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

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.

This year the research continued

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	name	ifc	category	geom
1	2	[null]	[null]	terrain	POINT Z (336434 6245891.4 31.6)
2	3	[null]	[null]	terrain	POINT Z (3364
3	4	[null]	[null]	terrain	POINT Z (336434 6245891
4	5	[null]	[null]	terrain	POINT Z (336434 6245891.6 20.4)
5	6	[null]	[null]	terrain	POINT Z (336434 6245891.8 20.6)

Figure 3: Example of a table created in PostgreSQL to store POINT 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	geo tex
1	57	Roundhouse-F6	IfcSite	building	MULTIPOINT Z (336098.5 6245[illegible]
2	60	Red Centre-H13	IfcMembe		
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 4
5	64	Science Theater-F13	IfcOpeningElement	building	MULTIPOINT Z (336342.7 6245628.1 29.4,...

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

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

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.

PrimalSpaceLayer and DualSpaceLayer: The two principal classes of the core module are the dedicated to the representation of the

Geometry of cells and external references

Because every space comes down to a CellSpace in indoorGML and any space can be a location of

Figure 9: UML diagram of IndoorGML core module

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

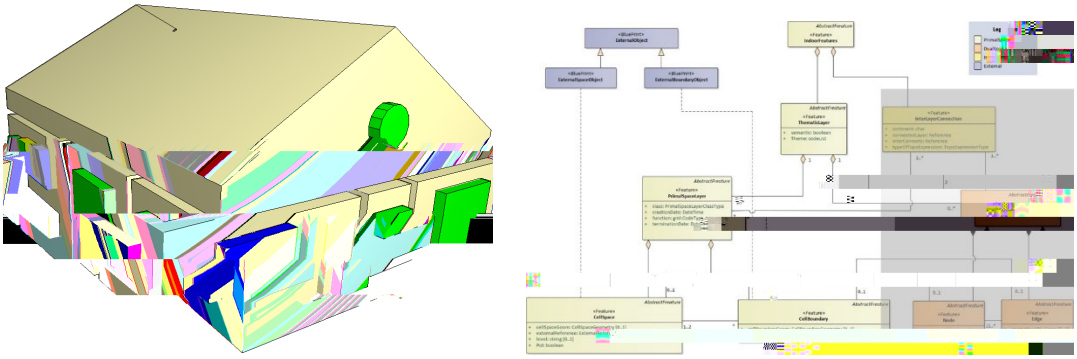

Figure 11: An indoorGML model with geometry only

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.

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="I[redacted]_891"
xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:xlink="http://[redacted]
xmlns:core="http://www.opengis.net/indoorgml/1.0/core"
xmlns:navi="http://www.opengi[redacted]gml/1.0/navigation"
xmlns:xsi[redacted]://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://[redacted]
http://schemas.opengis.net/indoorgml/1.0/indoorgmlnavi.xsd">
    <core:ThematicLayer gml:id="[redacted]">
        <semantic>true</semantic>
        <Theme>Topographic</Theme>
        <core:PrimalSpaceLayer gml:id="PSL_891">
            <navi:GeneralSpace gml:id="CS_1111_19">
                <core:cellSpaceGeometry>
                    <gml:Solid gml:id="CG-CS_1111_19">
                        <gml:exterior>
                            <gml:Shell>
                                <gml:surfaceMember>
                                    <gml:Polygon>
                                        <gml:[redacted]
                                            <gml:LinearRing>
                                                <gml:pos srsDim[redacted]on="3">7.650000 4.250000 0.000000</[redacted]
                                                <gml:pos srsDimension="3">11[redacted]
                                                <gml:pos srsDimension="3">[redacted]
                                                <gml:pos srsDimension="3">7.650000 4.25[redacted]0.000000</gml:pos>
                                            </gml:LinearRing>
                                        </gml:exterior>
                                    </gml:Polygon>
                                </gml:[redacted]
                                <gml:surfaceMember>
```

[...]

[...]

```
            <core:Edge gml:id="DSL_1111_TCS_891_17-CS_891_5">
                <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>
```

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.

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:

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

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)

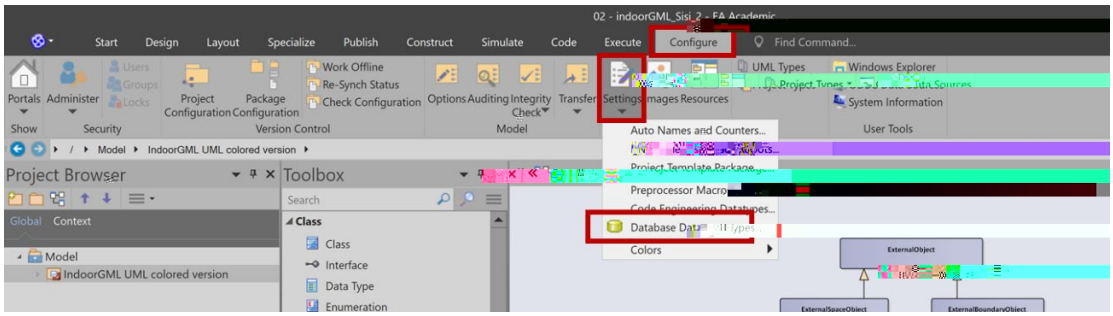

Figure 14: The interface of Enterprise Architect for mapping UML schema to SQL

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

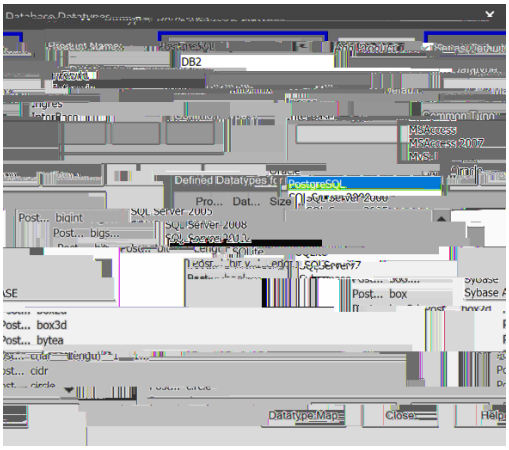

Figure 15: Set configurations of Database Datatypes

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

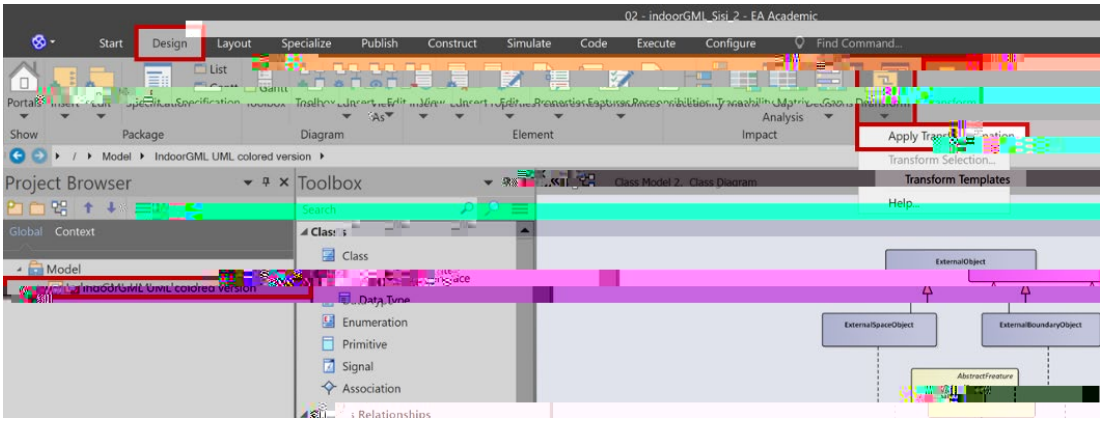

Figure 16: Select Design, Transform, and Apply Transformation

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

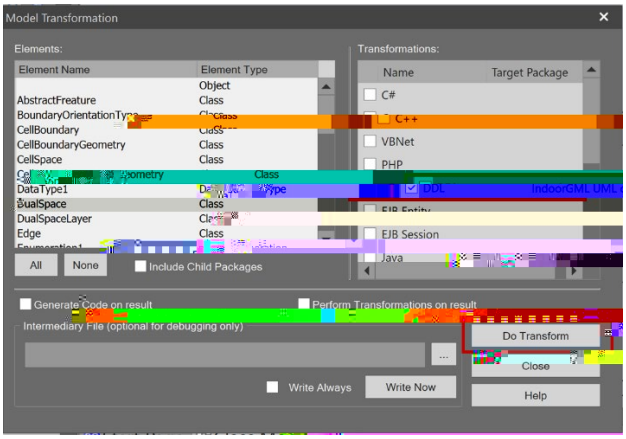

Figure 17: Do Transform

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

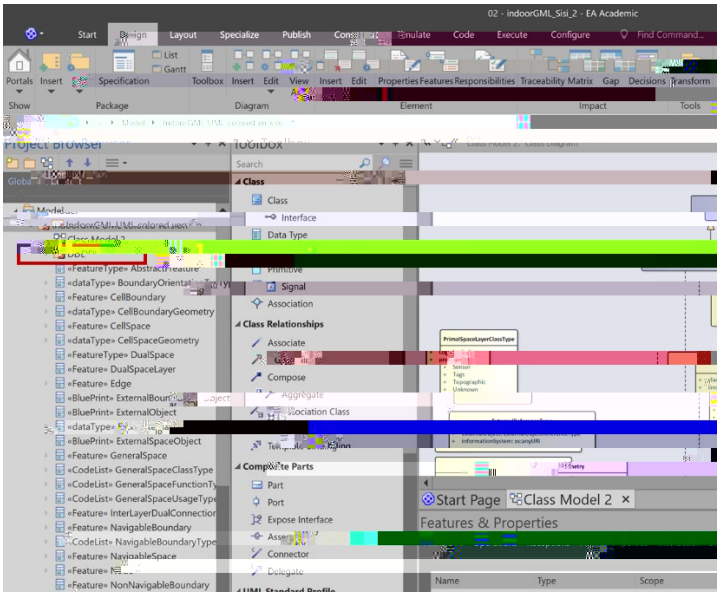

Figure 18: Select DDL folder

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