definition framework to support seamless indoor/outdoor navigation systems. Trans GIS 23(6):1273–1295

Yao Z, Nagel C, Kunde F, Hudra G, Willkomm P, Donaubauer A, Adolphi T, Kolbe TH (2018) 3DCityDB—a 3D geodatabase solution for the management, analysis, and visualization of semantic 3D city models based on CityGML. Open Geospat Data Softw Stand 3(5):1–26