Final report 2020

Methods for identification of free navigable space

Mitko Aleksandrov, Abdoulaye Diakité, Jinjin Yan, Wei Li and Sisi Zlatanova

Introduction

The research within this project is a part of four-year (2018-2022) international project named iNous focusing on seamless indoor-outdoor navigation for emergency response. The project is funded by South Korea government and leaded by Pusan University, South Korea. The overall idea of the project is to develop a workflow for navigation for the purpose of disaster management system as illustrated below (Figure1):

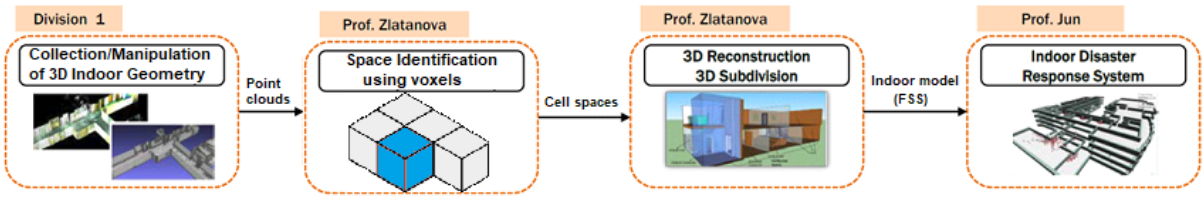


Figure 1. Workflow of the proposed Indoor disaster response system

In the third year of the project, the team of professor Zlatanova has concentrated on advancing the fee space identification from point clouds using voxels. The general scope of the research and developments have been defined as:

- Comparing algorithms for voxel space identification and subdivision
- Comparing algorithms for voxelisation of vector models
- Rounding-off the developments of IndoorGML and preparing examples
- Investigation of approaches to store voxels in a DBMS
- Developing algorithms to generate F-Spaces (functional).

The above objectives have been achieved by the following activities and developments:

- Literature review on voxelisation
- Preparing GML examples for IndoorGML
- Data schema and tests for storage of voxels.

Administrative information

In the specified period the following researchers were involved: Mitko Aleksandrov, Abdoulaye Diakité, Jinjin Yan, Wei Li and Sisi Zlatanova. In this period no specific meetings were organised with the other teams on this specific topic although a number of discussions were carried out regarding the IndoorGML.

Scientific project information

This year the research continued developing a voxel-based workflow for indoor modelling. Two major topics were considered 1) voxelization of geometric models and 2) management of voxel data. As discussed in (Diakite and Zlatanova, 2019), BIM and B-reps 3D representations can become very complex and lead to issues related to validity and intersections of solids or existence of tiny solids. Voxelisation of B-reps can be a promising approach to avoid computational complexity and validity issues. A series of algorithms were investigated and a journal papers is in process of preparation. As often discussed in the literature high resolution voxel models can result in large volumes of data. We have investigated several methods to for management of voxel spaces in PostGIS, using different spatial schemas. Parallel to the voxel-based workflow, we have continued the work on finalising IndoorGML 2.0

Algorithms for voxelisation

A literature review was completed on voxelisation approaches. Generally, voxelisation is a discrete approximation of an object or space with embedded objects. Voxel-based approaches are used in many domains, but most of the algorithms originate from computer graphics. The voxelisation can be subdivided in three general group referring to points, curves, surfaces and solids. Currently, no uniform approach exist that can be applied for all types of objects and therefor it is important to identify which algorithms are most appropriate for geographic objects. Geographic objects are related to each other and the relationship should be preserved in the voxelised space. This often relates as a 'half-voxel' problem, which can create all kind of issues from shift of objects, to reduction or increase of objects and even disappearing of objects.

Storage of Voxels

To make use of voxel data in different scenarios, an efficient storage and retrieval system is therefore needed. We investigate four data layouts for storing and managing the 3D raster data in PostgreSQL/PostGIS, namely - (1) a flat ARRAY table; (2) a POINT geometry table; (3) a MULTIPOINT geometry table, and (4) a PCPATCH table with the help of Pointcloud.

ARRAY Layout

The majority storage models can be adopted for voxel management is based upon the organisation of voxels in the flat ARRAY table, where each voxel is stored separately in a single row using common data types (INTEGER or NUMERIC). Each voxel attribute constitutes a separate field. The voxels are populated in a table and indexed using a B-tree index in the X, Y, and Z coordinates respectively.

	id integer	x numeric (8,1)	y numeric (8,1)	z numeric (8,1)	name character varying	ifc character varying	category objectclass
1	1	336385.0	6245328.2	27.0	Rupert Myers-M15	[null]	building
2	2	336385.0	6245328.2	27.2	Rupert Myers-M15	[null]	building
3	3	336385.0	6245328.2	27.4	Rupert Myers-M15	[null]	building
4	4	336385.0	6245328.2	27.6	Rupert Myers-M15	[null]	building
5	5	336385.0	6245328.2	27.8	Rupert Myers-M15	[null]	building

Figure 2 Example of a table created in PostgreSQL to store ARRAY

POINT Layout

A spatial POINT represents a single location on the Earth. This point is represented by a single coordinate (including either 2-, 3- or 4-dimensions). Points are used to represent objects when the exact details, such as shape and size, are not important at the target scales. For POINT table, we

keep semantic information, and create one geometry column with 3D point in PostGIS. Regarding to index, we create B-tree index on semantic columns, and GiST index on geom column.

	id integer	name character varying	ifc character varying	category objectclass	geom text
1	2	[null]	[null]	terrain	POINT Z (336434 6245891.4 31.6)
2	3	[null]	[null]	terrain	POINT Z (336434 6245891.4 31.8)
3	4	[null]	[null]	terrain	POINT Z (336434 6245891.6 20.2)
4	5	[null]	[null]	terrain	POINT Z (336434 6245891.6 20.4)
5	6	[null]	[null]	terrain	POINT Z (336434 6245891.6 20.6)

Figure 3: Example of a table created in PostgreSQL to store POINT layout.

MULTIPOINT Layout

MULTIPOINT is another geometry that consists of a collection of POINT. In this kind of layout, we consider regarding each object as one multipoint, that means, voxels in one MULTIPOINT geometry have same objID.

When using the multipoint geometry type, we need to consider how to cut the voxel data, that is, those Voxels stored in a multipoint object. We propose a semantic-based voxel data partitioning strategy. Specifically, we want to store all voxels with the same semantic information in a multipoint object, including object semantics and IFC semantics. Moreover, similar to POINT, two indexes (i.e., B-tree and GiST) are built on semantic columns and geom respectively.

	id integer	name character varying	ifc character varying	category objectclass	geom text
1	57	Roundhouse-E6	IfcSite	building	MULTIPOINT Z (336098.5 6245700.4 28.7,...
2	60	Red Centre-H13	IfcMember	building	MULTIPOINT Z (336303 6245537.9 42.6,3...
3	62	Red Centre-H13	IfcDoor	building	MULTIPOINT Z (336307.7 6245536.3 36.4,...
4	63	Red Centre-H13	IfcOpeningElement	building	MULTIPOINT Z (336303.8 6245544.3 45.3,...
5	64	Science Theatre-F13	IfcOpeningElement	building	MULTIPOINT Z (336342.7 6245628.1 29.4,...

Figure 4:. Example of a table created in PostgreSQL to store MULTIPOINT layout.

PCPATCH Layout

Pointcloud is a PostgreSQL extension for storing point cloud (LIDAR) data, where PcPatch can be regarded a potential structure used for management of voxel model in PostgreSQL. In our case, we collect a group of voxels with same semantic information into a PcPatch. Each patch should hopefully contain voxels that are near together. That is the same partition strategy as MULTIPOINT.

Following the Pointcloud schema 3 (PostgreSQL Pointcloud deals with all this variability by using a “schema document” to describe the contents of any particular LIDAR point.), we prepare a schema document to describe the contents of any particular voxel. Each voxel contains three dimensions, named X; Y; Z, and each dimension will be of INTEGER data type, with scaling 0:1. This schema document is stored in the pointcloud formats table, along with a pcid (i.e.,“pointcloud identifier”).

Different from multipoint representation, GiST index is created based on 2D bounds of the patch because it cannot be indexed directly on the PcPatch type. Fortunately, Pointcloud provides

PC EnvelopeGeometry(PCpatch) functions that can directly obtain bounding box as a PostGIS Polygon 2D. Thus, we can index 2D Polygon.

Data Output | Explain | Messages | Notifications

	id integer	name character varying	ifc character varying	category objectclass	pa text
1	1	Rupert Myers-M15	[null]	building	{"pcid":1, "npts":6925654, "srid":28356, "compr":"dimensional","dims":[{"pos":0,"na...
2	2	International House-...	[null]	building	{"pcid":1, "npts":2243609, "srid":28356, "compr":"dimensional","dims":[{"pos":0,"na...
3	3	UNSW Village-B10	[null]	building	{"pcid":1, "npts":22333289, "srid":28356, "compr":"dimensional","dims":[{"pos":0,"n...
4	4	Barker Street Carpar...	[null]	building	{"pcid":1, "npts":10120534, "srid":28356, "compr":"dimensional","dims":[{"pos":0,"n...
5	5	Law-F8	[null]	building	{"pcid":1, "npts":9510203, "srid":28356, "compr":"dimensional","dims":[{"pos":0,"na...

Figure 5: Example of a table created in PostgreSQL to store PCPATCH layout.

The different data schemas influence the spatial queries that can be performed. The ARRAY Layout is simple but the native PostGIS spatial operations can not be executed. Point and Multopint use the native PostGIS spatial data types, and all spatial indexing and spatial operations can be used. PC-Patch is a data type dedicated to point clouds and although a spatial data type, it cannot use the geometric functions and opetation of PostGIS. It has several dedicated functions. If the purpose of the data base is to maintain objects, the Miltipoint data type is most beneficial. However, if each voxel carries information, Mulipoint and PC-Patch are not appropriate. Point data type can be considered superior to Array because the spatial functions can be used. Semantic queries can be executed in all data layouts. Mutipoint and PC-Patch are advantageous to Pints and Arrays because the tables have less records and hence the queries are executed quicker. Figures below illustrate sematic queries.

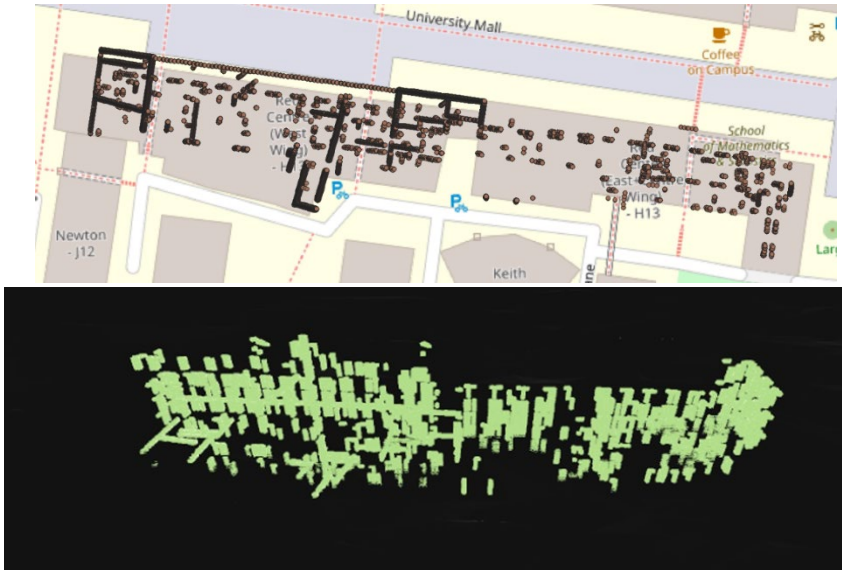


Figure 6: Query of all doors in a building: 2D (left) and 3D(right) view.

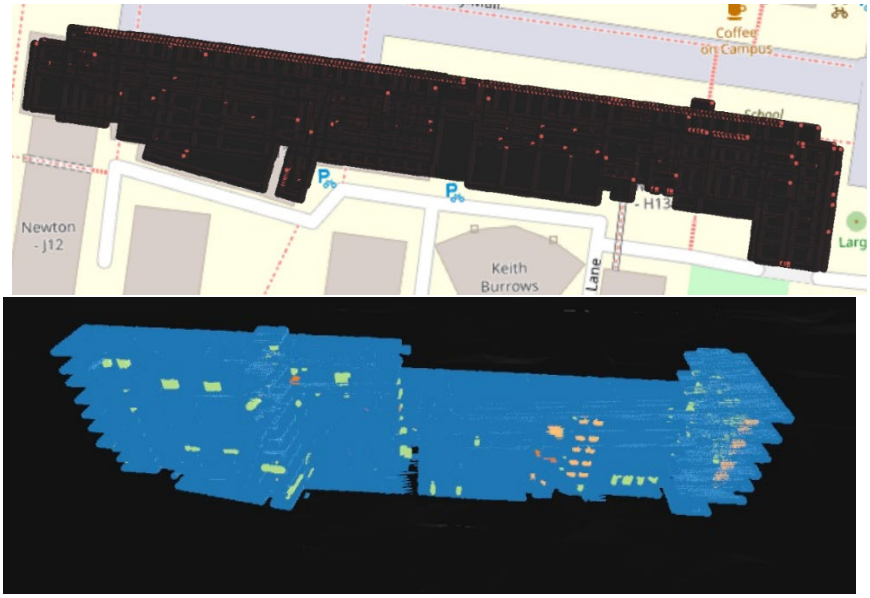


Figure 7: Sematic query of all Spaces, Windows, Doors and Slabs: 2D (left and 3D (right) view

IndoorGML 2.0

Several changes were brought compared to IndoorGML 1.x. Some classes and attributes were renamed, modified, or added in the standard for the sake of simplicity, clarity and genericness. To facilitate the comparison, Figure 1 illustrates the core module of IndoorGML1.x and IndoorGML2.0 that integrates the changes detailed in the following subsections.

Renamed Classes

PrimalSpaceLayer and DualSpaceLayer: The two principal classes of the core module are the dedicated to the representation of the primal space (3D geometry) and the dual space (3D topology). While they used to be referred to as PrimalSpaceFeatures and MultiLayeredGraph in IndoorGML1.x, they are now renamed respectively as PrimalSpaceLayer and DualSpaceLayer. This choice is obviously motivated by a clarification goal.

CellBoundary: According to the definition of IndoorGML, the primal space describes the indoor space as an aggregation of space units called cells. Those 3D space cells, and their boundaries were respectively called CellSpace and CellSpaceBoundary. In the IndoorGML2.0 proposal, the latter is renamed as CellBoundary.

Node and Edge: One of the common confusions involved in the use of IndoorGML1.x is related to the description of the topological network in the dual space. In fact, the network which is derived based on the Poincare´ Duality is composed of nodes representing the cells of the primal space and edges representing their adjacency and, under certain conditions, their connectivity. For the sake of completeness, one should note that 'connectivity' relationship, which is of importance for navigation applications, is a type of neighborhood relationship that corresponds to the adjacency graph of navigable spaces only. However, those graph components were called State for a node and Transition for an edge in IndoorGML 1.x. While such naming has not been motivated in the standard definition, it refers to navigation cases in which people are commonly staying in spaces, which gives their state. While the movement from one space to another is short-term and represents their transition. These notations then become confusing for non-navigation applications. Therefore, they are renamed into Node and Edge in IndoorGML2.0.

Modified classes and attributes

Geometry of cells and external references: The UML diagram of IndoorGML1.x suggests that there are dedicated classes for the geometric information related to the cell spaces and their boundaries, as well as the network. The concept of keeping geometry in separate classes has its roots in topological representations, i.e. common faces are to be maintained once in the model and they are 'reused' by the neighbouring solids. However, the indoor spaces used for navigation and other LBS applications most commonly do not share faces, which results in storing all faces of solids. Hence, the classes CellSpaceGeometry, CellSpaceBoundaryGeometry and Geometry (for the network) can be simply considered as attributes of those features. This is therefore clarified in the UML of IndoorGML2.0, where any CellSpace, CellBoundary, Node and Edge is provided with an attribute CellSpaceGeom, CellBoundaryGeom and Geometry (for Node and Edge). Similar process is done for the external references for which an attribute externalReference has been added to CellSpace and CellBoundary classes.

InterLayerConnection: The InterLayerConnection class is dedicated to the description of the links between different layers. While in IndoorGML1.x it was specifically dedicated to the connection between network components (MultiLayeredGraph), it is extended in IndoorGML2.0 to also account for interlayer connections between elements of the primal space. This is motivated by the fact that the notion of layer is improved in IndoorGML2.0. A layer can be solely composed of primal space features, dual space features or a combination of both. In the previous version of the standard, only interconnection between dual spaces of different primal spaces was possible.

New class and attributes

The need to improve the information organisation in the standard and its support for indoor features other than spaces led to the introduction of one new class and several attributes in the core module.

ThematicLayer: With IndoorGML1.x, a model is composed of one IndoorFeatures instance which aggregates two lists: one instance of PrimalSpaceFeatures which aggregates all the CellSpace elements existing in the model, and one instance of MultiLayeredGraph aggregating SpaceLayer components as well as InterLayerConnection instances. As mentioned previously, this enables the multi-layer mechanism only for the dual space (the networks). To make it possible to efficiently store several layers of geometry (CellSpace) and/or topology (Node and Edge), a reorganisation is needed. The ThematicLayer is proposed for IndoorGML2.0, as an aggregation of PrimalSpaceLayer and DualSpaceLayer instances to help defining layers separately with the ability to contain fully or partly components of the core module. The class comes with two attributes: semantic and Theme. The semantic is set as a boolean as it is simply an indication that there is semantic information associated to the PrimalSpaceLayer. Because the navigation module is currently the only semantic module available, a boolean is enough to indicate its presence for now. This is however susceptible to evolve in the future (e.g. into a codeList). The Theme attributes determines whether the layer is of type TOPOGRAPHIC, SENSOR, LOGICAL, TAGS or UNKNOWN. This codeList is already in use to determine the class type of a SpaceLayer entity in IndoorGML1.x. Therefore, it may also be subject to change in the future, once more relevant themes will be identified (e.g. we could introduce a LEGAL theme considering related work in a land administration standard (Alattas et al., 2017)).

PoI: The notion of PoI occupies a central place in LBS applications. As it name says, it corresponds to locations of interest for a specific purpose on a map. A 3D indoor map such as IndoorGML should therefore be able to represent such notion (Park et al., 2016). While recent works are investigating potential additional IndoorGML modules for this purpose (Claridades et al., 2019), we present here a preliminary approach that can directly fit to the current core module of the standard.

Because every space comes down to a CellSpace in indoorGML and any space can be a location of interest, may it be empty or occupied by a physical object (see Section 3), we propose to append the attribute PoI to the CellSpace class of the core module. For the moment, the attribute is just a boolean allowing to tag a cell as a PoI and implement specific considerations with respect to it in the corresponding application. Thanks to the relations between the primal and dual space in the standard, one can use the information on a navigation network for example and put a higher weight of edges leading to nodes associated with PoI cells.

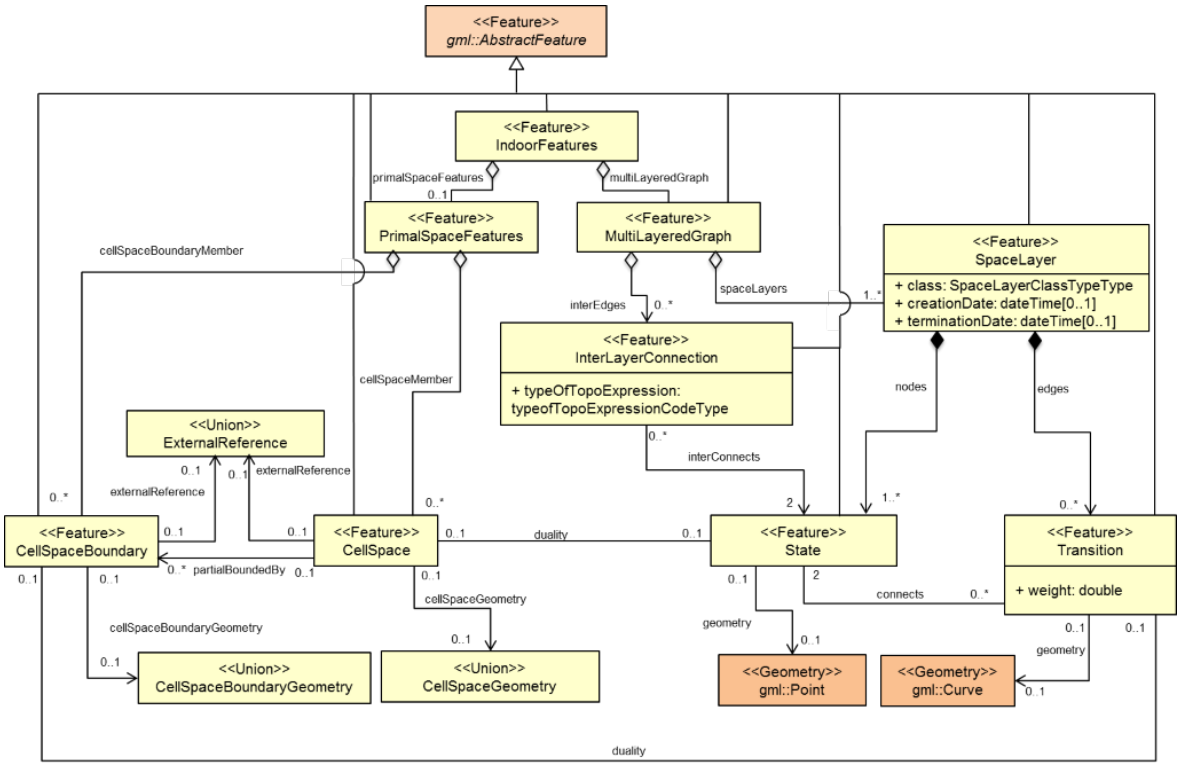


Figure 8: UML Diagram of IndoorGML core model 1.0

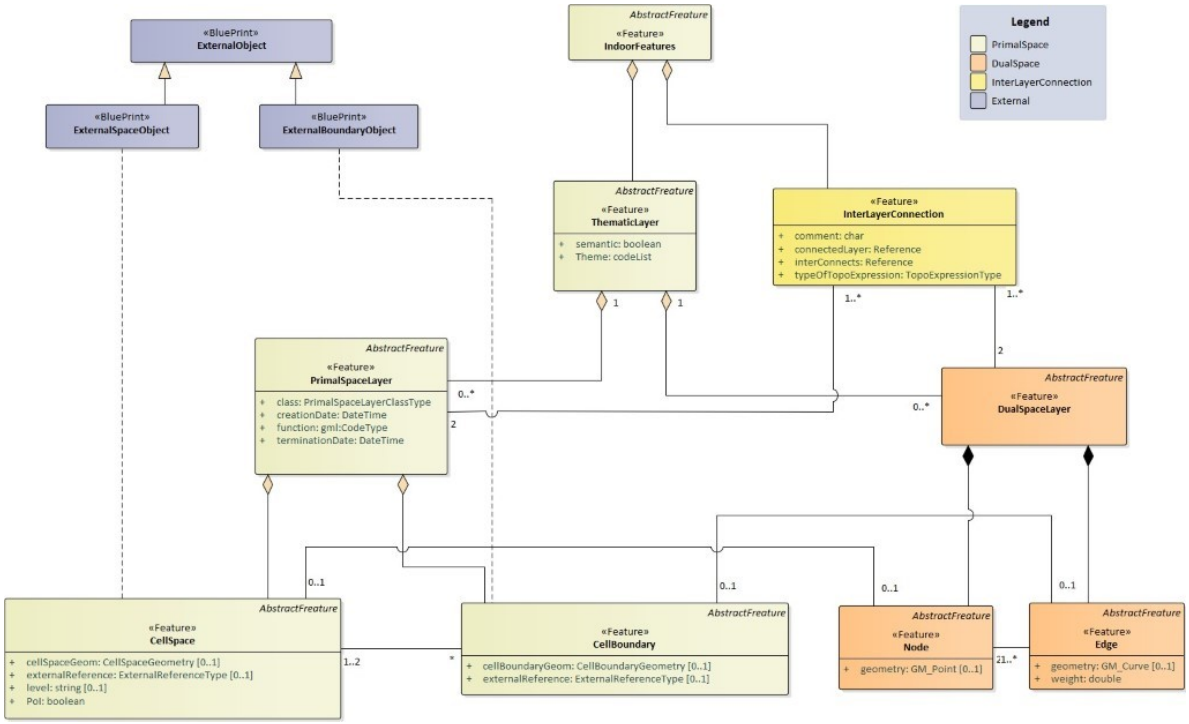


Figure 9: UML diagram of IndoorGML core module

GML examples

We provide some case illustrations of IndoorGML2.0 to demonstrate the flexibility offered by the new UML diagram. We explore different scenarios: (a) geometry only, (b) network only and (c) geometry + network combined. This involves the use of other common standards as data source and lead to specific use of the classes available. We use a BIM (IFC) model for the test (see Fig. 10), focusing mainly on its components that are relevant to IndoorGML.

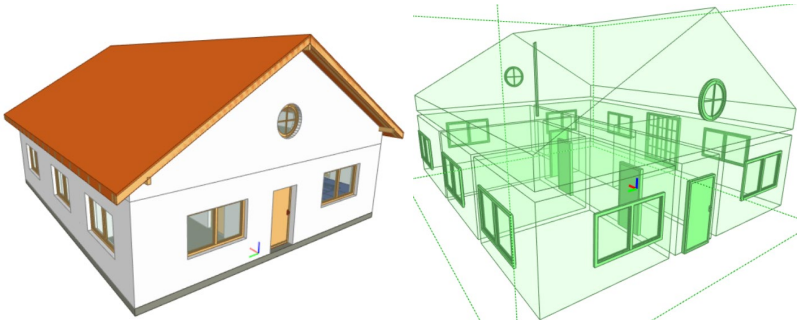


Figure 10: A BIM model (left) and its corresponding spaces and openings (right)

Geometry only

IndoorGML could be for the sole purpose of sharing the indoor spaces as CellSpace entities. Typically, every IfcSpace entity coming from an IFC model is a direct correspondence to a CellSpace in IndoorGML. The 3D geometry can be directly stored in the cellSpaceGeom attribute, and additionally, CellBoundary entities can also be directly stored if available.

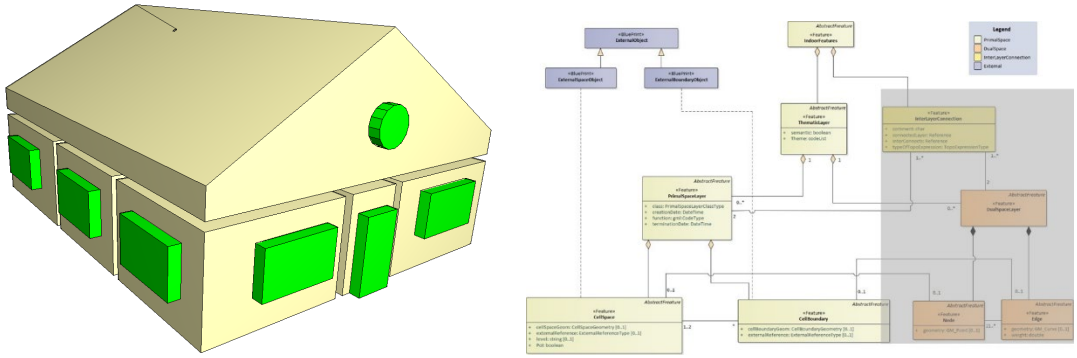


Figure 11: An indoorGML model with geometry only

Network only

Similarly to the previous example, an IndoorGML file could serve for the exchange of topological (network) information of a building model.

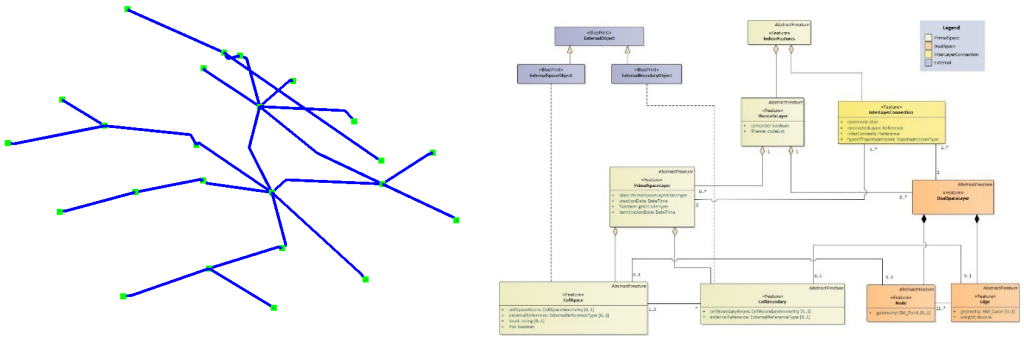


Figure 12: IndoorGML with topology (network) only.

In a scenario where the geometry is not needed, this could be a convenient option to use, guaranteeing lightweight files. Figure 12 shows an example of rooms and openings adjacency network computed from the input BIM model. The Node entities are computed using the centroid of the IfcSpace elements, while the Edge elements are obtained by connecting the nodes of adjacent spaces. Similarly to space boundaries, the information of the connectivity between the Ifc entities may be directly available in the model (e.g. using IfcRelSpaceBoundary relations). Note that for the case of the topology only, all the classes of the core model are still required, but the CellSpace entities will carry no geometric attribute.

Geometry, topology and semantic

Most of IndoorGML files are expected to carry those information (although semantic is currently not included in the core module). The image below shows the structure of an IndoorGML2.0 file with geometric description of CellSpace objects, as well as semantic information associated to the cells, based on the navigation module (e.g. navi:GeneralSpace). The network part carries topological information and the geometry of the network, with nodes and edges. This structure is similar for the previous example, with only the relevant part being described in the corresponding files.

```
<core:IndoorFeatures
gml:id="InFt_891"
xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:core="http://www.opengis.net/indoorgml/1.0/core"
xmlns:navi="http://www.opengis.net/indoorgml/1.0/navigation"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/indoorgml/1.0/core http://schemas.opengis.net/indoorgml/1.0/indoorgmlcore.xsd
http://schemas.opengis.net/indoorgml/1.0/indoorgmlnavi.xsd">
    <core:ThematicLayer gml:id="TmL_891">
        <semantic>true</semantic>
        <Theme>Topographic</Theme>
        <core:PrimalSpaceLayer gml:id="PSL_891">
            <navi:GeneralSpace gml:id="CS_1111_19">
                <core:cellSpaceGeometry>
                    <core:Geometry3D>
                        <gml:Solid gml:id="CG-CS_1111_19">
                            <gml:exterior>
                                <gml:Shell>
                                    <gml:surfaceMember>
                                        <gml:Polygon>
                                            <gml:exterior>
                                                <gml:LinearRing>
                                                    <gml:pos srsDimension="3">7.650000 4.250000 0.000000</gml:pos>
                                                    <gml:pos srsDimension="3">11.700000 4.250000 0.000000</gml:pos>
                                                    <gml:pos srsDimension="3">11.700000 4.250000 2.500000</gml:pos>
                                                    <gml:pos srsDimension="3">7.650000 4.250000 0.000000</gml:pos>
                                                </gml:LinearRing>
                                            </gml:exterior>
                                        </gml:Polygon>
                                    </gml:surfaceMember>
                                    <gml:surfaceMember>
```

[…]

```
        <core:DualSpaceLayer gml:id="DSL_1111">
            <core:Node gml:id="DSL_1111_ST0">
                <core:duality xlink:href="#CS_891_0"/>
                <core:connects xlink:href="#TCS_1111_22-CS_891_0"/>
                <core:geometry>
                    <gml:Point>
                        <gml:pos>9.210000 0.150000 1.400000</gml:pos>
                    </gml:Point>
                </core:geometry>
            </core:Node>
            <core:Node gml:id="DSL_1111_ST1">
            <core:Node gml:id="DSL_1111_ST10">
```

[…]

```
            <core:Edge gml:id="DSL_1111_TCS_891_17-CS_891_5">
            <core:Edge gml:id="DSL_1111_TCS_891_18-CS_891_8">
                <core:weight>1.0</core:weight>
                <core:connects xlink:href="#CS_891_18"/>
                <core:connects xlink:href="#CS_891_8"/>
                <core:geometry>
                    <gml:LineString>
                        <gml:pos>2.050000 7.845000 1.250000</gml:pos>
                        <gml:pos>2.926667 9.700000 1.200000</gml:pos>
                        <gml:pos>2.050000 9.850000 1.400000</gml:pos>
                    </gml:LineString>
                </core:geometry>
            </core:Edge>
        </core:DualSpaceLayer>
    </core:ThematicLayer>
</core:IndoorFeatures>
```

Other related topics

Within this project two other related topics were investigated. Substantial work was completed within IndoorGML 2.0 as well as extending the space-based concept to outdoor. Two critical papers for creating outdoor spaces where published.

Mapping UML schema to GML and SQL (Jinjin)

The UML diagram of IndoorGML 2.0 was automatically mapped to GML and SQL technical implementation. For the automatic mapping Enterprise Architect was used.

- **Mapping UML schema to GML**
  The Enterprise Architect offers interface to achieve this key step, which includes four steps: Code -> Export an XML Schema (XSD) File -> Set configurations -> Generate.

The first two steps can be seen in Figure 10. Figure 11 shows the rest two steps. In configurations setting, the Source Package of UML, encoding (default is Unicode (UTF-8)), the filename of the exported GML XSD file, as well as other preference options, should be specified. In this stage, we can preview and check the XSD to be exported by clicking the "View Schema". After clicking the bottom "Generate", the generating processes are monitored in the Progress box.

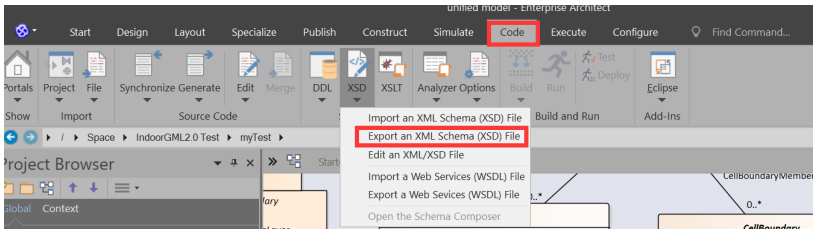


Figure 10: The interface of Enterprise Architect for mapping UML schema to GML

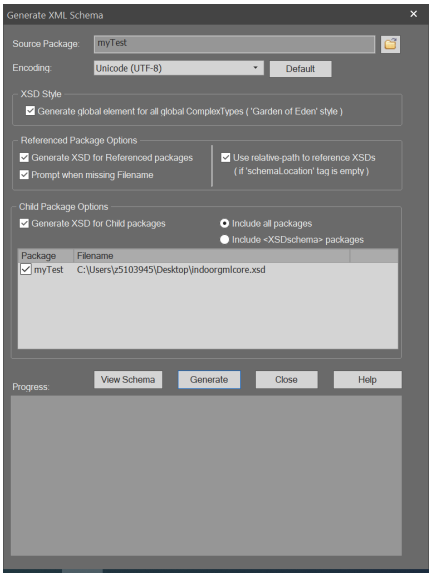


Figure 11: The interface of setting configures in Enterprise Architect

After finishing the whole process, we can find the exported XSD file. For instance, in the UML model, there are three classes: PrimalSpaceLayer, CellSpace and CellBoundary, in which each class has several attributes and the three classes have relationships. By following the four steps mentioned above, the three classes and their relationships are correctly recorded in the XSD file (Figure 13).

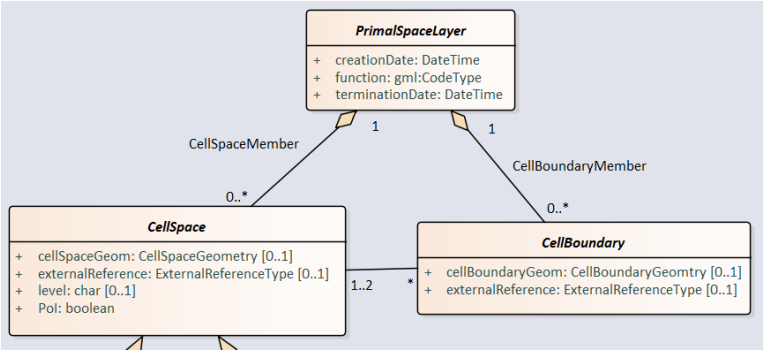


Figure 12: The UML diagram of three classes: PrimalSpaceLayer, CellSpace and CellBoundary

```
- <xs:complexType name="PrimalSpaceLayer" abstract="true">
    - <xs:sequence>
        <xs:element type="xs:dateTime" name="creationDate" maxOccurs="1" minOccurs="1"/>
        <xs:element type="gml:CodeType" name="function" maxOccurs="1" minOccurs="1"/>
        <xs:element type="xs:dateTime" name="terminationDate" maxOccurs="1" minOccurs="1"/>
        <xs:element type="CellBoundary" name="CellBoundary" maxOccurs="unbounded" minOccurs="0"/>
        <xs:element type="CellSpace" name="CellSpace" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
  </xs:complexType>
- <xs:complexType name="CellSpace" abstract="true">
    - <xs:sequence>
        <xs:element type="CellSpaceGeometry" name="cellSpaceGeom" maxOccurs="1" minOccurs="0"/>
        <xs:element type="ExternalReferenceType" name="externalReference" maxOccurs="1" minOccurs="0"/>
        <xs:element type="xs:string" name="level" maxOccurs="1" minOccurs="0"/>
        <xs:element type="xs:boolean" name="PoI" maxOccurs="1" minOccurs="1"/>
      </xs:sequence>
  </xs:complexType>
- <xs:complexType name="CellBoundary" abstract="true">
    - <xs:sequence>
        <xs:element type="CellBoundaryGeomtry" name="cellBoundaryGeom" maxOccurs="1" minOccurs="0"/>
        <xs:element type="ExternalReferenceType" name="externalReference" maxOccurs="1" minOccurs="0"/>
        <xs:element type="CellSpace" name="CellSpace" maxOccurs="2" minOccurs="1"/>
      </xs:sequence>
  </xs:complexType>
```

Figure 13: The XSD of the three classes: PrimalSpaceLayer, CellSpace and CellBoundary

Mapping UML schema to SQL

The Enterprise Architect also offers interface to map UML schema to SQL, which includes three steps: (i) Open EA file and Go to Configure -> Setting -> Database Datatypes; (ii) Select your file and then go to Design -> Transform -> Apply Transformation; and (iii) Select the DDL folder in the Project Browser and then go to Code -> Export a Database Schema (DDL).

I. Open EA file and Go to Configure -> Setting -> Database Datatypes (Figure 14)

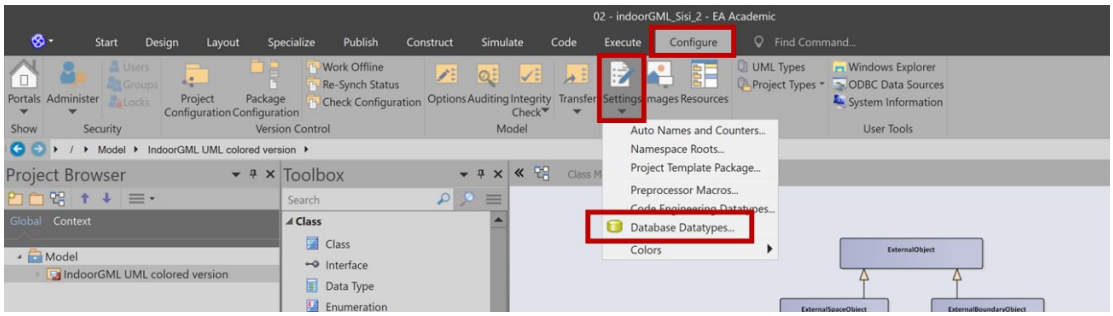


Figure 14: The interface of Enterprise Architect for mapping UML schema to SQL

In Database Datatypes select PostgreSQL and then make it as default choice (Figure 15).

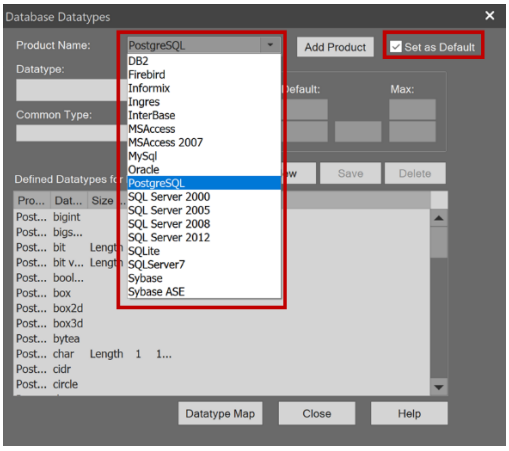


Figure 15: Set configurations of Database Datatypes

II. Select your file and then go to Design -> Transform -> Apply Transformation (Figure 16).

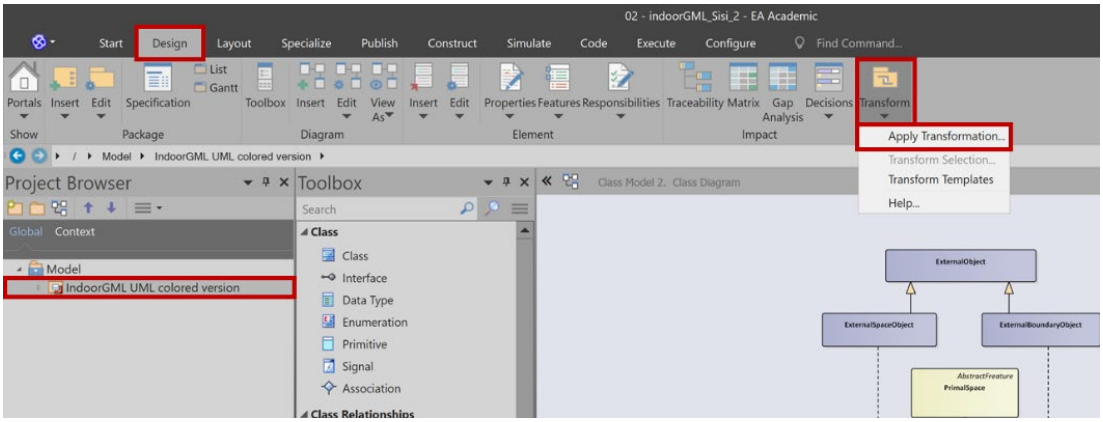


Figure 16: Select Design, Transform, and Apply Transformation

In the Model Transformation Select the Transformation as DDL and then press Do Transform (Figure 17).

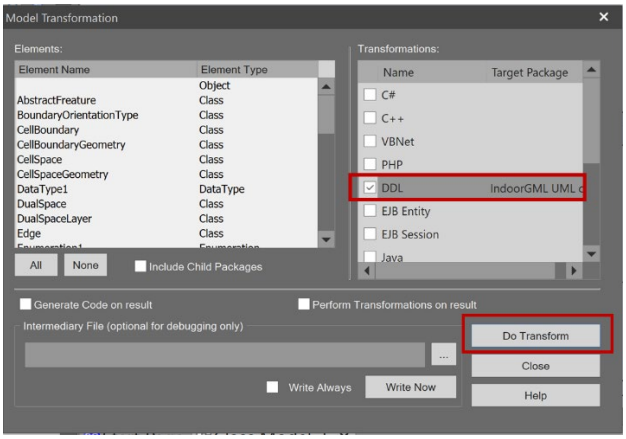


Figure 17: Do Transform

As result, a new folder will appear in the project browser (DDL folder) (Figure 18).

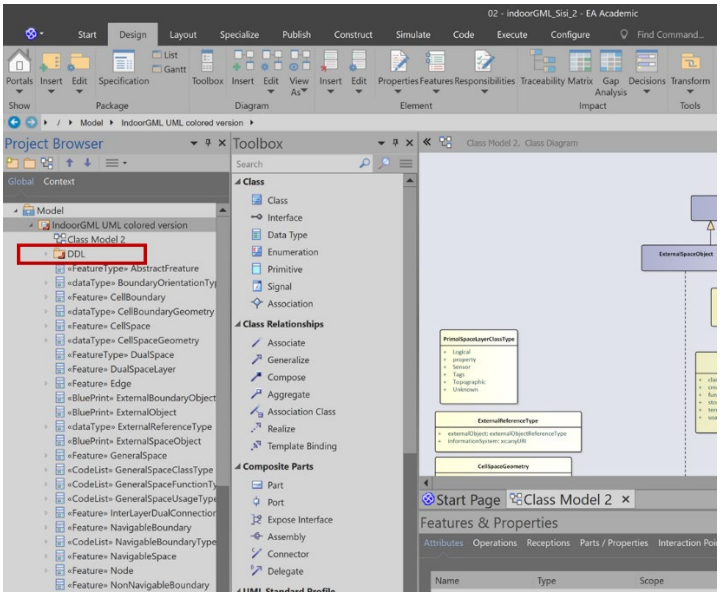


Figure 18: Select DDL folder

III. Select the DDL folder in the Project Browser and then go to Code -> Export a Database Schema (DDL) (Figure 19).

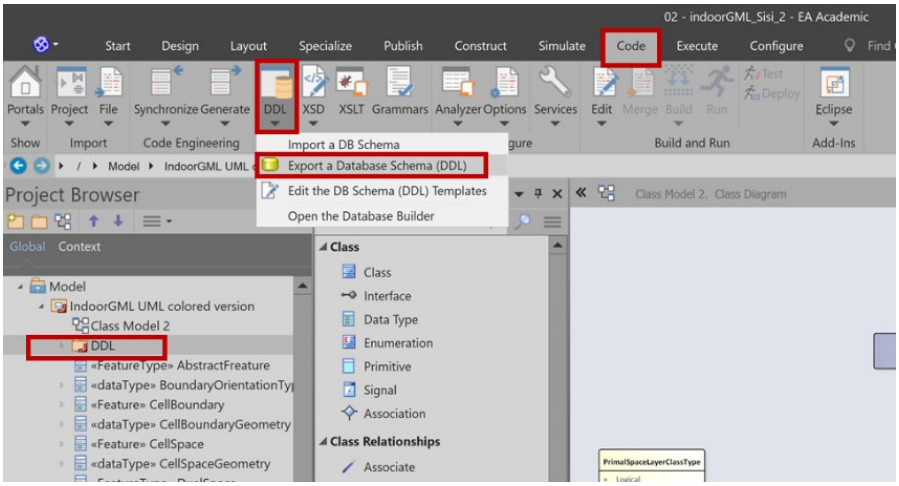


Figure 19: Select Code and Export a Database Schema (DDL)

Then select output file name and then press Generate (Figure 20 and 21).

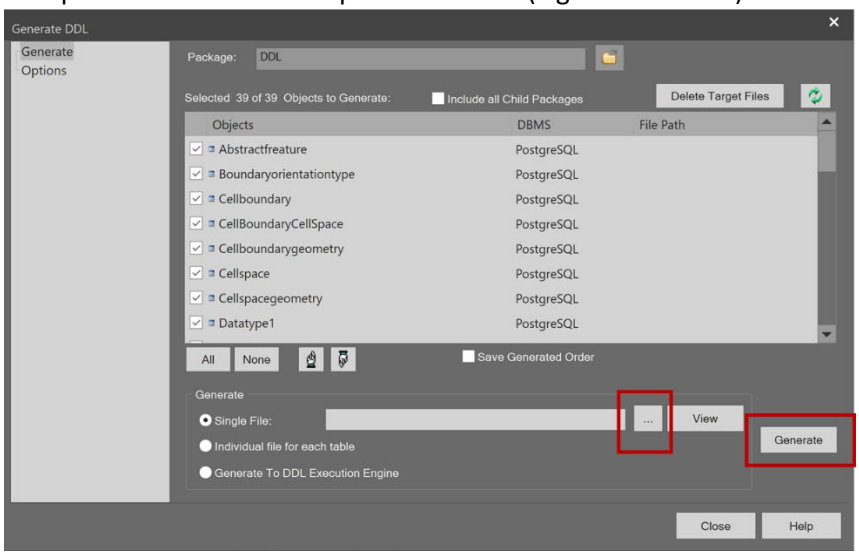
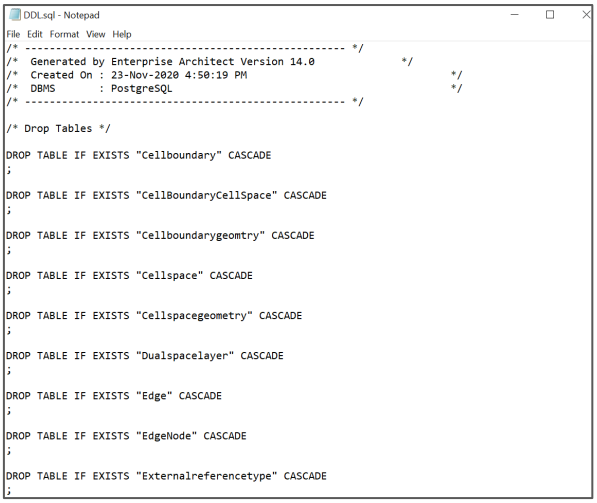


Figure 20: Set filename and path and generate



```
DDLsql - Notepad
File  Edit  Format  View  Help
/* ---------------------------------------------------- */
/*  Generated by Enterprise Architect Version 14.0          */
/*  Created On : 23-Nov-2020 4:50:19 PM                     */
/*  DBMS       : PostgreSQL                                 */
/* ---------------------------------------------------- */

/* Drop Tables */

DROP TABLE IF EXISTS "Cellboundary" CASCADE
;

DROP TABLE IF EXISTS "CellBoundaryCellSpace" CASCADE
;

DROP TABLE IF EXISTS "Cellboundarygeomtry" CASCADE
;

DROP TABLE IF EXISTS "Cellspace" CASCADE
;

DROP TABLE IF EXISTS "Cellspacegeometry" CASCADE
;

DROP TABLE IF EXISTS "Dualspacelayer" CASCADE
;

DROP TABLE IF EXISTS "Edge" CASCADE
;

DROP TABLE IF EXISTS "EdgeNode" CASCADE
;

DROP TABLE IF EXISTS "Externalreferencetype" CASCADE
;
```

Figure 21: The generated SQL scripts

Milestones

- Voxelisation: completed literature review.
- Management of voxels in DMS: investigated 4 approaches
- Completing IndoorGML 2.0:
- Space subdivision according to FSS using voxels: R-space (free navigable space) and O-spaces completed. An investigation on the way to automatically identify F-space is ongoing.

Conclusions

The project continues according the planning. A summary of the completed developments is given below:

- GML example for indoorGML 2.0
- SQL schema and SQL queries for voxels

Publications within this project

Diakité, A. A., Zlatanova, S., Alattas, A. F. M., and Li, K. J., 2020, Towards IndoorGML 2.0: Updates and Case Studies illustrations, Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci., XLIII-B4-2020, 337–344

Nikoohemat, S. A.A. Diakité, V. Lehtola, S. Zlatanova, G. Vosselman, 2020, Consistency grammar for 3D indoor model checking, Transaction in GIS , on-line (*doi*)

W. Li, S. Zlatanova, and B. Gorte, 2020, Voxel data management and analysis in PostgreSQL/PostGIS under different data layouts, ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., VI-3/W1-2020, 35–42, https://doi.org/10.5194/isprs-annals-VI-3-W1-2020-35-2020, 2020

Zlatanova, S., J. Yan, Y. Wang, A. Diakité, U. Isikdag, G. Sithole and J. Barton, 2020, Spaces in Spatial Science and Urban Applications—State of the Art Review, ISPRS Int. J. Geo-Inf. 2020, 9(1), 58;