Thomas H. Kolbe, Tatjana Kutzner
Technical University Munich (TUM)

Notes on the CityGML 3.0 Conceptual Model

Right before the end of 2017 we succeeded to integrate all CityGML 3.0 developments into one consolidated data model. Please note, that this is the very first version of the integrated "CityGML 3.0 Conceptual Model". We consider it to be "pre-alpha stage", i.e. some further corrections, refinements, and discussions within the OGC CityGML SWG will happen over the next three months. However, the model already includes all new concepts and features which are intended to become integral part of the next major release of the CityGML standard. It is, thus, useful to familiarize with the new package structure, the new thematic modules, the changes to the LOD concept, and the new CityGML Core model.

We highly recommend to not change your software systems according to this version of the data model yet, but the model can be already taken as a basis to envision required changes. The plan of the CityGML editors is to write the specification document and to correct / adapt the data model within the first 5 months of 2018, presenting the CityGML 3.0 Conceptual Model specification at the June 2018 OGC TC Meeting. After a subsequent discussion and improvement period of 3 months, the specification shall be finished for voting. As soon as this level is achieved, the CityGML 3.0 GML Encoding specification will be created. Hence, the CityGML 3.0 standard will consist of at least two parts: 1) conceptual model, 2) GML encoding. Further encoding specifications (e.g. relational database schema, JSON-based representation) may follow in the future.

New Features and Modules in CityGML 3.0

CityGML 3.0 brings a number of improvements, extensions, and new functionalities. The modifications to the core model and the thematic modules were carried out in a way to ensure backwards compatibility with CityGML 1.0 and 2.0. All CityGML 1.0 and 2.0 datasets can be transformed to the new model by just syntactical transformations. Nevertheless, this claim has to be verified in detail and mapping rules will have to be provided. Backwards compatibility is a major requirement for CityGML 3.0 in order to preserve investments by anybody providing CityGML tools, datasets, and extensions.

The following list gives an overview about the most important changes and new modules and functionalities. Some more details are provided in the following sections.

- The data model is now based directly on the ISO 191xx standards. The data model has been created using Enterprise Architect (EA) and makes use of the EA files provided by ISO and OGC on the underlying standards. The GML encoding (CityGML XML schema) will be fully automatically derived from the data model using the software tool Shapechange. Also the feature catalogue with the detailed overview and explanation of all classes, attributes, and relationships will be derived automatically. The EA file can be downloaded from the OGC CityGML.SWG Wiki: https://portal.opengeospatial.org/wiki/pub/CITYGMLswg/WebHome/CityGML_3.0_Consolidated_Draft_2017_12_22.eap.zip
- Changes to the LOD concept: LOD4 has been removed, so there remain LODs 0-3. The interior of objects (like indoor modeling for buildings and tunnels) can also be

expressed in different LODs 0-3 now. It is even possible to model the outside shell of a building in LOD1 while representing the interior structure in LOD2 or 3.

- New Core model: all spatial representations are rephrased based on the two pivotal abstract classes Space and SpaceBoundary. Geometry can now also be given by point clouds (external files or MultiPoint geometries). The new core model implements the new LOD concept.
- New Construction module: groups all classes which are similar over different types of constructions like buildings, tunnels, bridges, and introduces a new class "OtherConstruction" to represent other constructions not belonging to any of the other three modules.
- New Versioning module: allows to explicitly represent and exchange multiple versions of city objects (e.g. history or alternative designs) within one city model dataset. All objects can have bitemporal lifespan data.
- New Dynamizer module: defines concepts to represent and exchange time-varying data for city object properties as well as to integrate sensors with 3D city models. Data sources can be timeseries data represented inline, in external files, or from sensor web services (OGC Sensor Web Enablement or Internet of Things, IoT, in general).
- Revised Transportation module: transportation objects like Roads, Tracks, or Railways can now be subdivided into sections. TrafficArea and AuxiliaryTrafficArea are changed to TrafficSpace and AuxiliaryTrafficSpace. TrafficSpace can have an optional ClearanceSpace. 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 TrafficSpace object.

Changes to the CityGML Levels of Detail (LOD)

Details on the changes to the CityGML LOD concept are given in the 2016 paper "Proposal for a new LOD and multi-representation concept for CityGML" by Löwner, Gröger, Benner, Bijecki, and Nagel. The paper can be downloaded from here: https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-2-W1/3/2016/

The new CityGML Core Model

The new Core Model reflects the increasing need for better interoperability with other relevant standards in the field like IndoorGML, IFC, and LADM. A key feature of the new Core Model is that all spatial representations are rephrased based on the two pivotal abstract classes Space and SpaceBoundary. These classes are further subdivided into a number of subclasses. All geometric representations are associated with the semantic concepts of Space and SpaceBoundary. The feature classes in the thematic modules all represent their spatial characteristics almost exclusively using Space and SpaceBoundary classes and no longer have direct associations with geometry classes. This simplifies geometry handling of CityGML for software developers in general. The data model implements the new CityGML LOD concept (CityGML 3.0 Work Package 3).

Besides the existing geometry types also a point cloud representation for Spaces and SpaceBoundaries is being introduced. This allows, for example, to spatially represent the

building hull, a room within a building or just a single wall surface just by a point cloud. All thematic feature types incl. vegetation and transportation objects, city furniture etc. can be spatially represented by point clouds, too. Point clouds can either be represented inline within a CityGML file (using MultiPoint geometry) or just reference an external file of some common type (like LAS, LAZ etc.).

The introduction of the concepts of Space and SpaceBoundary and their consequent utilization in the specification of the feature types in the thematic modules allows for a direct mapping of CityGML objects representing buildings to IndoorGML and vice-versa. In addition, the CityGML building module now includes a class called BuildingConstructiveElement which is a subclass of OccupiedSpace. This class allows mapping constructive elements from BIM datasets (e.g. IFC classes IfcWall, IfcRoof, IfcBeam, IfcSlab etc.) directly onto objects of type BuildingConstructiveElement. The new volumetric construction components implement essentially the requirements from CityGML 3.0 Work Package 11. Because the relationship between spaces and their boundaries is now explicitly modeled in CityGML, e.g. a wall object inside a building can refer to the respective bounding InteriorWallSurface objects. This goes beyond the capabilities of IFC and is very suitable to represent BIM-as-Built models. This is also further supported by the new point cloud representation of Spaces and SpaceBoundaries.

The Space concept also has a subclass called LogicalSpace, which is used to model spaces that are not bounded by physical objects but defined according to logical / thematic considerations instead. An example is the definition of BuildingSubdivision allowing to represent building units like apartments or public spaces in buildings. In addition, building storeys are considered a logical and not a physical subdivision, because the boundaries of stories are not always clearly aligned / alignable with slabs. The specification of the concepts BuildingSubdivision, BuildingUnit, and Storey cover CityGML 3.0 Work Package 13. The notion of a logical space is also important to express ownership and usage rights as required in the realization of the ISO standard 19152 on Land Administration Domain Model (LADM).

There are also new relationship associations to express topologic, geometric, and semantic relations between spaces as well as between space boundaries. For example, it can be expressed that two rooms are adjacent or that one interior building installation (like a curtain rail) is overlapping with the spaces of two connected rooms. It can also be expressed that two wall surfaces are parallel and two others are orthogonal. Also distances between objects could be represented explicitly using geometric relations. In addition to spatial relations logical relations can be expressed. The Generics module defines a GenericSpaceRelation and GenericBoundaryRelation where the relation type is represented by a URI. This could directly be mapped onto an RDF triple representation where the RDF subject is pointing to the URI of the source Space / SpaceBoundary feature, the RDF object is pointing to the URI of the target Space / SpaceBoundary feature and the RDF predicate is denoted by the relation type URI. Also the external references known from CityGML 1.0 and 2.0 are 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 / databases, but these can now be additionally qualified by a relation type given by an additional URI. Again, this allows for direct mapping onto RDF triples. These improvements help to increase interoperability of CityGML with Semantic Web technologies.

The new class AbstractToplevelCityObject allows restricting the feature types that are allowed as members of a CityModel feature collection. This way, we can express in the data model that e.g. from the Building Module only the class Building can occur as a member of a CityModel and that Door or RoofSurface are not allowed as direct members of CityModel. This information is required for software packages that want to filter imports, exports, and visualization according to the general type of city object (e.g. only show buildings, solitary vegetation objects, and roads). Application Domain Extensions of CityGML should also make use of this concept, such that software tools can learn from parsing the ADE XML schema what are the main, i.e. the toplevel feature types of the extension. Please note that this does not affect the general possibility of a Web Feature Service to query also subfeatures like a door or window of a Building object. The resulting features of a WFS GetFeature operation are represented as members of a WFS:FeatureCollection object which allows to have arbitrary geographic features as member elements.

The new Construction module

A new thematic module is being introduced called “Construction”. This module defines the concepts that are common to all kinds of man-made constructions like buildings, bridges, tunnels, and other constructions. It comprises also the definition of the different kinds of thematic surfaces like RoofSurface, GroundSurface, WallSurface etc. The thematic modules Building, Bridge, and Tunnel are now defined as further specializations of the concepts provided in the Construction module. This has lead to a substantial simplification of the data models of these modules. The new feature type OtherConstruction, which should be used to model those man-made structures that are neither buildings, tunnels, nor bridges (e.g. large chimneys, city walls etc; see CityGML 3.0 Work Package 9), has been incorporated into the Construction module.

All construction objects (including buildings, tunnels, bridges, other constructions) now can specify multiple elevation levels with regard to different construction height points (e.g. top point, ridge point, lowest point on terrain). Also multiple measured height properties are allowed where the high and low reference has to be stated explicitly. Constructions can store a date of construction, demolition, and multiple dates of renovations. Again, these properties are inherited to the specialized classes Building, Tunnel, Bridge, and OtherConstruction. In fact, these properties were adopted from the INSPIRE BU data theme resolving the problem of the ambiguity of the measuredHeight property of Building objects in CityGML 1.0 and 2.0 and further improving the interoperability of CityGML 3.0 with INSPIRE.

The new Versioning module

CityGML 3.0 introduces bitemporal timestamps for all objects. Besides the well-known attributes “creationDate” and “terminationDate” from CityGML 2.0, all objects now can have a second lifespan expressed by the attributes “validFrom” and “validTo”. The naming and the semantics of these four attributes are aligned with the INSPIRE data specifications. “validFrom” and “validTo” represent the lifespan of an object – or more precise: the specific version of an object – with regard to its existence in the real world. “creationDate” and “terminationDate” refer to the time period over which the respective version of the object is considered an integral part of the 3D city model. This bitemporal model allows for queries

like “How did the real world looked like at a specific point in time?” and “How did the 3D city model looked like at a specific point in time?”.

Since CityGML 3.0 is based on GML 3.2.1 / 3.3, each geographic feature now has two identifiers: the “identifier” property and the “gml:id” attribute. The value of the “identifier” property is intended to be stable along the lifetime of the real world object. This means, it should be a globally unique value which also does not change when some attribute values (or the geometry) of the object are being changed. The “gml:id” attribute is intended to be constructed from the “identifier” with a concatenated timestamp (the “creationDate” value) to mark the respective version of the object. Everytime an object will be modified, the previous version should receive a proper “terminationDate” value and a new version of the object with the proper “creationDate” and a corresponding new “gml:id” will have to be created. This way not just the current version of a 3D city model can be represented in CityGML, but the entire history (or just some part of the history) can be represented and exchanged. This is useful to feed the 3D city model including its change history to some analytical tools, which e.g. try to detect transformation or change processes in the city over time.

The Versioning module also defines two new feature types called “Version” and “VersionTransition” allowing to explicitly define named states (“versions”) of the 3D city model and denoting all the objects (in their specific versions) belonging to such states. “VersionTransitions” allow to explicitly link different versions of the 3D city model by describing the reason of change as well as enumerating all modifications. Versions are also allowed to fork in order to represent alternative future developments of a city (e.g. to represent competing designs for some urban planning). Since “Version” and “VersionTransition” are being modeled as features, version management can be performed by a standard OGC Web Feature Service (WFS) without the need of additional vendor-specific extensions.

Details on the versioning concept are given in the 2015 paper “Managing Versions and History Within Semantic 3D City Models for the Next Generation of CityGML” by Chaturvedi, Smyth, Gesquière, Kutzner, and Kolbe. The paper can be downloaded from here: https://mediatum.ub.tum.de/doc/1276238/362001.pdf

Like with all CityGML modules, the usage and support of Versioning is optional.

The new Dynamizer module

The Dynamizer module has been developed to improve the usability of CityGML for different kinds of simulations on the one hand, and to facilitate the integration of sensors (and the Internet of Things IoT in general) with 3D city models on the other hand. The integration of sensors with 3D city models is important e.g. in the context of smart cities and digital twins.

Both simulations and sensors provide dynamic variations of some measured or simulated properties like, for example, the electricity consumption of a building. The variations of the value are typically represented using timeseries data. The data source of the timeseries data are either sensor observations (e.g. from a smart meter), prerecorded load profiles (e.g. from an energy company), or the results of some simulation run.

Dynamizers are special objects linking the timeseries data (from either source) to a specific attribute / property of a specific object within the 3D city model. The timeseries data can be given in different representations (OGC TimeseriesML, OGC Observations & Measurements, tabulated data in external files like CSV or MS Excel) or as a link to a sensor service like the OGC SOS or SensorthingsAPI. A Dynamizer references the attribute (e.g. geometry, thematic data, or appearance) of an object within a 3D city model providing dynamic values, effectively overriding the static value of the referenced object attribute. When a Dynamizer refers to sensor observation data, it establishes an explicit link between the sensor and the respective property of the city model object that is measured by it. By creating 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 or include these special types of features.

Details on the Dynamizer concept and some demonstration applications showing a) the integration of weather sensors with the city model, and b) the representation of the time-varying results of a solar potential analysis are given in the recent Engineering Report on the OGC Future Cities Pilot – Phase 1 (FCP1): http://docs.opengeospatial.org/per/16-098.html

Like with all CityGML modules, the usage and support of Dynamizers is optional.

The revised Transportation module

The Transportation module defines classes for the representation of central elements of the traffic infrastructure. In order 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.

Transportation objects like Roads, Tracks, or Railways can now be subdivided into sections. Sections can be regular road, track or railway legs, intersection areas, or roundabouts. Intersection areas as well as roundabouts can belong to multiple Road or Track objects avoiding the redundant representation of shared spaces. TrafficArea and AuxiliaryTrafficArea are changed to TrafficSpace and AuxiliaryTrafficSpace. Each TrafficSpace can have an optional ClearanceSpace. Since the geometry of spaces can now be also represented by point clouds, the ClearanceSpace could be modeled directly from the result of a mobile laser scanning campaign. 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 TrafficSpace object. Each TrafficSpace can be linked to predecessor and successor TrafficSpaces. This information is typically used (and required) in navigation systems and traffic simulations.

Details and examples for the new transportation module are given in the 2017 paper "CityGML and the streets of New York - A proposal for detailed street space modelling" by Beil and Kolbe. The paper can be downloaded from here: https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-4-W5/9/2017/

Additional Remarks and Open Questions

There are a number of additional remarks and open questions. Here are some (this list definitely is not complete – please provide any further questions not addressed so far):

- Some data types in the EA file currently still refer to the old ISO 19103:2005 UML model. We will correct this, so that only data types from the ISO 19103:2015 UML model will be used.
- We did not yet derive XSDs from the EA file using ShapeChange and suppose that this is not yet working correctly as some tagged values still need to be set. This will also be fixed in the next weeks.
- Textual definitions of all CityGML concepts are not assigned to all classes, attributes, and associations yet. This will be completed over the next weeks. This information will be used to automatically derive the feature catalogue for CityGML 3.0.
- The subset connector in the "Core" module defines that the association between "AbstractVoid" and "AbstractVoidSurface" represents a partial set of all possible associations defined by the association between "AbstractSpace" and "AbstractSpaceBoundary".
- Should the class "CityModel" be associated with the class "Dynamizer" (similar to the association between "CityModel" and "Appearance") or should Dynamizer features only appear as properties / child elements of those city objects which contain dynamic data?
- How does the new model affect compatibility with LandInfra / InfraGML?
- How are existing CityGML ADEs affected by the new CityGML 3.0 data model?
- The new Core model was first presented at the OGC TC Meeting in September 2017 in Southampton. There was a discussion about the naming of some of the classes like AbstractUnoccupiedSpace versus AbstractVoid. We have modeled "Room" as a subclass of AbstractUnoccupiedSpace. This is not completely true, because the furniture and the installations within a room occupy some of the free space. There is also a discussion on how to represent windows and doors in walls. The current approach is to represent the opening as a "Void" object, in which a window or door object will be embedded. Should the classes "Door" and "Window" be subclasses of "AbstractVoid" or should they better be just associated with a (non-abstract) "Void" class? We are interested in hearing your opinion!