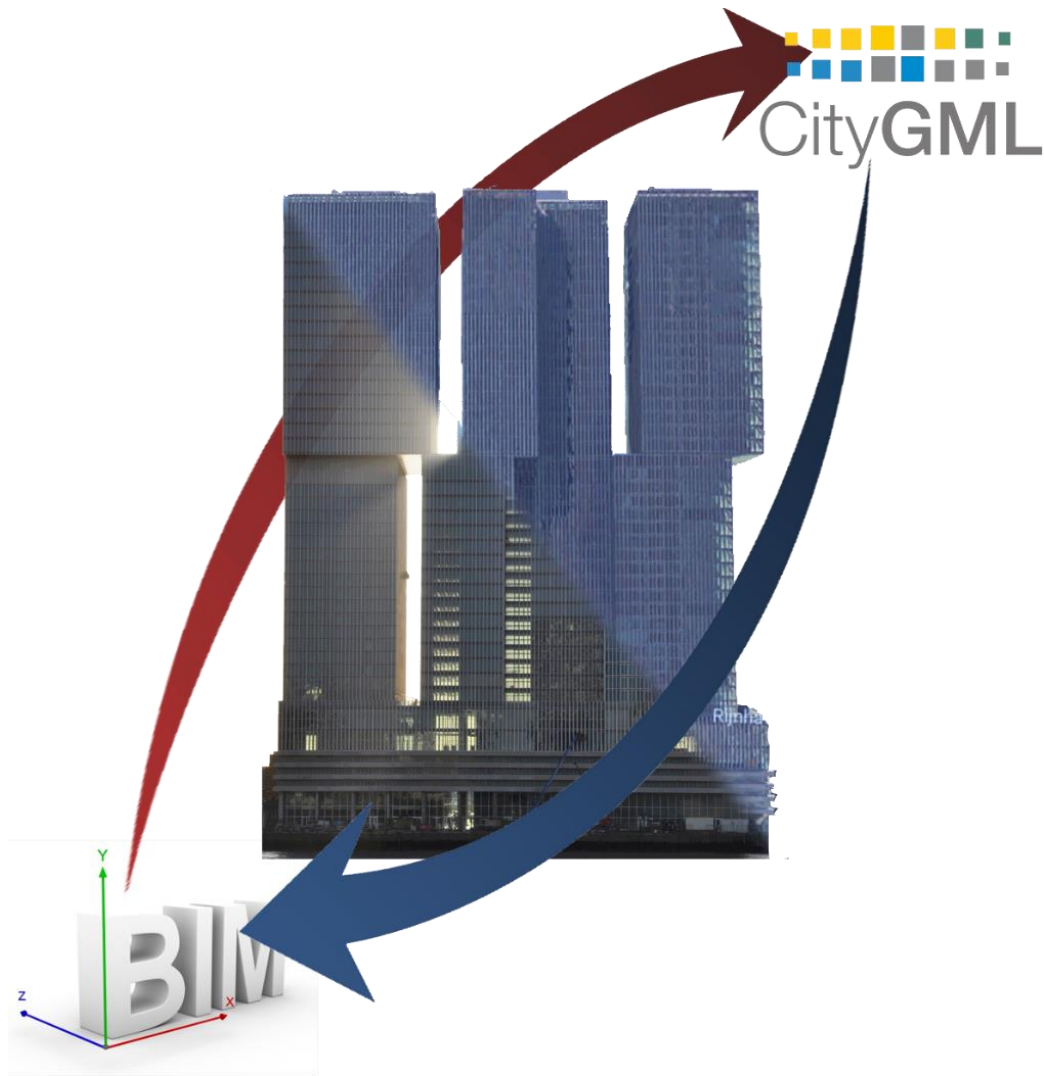


MSc thesis in Geomatics for the Built Environment

Automatic Conversion of CityGML to IFC



Nebras Salheb

2019

Automatic Conversion of CityGML to IFC

A thesis submitted to the Delft University of Technology in partial fulfillment of
the requirements for the degree of

Master of Science in Geomatics for the Built Environment

By

Nebras Salheb

October 2019

Nebras Salheb: Automatic Conversion of CityGML to IFC (2019)
This work is licensed under a Creative Commons Attribution 4.0 International License. To view a copy of this license, visit
<http://creativecommons.org/licenses/by/4.0/>.

The work in this thesis was carried out in the:



3D geoinformation group
Department of Urbanism
Faculty of the Built Environment & Architecture
Delft University of Technology

Supervisors: Prof.dr. Jantien Stoter
Dr. Ken Arroyo Ohori
Michiel Boelhouwer
Co-reader: Dr. Pirouz Nourian

ABSTRACT

This study presents a methodology to convert from the most dominant 3D city model standard in 3D GIS to BIM. Namely, CityGML to IFC. IFC is chosen because it is the common open standard to exchange data in the BIM world. For the aim of this study, the two standards are divided into 5 comparable subparts; Semantics, Geometry, Geographical coordinates, Topology, and Encoding. The characteristics of each of these subparts are studied and a theoretical conversion method is proposed for it from the first standard to the other. This is done by performing a semantic and geometrical mapping between the two standards, converting the georeferencing from Global to local, converting the encoding that the two standards use from XML to STEP, deciding which topological relations are to be retained, and providing a basic implementation that is created using Python to combine the above tasks. The work presented in this thesis can provide a foundation for future work in converting CityGML to IFC. It provides an insight into the relationship between the two standards and a methodology for the conversion from one to the other, and the process of developing software to perform such conversion. This is done in a way that can be extended for future specific needs.

ACKNOWLEDGEMENTS

Finishing my graduation thesis comes with challenges that I have overcome with the support of people around me. My deepest appreciation goes to Ken Arroyo Ohori whose continues support and advice were innumerably valuable throughout the thesis work. Special thanks also go to Jantien Stoter whose comments made an enormous contribution to my work and who provided me with big learning opportunities both inside and outside the university. I would also like to thank all the others from the 3D geoinformation group for being there whenever I need any help or advice including Kavisha Kumar, Abdoulaye Diakite, and Hugo Ledoux.

I would like to thank all the other teachers from the MSc Geomatics at the TU Delft, particularly Pirouz Nourian and Mathias Lemmens for their contribution during the time of this thesis.

I would like to thank all the colleagues from the department of “Basisinformatie” at the municipality of Rotterdam who provided me with all the resources possible to make my work there a success. I would like to particularly thank Michiel Boelhouwer who directly supervised me and generously provided me with all the connections and knowledge that I required. Furthermore, I would like to express my deepest gratitude to the following for their time and knowledge: Timo Erinkveld, Christian Veldhuis, Christian Wisse, and Alexander Boersema.

I would also like to express my gratitude to my friends in the Netherlands for their moral support and warm encouragement. And to my family who no matter how far they are, they are always in my heart.

CONTENTS

1	Introduction	14
1.1	Background	14
1.2	Motivation and problem statement.....	14
1.3	Research question.....	19
1.4	Use Case.....	Error! Bookmark not defined.
1.5	Scientific relevance	20
1.6	Scope	20
1.7	Thesis outline	21
2	Background.....	22
2.1	CityGML	22
2.1.1	Multi-scale representation.....	22
2.1.2	Semantics	24
2.1.3	Geometry.....	27
2.1.4	Coordinates	27
2.1.5	Topology	28
2.1.6	Encoding	29
2.2	IFC.....	32
2.2.1	Semantics	33
2.2.2	Geometry.....	34
2.2.3	Coordinates	34
2.2.4	Topology	36
2.2.5	Encoding	36

3	Related work.....	38
3.1	Matching IFC and CityGML schemas	38
3.2	Conversion of 3D-Geoinformation to BIM.....	38
3.3	Conversion of IFC to CityGML	40
3.4	Georeferencing BIM	41
3.5	Software	42
3.5.1	FZKViewer	42
3.5.2	CityGML2CAD	42
4	Methodology.....	43
4.1	Overview of methodology.....	43
4.1.1	Source Data.....	46
4.1.2	Filtering the Dataset.....	46
4.1.3	Creating IFC dataset	47
4.1.4	Make the data accessible for users.....	47
4.2	Encoding.....	48
4.3	Geometry	50
4.4	Coordinates.....	54
4.5	Semantics	59
4.6	Topology	65
4.7	User accessibility.....	67
4.8	Methodology results.....	70
5	implementation	73
5.1	Program Description	73
5.1.1	how to use program.....	74
5.2	Data description.....	75

5.2.1	Rotterdam3D.....	75
5.2.2	Other data sources.....	76
5.3	Tools.....	76
6	Validation	78
6.1	FZK Viewer.....	78
6.2	IfcCheckingTool.....	81
6.3	Checking other files	84
6.4	IFCRELAGGREGATES	85
6.5	Checking with users	86
7	Conclusions	88
7.1	Research questions	88
7.2	Discussion	91
7.3	Limitations	92
7.4	Future work	92
8	Appendices	95
A.	Revit error.log	95
B.	Complete Program.....	98
C.	methodology resulting data model	106
9	References	107

TABLE OF FIGURES

Figure 1- From CityGML to BIM.....	16
Figure 2- IFC can be added to the export possibilities of Rotterdam3D	17
Figure 3- Importing CityGML to a BIM software such as Revit to provide context for BIM models	18
Figure 4- Contextual design of a new building on the Wilhelminapier requires the BIM models of the immediate surroundings.....	20
Figure 5- Top-level class hierarchy of CityGML.	24
Figure 6- UML diagram of CityGML's building model	26
Figure 7- Polygon primitive example	28
Figure 8- relationship from Window towards the Room according to CityGML schema	28
Figure 9- Rectangular GroundSurface in CityGML (XML data format)	31
Figure 10-IFC example	32
Figure 11-IFC building	33
Figure 12- Separation of geometry and semantics in the IFC data model.....	34
Figure 13- IFCSite as part of the spatial structure	35
Figure 14- Two options for the relationship from IfcWindow towards the IfcSpace according to user requirements	36
Figure 15- Ground/Slab rectangular surface in IFC (STEP data format)	37
Figure 16- BIM creation processes for new and existing buildings	39
Figure 17- Workflow of the solution	39
Figure 18- FZKViewer convert bridge from CityGML (left) to IFC (right)	42
Figure 19- Overview of methodology.....	44
Figure 20- Research workflow from CityGML to IFC.....	45
Figure 21- Webtool place in the workflow	47
Figure 22-Adding IFC to Rotterdam 3D export	48
Figure 23-example of CityGML GroundSurface geometry to be converted to IFC.....	51
Figure 24- Resulting geometry model for IFCSLAB	52
Figure 25- Converting from CityGML geometry to IFC.....	53
Figure 26- Creating a reference point based on minimum values	54

Figure 27- Converting calculating local referencing to all points	55
Figure 28- Transformation file example	56
Figure 29- from CityGML to IFC geographical transformation.....	57
Figure 30- The data model of CityGML Building.....	61
Figure 31-Rotterdam3D visualization.....	62
Figure 32_Semantic mapping application	63
Figure 33- Defining the object type (IFCWall) and the namespace (bldg:WallSurface) based on the source entity in CityGML (WallSurface).....	64
Figure 34- the use of IfcRelAggregates to establish a spatial structure including site, building, building section and storey (“IfcSpatialStructureElement,” 2019).....	66
Figure 35-visualizing CityGML for users using cesium.....	67
Figure 36- Incorporating methodology within 3DCityDB	68
Figure 37-Incorporating conversion tool within the frame of FME server.	69
Figure 38- The complete methodology resulting data model	71
Figure 39- Source CityGML example dataset	72
Figure 40- Resulting IFC dataset	72
Figure 41- Snapshot of the GitHub repository.....	73
Figure 42-Rotterdam3D Building Model.....	75
Figure 43- CityGML Building (left) Composed of 2D surfaces. IFC Building (right) with enhanced geometry composed of SweptSolid	92

1 INTRODUCTION

1.1 BACKGROUND

Since 2008 there was an increase in the use of Building Information Modeling (BIM) within the construction industry. It has been increasingly adopted to improve the construction process to save both time and money, and to decrease the number of requests of information (Lee & Hollar, 2013, p. 1). Similarly, Geographic information systems (GIS) have been increasingly used to generate detailed 3D data (Ohori, Diakit , Ledoux, Stoter, & Krijnen, n.d.).

Both GIS and BIM are able to provide 3D data. However, there are major differences between the two in terms of their characteristics and focus. The main focus of the BIM field is on the information regarding building sites and its design and construction. Hence, it contains information about all the physical elements that comprise an individual building as it is designed or built. This results in that BIM is more detailed and semantically rich than GIS. On the other hand, GIS has less detailed but more updated datasets describing the environment in a wider area (Ohori, Diakit , Ledoux, Stoter, & Krijnen, 2018).

The trend of increased usage of both BIM and 3D GIS, and the similarity between the two has also led to an increase in the overlap between them. A possible application of such overlap is providing context data for BIM models through importing 3D GIS to BIM, which is the focus of this research.

1.2 MOTIVATION AND PROBLEM STATEMENT

In the process of creating a new BIM project, architects and engineers go through different steps from creating a schematic model to creating a digital representation of the building in BIM. Throughout these steps, GIS can be integrated. That means blending a layer of geospatial context into the BIM model. This is beneficial because GIS and BIM operate on different scale levels. GIS provides information at the city level. While BIM data applies to designing and building a specific shape or structure. In the BIM model the physical structure is designed at an object level, for example sketching and placing a door, window or wall. By adding GIS data, BIM model can be managed within the context of a larger and smarter landscape. For example, a building will be

connected to a parcel of land, utilities, and roads (“GIS and BIM Integration Will Transform Infrastructure Design,” 2018).

Currently, the above-mentioned GIS integration within BIM is challenging, due to the different data models used for both, hence the lack of interoperability between the two. The aim of the research is to help to bridge this gap by providing a method to convert common data models from 3D GIS to BIM.

The dominant GIS standard for storing city models is CityGML which is “*an open data model and XML-based format for the storage, analysis, representation, and exchange of semantic 3D city models. The common information model behind CityGML defines classes and relations for the most relevant topographic objects in cities and regional models with respect to their geometrical, topological, semantic and appearance properties*” (Gröger, Kolbe, Nagel, & Häfele, 2012). The CityGML information model is becoming the de-facto standard for storing and managing urban geographical information.

On the other hand, designers and engineers usually design the buildings using BIM models which are stored in different data formats one of which is IFC. An IFC dataset is an open data standard commonly used to develop and exchange BIM models. By taking the surrounding CityGML data and converting it to IFC; The resulting IFC models could give context to designers directly in their software, this would help them in multiple design-related issues such as visualizing the shadow that their new building casts on neighboring buildings, assessing quickly whether the gardens of neighbors are visible...etc. see *Figure 2* and *Figure 3*.

It is difficult for BIM users to use CityGML in their models because of the lack of support in BIM software for 3D Geo-information in general and for CityGML in particular. The aim of this research is to make Geo-information models more useful for them by developing a method for automatic conversion of CityGML dataset to IFC.

This conversion can also encourage the production of BIM models that are possible to be prepared and later be exported to CityGML by being properly Georeferenced and following a certain standard.

Another application for converting CityGML to IFC is creating complete BIM models for buildings that require one, this can be done by applying this conversion to CityGML with a high level of detail such as LOD3 or LOD4, as shown in the figure below:

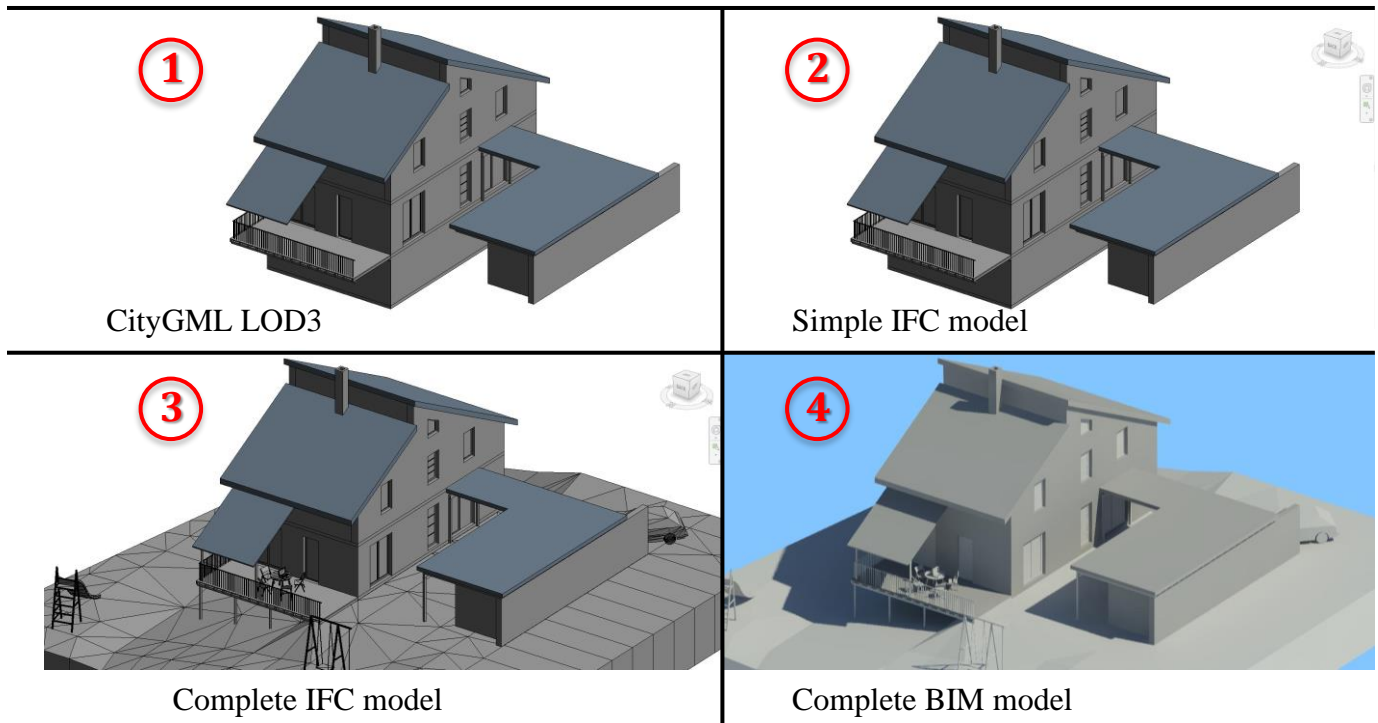


Figure 1- From CityGML to BIM

In the above figure, a building modeled in CityGML LOD3 is converted to a simple IFC model. and then the IFC model is improved by adding the necessary elements and attributes to create a BIM model.

Another possible application of the CityGML to IFC conversion methodology is the use of the resulting models to create a simplified BIM model of buildings and then develop the models using BIM software. This can be useful to create thematic representations of buildings to show the location of users for example.

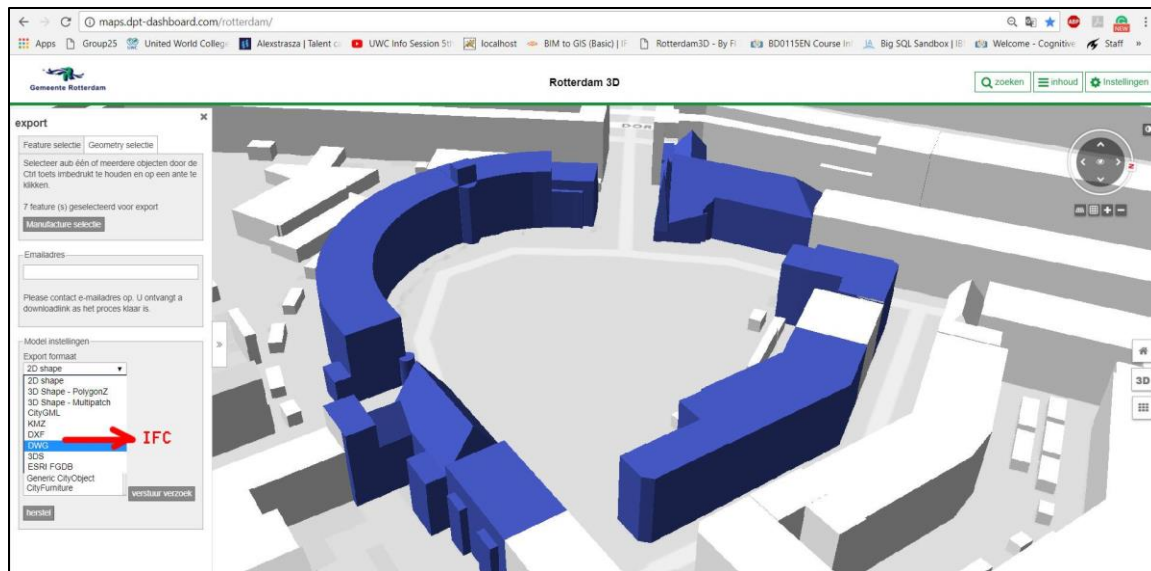
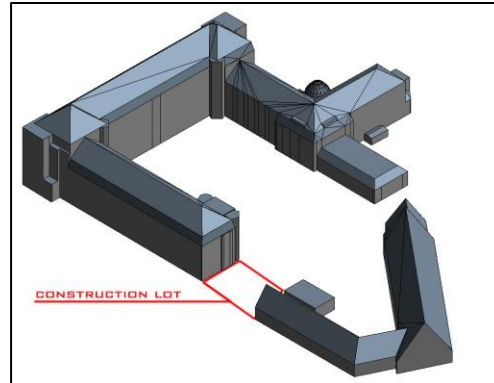
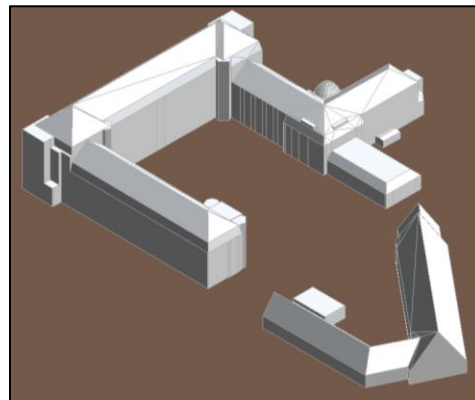


Figure 2- IFC can be added to the export possibilities of Rotterdam3D

- 1- For a construction lot, The surrounding buildings are defined in the CityGML dataset.



- 2- The surrounding buildings are then exported from CityGML to IFC.



- 3- Lastly, the surrounding buildings are imported to the BIM software of choice and the new building is then developed within its context.

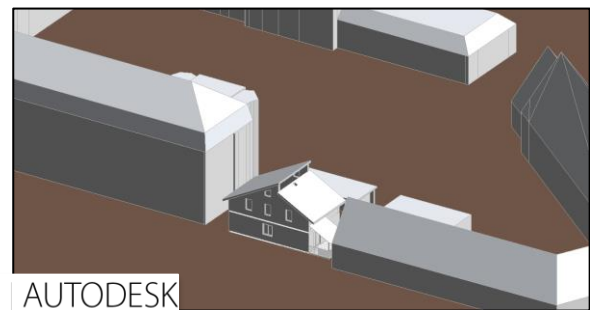


Figure 3- Importing CityGML to a BIM software such as Revit to provide context for BIM models

1.3 RESEARCH QUESTION

HOW TO MAKE 3D CITYMODELS ACCESSIBLE IN DESIGN & CONSTRUCTION SOFTWARE?

The aim is to propose a method to convert CityGML data to IFC. To apply this transformation, the following sub-questions should be answered:

- *What are the requirements for CityGML to IFC conversion?*
- *How to map semantics from CityGML to their equivalents in IFC?*
- *How to generate IFC Geometry Resource from CityGML?*
- *How to spatially reference the resulting IFC models?*
- *What kind of topology should be retained in the resulting IFC model?*
- *How to make the conversion methodology for CityGML to IFC accessible for users?*

1.4 USE CASE

To clarify the practical implementation of the project, the following use case is introduced:

An architectural firm is designing a new tower on the south side of “Wilhelminapier” in Rotterdam. In order to make the tower more coherent with the surrounding, they need to incorporate the surrounding buildings as 3D models in their BIM model during the design phase. The aim of this research is to facilitate this procedure by allowing the architects the company to download the surrounding area of the project from the city’s 3D model in IFC format *Figure 4*.

For this conversion to be successful, several metrics are set for the resulting model to fulfill. The amount of these metrics that the resulting BIM can grasp will be considered as an indication of the performance and completeness of the resulting BIM model. BIM Performance generally refers to many metrics that cover the abilities and deliverables of a BIM model. In this research, the following metrics are set based on different conversations with BIM users and creatures and bounded by the scope of research; The ability of the model to visualize reality, simulate shadows, detect clashes between elements, heat simulation, and others.

to fulfill these requirements, the model should include specific characteristics in terms of geometry, topology, and semantics. It is also worth noting that the above-mentioned use case is

merely an example, and the methodology will be tested on multiple other use cases and different city objects (other than buildings).

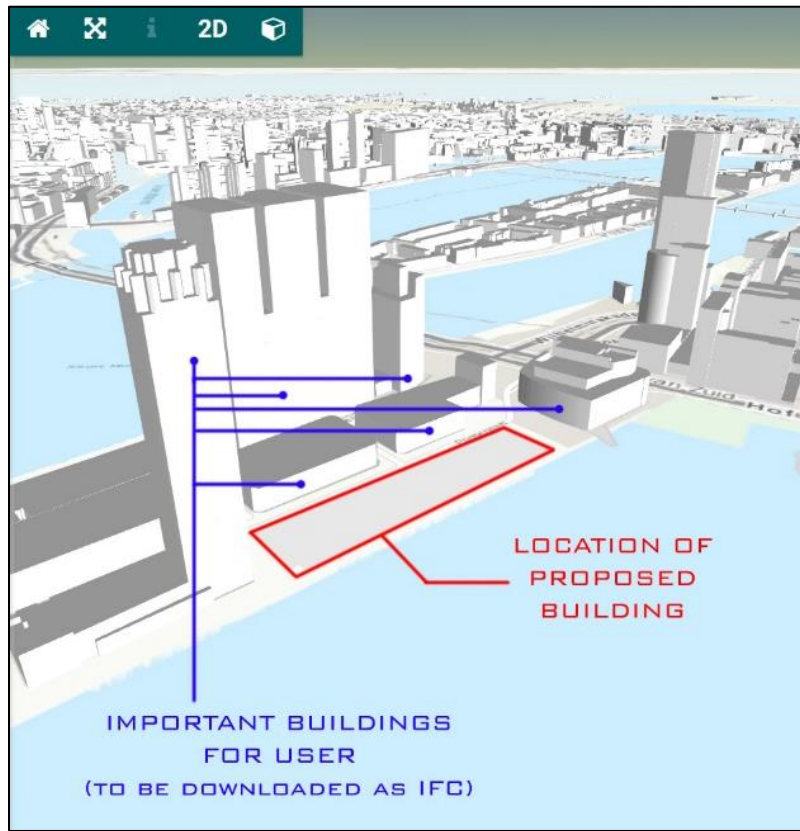


Figure 4- Contextual design of a new building on the Wilhelminapier requires the BIM models of the immediate surroundings

1.5 SCIENTIFIC RELEVANCE

This research aims at bridging the gap between 3D GIS and BIM. This field is quite extensively studied in terms of converting BIM to 3D GIS, but the research is limited in the opposite conversion (from 3D GIS to BIM). Moreover, no research is found dealing with this issue on the data level which is focused here in this research.

1.6 SCOPE

The research focuses on converting CityGML to IFC datasets since these are the most relevant data formats for the aim of the research. Other geographic features that are presented in GML but not in CityGML will not be focused on, such as noise and pollution models.

Moreover, Other BIM data formats such as .RVT and .NWD are not included in the research because of their proprietary nature and less relevance for the aim the research. Other IFC formats such as .ifcXML and .ifcZIP will not focused on since .ifc is the common data exchange format.

This research aims at producing semantically accurate IFC buildings from CityGML, and making sure that these models can be imported in BIM software. Other CityGML features are converted into semantically accurate IFC objects when possible, otherwise, it can be converted to generic IFC objects within the scope of this research.

Developing a web-based tool to disseminate the resulting IFC dataset for users is important to practically give access for the user to the developed methodology. Therefore, it is studied but it is not the focus of the research.

1.7 THESIS OUTLINE

Chapter 1 contains the introduction of the research where the problem statement and the problem background are introduced and the scientific relevance and the scope. Chapter 2 contains background information regarding the methodology including a description of both CityGML and IFC and an overview of the related work. Chapter 3 covers related work. Chapter 4 covers the conversion methodology for semantics, geometry, data, and georeferencing. The implementation of the methodology is covered in Chapter 5 including checking the validity of the results and making the data accessible for users. Chapter 6 includes the conclusions, recommendations and future work derived from this research.

2 BACKGROUND

To propose a method for converting CityGML data to IFC, a proper understanding of the different aspects of both standards is needed. In this chapter, these different aspects are presented and compared to provide a good basis for developing a methodology.

2.1 CITYGML

CityGML (Consortium., 2012.) is the most common standard to store and exchange 3D city models that includes semantics in the 3D GIS domain. It has a structured way to describe the geometry and semantics of topographic features such as buildings and roads (Arroyo Ogori, Diakit , Krijnen, Ledoux, & Stoter, 2017). The representation of spatial objects in CityGML is based on the GML3 (Geography Markup Language 3) schema.

In CityGML objects can be represented in five different levels of detail. These objects represent four different aspects of virtual 3D city models; semantics, geometry, topology, and appearance. These aspects will be discussed in this subchapter with a particular focus on building features since it is the focus of the research.

2.1.1 Multi-scale representation

There are five levels of detail (LOD) in CityGML, which can help both the provider and the customer to indicate the quality class of a CityGML dataset and makes the datasets comparable in terms of granularity, complexity, and accuracy (“Kolbe - 2009 - Representing and Exchanging 3D City Models with Ci.pdf,” p. 18).

Objects become more detailed with the increased LOD and it differs regarding its geometry and thematic representation, and objects can be represented in different LODs simultaneously. These LODs derive from data collection processes and facilitate efficient visualization and data analysis.

LODs of buildings feature are represented in the following table:

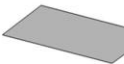
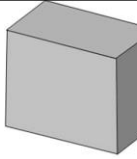
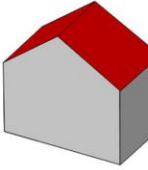
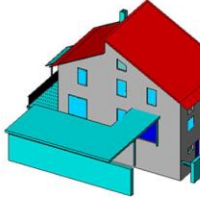
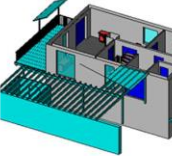
Building Feature	Geometry	LOD-0	LOD-1	LOD-2	LOD-3	LOD-4
					 LoD3	 LoD4
Building Footprint	<code>gml:MultiSurface</code>	✓	✓	✓	✓	✓
Building Volume	<code>gml:MultiSurface</code>		✓	✓	✓	✓
Facades of Building	<code>gml:MultiSurface</code>		✓	✓	✓	✓
Outer Building Installation	<code>ImplicitGeometry</code>			✓	✓	✓
Openings (Door and Window)	<code>gml:MultiSurface</code>				✓	✓
Rooms	<code>gml:Solid</code> or <code>gml:MultiSurface</code>					✓
Interior Building Installations	<code>ImplicitGeometry</code>					✓

Table 1-CityGML Building Features and associated geometries (Gröger & Plümer, 2012),(Kavisha, 2015), own work.

2.1.2 Semantics

In CityGML features are an abstraction of real-world objects and modeled by classes that are specified using UML notation. Geographic features may have an arbitrary number of spatial and non-spatial attributes. Object-oriented modeling can be applied in order to create a specialization and aggregation hierarchies. As shown in Figure 5- Top-level class hierarchy of CityGML (Kolbe—2009—*Representing and Exchanging 3D City Models with Ci.pdf*, n.d.).

CityGML provides class definitions, normative regulations, and explanations of the semantics for the most important geographic features within virtual 3D city models including buildings, DTMs, water bodies, vegetation, and city furniture. (Kolbe—2009—*Representing and Exchanging 3D City Models with Ci.pdf*, n.d.)

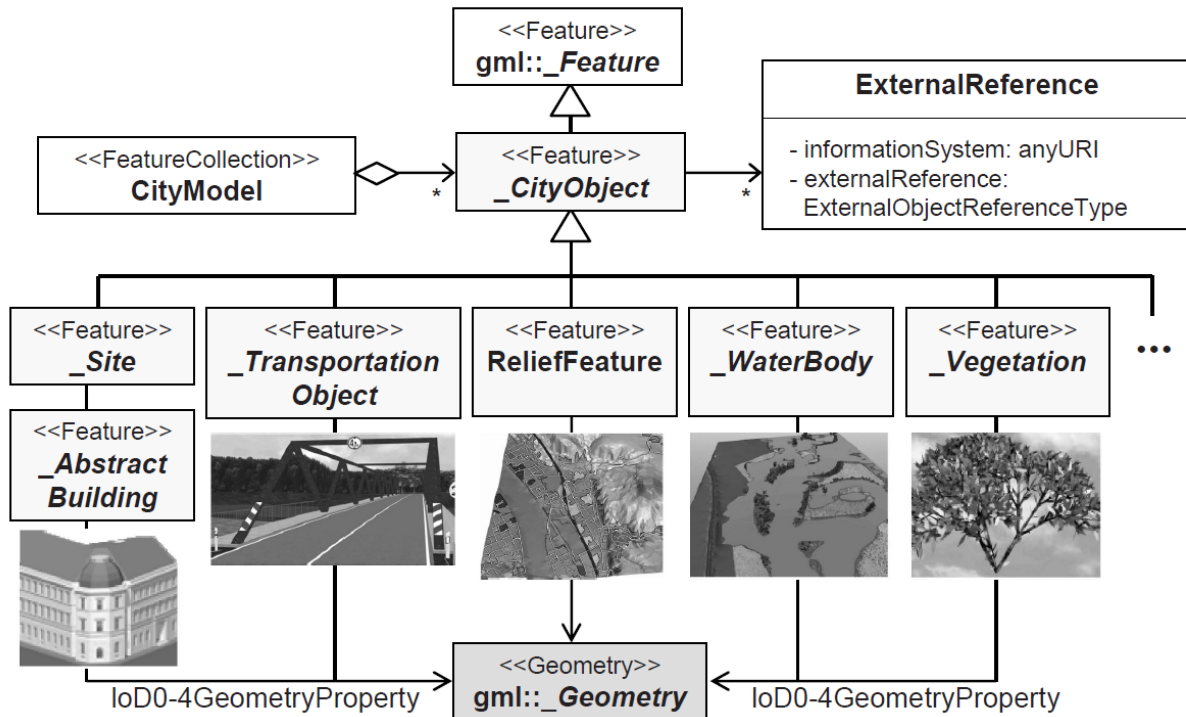
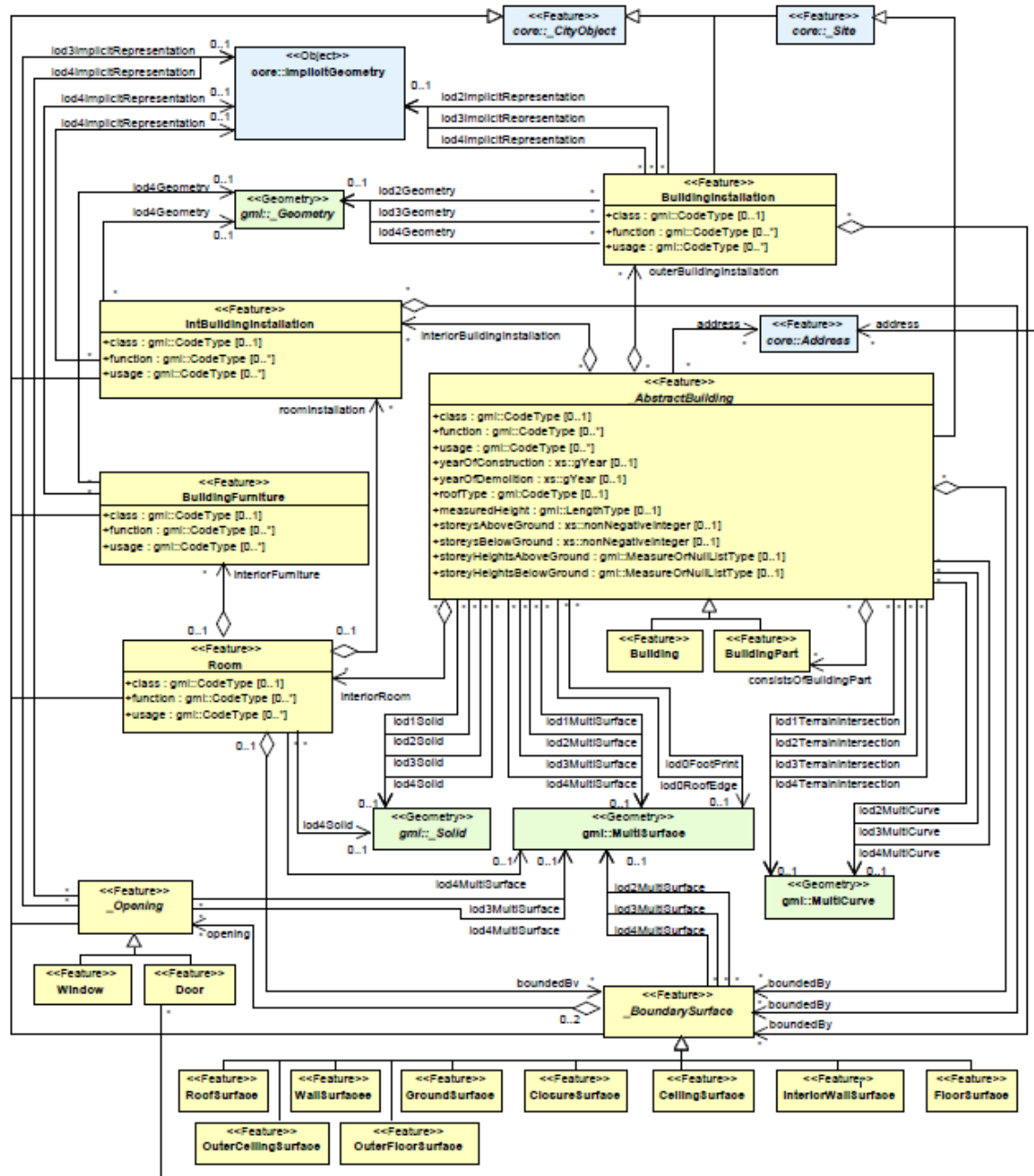


Figure 5- Top-level class hierarchy of CityGML (Kolbe—2009—*Representing and Exchanging 3D City Models with Ci.pdf*, n.d.).

Looking at the data model of buildings. Figure 6. We find that a *Building* or a *BuildingPart* is described by optional attributes inherited from *AbstractBuilding*.

Starting from LOD2 the boundary surfaces of *Buildings* and *BuildingParts* can be represented as semantic objects. *BoundarySurface* is the abstract superclass of these thematic objects that include as subclasses; *RoofSurface*, *WallSurface*, *GroundSurface*, *ClosureSurface*, *CellingSurface*,

FloorSurface ... etc. They also derive from the class *CityObject* to inherit its attributes and relations.



2.1.3 Geometry

CityGML is a subset of the GML3 model where geometries of geographic features are represented as objects that have an identity and further geometric substructures. These objects have a geometry that is described using GML.

Composite geometries like *CompositeSurface* must be topologically connected and isomorphic to a primitive of the same dimension (e.g. *Surface*)

Aggregate geometries like *MultiSurface* or *Multi-Solid* do not impose topological constraints and thus their components can also permeate each other or be disjoint.

Volumetric geometries are modeled according to Boundary Representation where each solid is bounded by a closed surface (typically *CompositeSurface*).

ImplicitGeometry which is an extension of GML3 that refer to a shape geometry in a local coordinate system that can be reused. For example, it can be used for tree and city furniture.

Curved Geometries are not supported; hence it needs to be approximated.

Buildings and building objects' geometrical representations are defined implicitly. Which means that an object is defined by attributes that define its sub-elements which are then combined to form the complete object.

2.1.4 Coordinates

In CityGML all coordinates belong to a world coordinates reference system (CRS) and local transformations are not allowed. In addition to 3D CRS, GML support 2D+1D CRS which means that x and y coordinates will have a different reference system than the vertical coordinates z.

The advantage of having only absolute coordinates is that each geometry belongs to exactly one fixed place in space which helps in creating and maintaining spatial indexes in geodatabases or geoinformation systems. However, one big disadvantage is that shape definition cannot be reused. Here *ImplicitGeometry* can be used. *ImplicitGeometry* has, in addition to a local coordinate system, a transformation matrix and an anchor point in a world CRS.

Because of the explicit association of each geometry with CRS; Integrating different CityGML datasets can be done by applying the respective geodetic datum and coordinates transformation.

2.1.5 Topology

The topology model of GML3 follows the line of full decomposition of n-dimensional topological primitives into (n-1)-dimensional primitives, which again are decomposed down to the level of nodes (0D). Each primitive becomes an object with ID; Figure 7- Polygon primitive example(Kolbe, 2009). (Kolbe, 2009)

```
<gml:Polygon gml:id="ID_b1b63cd4-2a3e-46fa-bed4-3b4fb27a6c10">
  <gml:exterior>
    <gml:LinearRing>
      <gml:posList>93980.525 432736.058 -1.906429999999905
63982.817 432733.142 -1.906429999999905 63978.625
432729.81 -1.906429999999905 63976.321 432732.709 -
1.906429999999905 63979.07 432734.899 -
1.906429999999905 63980.525 432736.058 -
1.906429999999905</gml:posList>
    </gml:LinearRing>
  </gml:exterior>
</gml:Polygon>
```

Figure 7- Polygon primitive example

In CityGML the schema of the relationship between elements is well defined, for example the relation between a window and a room is defined in the schema as shown in the following Figure 8- relationship from Window towards the Room according to CityGML schema Figure 8:

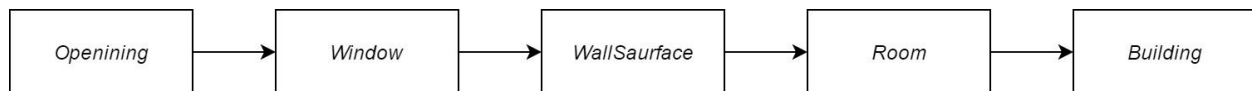


Figure 8- relationship from Window towards the Room according to CityGML schema

2.1.6 Encoding

CityGML is an application schema for GML3 that is based on XML. It provides a common definition of basic entities, attributes and relations of a 3D city model. Being based on XML; CityGML has a tree representation of this data and it usually preferred on the server-side, this tree will create a hierarchal structure that reaches down to individual features and attributes. A typical CityGML project will consist of a collection of XML files. Where each will represent a part of the dataset. besides, some image files can also be included. These files represent the textures and it is pointed at in the XML files. The CityGML datasets can be divided into different areas or LODs or based on object types. (Gröger et al., 2012). See examples in Figure 7 and Figure 9.

```

<?xml version="1.0" encoding="UTF-8"?>
<core:CityModel
  xmlns:bldg="http://www.opengis.net/citygml/building/1.0"
  xmlns:luse="http://www.opengis.net/citygml/landuse/1.0"
  xmlns:app="http://www.opengis.net/citygml/appearance/1.0"
  xmlns:xAL="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0"
  xmlns:base="http://www.citygml.org/citygml/profiles/base/1.0"
  xmlns:frn="http://www.opengis.net/citygml/cityfurniture/1.0"
  xmlns:smil20lang="http://www.w3.org/2001/SMIL20/Language"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:tex="http://www.opengis.net/citygml/texturedsurface/1.0"
  xmlns:wtr="http://www.opengis.net/citygml/waterbody/1.0"
  xmlns:smil20="http://www.w3.org/2001/SMIL20/"
  xmlns:grp="http://www.opengis.net/citygml/cityobjectgroup/1.0"
  xmlns:core="http://www.opengis.net/citygml/1.0"
  xmlns:veg="http://www.opengis.net/citygml/vegetation/1.0"
  xmlns:sch="http://www.ascc.net/xml/schematron"
  xmlns:gen="http://www.opengis.net/citygml/generics/1.0"
  xmlns:dem="http://www.opengis.net/citygml/relief/1.0"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:tran="http://www.opengis.net/citygml/transportation/1.0">
  <gml:boundedBy>
    <gml:Envelope srsName="EPSG:28992" srsDimension="3">
      <gml:lowerCorner>93976.321 432729.81 -
1.906429999999905</gml:lowerCorner>
      <gml:upperCorner>93982.817 432736.058 -
1.906429999999905</gml:upperCorner>
    </gml:Envelope>
  </gml:boundedBy>
  <core:cityObjectMember>
    <bldg:GroundSurface>
      <bldg:lod2MultiSurface>
        <gml:MultiSurface srsName="EPSG:28992" srsDimension="3">
          <gml:surfaceMember>
            <gml:Polygon gml:id="ID_b1b63cd4-2a3e-46fa-
bed4-3b4fb27a6c10">
              <gml:exterior>
                <gml:LinearRing>
                  <gml:posList>93980.525
432736.058 -1.906429999999905 93982.817 432733.142 -1.906429999999905 93978.625
432729.81 -1.906429999999905 93976.321 432732.709 -1.906429999999905 93979.07
432734.899 -1.906429999999905 93980.525 432736.058 -1.906429999999905</gml:posList>
                </gml:LinearRing>
              </gml:exterior>
            </gml:Polygon>

```

```
        </gml:surfaceMember>
      </gml:MultiSurface>
    </bldg:lod2MultiSurface>
  </bldg:GroundSurface>
</core:cityObjectMember>
</core:CityModel>
```

Figure 9- Rectangular GroundSurface in CityGML (XML data format)

2.2 IFC

BIM has emerged as one of the key streams in construction and civil engineering research within the last decade (Yalcinka & Singh, 2015). It means the semantic modelling of objects and processes in the field of AEC (Architecture/Engineering/Construction), Facilities Management and CAAD (Computer-aided architectural design). Like in CityGML, thematic objects are represented with their 3D spatial properties and interrelationships.

IFC (Standard, 2005) is the typical data exchange standard for BIM data, it provides a very detailed semantic model for 3D building representations using constructive elements like beams, walls etc. IFC is developed to represent building information over the whole building and to facilitate data transfer between BIM modelling software (Eastman., Teicholz., Sacks., & Liston., 2011). IFC models contain structures that are a combination of geometric and non-geometric data as shown in Figure 10. IFC is the BIM format of choice in this research because of its non-proprietary nature and its dominance as the main exchange format between AEC software (Shen et al., 2010).

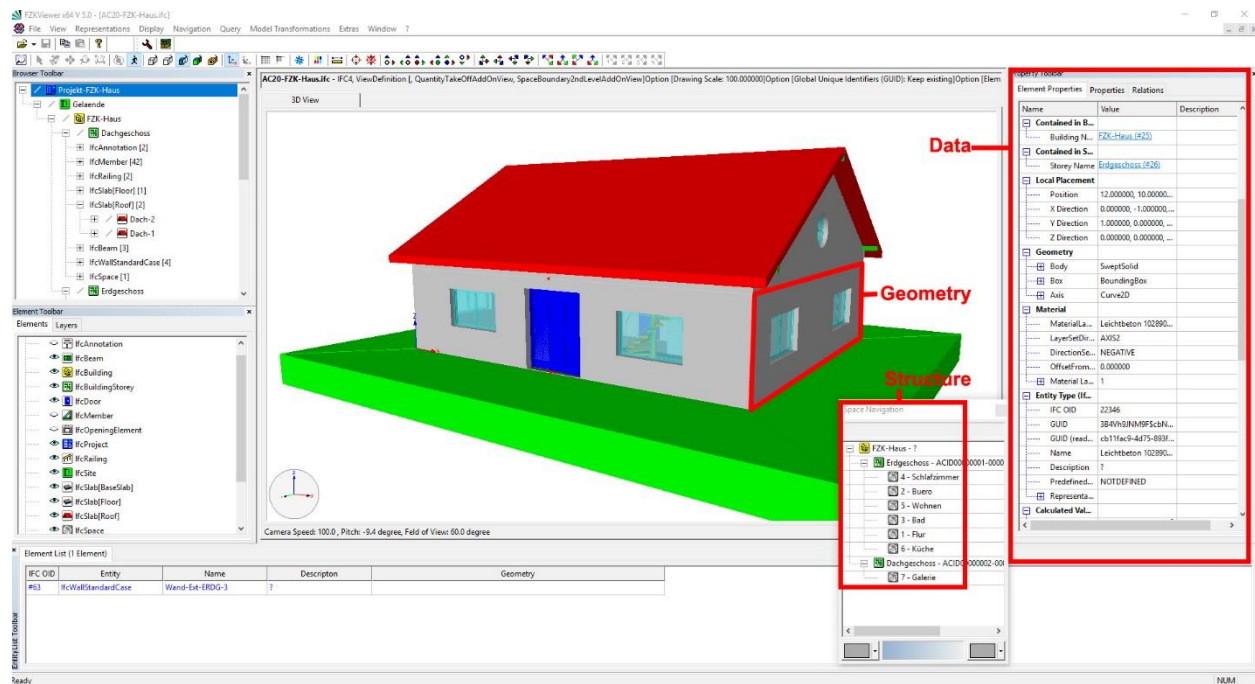


Figure 10-IFC example

2.2.1 Semantics

Since the scope of IFC is restricted to buildings and sites, no topographic feature classes like terrain, vegetation, water bodies etc. are included. IFC is a semantic model like CityGML, but with a different scope and at a different scale (El-Mekawy, Östman, & Hijazi, 2012, p. 160). And unlike CityGML there is no formal approach is adapted for multi scale representation (Borrmann, Kolbe, Donaubaue, Steuer, & Jubierre, 2013, p. 1)

An IFC building model (Figure 11) is denoted with hierarchical spatial structure beginning with the classes *IFCProject* and *IFCSite*. Class *IFCSpatialStructureElement* links the building elements and the upper arrangement of buildings (project, site, building storey, etc.). A building (*IFCBuilding*) in IFC can have at least one storey or multiple stories (*IFCStorey*). Each building storey can have zero or more spaces (*IFCSpace*) (Kavisha, 2015).

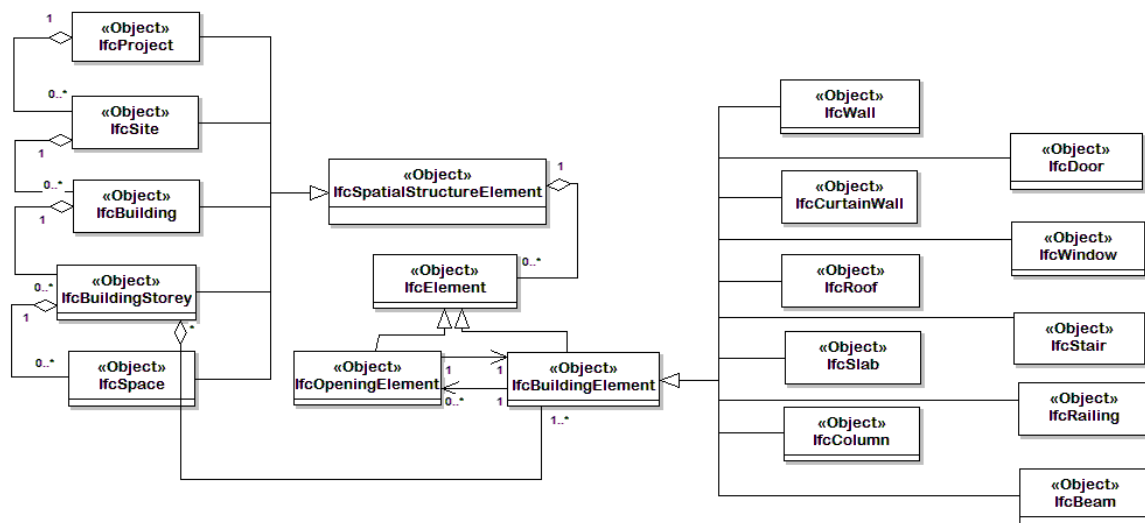


Figure 11-IFC building (El-Mekawy et al., 2012)

2.2.2 Geometry

There are distinctive geometrical models characterized in IFC for example; CSG (Constructive Solid Geometry), BRep or Sweeping. The semantic implementations of objects are unambiguously mapped in IFC, for instance, *IFCWall*, *IFCRoof*, etc. suggests its semantic implications and its clear geometric representation. Similarly, to CityGML a strict separation between geometry and semantics are implemented Figure 12. In IFC there are 653 entities, however, the majority of these classes are used to create a spatial relation between elements and their geometric representation.

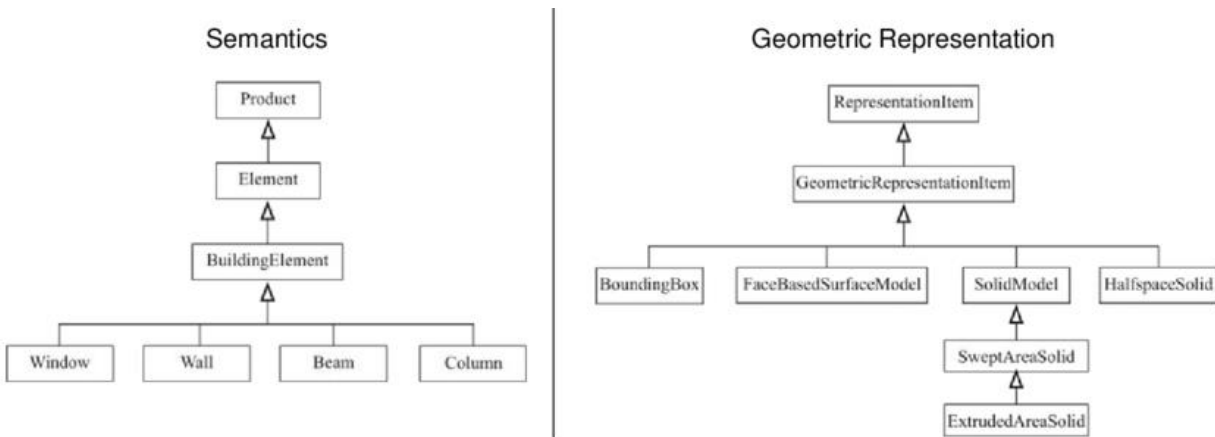


Figure 12- Separation of geometry and semantics in the IFC data model

2.2.3 Coordinates

IFC has classes that can describe the information required for georeferencing. *IFCSite* (buildingsmart, 2019a) can have the information of a geographic reference point for the project site in WGS84 with Longitude, Latitude and Elevation. If these values are given, it provides the absolute placement in relation to the real world. The geographic reference point would be the location of the point 0.,0.,0. Of the local placement of the *IFCSite*. Figure 13 shows *IfcSite* as part of the spatial structure. It shows the relation between it and to the following classes:

The IFC entity *IfcGeometricRepresentationContext* is used to define the coordinate space of an IFC model in 3D and optionally the 2D plan of such a model. This entity can be used to offset the project coordinate system from the global point of origin using the *WorldCoordinateSystem* attribute, it defines the Precision under which two given points are still assumed to be identical, and it defines the direction of the *TrueNorth* relative to the underlying coordinate system. The

latter attribute defaults to the positive direction of the y-axis of the WorldCoordinateSystem (Diakite, 2018, p. 2).

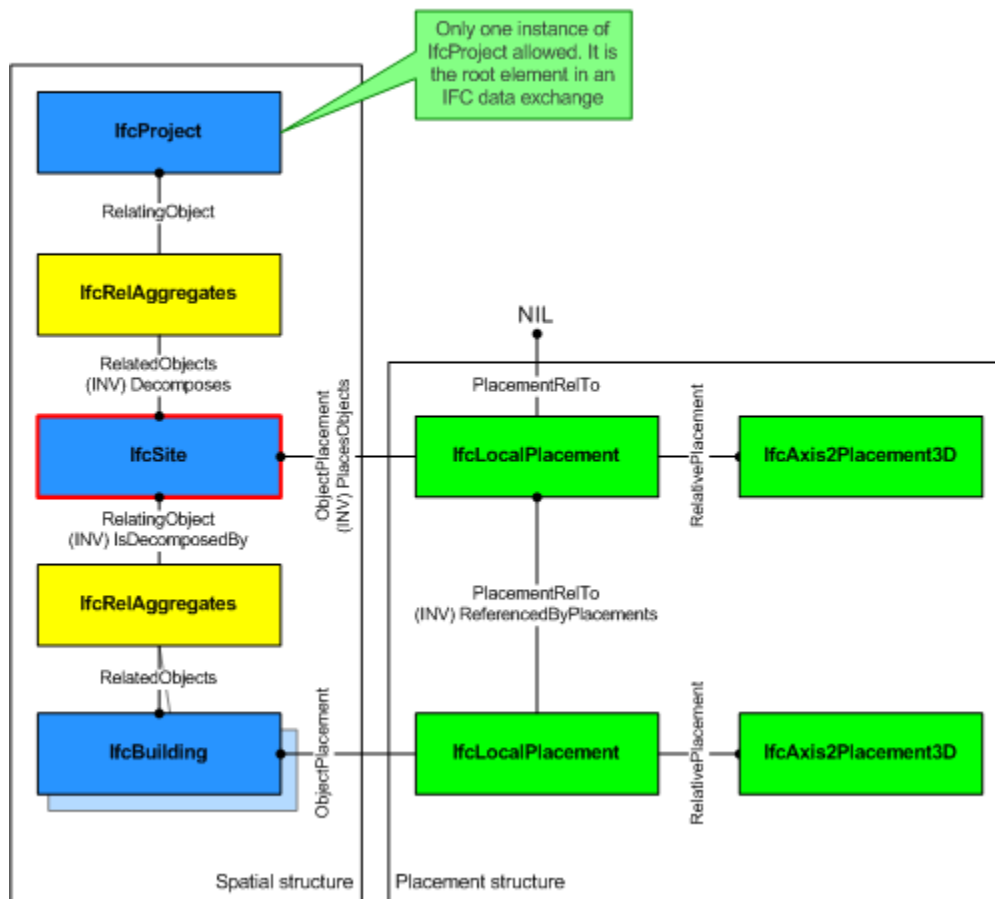


Figure 13- IFCSite as part of the spatial structure (buildingsmart, 2019a)

2.2.4 Topology

The IFC structure is designed to support dynamic modelling, that provides the users with the flexibility to represent their building data. All the objects in IFC can be created using some core elements, these core elements contain the general information. This flexibility in IFC is the reason behind the different ways in connecting two different elements in IFC. For example, in the following figures the relationship from an *IfcWindow* towards the *IfcSpace* is changing according to user requirements because this relation is not defined in the schema:

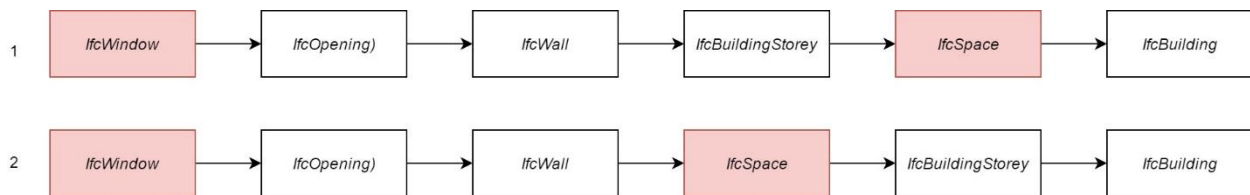


Figure 14- Two options for the relationship from *IfcWindow* towards the *IfcSpace* according to user requirements

2.2.5 Encoding

The IFC architecture has an entity relation model that is based on *Express* relation. It consists of hundreds of entities that have a hierarchy that is based on object inheritance relation (buildingsmart, 2019b).

IFC has multiple file formats; IFC-SPF, IFC-XML and IFC-ZIP. The most common file format is IFC-SPF which has a STEP encoding and have the file extension *.ifc*. *STEP* is a plain text format that is human readable and more compact compared to XML Figure 15- Ground/Slab rectangular surface in IFC (STEP data format) Figure 15.

```

ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('ViewDefinition[CoordinationView_V2.0]'), '2;1');
FILE_NAME('0ground.gml', '2018-01-05T00:07:21', (''), (''), 'IFC 2x3 writer version ', 'FZK Viewer x64',
'');
FILE_SCHEMA(('IFC2X3'));
ENDSEC;
DATA;
#101 = IFCORGANIZATION ($, 'IAI/FZK', 'Research Centre Karlsruhe', $, $);
#104 = IFCPERSON ($, 'IAI/FZK', 'IAI/FZK', $, $, $, $, $);
#103 = IFCPERSONANDORGANIZATION (#104, #101, $);
#105 = IFCAPPLICATION (#101, 'Version 4.8', 'FZKViewer', 'FZKViewer');
#102 = IFCOWNERHISTORY (#103, #105, .READWRITE., .NOCHANGE., $, $, $, 1515107241);
#109 = IFCCARTESIANPOINT ((0., 0., 0.));
#110 = IFCDIRECTION ((0., 0., 1.));
#111 = IFCDIRECTION ((1., 0., 0.));
#108 = IFCAXIS2PLACEMENT3D (#109, #110, #111);
#112 = IFCDIRECTION ((1., 0., 0.));
#107 = IFCGEOMETRICREPRESENTATIONCONTEXT ($, 'Model', 3, 1.E-005, #108, #112);
#114 = IFCSIUNIT (*, .LENGTHUNIT., $, .METRE.);
#113 = IFCUNITASSIGNMENT ((#114));
#106 = IFCPROJECT ('1FXEOs0FX9duLKeAKj5gqx', #102, 'core:CityModel', ", $, $, $, (#107), #113);
#116 = IFCAXIS2PLACEMENT3D (#117, #118, #119);
#120 = IFCLOCALPLACEMENT (#2734926363768, #116);
#128 = IFCCARTESIANPOINT ((4.204000, 6.248000, -1.906430));
z#129 = IFCCARTESIANPOINT ((6.496000, 3.332000, -1.906430));
#130 = IFCCARTESIANPOINT ((2.304000, 0., -1.906430));
#131 = IFCCARTESIANPOINT ((0., 2.899000, -1.906430));
#132 = IFCCARTESIANPOINT ((2.749000, 5.089000, -1.906430));
#127 = IFCPOLYLOOP ((#128, #129, #130, #131, #132));
#126 = IFCFACEOUTERBOUND (#127, .T.);
#125 = IFCFACE ((#126));
#124 = IFCOPENSHELL ((#125));
#123 = IFCSHELLBASEDSURFACEMODEL ((#124));
#137 = IFCCOLOURRGB ($, 0.400000, 0.400000, 0.400000);
#136 = IFCSURFACESTYLESHADING (#137);
#135 = IFCSURFACESTYLE ($, .POSITIVE., (#136));
#134 = IFCPRESENTATIONSTYLEASSIGNMENT ((#135));
#133 = IFCSTYLEDITEM (#123, (#134), $);
#122 = IFCSHAPE REPRESENTATION (#107, 'Body', 'SurfaceModel', (#123));
#121 = IFCPRODUCTDEFINITIONSHAPE ($, $, (#122));
#115 = IFCSLAB ('2qlJxYRTDD09U2BKC4zG68', #102, 'bldg:GroundSurface', ", $, #120, #121, $,
.BASESLAB.);
ENDSEC;
END-ISO-10303-21;

```

Figure 15- Ground/Slab rectangular surface in IFC (STEP data format)

3 RELATED WORK

3.1 MATCHING IFC AND CITYGML SCHEMAS

Matching IFC and CityGML schemas means investigating the following; which attributes and entities correspond to each other, the different semantic meaning of objects, and the loss of information as a result of direct transformation (El-Mekawy et al., 2012, p. 162).

El-Mekawy, Östman and Hijazi (2012) investigated the matching between IFC and CityGML schemas based on a manual and practical approach. The focus of the mapping evaluation here is divided into two parts:

1- The *product extension*: the *product extension* is the spatial structure of buildings and the relation between these structures and building elements. The research defines the product extension classes properties that can be transformed between IFC to CityGML. Which are:

IfcBuilding, IfcFurnishingElement, IfcSpace, IfcAnnotation.

2- The *shared building elements*: similarly, to the *Product Extension*, a lot of the entities in the *shared building elements* are used to define spatial characteristics and representations of building objects hence not applicable for the conversion. the research defines the entities that are applicable for conversion.

The *IfcProductExtension* and *IfcSharedBuildingElement* domains. Combined, they have 83 classes/entities in their schemas.

3.2 CONVERSION OF 3D-GEOINFORMATION TO BIM

The research in conversion from CityGML to IFC is limited because it is considered less useful than the opposite conversion (Arroyo Ohori et al., 2017, p. 9). However, there is some research done in this field but not on the data level as it is in this research. In Volk, Stengel, & Schultmann, 2014 methods for creating BIM from existing models is discussed starting from capturing the data to processing the data and object recognition to model the required BIMs see *Figure 16* These methods can be considered GIS to BIM (DIAKITÉ & STOTER, 2017).

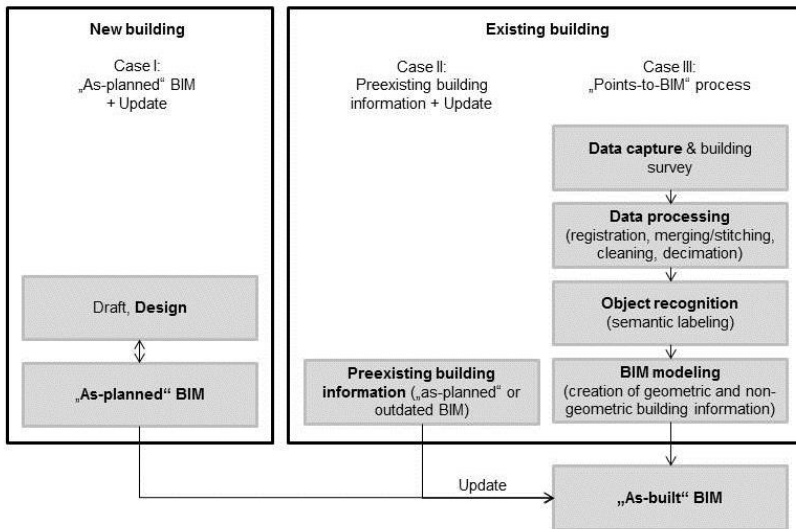


Figure 16- BIM creation processes for new and existing buildings (Volk et al., 2014)

In Diakité & Stoter, 2017 an interface to translate Geodata into IFC data is developed by applying the workflow shown in Figure 17. The method aims to represent subsurface information in IFC, in which the class IfcSpace is used.

In our research, similar method for georeferencing the IFC model can be used and IfcSpace can also be used when there is uncertainty of which IfcClass to be used since it has proven sufficient in (Diakité & Stoter, 2017).

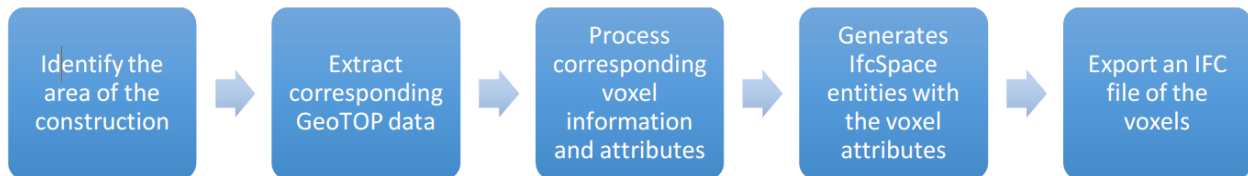


Figure 17- Workflow of the solution (Diakité & Stoter, 2017)

3.3 CONVERSION OF IFC TO CITYGML

The opposite conversion from IFC to CityGML is more extensively discussed because it is deemed more useful and it implies simplifying and removing details and unnecessary information in the data. In Arroyo Ohori et al. (2017) a methodology for this transformation is developed and it deals with the models starting from data level which is similar to this research. This methodology starts with converting the IFC data using IfcOpenShell to export a series of .obj files that represent the model. For this it is mainly looked at subentities of IfcBuildingElement. And then the individual .obj files are transformed into CGAL Nef polyhedron for every object, the resulted Nef polyhedra were each of them processed to generate one Nef polyhedron by a group of processes of Boolean operations. The resulting polyhedron contains all the relevant information of the building (Arroyo Ohori et al., 2017).

Donkers (2013) provides a methodology of generating LOD3 buildings models out of IFC. In addition, a prototype implementation is made that is able to successfully apply the conversion. The conversion process consists of three steps: the first step consists of identifying the semantics of an IFC dataset and subsequently mapping them to a CityGML semantics model. The research also deals with geometric transformations in which the exterior shell of an IFC model is extracted using transformation based on morphological operations and Boolean generating an external shell similarly to Ohori et al. (2018). Lastly the model is modified as necessary form further operations and analysis.

3.4 GEOREFERENCING BIM

Diakite, (2018) proposed an overview about the situation of georeferencing BIM with the focus on models designed in Revit and exported as IFC. Some BIM software have a methodology to georeference models, this methodology can give an insight about how to produce Georeferenced IFC from CityGML. For example, Revit provides multiple ways to feed georeferencing data to BIM models.

In Autodesk (2018) a method to link CAD files that contain real-world (GIS) coordinates to a Revit model is proposed, in order to acquire those coordinates directly in the Revit model. This is done by linking a DWG file containing geo-referenced coordinates and adjusting the location of the BIM project accordingly.

Another possible method to georeference the BIM model using Autodesk software done by generating a shared reference point for Revit using Autodesk Civil that can be imported in as an XML file to Revit Table 2 and link this point to the local coordinate system that exists in the project.

Table 2- Shared reference point XML file

```
<?xml version="1.0" encoding="utf-8"?>
<CoordSysZup Units="M">
  <OriginX>67561.327969</OriginX>
  <OriginY>444577.20961</OriginY>
  <OriginZ>0</OriginZ>
  <RotationInXYPlane>5.9628533378646953</RotationInXYPlane>
</CoordSysZup>
```

3.5 SOFTWARE

The following software are developed to deal with the issue of converting CityGML to CAD and BIM formats.

3.5.1 FZKViewer

FZKViewer is tool for the visualization of semantic data models in the fields of BIM and GIS. FZKViewer is able to export CityGML as IFC (Informatik, 2017). However, it has the following limitations; It is limited in terms of georeferencing the data, where it can read the coordinates of the data, these coordinates are lost when CityGML is converted into IFC. FZKViewer is able correctly convert buildings up to LOD2 in terms of geometry and semantics. However, information is lost when trying to convert buildings in LOD3 or LOD4 or other city objects such as bridges *Figure 18*. lastly, the proprietary nature of the software also does not allow for improvement.

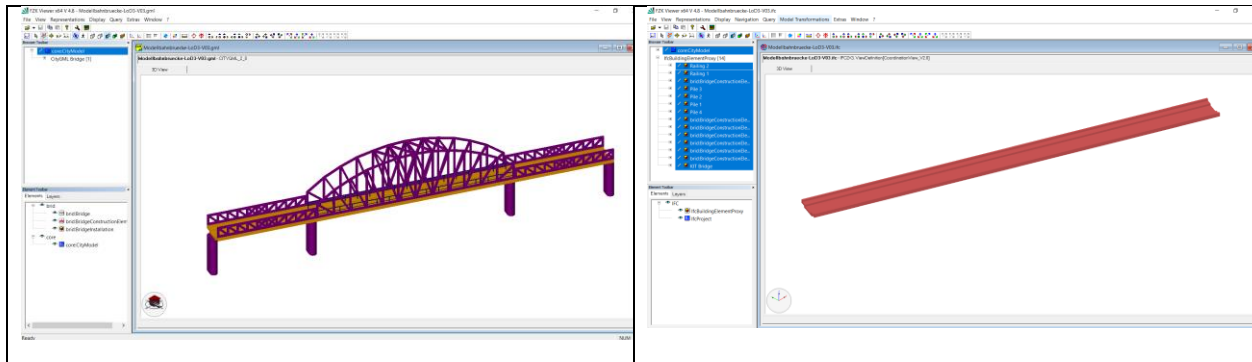


Figure 18- FZKViewer convert bridge from CityGML (left) to IFC (right)

3.5.2 CityGML2CAD

CityGML2CAD helps in bridging the gap between CityGML and CAD. It helps in converting CityGML models into various 3D formats so that the data can be used in CAD applications. It supports the following data formats; Sketchup, 3D studio, Object, Open inventor .iv, AC3D, .ive and .osg. and It is also supporting Georeferencing the resulting data. The limitations of the software are that it is of a proprietary nature and it supports up to level 1 maturity level BIM but does not extend to up to level 4 as the aim of this research.

4 METHODOLOGY

4.1 OVERVIEW OF METHODOLOGY

IFC and CityGML differs widely, hence the first step in the methodology is that the standards are divided into different components defining, these are; semantics, geometry, topology, encoding and georeferencing. Next each of these components is studied and a theoretical conversion method is proposed for it.

Next step is combining all these conversion requirements in one implementation and then applying an incremental development on this implementation starting from converting a simple dataset to a complete dataset and different datasets. Next, a collection of software is selected to check the results. By checking the resulting on this software the methodology is the improved accordingly and the implementation is debugged in an iterative process. In the end of this process the conversion implementation is finished. This implementation is then presented to different users and the ability to make it accessible to users is studied. Finally, conclusions and future research are derived. And the program is made accessible for users through GitHub.

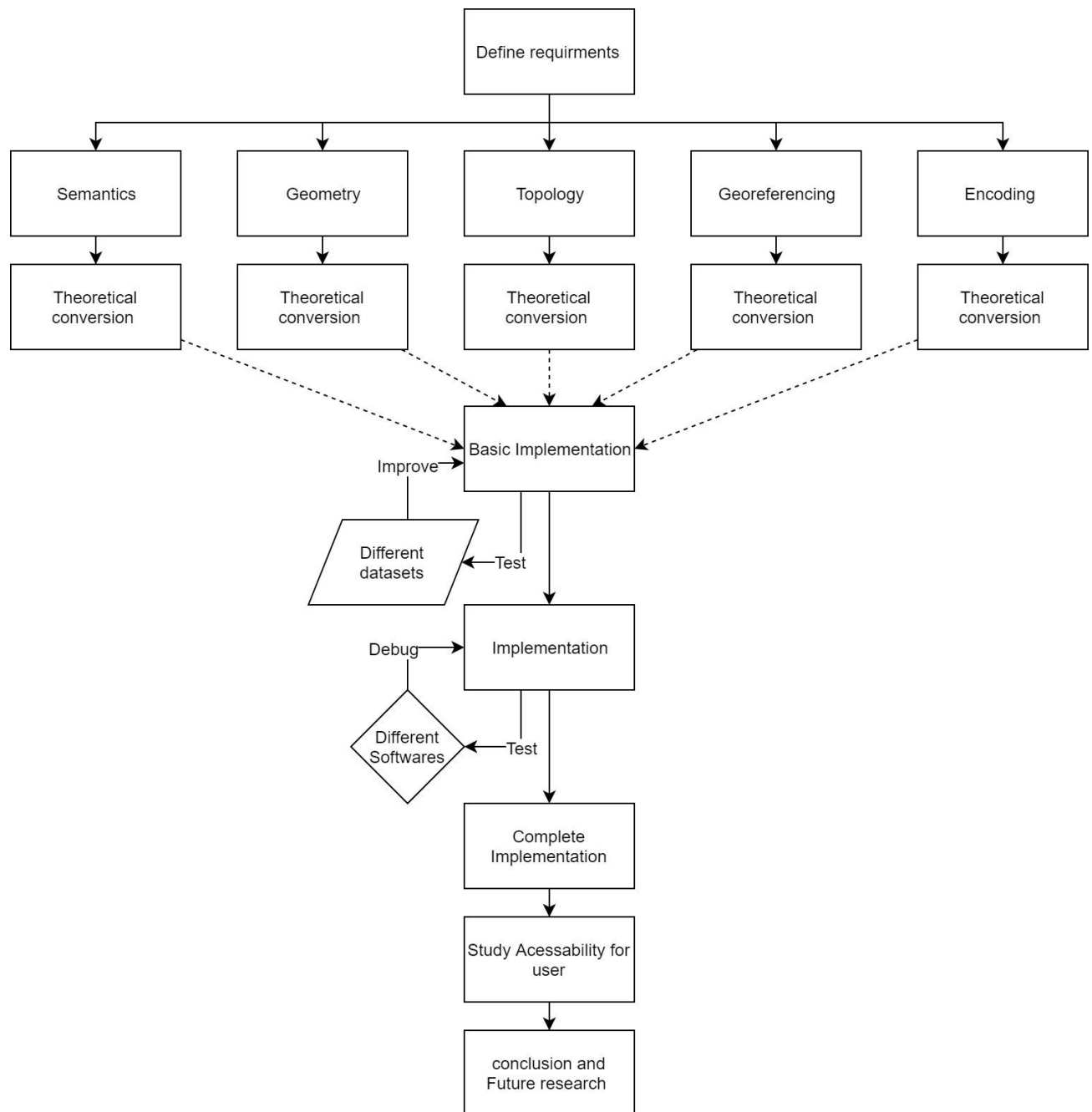


Figure 19- Overview of methodology

To be able to apply the conversion between CityGML and IFC both data models should be studied and specified. For this purpose, practical data models are used, such as Rotterdam3D and practical BIM models that were provided by municipality of Rotterdam. This helps to avoid different definitions because it is based on real world data.

To accurately convert from CityGML to IFC the data model from CityGML are redesigned to incorporate with IFC. The difference in data models between the two is quite big where CityGML is hierarchical IFC is not. This process is done by defining a semantic and geometrical mapping from the first standard to the other. And by presenting a transformation too to convert from the first encoding to the other.

Moreover, a practical implementation is presented to This transformation part of the process is implemented using Python, which enabled performing specialized operations on a feature's geometry, attributes, and coordinate system. The results are be evaluated by looking at what is possible with other software such as FME. Other tools that will be used is explained in 5.3. Hence the complete work flow will look as following Figure 20:

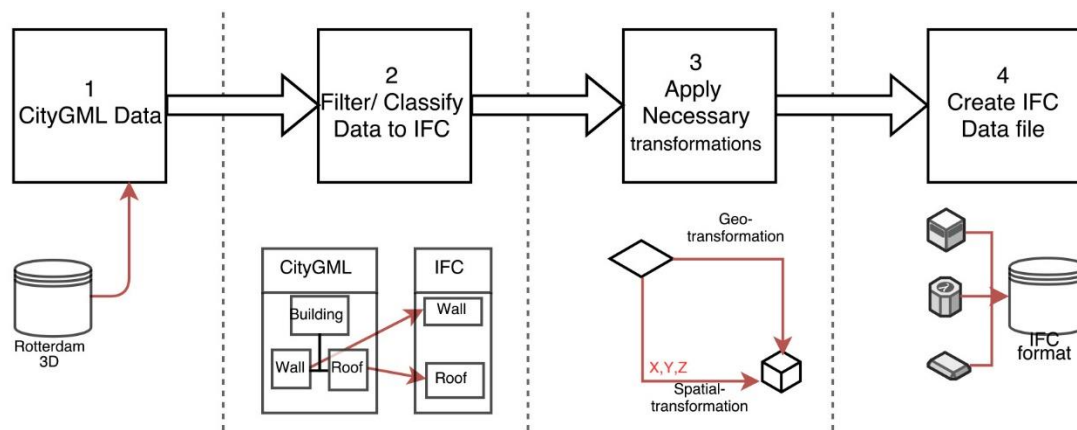


Figure 20- Research workflow from CityGML to IFC

The research is conducted in the municipality of Rotterdam. This helps in acquiring the CityGML from the newly developed Rotterdam3D 2.0 which is provided by the municipality. This model is used as an example of the source data. The municipality also gathers BIM models for the newly designed buildings. These models are used to give insight about the used and delivered IFC models. Moreover, conducting the research within the municipality gives insight about the needs

of different stakeholders and expected users of the developed methods. In addition, the resulting program can be incorporated in a Web Viewer for example in Rotterdam3D 2.0 to directly export models from the CityGML view to IFC file format.

4.1.1 Source Data

Rotterdam 3D 2.0 is the main source of CityGML dataset in this research and its focus see *Figure 29*, this data is in LOD2 and based on both BAG and Rotterdam-Height model (Hoogtebestand) it contains the following features:

- Buildings
- Terrain model
- Trees
- Street lanterns
- Underground Cables
- Other especially created city objects such as the Erasmus bridge.

4.1.2 Filtering the Dataset

BIM models are generally more rich and detailed datasets than CityGML. Hence, it is preferable to convert as much information as possible from the source CityGML dataset. However, some features could be ignored within the scope of the project. For example:

- Vegetation: because IFC2x3 does not have specific classes that is appropriate for vegetation so could be either filtered out or converted to a generic object class such as *IfcBuildingElementProxy*.
- Textures: because IFC2x3 does not support texture data types.

For the above-mentioned features that requires to be filtered, the filtration process is done automatically because of the implementation method of choice, this happens as following;

The software will look for features with a certain tag (say: building) and convert these features. If a future tag is not incorporated in the program then it will be left out of the conversion automatically (say: vegetation).

Other filtration of the data is done based on areas, feature type or featured ID. This kind of filtration is done separately using tools such as FME and other tools as mentioned in subchapter 5.3.

4.1.3 Creating IFC dataset

CityGML has an XML schema. While on the other hand IFC has a STEP format. Python will be used to read the source XML data, parse the data, apply transformation and write the STEP file (STEP, 2017) for IFC.

Developing a tool to apply such task can be quite complicated. Hence it is proposed to develop the tool incrementally, starting from converting one single element (such as wall) to a simple obj file and then creating a complete building and then including other city objects.

4.1.4 Make the data accessible for users

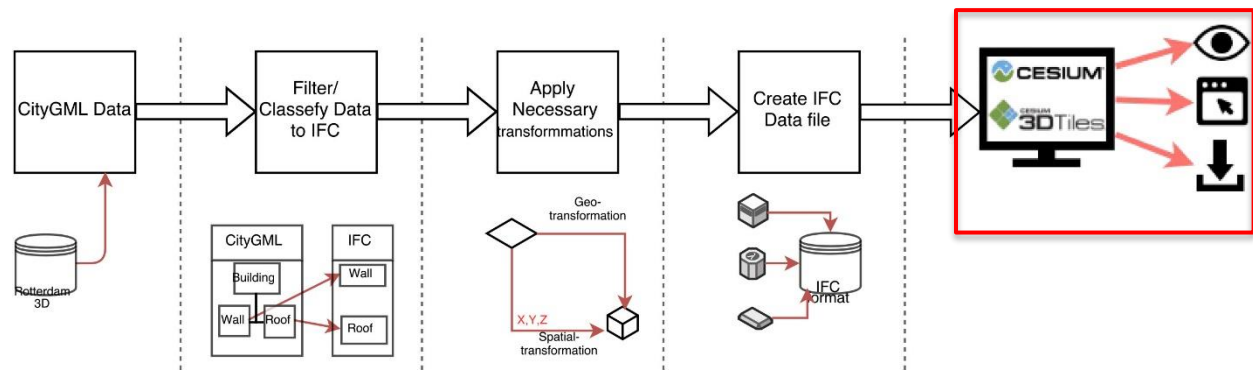


Figure 21- Webtool place in the workflow

To make the resulting data accessible for users a tool to select and download the data can be developed Figure 21. For this purpose, Cesium, which is an open-source JavaScript library for 3D maps, can be used in combination with 3D Tiles to stream the 3D geospatial datasets on the web for both CityGML and IFC.

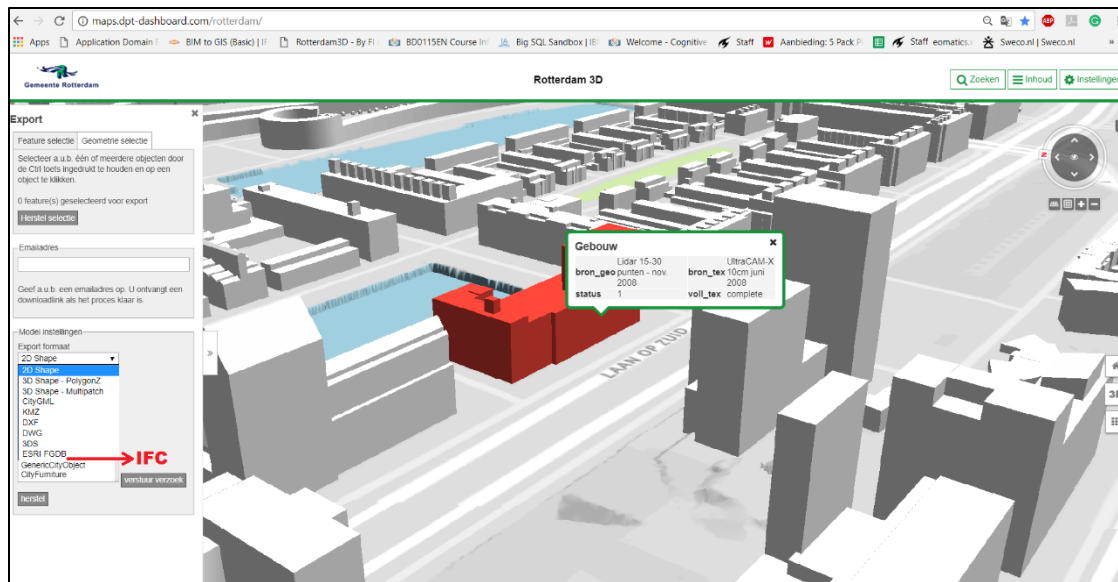


Figure 22-Adding IFC to Rotterdam 3D export

4.2 ENCODING

The first component to be studied in the conversion is Encoding. Because this is the first major obstacle that can obstruct the conversion:

A CityGML file is a hierarchy that ultimately reaches down to individual objects and their attributes. These objects have a geometry that is described using GML (Consortium., 2012.). On the other hand, IFC model contains both geometric (3D and 2D) and non-geometric data about the building project. Technically the schema defines an entity-relationship model based on “EXPRESS” which means that an element is defined in the system and both its type and instance can have attributes and properties attached to them. The properties themselves has a specific structure; they are normally grouped in property sets. IFC also defines relationships between the building elements. (areo, 2016). These difference between the two data models requires remodeling of the source elements from the hierarchal CityGML to their counterparts of the non-hierarchal structure of IFC data model.

To generate a unique id for every object a function is created *guid()* which will automatically generate a unique id for every time it is called. For example: *'dcdc161f86a246cfb37dcb'*

The function *CityGML2IFC(path,dst)* is the main function, in which all the other functions are called. Here *path* refers to the source CityGML file and *dst* is the destination IFC file.

the following steps are implemented inside *CityGML2IFC(path,dst)*;

- 1- The source CityGML file is parsed using *xml.etree.ElementTree*
- 2- The CityGML version is identified based on the root tag. The name spaces are created based on the CityGML version and then as dictionaries with keys and value.
- 3- The results from the conversion are printed in a destination file. With the format: *dst.ifc*
- 4- The file *dst.ifc* starts with the mandatory header part with the following information: File description, name and schema. In which other IFC compatibility formatting is added such as *time* and file destination. Followed by the *data* part. (“STEP-file, ISO 10303-21,” 2017). Which consists of a sequence of entities, where each line represents a different entity. The names of these entities are defined by a sequence of numbers that is generated using a counter starting from 1000. The first part of the data part is filled with the following information that should exist in every IFC project:

- *IFCORGANIZATION*
- *IFCPERSON*
- *IFCPERSONANDORGANIZATION*
- *IFCAPPLICATION*
- *IFCOWNERHISTORY*
- *IFCCARTESIANPOINT*
- *IFCDIRECTION*
- *IFCDIRECTION*
- *IFCAXIS2PLACEMENT3D*
- *IFCDIRECTION*
- *IFCGEOMETRICREPRESENTATIONCONTEXT*
- *IFCSIUNIT*
- *IFCUNITASSIGNMENT*
- *IFCPROJECTID*
- *IFCSITEID*

4.3 GEOMETRY

The CityGML geometry of buildings consists mainly of 2D surfaces. On the other hand, IFC objects are built using different geometry representations including; sweep volumes, explicit faceted surface models and CSG (Arroyo Ohori et al., 2017). To be able to convert an element from CityGML to IFC; the accurate matching geometry should be created. And since the focus of this research is on converting buildings classes the representations of IFCWall and IFCSlab is particularly interesting since it is possible to model the whole LOD2 building geometry with these two classes. In IFC 2x3, the use of 'SweptSolid' and 'Clipping' representations is supported for these classes. In addition, the general representation types 'Brep', 'SurfaceModel' and 'BoundingBox' are allowed (buildingsmart, 2017).

To explain the conversion methodology the following example will be shown Figure 23. The same process is repeated for every surface element in CityGML. The source file in CityGML is a ground surface that is defined by a linear ring that consists of 6 points:

```

<bldg:boundedBy>
  <bldg:GroundSurface>
    <bldg:lod2MultiSurface>
      <gml:MultiSurface>
        <gml:surfaceMember>
          <gml:Polygon gml:id="RCID_98b8908e-480d-4e67-8cb8-
2967c0d78523">
            <gml:exterior>
              <gml:LinearRing gml:id="RCID_98b8908e-480d-4e67-
8cb8-2967c0d78523_E_1_1">
                <gml:posList>94713.25000000000000000000
433753.299999999990000000 -1.2266500000002190000
94712.259999999995000000 433751.789999999980000000 -
1.2266500000002190000 94700.619999999995000000
433759.409999999970000000 -1.2266500000002190000
94703.25000000000000000000 433763.419999999980000000 -
1.2266500000002190000 94709.589999999997000000
433759.25000000000000000000 -1.2266500000002190000
94707.9600000000006000000 433756.7700000000020000000 -
1.2266500000002190000 94713.250000000000000000
433753.299999999990000000 -1.2266500000002190000</gml:posList>
              </gml:LinearRing>
            </gml:exterior>
          </gml:Polygon>
        </gml:surfaceMember>
      </gml:MultiSurface>
    </bldg:lod2MultiSurface>
  </bldg:GroundSurface>

```

Figure 23-example of CityGML GroundSurface geometry to be converted to IFC

The total number of values in *gml:posList* is 21 defining the coordinates of 7 points because the end of the ring is the same as the beginning.

Here, using element tree *tree.findall('./{%s}posList' % ns_gml)* is used to create list of a *IFCCartesianPoint* entities.

IFCCartesianPoint; is a point defined by three dimensional cartesian coordinates system. The list of these points forms an *IFCFaceOuterBound*. Which defines the outer boundary of the face; *IFCFace*. To define the geometric representation of the face the following entities inheritance is defined:

IFCOPENSHELL

↳ *IFCSHELLBASEDSURFACEMODEL*

↳ *IFCPRODUCTDEFINITIONSHAPE*

↳ *IFCSLAB*

The complete resulting IFC Geometry model is shown in the figures below Figure 24, Figure 25.

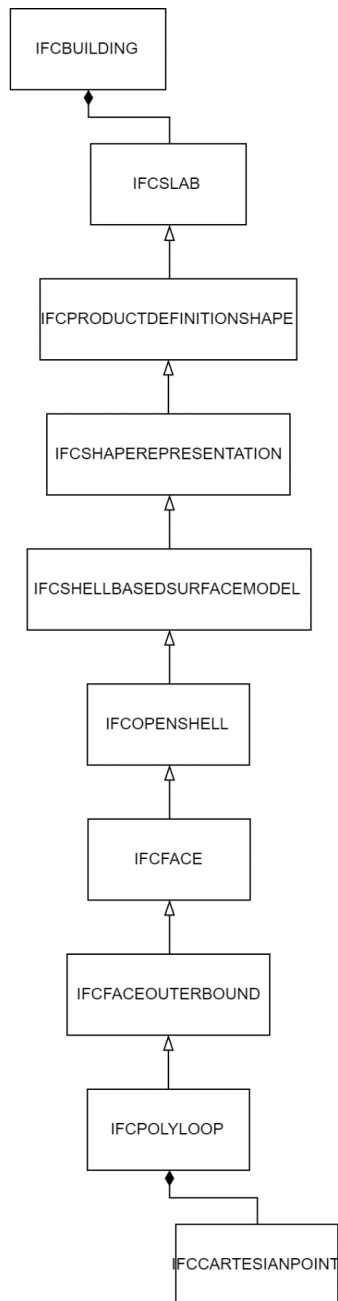


Figure 24- Resulting geometry model for IFCSLAB

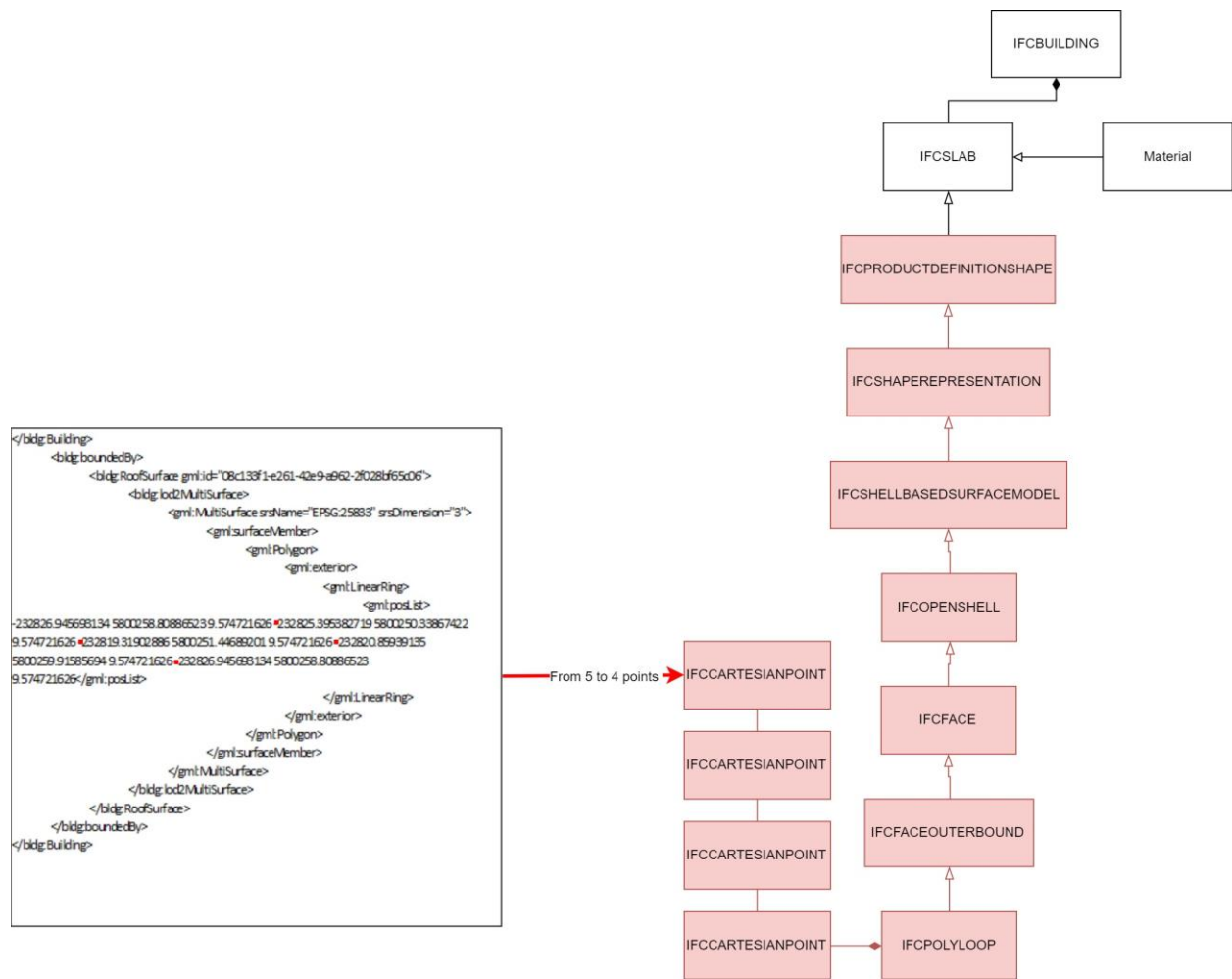


Figure 25- Converting from CityGML geometry to IFC

4.4 COORDINATES

In CityGML, Coordinates belong to a world coordinates reference system and no local transformations are allowed. On the other hand, IFC files are locally referenced. It is not very common for BIM models to be geographically referenced, which increases the importance in the research to produce a correctly georeferenced IFC file. Here a method is proposed and implemented to georeference the resulting IFC datafile out of the source CityGML files:

Firstly, a reference point for the model is created, this is done by iterating over all points in the model and selecting the minimum value for all the points. For example; in Figure 26 a reference point for a GroundSurface in CityGML is created, with an EPSG:28992 coordinate system.

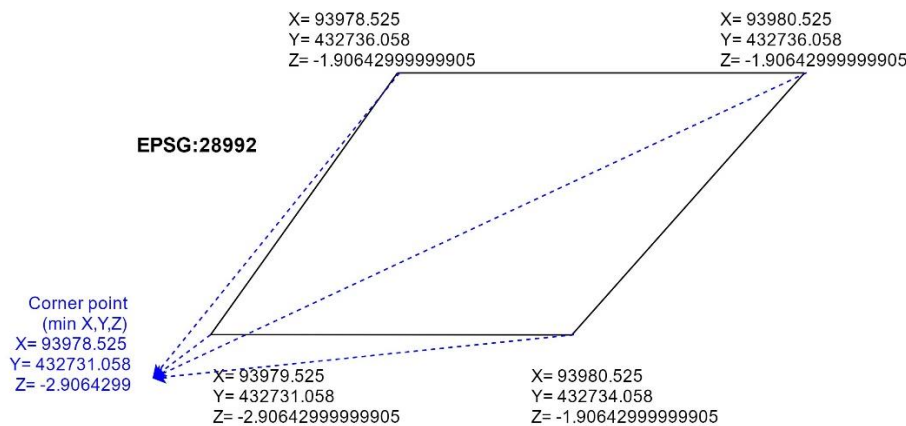


Figure 26- Creating a reference point based on minimum values

Secondly, the values of all points are converted to local referencing in relation to the reference point. The local referencing is calculated by subtracting the values of all points from the reference point value. As shown in Figure 27.

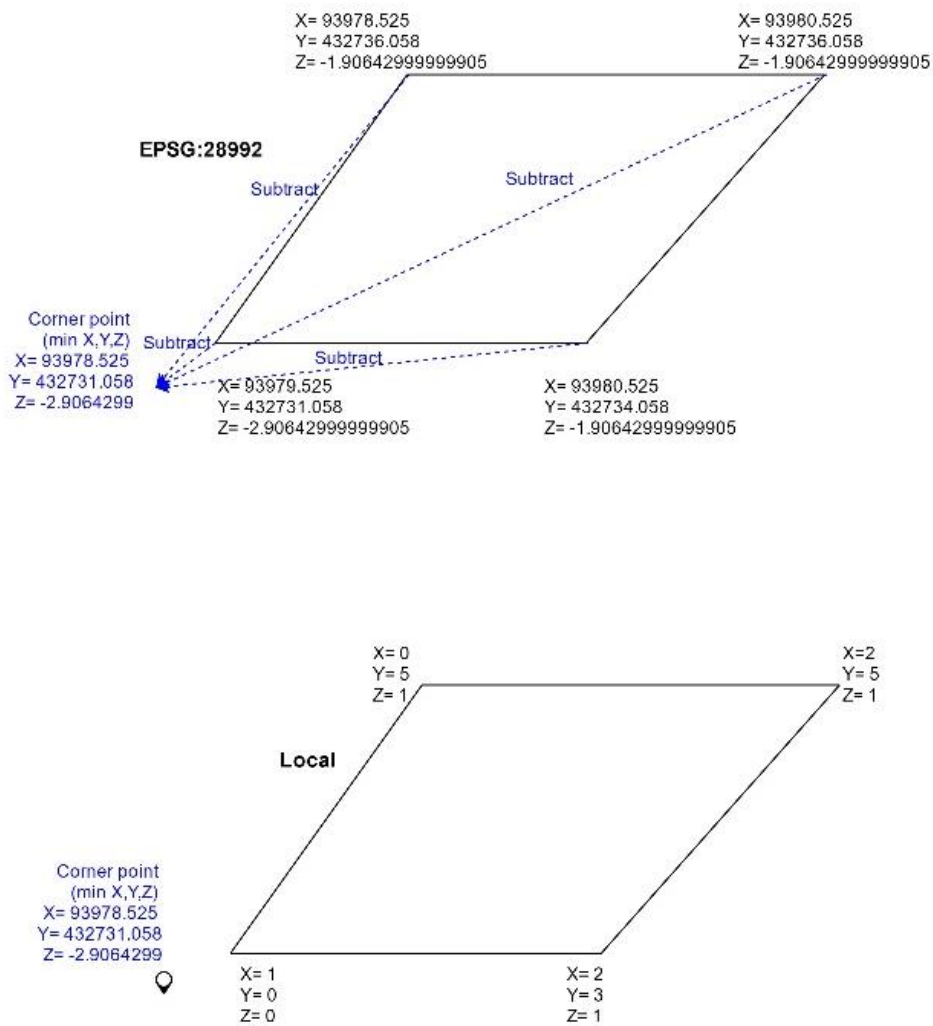


Figure 27- Converting calculating local referencing to all points

Lastly, latitude, longitude in WGS 84 and elevation of a project are dedicated to the IfcSite class as shown in (Diakite, 2018). BIM model coordinates are usually stored in an IFC transformation file as shown in Figure 28. However, since EPSG:28992 system in the Netherlands is based on XY coordinates the rotation degree is not needed.

```
<?xml version="1.0" encoding="utf-8"?>
<CoordSysZup Units="M">
  <OriginX>67561.327969</OriginX>
  <OriginY>544577.20961</OriginY>
  <OriginZ>2.3</OriginZ>
  <RotationInXYPlane>5.9628533378646953</RotationInXYPlane>
</CoordSysZup>
```

Figure 28- Transformation file example

The resulting program is able to achieve the following steps for every element (such as a ground surface) *Figure 29*:

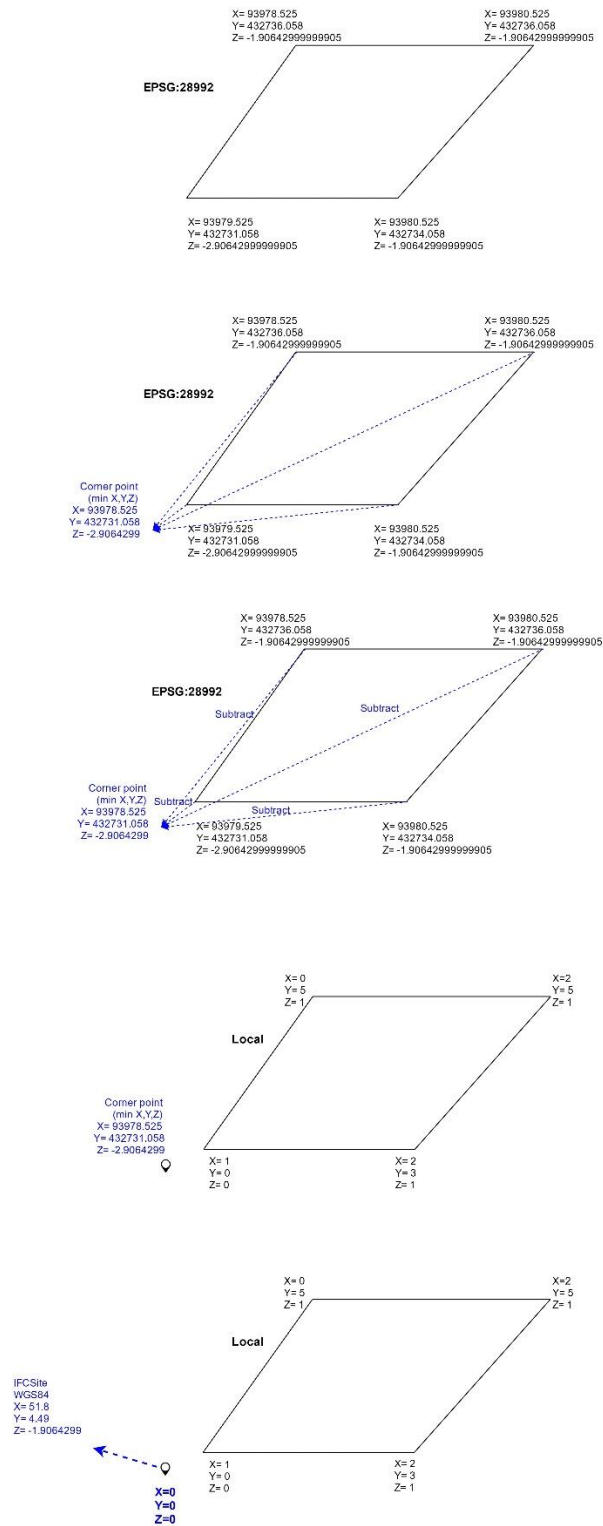


Figure 29- from CityGML to IFC geographical transformation

To implement the above conversion, all points in the project are found using:

```
tree.findall('./{%s}posList' % ns_gml)
```

and then to perform the coordinates conversion the following functions are defined and used:

find_reference_point(l), which takes a list of lists as input and will return the corner point as a list (point with least x and least y and least z). similarly, the function *find_max_point(l)* is defined which will take the maximum value instead.

The function *move_to_local(local_pont, l)* converts a list of points (*l*) into local coordinates by subtracting the local point value (*local_pont*) from them.

The function *from_EPSG28992_TO_WGS84* convert a point from EPSG28992 coordinate system to WGS84.

4.5 SEMANTICS

For semantics mapping, different criteria are used to create semantics mapping between the IFC domain and the CityGML domain. These criteria include the:

- There is an existing class in IFC that matches the source CityGML class
- The possibility to match the geometry
- The matching practiced in different research for example in In (Kavisha, 2015) the classes and notations of the two data models are collected and semantic mapping is created.
- The different practices in software that deals with this problem. Most notably FZK Viewer (Hütter, 2016)

That helped to match the semantic terminologies of objects and classes used in the data models. as shown in Table 3:

CityGML	IFC
AbstractBuilding	IfcBuilding
-GroundSurface	IfcSlab
-FloorSurface	-GroundSlab
-CeilingSurface	-FloorSlab
	-CeilingSlab
RoofSurface	IfcRoof
-WallSurface	IFCWall
-InteriorWallSurface	-Interior Wall
	-Exterior Wall
WallSurface	IfcCurtainWall

GenericCityObject	IfcBuildingElementProxy
SolitaryVegetationObject	IfcBuildingElementProxy
Opening	IfcOpeningElement
Door	IfcDoor
Window	IfcWindow
BuildingInstallation	<ul style="list-style-type: none"> - IfcBeam, - IfcColumn, - IfcCovering, - IfcStair, - IfcRailing, - IfcRamp

Table 3- IFC-CityGML Mapping (Kumar & Saran, 2015), own work

Looking at the diagram CityGML in different LODs can be converted to IFC models preserving most of their semantic information. (El-Mekawy et al., 2012). In this study the data model of CityGML Building is shown below Figure 30:

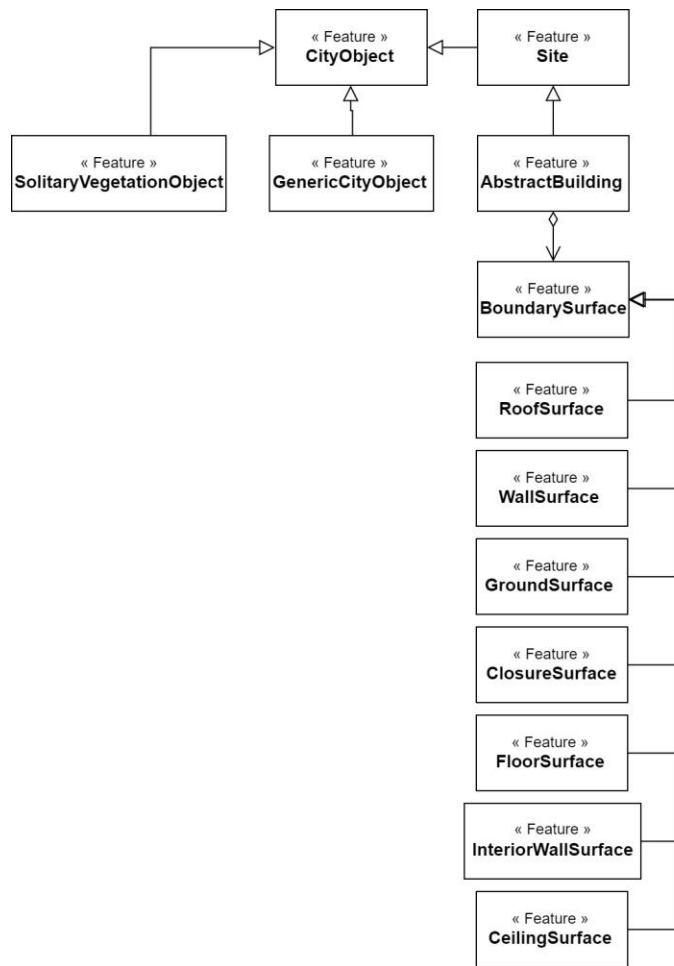


Figure 30- The data model of CityGML Building

Taking the Rotterdam3D data set, it has the following object types *Building*, *GenericCityObject*, *SolitaryVegetationObject*. As shown in Figure 31:

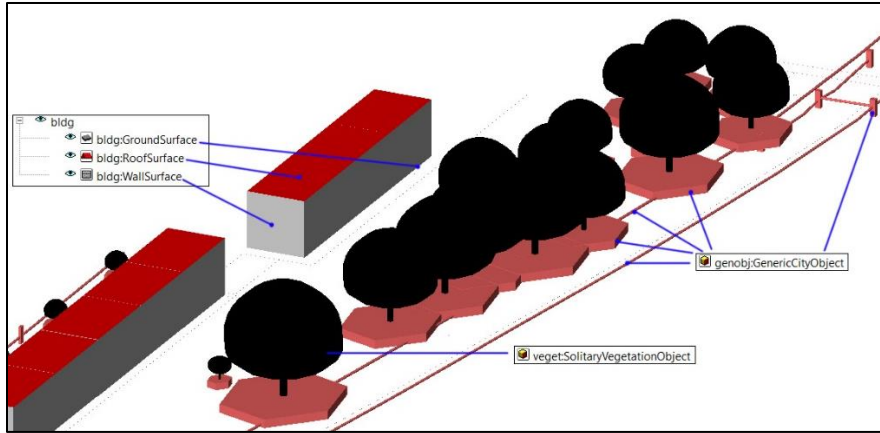


Figure 31-Rotterdam3D visualization

The data model for the features in 3DRotterdam features is depicted below:

Based on the above proposed semantic mapping, to implement the semantic transformation the following mapping is performed between the two data models Figure 32:

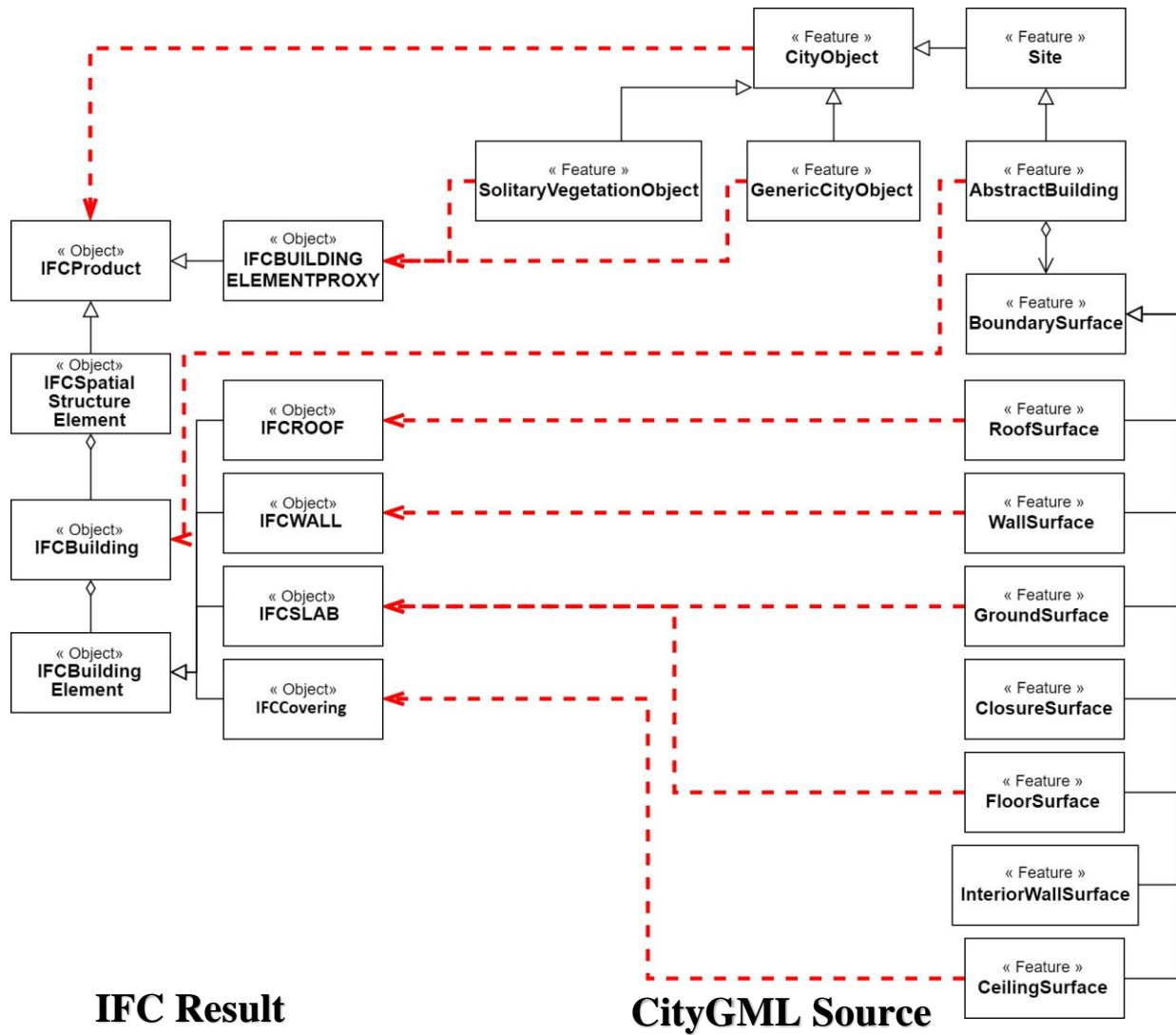


Figure 32_Semantic mapping application

To perform the above-described transformation the following steps are done:

Lists are created for every class that is in CityGML file. For the project here these classes are: *CityObjects*, *buildings*, *generic*, and *other*

The list *CityObjects* contains all the features that belong to the superclass *CityObjectMember*. The features here are *buildings*, *generic*, and *other*.

Next, for every building an *IFCBUILDING* entity is created. Next bounding elements of the building are defined and its entities are generated along with its geometry. These entities are defined based on their feature source in CityGML. See the example below:

```
if (Boundary.find('{%s}WallSurface' %ns_bldg)) :

    print(ifcsurfaceid, " = IFCWALL (", guid(), ", ",
$, 'bldg:WallSurface', ' ', $, $, ", ifcproductdefiniteshapeid, ", $);") )
```

Figure 33- Defining the object type (IFCWall) and the namespace (bldg:WallSurface) based on the source entity in CityGML (WallSurface)

Similarly, the other building elements are converted as follows:

Source CityGML Feature	Destination IFC object	IFC namespace
GroundSurface	IFCSLAB	IFCSLAB
FloorSurface	IFCSLAB	GroundSlab
RoofSurface	IFCROOF	RoofSlab
WallSurface	IFCWALL	bldg:WallSurface
InteriorWallSurface	IFCWALL	bldg:WallSurface
CeilingSurface	IFCCovering	CoveringSlab

Table 4- Semeatics mapping for building elements in the application

GenericCityObject in CityGML is converted to *IfcBuildingElementProxy*. The matching between *GenericCityObject* and *IfcBuildingElementProxy* Is logical. since *IfcBuildingElementProxy* provides the same functionality as *IfcBuildingElement* but without having a defined meaning.

Similarly, *SolitaryVegetationObject* in CityGML is converted to *IfcBuildingElementProxy*. The matching between *SolitaryVegetationObject* and *IfcBuildingElementProxy* is because there is no support for vegetation in IFC-2x3.

The application of the conversion methodology is similar to the above building elements.

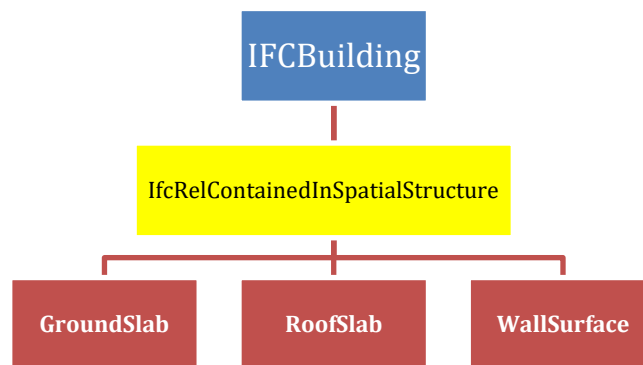
Source CityGML Feature	Destination IFC object	IFC namespace
<i>GenericCityObject</i>	<i>IfcBuildingElementProxy</i>	<i>IFCSLAB</i>
<i>SolitaryVegetationObject</i>	<i>IfcBuildingElementProxy</i>	<i>GroundSlab</i>

Table 5- Semantics mapping for other elements in the application

4.6 TOPOLOGY

It is important to show the spatial structure of the project elements using *IfcRelAggregates*, for example, see Figure 34. This is particularly important for some BIM software to read the IFC file correctly. To complete this relation *IfcRelAggregates* is used to represent the physical containment of the buildings in the *IFCProject*. this creates a certain level for the building in the spatial structure. This allows the use of *IFCRelContainedInSpatialStructure* to assign sub-elements of the building.

Hence the following relation can be established:



This is used to assign elements to a certain level of the spatial project structure. However, it is worth noting that an element can be assigned once to a certain level of spatial structure as shown in Figure 34. However, using *IfcRelContainedInSpatialStructure* spatial containments can be assigned on multiple levels. Hence a wall can be contained in both a building and a building store.

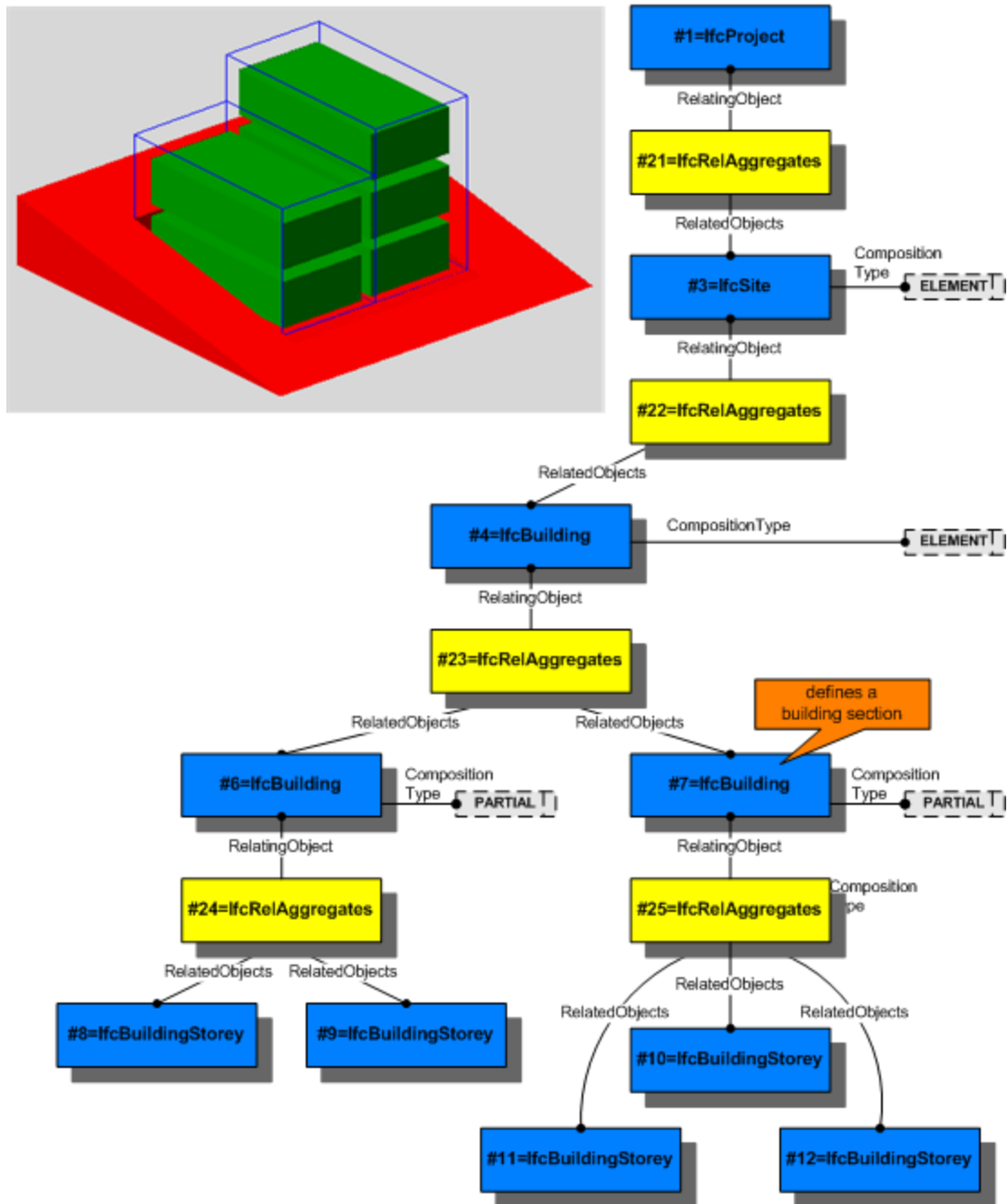


Figure 34- the use of *IfcRelAggregates* to establish a spatial structure including site, building, building section and storey (*"IfcSpatialStructureElement,"* 2019)

4.7 USER ACCESSIBILITY

The implementation was done in a simple way with the aim of allowing it to be incorporated in different applications.

CityGML is usually visualized using Cesium while 3DCityDB is running on the back end (“3DCityDB | cesiumjs.org,” n.d.). As shown in the figure below Figure 35:

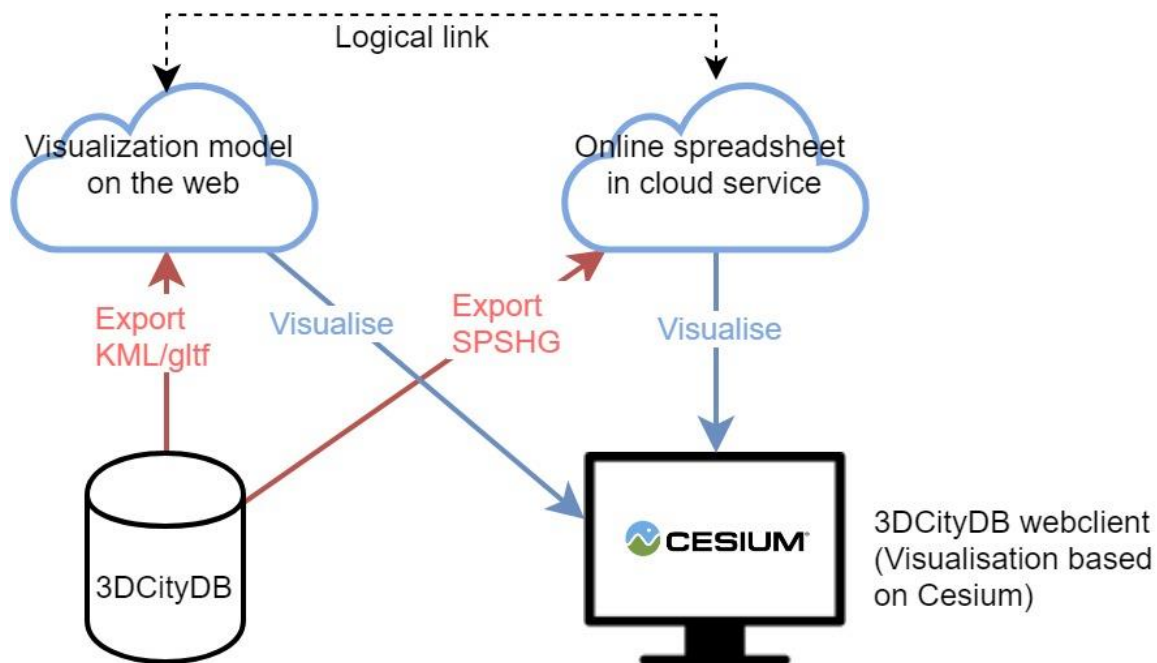


Figure 35-visualizing CityGML for users using cesium

The program that is resulted from the methodology can be incorporated in the above relation between CityGML and Cesium in two different ways:

The first possibility is by editing the 3DCityDB structure to incorporate the program *CityGML2IFC*, as shown in the figure below:

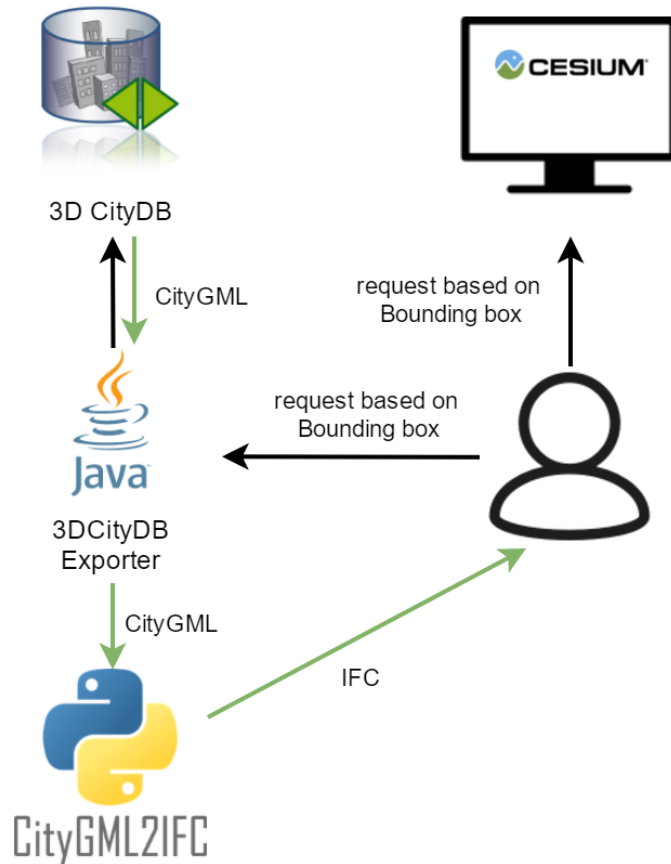


Figure 36- Incorporating methodology within 3DCityDB

3DCityDB uses a Java-based 3DCityDB-Exporter and it is open source. The idea here is to edit the source code of it to incorporate the python script of our method.

The second possibility is to incorporate the program in the real-time server of choice. For example, for 3D Rotterdam FME Server provides the user with the data of their need in real-time (when requested). This data is generated with the help of FME based transformers. On the other hand, FME can accept Python-based code to transform data using a special tool called *Python Caller*. Therefore, the methodology can be edited to adhere to the *Python Caller* tool requirements and be incorporated with the FME server service as shown in the figure below:

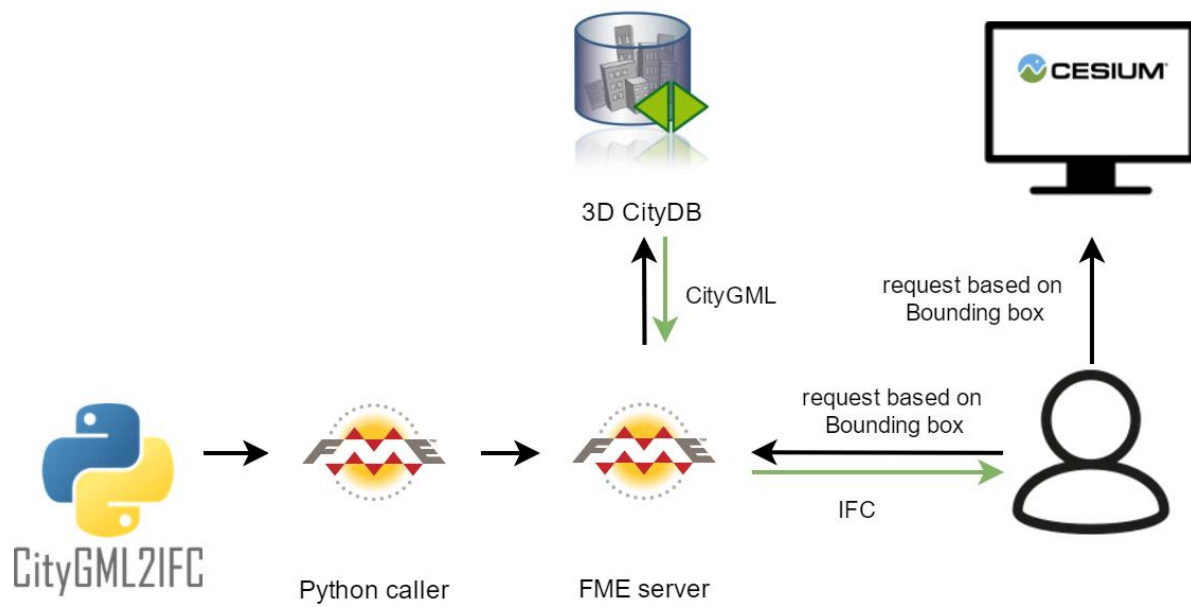


Figure 37-Incorporating conversion tool within the frame of FME server.

The two possibilities above, though explored, were not fully implemented due to the restrains of the research time and scope.

4.8 METHODOLOGY RESULTS

The methodology that is described in chapter 4 is a result of trial and error and a validation process that is discussed in detail in Chapter 6. The complete program that was resulted from the methodology can be found here (Complete Program). Taking controlled CityGML data set as input (Rotterdam 3D for example), the methodology can convert the data set to an IFC2x3 data set that is readable by all BIM software and editable with some (e.g. Revit). the ability of a BIM modeling software also depends on the ability of the software itself to deal with IFC files.

In addition, the resulting BIM models are geographically referenced with a local coordinates system. And it has the necessary topological relations to be accurate schematically. Some attributes are carried to the IFC file from CityGML, however, no particular focus was given to this point because of two factors:

- Attributes are dataset dependent. Hence, the program should be edited accordingly for each different dataset.
- IFC data model is expected to provide sufficient ability to retain most of the attributes from CityGML in a straight forward way. However, that requires checking which attributes are to be retained.

The complete methodology resulting model is shown below Figure 38:

(for higher resolution see appendix methodology resulting data model)

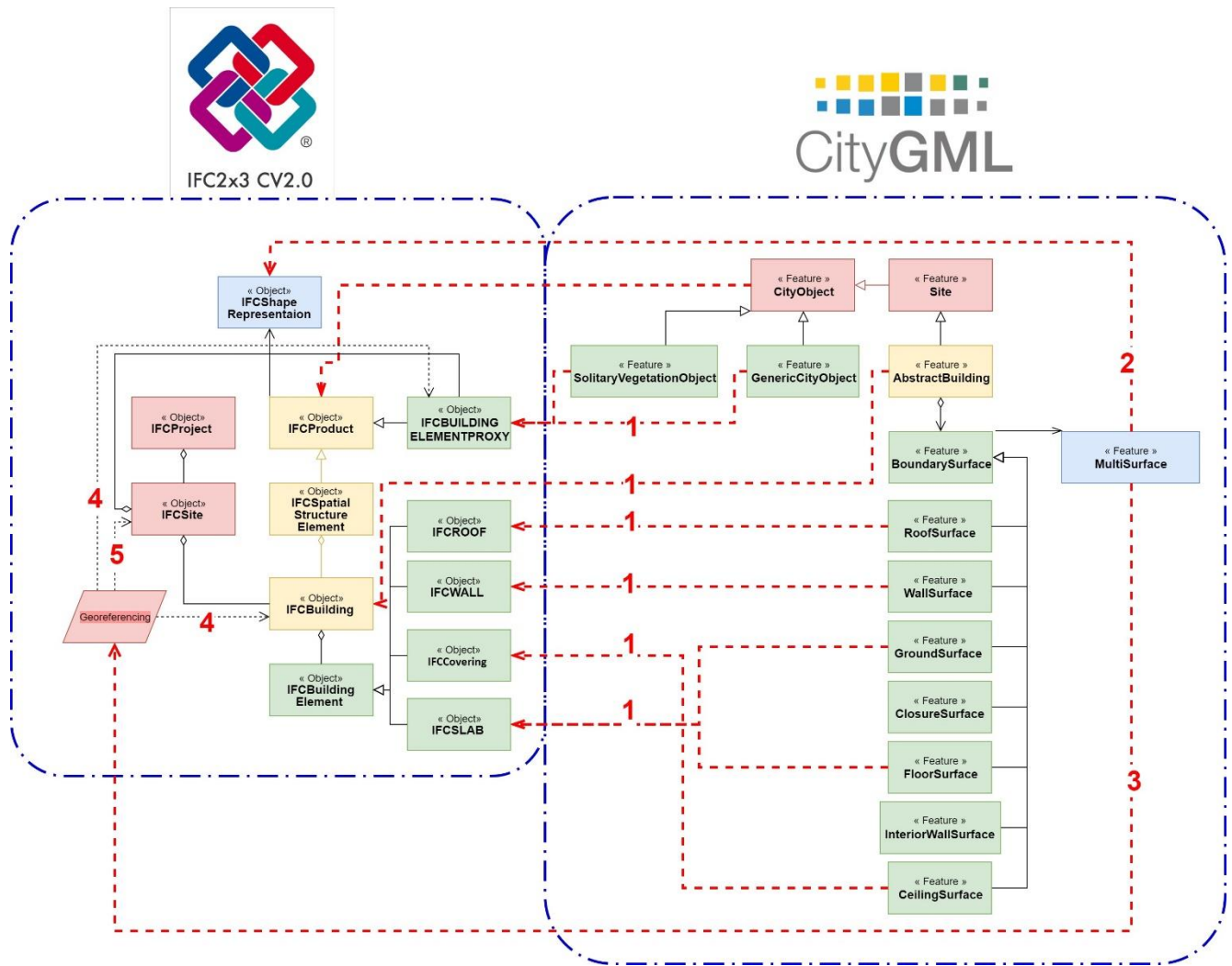


Figure 38- The complete methodology resulting data model

In the figure above the following transformations are shown numbered:

- 1- Semantic mapping from CityGML features to IFC objects.
- 2- Creating Geometry resources for IFC objects based on source CityGML geometry.
- 3- Creating Georeferencing point from the CityGML dataset.
- 4- Georeferencing IFC objects.
- 5- Storing Georeferencing information in the IFCSite object.

The resulting semantic conversion will look as following:

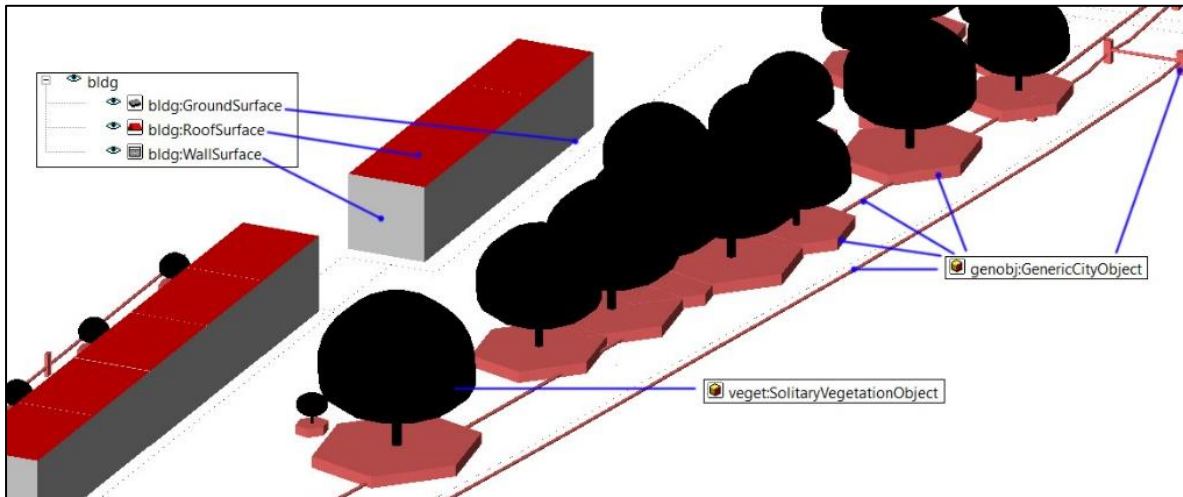


Figure 39- Source CityGML example dataset

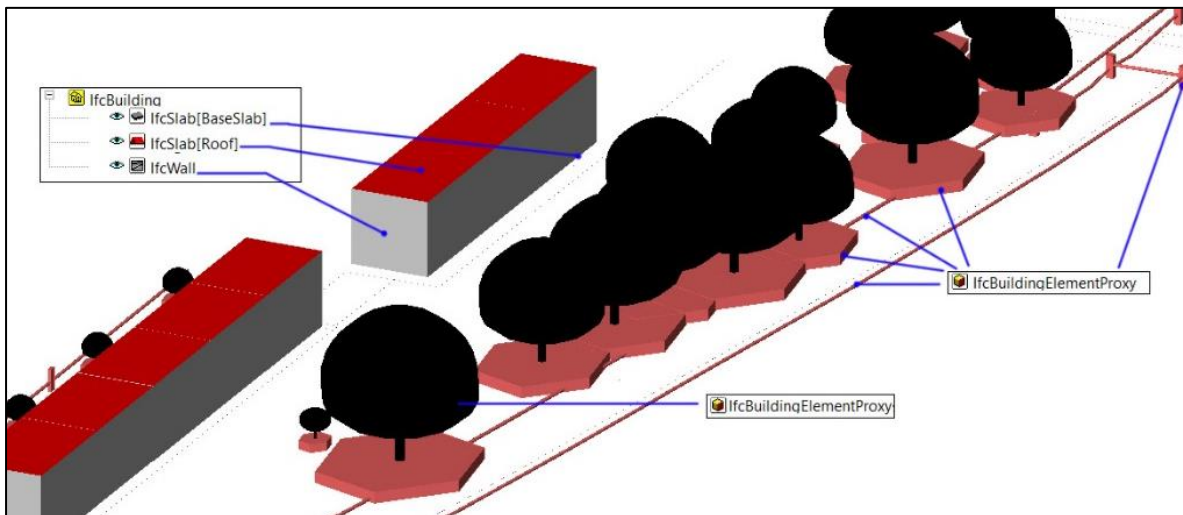


Figure 40- Resulting IFC dataset

5 IMPLEMENTATION

5.1 PROGRAM DESCRIPTION

The main implementation part consists of a program named “CityGML2IFC.py” it is a script file written in Python 3. When compiled the program will convert a source file in CityGML to destination file in IFC.

The complete code could and license information be found on GitHub here <https://github.com/nsalheb/CityGML2IFC>

nsalheb / CityGML2IFC

Watch 1 Star 1 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Security Insights Settings

No description, website, or topics provided. Edit

Manage topics

10 commits 1 branch 0 releases 1 contributor GPL-3.0

Branch: master New pull request Create new file Upload files Find File Clone or download

nsalheb Add files via upload	Latest commit c2c5a21 now
CityGML2IFC.py	Add files via upload 2 minutes ago
LICENSE.txt	Add files via upload now
Readme	Update Readme 7 minutes ago
Source.gml	Add files via upload 2 minutes ago

Readme

Program Description
The main implementation part consists of a program named “CityGML2IFC.py” it is a script file written in Python 3. When compiled the program will convert a source file in CityGML to a destination file in IFC.

License
and the program is licensed under General Public License v3.0

Participation
It is made with the help of Kavisha Kumar <https://3d.bk.tudelft.nl/kavisha/>.
Kavisha's GitHub <https://github.com/kkimmy>.

Used Modules
The following modules are imported and used in the program; these modules should be preinstalled before running the program:
xml.etree.ElementTree _ Is used here for parsing the XML data
os _ To interact with the operating system where the computer is running for example: reading time and file bath.
time_ To read the current time and stored in the created IFC files
itertools _ Is used to create a hashtaged unique id with an incremental value starting from a given value
sys _ Is used to allow files to be written on the hard disk
numpy _ To perform mathematical operation such as finding minimum value or subtract arrays
uuid_ To automatically generate unique IDs
pyproj _ To convert the resulting files projection

How to use program

- 1- Make sure that python 3 is installed.
- 2- Make sure all the necessary modules are installed, particularly:
 - a. numpy
 - b. pyproj
- 3- Download the program CityGML2IFC.py
- 4- the program will convert a source file in CityGML to destination file in IFC.
- 5- Change the name of your source CityGML file to: “Source.gml”
- 6- Compile (Run) the program CityGML2IFC.py.
- 7- A file called Result.ifc will appear. This file is the result of the conversion.
- 8- Check Result.ifc on the BIM software of your choice.

Figure 41- Snapshot of the GitHub repository

The following modules are imported and used in the program; these modules should be preinstalled before running the program:

XML.etree.ElementTree	Is used here for parsing the XML data
os	To interact with the operating system where the computer is running, for example, reading time and file bath.
time	To read the current time and stored in the created IFC files
itertools	Is used to create a hashtagged unique id with an incremental value starting from a given value
sys	Is used to allow files to be written on the hard disk
numpy	To perform mathematical operations such as finding minimum value or subtract arrays
UUID	To automatically generate unique IDs
pyproj	To convert the resulting files projection

5.1.1 how to use program

- 1- Make sure that python 3 is installed.
- 2- Make sure all the necessary modules are installed, particularly:
 - a. numpy
 - b. pyproj
- 3- Download the program *CityGML2IFC.py*
- 4- The program will convert a source file in CityGML to destination file in IFC.
- 5- Change the name of your source CityGML file to: "Source.gml"
- 6- Compile (Run) the program *CityGML2IFC.py*.
- 7- A file called *Result.ifc* will appear. This file is the result of the conversion.
- 8- Check *Result.ifc* on the BIM software of your choice.

5.2 DATA DESCRIPTION

5.2.1 Rotterdam3D

Since the research focuses on converting CityGML data to IFC. The CityGML data model is of great importance. The municipality of Rotterdam made Rotterdam3D available for this research. Rotterdam3D is an open data produced and provided by the municipality of Rotterdam.

Rotterdam3D consists of LOD1 and LOD2 building simultaneously where the viewer of the data is designed to let the user view the required level of detail. Buildings in LOD1 is modeled as valid solids.

Rotterdam3D includes the following features: buildings, trees, ground level, design, and building information. Pipes and cables are also part of the 3D city model but are modeled as generic city objects. In Rotterdam 3D buildings are in LOD2, with the following model Figure 42;

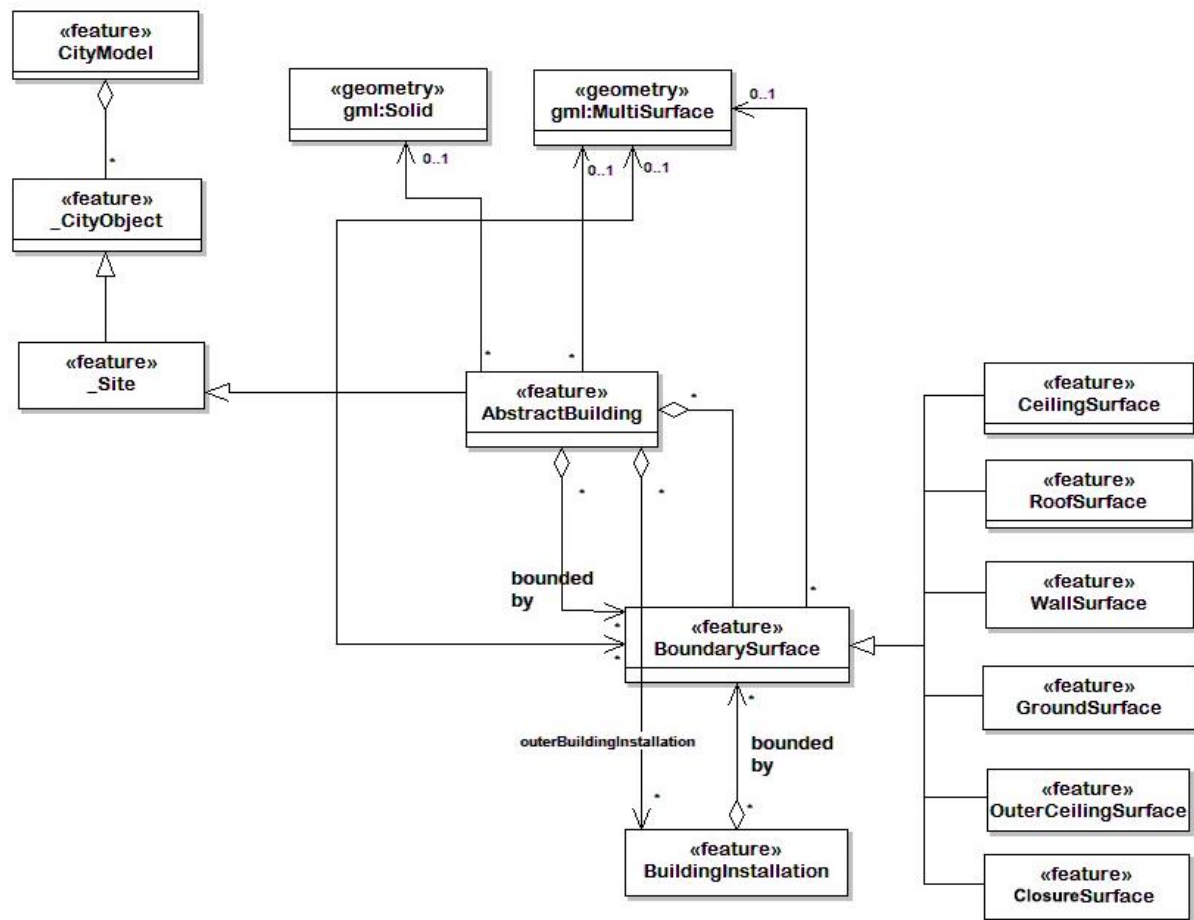


Figure 42-Rotterdam3D Building Model

5.2.2 Other data sources

Rotterdam3D 2.0 is the main source of CityGML data sets. It is available for the researcher since the research was conducted in cooperation with the municipality of Rotterdam. The municipality also provided examples of BIM models. Simplified IFC models are also produced using BIM software such as ArchiCAD and Revit.

Open IFC Model Repository

Examples of IFC models building were at “Open IFC Model Repository”
<http://openifcmodel.cs.auckland.ac.nz/Model/Download>

The resulting conversion method is then be tested on other publicly available CityGML datasets such as Berlin 3D (<http://www.businesslocationcenter.de/en/downloadportal>).

5.3 TOOLS

Other tools the tools that are required for the aim of the research is diverse depending on the different aspects of the project which are: CityGML, BIM, conversion and web development tools. These tools are:

- 1- FME: to view and select and apply basic transformations on the CityGML datasets. It's also useful to view and apply transformations on IFC datasets.
- 2- FZK viewer: useful and fast to view data for both formats; CityGML and IFC. Moreover, it can apply basic transformations between the two data formats which are useful for testing the developed transformation.
- 3- Python: is the main tool to apply the transformation including reading, parsing and writing the datasets. For the purpose of the project, different Python libraries are helpful such as:
- 4- Lxml: to parse the CityGML data sets
- 5- NumPy: This helps for dealing with large, multi-dimensional arrays and matrices.
- 6- ArchiCAD and Revit: both are BIM software and they are the most commonly used by users in the BIM world. This software will be used to visualize and test the resulting IFC datasets. Moreover, they will be used to produce samples for IFC data. For example, a wall

can be georeferenced in Revit and exported as IFC to acquire an example of a georeferenced IFC element.

- 7- IfcCheckingTool (<https://www.iai.kit.edu/1649.php>): which is an analysis tool for checking the semantic and syntactic correctness of IFC data. It can be used to check the final resulting data when needed.
- 8- IFCdiff <http://cgcad.thss.tsinghua.edu.cn/liuyushen/ifcdiff/>: A content-based automatic comparison approach for IFC files with hierarchical structures
- 9- <https://www.nist.gov/services-resources/software/ifc-file-analyzer> IFC The IFC File Analyzer generates a spreadsheet or CSV files from an IFC file.
- 10- Other tools for IFC can be found in: <http://www.ifcwiki.org/index.php/Freeware>

6 VALIDATION

As mentioned in Chapter 4.1. incremental development was done to make the methodology complete. In this chapter, the validation procedure is described in which the resulting model was tested. After every test, the methodology was adjusted and the conversion was developed to reach the complete software hence the “incremental” description.

This procedure to check the results is as follows:

- 1- FZK viewer: check if the results show probably on it FZK viewer (visual inspection).
- 2- Check with BIM viewer software (DDS-CAD viewer, Arredro BIM viewer).
- 3- Check with Building information modeling software (Revit, ArchiCAD).
- 4- FZK viewer: check message log, if any message report
- 5- Use IfcCheckingTool tool to see advanced errors
- 6- Check with users.

Further reports about this procedure are discussed in the following subchapters.

6.1 FZK VIEWER

In the beginning, initial conversion for LOD2 buildings was performed, the resulting IFC building was able to be viewed on FZK viewer after ignoring the error messages. However, for one building there were 32 errors found divided into 7 categories. These errors were a result of inaccurate conversion and provided an indication to certain areas of the program to be improved by resolving these errors, the following table describes these errors and how they were solved:

Error type	Error name	How the error is solved
Data prepare	Illegal GUID found, 10 errors:(in every element including 2 times for both IFCsite and IFCbuilding)	Some elements had incorrect GUID, for example, because of too many characters in GUID or including a space in the name. This problem was solved by giving accurate IDs for all elements. As shown in the example below: 'IFC-wall'→ '36d1601925d54a5ca6a4cd'
EccoError	Runtime error: incomplete assignment, 7 errors	In STEP encoding; when assigning an object to multiple other objects the comma should be removed at the end of the list as shown in the example below: [#109,#110,] → [#109,#110]
EccoError	Runtime error: unresolved reference, 2 errors	References to unexisting objects were removed
EccoError	Runtime error: missing parameter for construction type IFCRoot, 1 error	For unknown parameters, the sign: "\$" is added. To avoid these kinds of errors: <i>missing parameter</i>
EccoError	Runtime error: too many parameters for every wall, 4 errors	The number of parameters should be accurate according to the IFC standard.
EccoError	Runtime error: <i>parameter expected</i>	For unknown parameters, the sign: "\$" is added. To avoid these kinds of errors: <i>parameter expected</i>

MapView_002	(can't find file material in FZKprogram)	This is a program-related error, When the capabilities of a program, In this case, FZK viewer, is unable to view a defined material in the data file.
-------------	--	--

Table 6- Errors after initial conversion and how they were solved

After fixing the above errors, FZK viewer was able to view the results unlike other software, see the table below:

Error Type	Result on program (readable= ✓ / not readable = X)					
	FZK	DDS-CAD viewer	Arreddo BIM viewer	BIM Vision	ArchiCAD	Revit
Illegal GUID	✓	X	X	X	X	X
Incomplete Assignment	✓	X	X	X	X	X
Unresolved reference	✓	X	X	X	X	X
Missing parameter	✓	X	X	X	X	X
Too many parameters	✓	X	X	X	X	X
Parameter expected	✓	X	X	X	X	X
Material missing	✓	X	X	X	X	X

Table 7- The result after fixing all the above errors, the file is still not visible in different BIM software

Also, noticeably some errors are related to the program itself, for example, the missing material above is considered an error in FZK while in other viewers is not. This error is fixed by adding material entities to the files and creating a relation between the different entities.

Another, example of a program-related error is shown in (“IfcOpenShell / Forums / Help: Creating profile definition with coordinates,” 2019) where most viewers will fail to create a closed face from the polyline and therefore may extrude the line segments into faces or show nothing at all. This is because *IfcPolylines* are not closed implicitly unlike *IfcPolyLoops*.

6.2 IFCHECKINGTOOL

The IfcCheckingTool is an analysis tool for checking the semantic and syntactic correctness of IFC data. The check considers IFC Schema versions as of IFC2X3 in the file formats SPF (STEP Physical File) and ifcXML. In an automatically generated interactive report, the results can be sorted according to different criteria. As far as possible, a hyperlink to the corresponding definition in the IFC specification is output for each error, and the error within the instance document can be displayed via an EXPRESS navigation window.

After running the tool on the above city object that does not contain any errors according to FZK message log, the following report is generated:

Results:	
Total Number of Errors	68
Total Number of Warnings	0
Total Number of Comments	2
Number of Ecco Errors	60
Object Statistic:	
Object Types	5

Object Instances	9
Object	Amount
IFCBUILDING	1
IFCPROJECT	1
IFCSITE	1
IFCSLAB	2
IFCWALL	4
Full Statistic:	
Types	24
Instances	95
Object	Amount
IFCAPPLICATION	1
IFCAXIS2PLACEMENT3D	1
IFCBUILDING	1
IFCCARTESIANPOINT	31
IFCDIRECTION	3
IFCFACE	6
IFCFACEOUTERBOUND	6
IFCGEOMETRICREPRESENTATIONCONTEXT	1
IFCOPENSHELL	6

IFCORGANIZATION	1
IFCOWNERHISTORY	1
IFCPERSON	1
IFCPERSONANDORGANIZATION	1
IFCPOLYLOOP	6
IFCPRODUCTDEFINITIONSHAPE	6
IFCPROJECT	1
IFCRELCONTAINEDINSPATIALSTRUCTURE	1
IFCSHAPEREPRESENTATION	6
IFCSHELLBASEDSURFACEMODEL	6
IFCSITE	1
IFCSIUNIT	1
IFCSLAB	2
IFCUNITASSIGNMENT	1
IFCWALL	4

According to the report above there are 68 errors, these errors consists mainly of a missing definition for example:

- *No view definition in the header*
- *No material definition*

The later problem was fixed by assigning different materials for every kind of element: (wall, roof, ground). However, adding materials increased the size of the file exponentially and can be ignored if not needed. This increase in size is very clear and affect performance on big files size (example below):

With materials 17MB

Without materials 5.8 MB

6.3 CHECKING OTHER FILES

Other files were checked for errors with IfcCheckingTool to control results, namely the default file; *IfcOpenHouse.ifc*. Then file came without errors. It was also tested on the other software and it worked on all of them as shown below:

Filename	Result on program (readable= ✓ / not readable = X)					
	FZK	DDS-CAD viewer	Arreddo BIM viewer	BIM Vision	ArchiCAD	Revit
IfcOpenHouse.ifc	✓	✓	✓	✓	✓	✓

6.4 IFCRELAGGREGATES

Adding correct *IFCRELAGGREGATES* (see chapter Topology) has fixed the problem with the conversion making the files readable with the following software:

Filename	Result on program (readable= ✓ / not readable = X)					
	FZK	DDS-CAD viewer	Arredro BIM viewer	BIM Vision	ArchiCAD	Revit
New.ifc	✓	✓	✓	✓	✓	✓

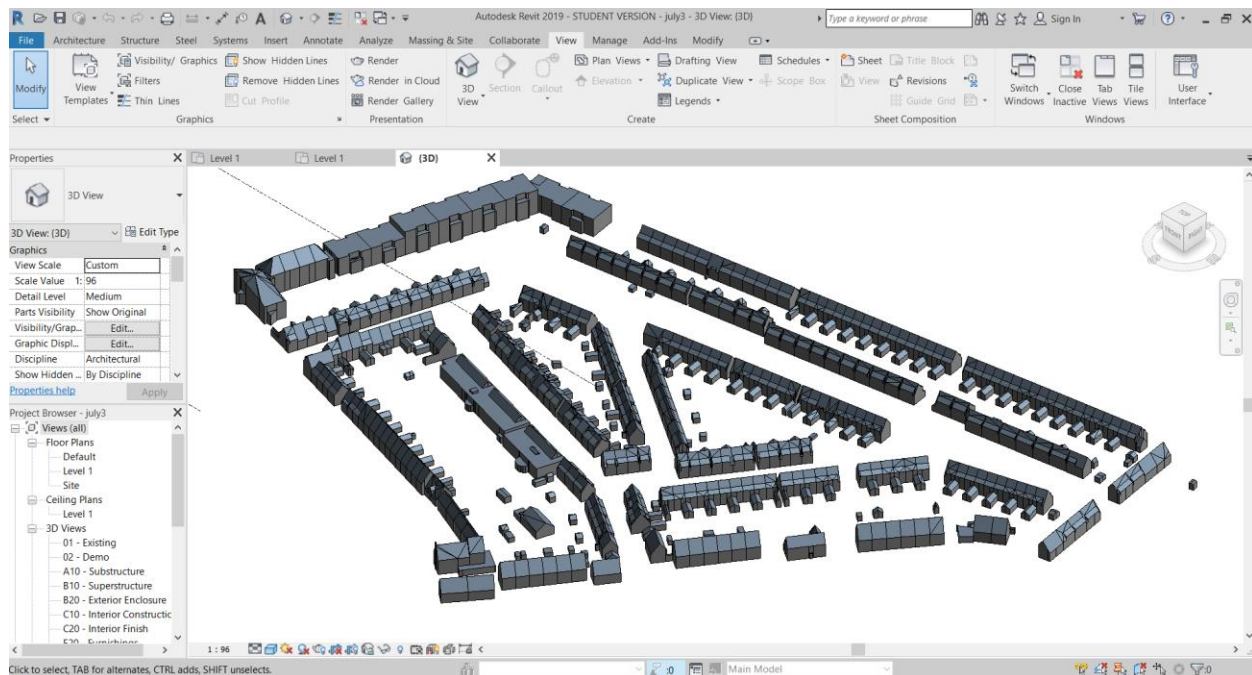
With ArchiCAD and Revit you can add elements to the resulting file. With Revit, you can even edit the resulting file. Basic editing with Revit includes:

Moving elements such as walls and roofs

Copy and paste elements

Applying an array copy

Creating facades and sections



Revit is also able to provide a list of found errors, see (A Revit error.log). these errors were also traced and fixed (as much as possible in the final conversion).

6.5 CHECKING WITH USERS

Lastly, the resulting files are checked with different users from different fields. namely: Architecture, building engineering, Geomatics, Public sector/GIS.

The results are shown in the table below:

Question	Answer
Percentage of users who were able to open the data with the software of their choice?	100%
Could the data be helpful with your work?	80% Yes 20% Maybe
In what ways can the data file be helpful for your work?	1- It becomes possible to exchange 3D-data between GIS and BIM. That is good for all of us. 2- Existing information 3- Give a context to design 4- Visualization 5- The exchange between software
In what way you think the data file can be more useful?	1- Detail information 2- Including terrain model 3- Metadata enrichment

	4- when it is defined as an element can be used for structural and environmental simulations in a BIM environment
What are the limitations of datafile?	1- the base level of all buildings is the same 2- Not possible to edit the file after the transfer

Table 8- Results from user questionnaire

Taking the opinions of the above users has helped to identify the limitation of the conversion against the requirements of the users. Some parts that were not focused on in the research, were for particular importance to the users. Namely;

- There was a request for a more data/meta data-rich IFC model. however, this depends on the amount of data that is provided with the original CityGML model. in addition to the effectiveness of the methodology itself.
- Including the terrain model, which is outside of the scope of the research.
- The ground floor of all buildings was on the same high, which is a limitation of the source CityGML data set.

There were also a few positive results that were derived from the questionnaire. Namely;

- The users expressed the possible helpfulness of the data.
- The users were able to open and use/view the data on their software of choice.

7 CONCLUSIONS

The aim of the present research was to examine the possibility of making 3D city data accessible in the design and construction software. This is done with the aim of using open source data format namely CityGML and IFC. Where different aspects of both formats are determined and studied in a comparable way.

Next, this work was undertaken to design a methodology to convert CityGML to IFC and a prototype software is developed to perform such conversion.

This is done in a way that allows the extensibility and reusability of the software in different applications such as, making the conversion accessible in a web viewer or incorporating its data transformation software.

7.1 RESEARCH QUESTIONS

To draw conclusions from this research, the research questions stated in Chapter 1.3 are answered here. The main research question of the study is:

HOW TO MAKE 3D CITYMODELS ACCESSIBLE IN DESIGN & CONSTRUCTION SOFTWARE?

To answer the above question the study has identified CityGML as a source of 3D Geo-information data and provided a methodology to convert this data to IFC. Through this conversion, the IFC data can be then used within the different design and construction software such as Revit & ArchiCAD.

In order to accomplish the above conversion, the first step was to identify the requirements of the conversion between the two standards. Hence the study had to answer the following sub-question:

- *What are the requirements for CityGML to IFC conversion?*

By studying the two standards each standard has its own characteristics that were identified. And to make the conversion successful these characteristics had to be matched from the source data to the destination data. These characteristics are defined in the following main categories:

- Semantics
- Geometry
- Coordinates
- Topology
- Encoding

These structures were generally kept throughout the research. Where for every standard these five categories had been looked into to make a successful conversion. Hence for each of these categories, a sub-question can be derived. Therefore, the second sub-question in this research was:

- *How to map semantics from CityGML to their equivalents in IFC?*

Compared to CityGML, IFC has a high number of classes. Therefore, to find a matching class in IFC multiple classes in IFC are studied and previous research is done to create a basic semantic matching. In subchapter 4.5, This mapping was extended based on the requirements of this research and the particular data set that is being tested. Next, the above matching is extended and described in the data models, and programming implementation is described. To apply the above; a theoretical description of the classes and their geometrical resources had to be comparable. Therefore, from the need to generate the geometrical resource for the above classes the third sub-question in this research is derived:

- *How to generate IFC Geometry Resource from CityGML?*

In CityGML in general and in the datasets that were studied here in particular; geometry resource for all features consists of polygonal faces. On the other hand, IFC objects can accept multiple kinds of geometry resources polygonal faces are also used for IFC elements to achieve direct matching. The main difference between CityGML and IFC is that the geometry is included in the same hierarchy as the semantics in CityGML, While in IFC there is a complete separation between the two. This kind of conversion and separation is presented in subchapter 4.3. During the process of creating the accurate geometry for IFC the fourth research sub-question is dealt with:

- *How to spatially reference the resulting IFC models?*

In CityGML Coordinates belong to a world coordinates reference system and no Local transformations are allowed, On the other hand, IFC files are locally referenced. in subchapter 4.4 a method is proposed and implemented to georeference the resulting IFC data file out of the source CityGML files by automatically creating a reference point for the project and storing it in the IFC class. This has resulted in a fully referenced IFC project.

The last characteristic to be converted is the topology which is related to the following sub-question:

- *What kind of topology should be retained in the resulting IFC model?*

In subchapter 2.2.4 it was shown that topology in IFC can be represented in different ways. However, the current study found that a certain topological relation is necessary to have an accurate IFC model that can be used on different BIM software, namely, aggregation and containment in spatial structure. This relation was described theoretically and implemented in programming.

The topological relation of in CityGML exists implicitly due to the hierarchal nature of XML files in general and CityGML datasets in particular, and this results on relations that are fixed according to the standard. In contrast, IFC's topological relation has to be represented explicitly. This also allows to have more flexibility of possible topological relations according to the user's needs.

Other possibility, is to generate different IFC geometry from CityGML based on different topological requirement. For example, a pipe in CityGML is converted to 2D surfaces for the purpose of this research. Instead, if the user requires creating a connection from this pipe to other pipes or city furniture; the pipe can be converted to a *SweptSolid* in IFC with a beginning and end points, these points can then be connected to other points that belongs to other pipes or City Furniture.

After having a complete and working conversion a final sub-question has to be answered is:

- *How to make the conversion methodology for CityGML to IFC accessible for users?*

To make the conversion method accessible for users. A simple program is developed with a clear way for usage has been provided. The way that the program is developed allows it to be incorporated within a data transformation software such as *FME* or incorporating it within a web viewer such as Cesium as shown in subchapters 4.7.

7.2 DISCUSSION

This research provides a basic framework of conversion from CityGML to IFC. It is possible with additional work and adjustment to extend the work to include more feature classes and other versions of CityGML and IFC.

It is clear from the research and practice that the complexity of IFC also comes with flexibility, in contrast to the strict rules of CityGML. This leads to the conclusion that there could be different ways to convert some elements from CityGML to IFC. This can lead to a different end result, for example, two elements that belong to one class in CityGML can be converted to different classes in IFC based on a specific attribute to generate a more semantically reach IFC file. Another possibility is to generate a different geometry for an object based on our requirements. For example, a pipe can be constructed as a *SweptSolid* instead of *MultiSurface* so that we can add the necessary topological connections to the other pipes in the IFC project.

Different BIM software deals with IFC data in different ways. This was helpful to provide different readings to debug the errors. It was evident also that commercial software such as Revit expected only completely accurate IFC models to be imported into it, unlike free software such as FZK Viewer. This could be because commercial software tries to push its own proprietary data formats.

It is important to make the conversion adequate to the needs of the user so that unnecessary information is left out from the conversion to reduce file size and for simplicity. One approach to be taken is to make the conversion as complete as possible by including all the possible information. Next, we can leave out parts of this complete conversion according the user needs, this can be done by developing specialized tools or be simply creating different versions of the software as done in this research where two versions of the program are provided; “with material” and “without material” where the later create a smaller in size files.

7.3 LIMITATIONS

This study presents a methodology of conversion with certain assumptions of the state of the source CityGML data. Any changes with the source data require some adjustments on the methodology this is particularly difficult for researchers with no programming background. One important limitation that could be faced with other data sets is then geographical reference system. Where this research is done on data with an *EPSG:28992 coordinate system*. Data that belongs to a different coordinates system would require some adjustments accordingly.

The study focuses on producing data in *IFC2x3* format, while during the time of the study (January 2019) *IFC4* was introduced and it is expected to become more prevailing in the upcoming period hence the required adjustment should be considered.

7.4 FUTURE WORK

Some aspects of the research were left for future studies. For example, the conversion could expand from merely converting the geometry from the CityGML to IFC. to enhancing this geometry by adding volume to the surface as described in the figure below:

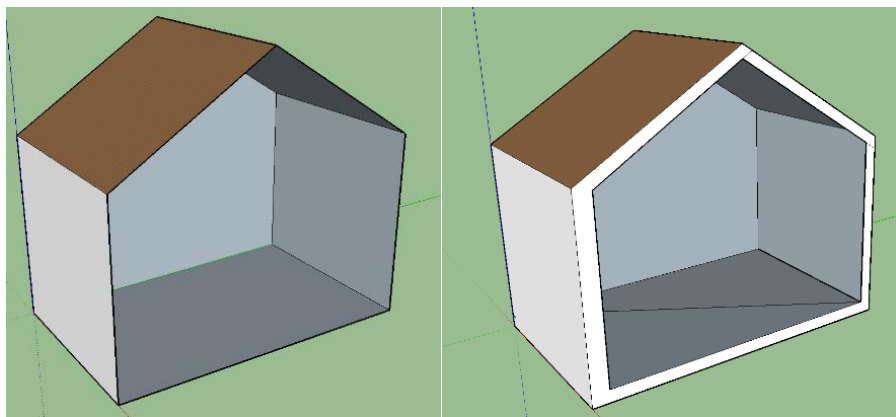
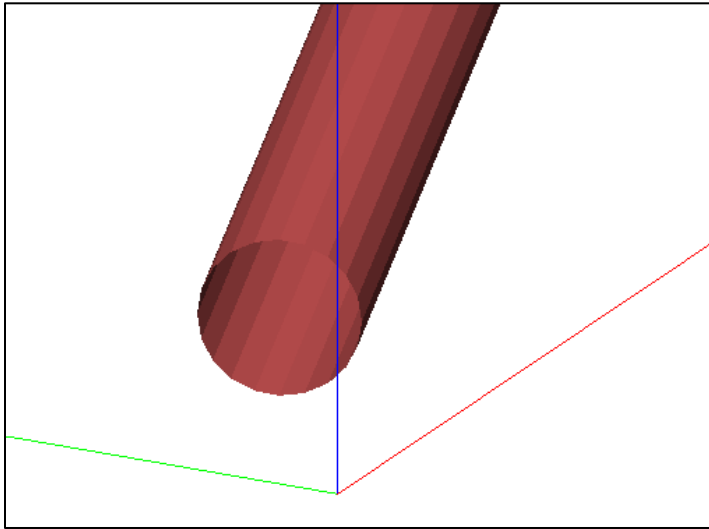


Figure 43- CityGML Building (left) Composed of 2D surfaces. IFC Building (right) with enhanced geometry composed of SweptSolid

This will be helpful to calculate Human space usage and buildings heat exchange. Other example regards converting pipes; in this research *CityGML2IFC* converts pipes in a direct way from *GenericCityObject* (2D faces) to *BuildingElementProxy* (2D faces). Instead, the methodology can be extended to provide the user the possibility to generate pipes that are composed of *SweptSolid* to create topological relations as mentioned before.



Other area of future research regards providing the methodology the ability to deal with CityGML datasets that belongs to a CRS other than *EPSG:28992*. This will lead to a higher level of complexity, because in this research converting from CityGML datasets that are based on *EPSG:28992* to the local coordinates system of IFC files is basically converting between two cartesian systems. Hence it is important to explore the possibility of converting CityGML datasets that are based on a CRS that is not Cartesian (*WGS84* for example).

Focusing more on including more attributes from CityGML to be transferred to IFC. Because when it comes to buildings; IFC is more detailed than CityGML hence all building information that are available in CityGML is possible to be transferred to IFC. But this requires checking each attribute from different datasets and including these attributes in the conversion. This can be done using a validation process similar to what is described in Chapter 6.

Other area of future work is including more classes from CityGML and higher LODs (up to LOD4). This is important for creating BIM models out of CityGML data as shown in figure 1. In

addition, this is important to avoid information loss when converting to IFC and to avoid information loss in case these data was converted back to CityGML.

Other possible area of future work is Including IFC4 in the conversion. Since, IFC4 is gaining more popularity and it addresses some issues that the users have with IFC2x3.

Additional future work possibility is creating a user interface for easily navigating the CityGML model and exporting the data to IFC. or include such the tool in existing software such as in www.3drotterdam.nl as discussed in Chapter 4.7 .

8 APPENDICES

A. REVIT ERROR.LOG

Revit is expecting integers instead of floats in IFCSite coordinates

```
/******  
*****
```

* ReadStepFile diagnostics.

* Date: Mon Jul 02 16:20:30 2018

* Produced by: The EXPRESS Data Manager Version 5.02.0100.07 : 28 Aug 2013

* Module: EDMstepFileFactory/EDMstandAlone

* Host: DESKTOP-L54SPS3

* Database: C:\Users\nebra\AppData\Local\Temp\{0A0EB9C9-7646-4E2D-9DC8-267E7272A059}\ifc

* Database version: 5507

* Database creation date: Mon Jul 02 16:20:29 2018

* Model to populate: DataRepository.ifcimport

* Header model to populate:

* Step file name: New_vision.ifc

* EDMuser: sdai-user

* EDMgroup: sdai-group

* License ID and type: 5605 : Permanent license. Expiry date:

* EDMstepFileFactory options: 010201000002

```
*****  
*****/
```

/* 29 */ #102, 'Rotterdam', 'Description of Default Site Rotterdam', 'LandUse', \$, \$, \$,
.ELEMENT., (4.513888922738401

*** Error in column number: 156 ***

*** Recoverable Error: Inconsistency: INTEGER expected - REAL found. *** on line 29.

, 51.88893480730484

*** Error in column number: 175 ***

*** Recoverable Error: Inconsistency: INTEGER expected - REAL found. *** on line 29.

, 1.75463243424257

*** Error in column number: 193 ***

*** Recoverable Error: Inconsistency: INTEGER expected - REAL found. *** on line 29.

), (4.513838893294704

*** Error in column number: 215 ***

*** Recoverable Error: Inconsistency: INTEGER expected - REAL found. *** on line 29.

, 51.88891307816326

*** Error in column number: 234 ***

*** Recoverable Error: Inconsistency: INTEGER expected - REAL found. *** on line 29.

, -0.694959999995253

*** Error in column number: 254 ***

*** Recoverable Error: Inconsistency: INTEGER expected - REAL found. *** on line 29.

Operation terminated with error: Error/warning during STEP File read operation.

Fatal errors.....: 0

Recoverable errors.....: 6

Warnings.....: 0

Info.....: 0

total.....: 6

Lines.....: 114

B. COMPLETE PROGRAM

```
import xml.etree.ElementTree as ET
import os
import time
import itertools
import sys
import numpy
import uuid
from pyproj import Proj, transform

def guid():
    x=str(uuid.uuid4())
    newstr = x.replace("-", "")
    return ("'" +newstr[:22]+'")

def find_reference_point(l):
    #takes a list of lists as input
    #will return the corner point as a list(point with least x and least y and least
z)
    x_list = []
    y_list = []
    z_list = []
    reference_point = []

    for x in l:
        x_list.append(x[0])
    minimum_x = numpy.min(x_list)
    reference_point.append(minimum_x)
    for x in l:
        y_list.append(x[1])
    minimum_y = numpy.min(y_list)
    reference_point.append(minimum_y)
    for x in l:
        z_list.append(x[2])
    minimum_z = numpy.min(z_list)
    reference_point.append(minimum_z)
    #return (tuple(reference_point))
    return (minimum_x,minimum_y,minimum_z)

def find_max_point(l):
    #takes a list of lists as input
    #will return a point with maxium values as a list(point with max x and max y and
max z)
    x_list = []
    y_list = []
    z_list = []
    max_point = []
    for x in l:
        x_list.append(x[0])
    max_x = numpy.max(x_list)
    max_point.append(max_x)
    for x in l:
        y_list.append(x[1])
    max_y = numpy.max(y_list)
    max_point.append(max_y)
    for x in l:
        z_list.append(x[2])
```

```

max_z = numpy.max(z_list)
max_point.append(max_z)
#return (tuple(max_point))
return (max_x,max_y,max_z)

def move_to_local(local_pont, l):
    #will convert list of points into local coordinates by subtracting the local
    point value from them
    local_points_list=[]
    for x in l:
        result = numpy.subtract(x, local_pont)
        local_points_list.append(numpy.ndarray.tolist(result))
    return local_points_list

def from_EPSG28992_TO_WGS84(x1,y1,z):
    # convert coordinates from EPSG28992 TO WGS84
    inProj = Proj(init='epsg:28992')
    outProj = Proj(init='epsg:4326')
    x2,y2 = transform(inProj,outProj,x1,y1)
    return (x2,y2,z)

def chunks(l, n):
    #break the list "l" into sub-lists each has the length of "n"
    n = max(1, n)
    return [l[i:i + n] for i in range(0, len(l), n)]

def CityGML2IFC(path,dst):
    tree = ET.parse(path)
    counter = itertools.count(1000)
    #counter is used to create a hashtaged unique id with an incremintal value
    starting from the value given above

    root = tree.getroot()
    #define name spaces
    if root.tag == "{http://www.opengis.net/citygml/1.0}CityModel":
        # -- Name spaces
        ns_citygml = "http://www.opengis.net/citygml/1.0"
        ns_gml = "http://www.opengis.net/gml"
        ns_bldg = "http://www.opengis.net/citygml/building/1.0"
        ns_tran = "http://www.opengis.net/citygml/transportation/1.0"
        ns_veg = "http://www.opengis.net/citygml/vegetation/1.0"
        ns_gen = "http://www.opengis.net/citygml/generics/1.0"
        ns_xsi = "http://www.w3.org/2001/XMLSchema-instance"
        ns_xAL = "urn:oasis:names:tc:ciq:xsdschema:xAL:1.0"
        ns_xlink = "http://www.w3.org/1999/xlink"
        ns_dem = "http://www.opengis.net/citygml/relief/1.0"
        ns_frn = "http://www.opengis.net/citygml/cityfurniture/1.0"
        ns_tun = "http://www.opengis.net/citygml/tunnel/1.0"
        ns_wtr = "http://www.opengis.net/citygml/waterbody/1.0"
        ns_brid = "http://www.opengis.net/citygml/bridge/1.0"
        ns_app = "http://www.opengis.net/citygml/appearance/1.0"
        # -- Else probably means 2.0
    else:
        # -- Name spaces
        ns_citygml = "http://www.opengis.net/citygml/2.0"
        ns_gml = "http://www.opengis.net/gml"
        ns_bldg = "http://www.opengis.net/citygml/building/2.0"
        ns_tran = "http://www.opengis.net/citygml/transportation/2.0"
        ns_veg = "http://www.opengis.net/citygml/vegetation/2.0"
        ns_gen = "http://www.opengis.net/citygml/generics/2.0"
        ns_xsi = "http://www.w3.org/2001/XMLSchema-instance"
        ns_xAL = "urn:oasis:names:tc:ciq:xsdschema:xAL:2.0"

```

```

ns_xlink = "http://www.w3.org/1999/xlink"
ns_dem = "http://www.opengis.net/citygml/relief/2.0"
ns_frn = "http://www.opengis.net/citygml/cityfurniture/2.0"
ns_tun = "http://www.opengis.net/citygml/tunnel/2.0"
ns_wtr = "http://www.opengis.net/citygml/waterbody/2.0"
ns_brid = "http://www.opengis.net/citygml/bridge/2.0"
ns_app = "http://www.opengis.net/citygml/appearance/2.0"

nsmap = {
    None : ns_citygml,
    'gml': ns_gml,
    'bldg': ns_bldg,
    'xsi' : ns_xsi,
    'xAL' : ns_xAL,
    'xlink' : ns_xlink,
    'dem' : ns_dem
}

dmy=time.strftime("%Y-%m-%dT%H:%M:%S", time.gmtime(os.path.getmtime(path)))
#dmys will print the current time in IFC compatible formay
dmys=" "+dmy+" "

cityObjects = []
buildings = []
generic=[]
other = []
#-- Find all instances of cityObjectMember and put them in a list
for obj in root.getiterator('{%s}cityObjectMember'% ns_citygml):
    cityObjects.append(obj)

for cityObject in cityObjects:
    #createa a list iof different city objects
    for child in cityObject.getchildren():
        if child.tag == '{%s}Building' %ns_bldg:
            buildings.append(child)
        elif child.tag == '{%s}GenericCityObject' %ns_gen:
            generic.append(child)
        else :
            other.append(child)

i=0
FILE = open(dst,"w")
sys.stdout = FILE
#sys.stdout = FILE means that every print statment will be saved instead in the
file
#-----
#-----
#define the ultimate reference point so that all points will have positive/small
values
points_list_complete = []
pos2 = tree.findall('..{%s}posList' % ns_gml)
for pos in pos2:
    x = ([float(val) for val in (pos.text).strip().split(' ')])
    #points_list_complete.append((x)[:3])
    points_list_complete.append(x)
reference_point=find_reference_point(points_list_complete)

reference_point_wgs84=from_EPSG28992_TO_WGS84(reference_point[0],reference_point[1],
reference_point[2])

```

```

max_point=find_max_point(points_list_complete)
max_point_wgs84=from_EPSG28992_TO_WGS84(max_point[0],max_point[1],max_point[2])
# -----
ifcprojectid="#" + str(next(counter))
ifcsiteid="#" + str(next(counter))
wall_id_list=[]
ground_id_list=[]
roof_id_list=[]
floor_id_list=[]
print("\nISO-10303-21;"
"\nHEADER;"
"\nFILE_DESCRIPTION(('ViewDefinition[CoordinationView_V2.0]'), '2;1');"
"\nFILE_NAME ('', (os.path.basename(path)), '', '', dmys,);"
"\nFILE_SCHEMA (('IFC2X3'));"
"\nENDSEC;"
"\n\nDATA;"
"\n#101 = IFCORGANIZATION ($, 'MSC_Geomatics', 'TU_Delft', $, $);"
"\n#104 = IFCPERSON ($, 'Nebras_salheb', 'TU_Delft', $, $, $, $);"
"\n#103 = IFCPERSONANDORGANIZATION (#104, #101, $);"
"\n#105 = IFCAPPLICATION (#101, 'CityGML2IFC', 'CityGML2IFC', 'CityGML2IFC');"
"\n#102 = IFCOWNERHISTORY (#103, #105, .READWRITE., .NOCHANGE., $, $, $,
1528899117);"
"\n#109 = IFCCARTESIANPOINT ((0., 0., 0.));"
"\n#110 = IFCDIRECTION ((0., 0., 1.));"
"\n#111 = IFCDIRECTION ((1., 0., 0.));"
"\n#108 = IFCAXIS2PLACEMENT3D (#109, #110, #111);"
"\n#112 = IFCDIRECTION ((1., 0., 0.));"
"\n#107 = IFCGEOMETRICREPRESENTATIONCONTEXT ($, 'Model', 3, 1.E-005, #108,
#112);"
"\n#114 = IFCSIUNIT (*, .LENGTHUNIT., $, .METRE.);"
"\n#113 = IFCUNITASSIGNMENT ((#114));"
"\n#115= IFCMATERIAL('K01-1');"
"\n#116= IFCMATERIAL('K01-2');"
"\n#117= IFCMATERIAL('K01-3');"
"\n#118= IFCMATERIAL('K01-4');"
"\n#119=IFCLOCALPLACEMENT($,#108);"
"\n", ifcprojectid, " = IFCPROJECT (" ,guid(),",", #102, 'core:CityModel', '', $,
$, $, (#107), #113);"
"\n", ifcsiteid, " = IFCSITE (" ,guid(),",", #102, 'Rotterdam', 'Description of
Default Site Rotterdam', 'LandUse', $, $, $,
.ELEMENT., "", max_point_wgs84, "", reference_point_wgs84, $, $, $, $);"

for building in buildings:
    ifcbuildingid = "#" + str(next(counter))
    print(ifcbuildingid, " = IFCBUILDING (" ,guid(),",", #102, 'bldg:Building', $,
$, $, $, $, $, $, $, $, $, $);"
    ifcsurfaceid_list=[]
    BoundedBy=building.findall('.//{%s}boundedBy' %ns_bldg)
    for Boundary in BoundedBy:
        #iterate over every boundedby class
        surfaces=Boundary.findall('.//{%s}surfaceMember' %ns_gml)
        for surface in surfaces:
            ifc_id_list=[]
            pl=0
            pos=surface.find('.//{%s}posList' %ns_gml)
            points_list= [float(val) for val in (pos.text).strip().split(' ')]
            #points list is the list of the bounding points eg. (5 points for
rectangle) every three points have the value of x and y and z
            points_list_chunks=chunks(points_list,3)
            #reference_point=find_reference_point(points_list_chunks)
            bounding_points=move to local(reference_point,points list chunks)

```

```

        while pl<len(bounding_points):
            #iterate over every point in the boundary and create an elemetn from
            this point and store the id in ifc_id_list
            ifc_id="#" + str(next(counter))
            print((ifc_id, " = IFCCARTESIANPOINT ((" , (str(bounding_points
[pl])).strip("[]"), "));")
            ifc_id_list.append(ifc_id)
            pl=pl+1
        ifcpolyloopid="#" + str(next(counter))
        print(ifcpolyloopid, " = IFCPOLYLOOP ((" , end=' ')
        #note; end= is used to identify what tp print after printstatment
have ended

        loop_string = ""
        for Element_id in ifc_id_list:
            loop_string += (str((Element_id)).strip("'"))
            loop_string += (" , ")
            loop_string2 = loop_string[:-1]
            print(loop_string2, "));")

        ifcfaceouterboundid="#" + str(next(counter))
        print(ifcfaceouterboundid, " = IFCFACEOUTERBOUND (" , ifcpolyloopid, " ,
.T.);")

        ifcfaceid="#" + str(next(counter))
        print(ifcfaceid, " = IFCFACE ((" , ifcfaceouterboundid, "));")

        ifcopenshellid="#" + str(next(counter))
        print(ifcopenshellid, " = IFCOPENSHELL ((" , ifcfaceid, "));")

        ifcshellbasedsurfacemodelid="#" + str(next(counter))
        print(ifcshellbasedsurfacemodelid, " = IFCSHELLBASEDSURFACEMODEL
((" , ifcopenshellid, "));")

        ifcshaperepresentationid="#" + str(next(counter))
        print(ifcshaperepresentationid, " = IFCSHAPEREPRESENTATION
($ , 'Body' , 'SurfaceModel' , (" , ifcshellbasedsurfacemodelid, "));")

        ifcproductdefiniteshapeid="#" + str(next(counter))
        print(ifcproductdefiniteshapeid, " = IFCPRODUCTDEFINITIONSHAPE ($ , $ ,
(" , ifcshaperepresentationid, "));")

        if(Boundary.find('{%s}GroundSurface' %ns_bldg)):
            ifcsurfaceid="#" + str(next(counter))
            print(ifcsurfaceid, " = IFCSLAB (" , guid(), " , $ , 'GroundSlab' , '
' , $ , $ , " , ifcproductdefiniteshapeid, " , $ , .BASESLAB.);")
            ifcsurfaceid_list.append(ifcsurfaceid)
            ground_id_list.append(ifcsurfaceid)

        if(Boundary.find('{%s}FloorSurface' %ns_bldg)):
            ifcsurfaceid="#" + str(next(counter))
            print(ifcsurfaceid, " = IFCSLAB (" , guid(), " , $ , 'GroundSlab' , '
' , $ , $ , " , ifcproductdefiniteshapeid, " , $ , .BASESLAB.);")
            ifcsurfaceid_list.append(ifcsurfaceid)
            floor_id_list.append(ifcsurfaceid)

        if(Boundary.find('{%s}RoofSurface' %ns_bldg)):
            ifcsurfaceid="#" + str(next(counter))
            print(ifcsurfaceid, " = IFCROOF (" , guid(), " , $ , 'RoofSlab' , '
' , $ , $ , " , ifcproductdefiniteshapeid, " , $ , .ROOF.);")
            ifcsurfaceid_list.append(ifcsurfaceid)

```

```

        roof_id_list.append(ifcsurfaceid)

        if(Boundary.find('{%s}WallSurface' %ns_bldg)):
            ifcsurfaceid="#" +str(next(counter))
            print(ifcsurfaceid," = IFCWALL (" , guid()," ,
$, 'bldg:WallSurface', ' ', $,$,$," ,ifcproductdefiniteshapeid," , $);")
            ifcsurfaceid_list.append(ifcsurfaceid)
            wall_id_list.append(ifcsurfaceid)

        if(Boundary.find('{%s}InteriorWallSurface' %ns_bldg)):
            ifcsurfaceid="#" +str(next(counter))
            print(ifcsurfaceid," = IFCWALL (" ,guid()," ,
$, 'bldg:WallSurface', ' ', $,$,$," ,ifcproductdefiniteshapeid," , $);")
            ifcsurfaceid_list.append(ifcsurfaceid)
            wall_id_list.append(ifcsurfaceid)

        if(Boundary.find('{%s}CeilingSurface' %ns_bldg)):
            ifcsurfaceid="#" +str(next(counter))
            print(ifcsurfaceid," = IFCCovering (" ,guid()," ,
$, 'CoveringSlab', ' ', $,$,$," ,ifcproductdefiniteshapeid," , $,$,$);")
            ifcsurfaceid_list.append(ifcsurfaceid)
            roof_id_list.append(ifcsurfaceid)

    if (ifcsurfaceid_list):
        print("#"+str(next(counter))," = IFCRELAGGREGATES (" ,guid()," , #102, $,
$, " ,ifcprojectid," , (" , ifcbuildingid,"));")
        print("#"+str(next(counter))," = IFCRELCONTAINEDINSPATIALSTRUCTURE
(",guid()," , #102, $, $, (" , end=' ' )
        loop_string = ""
        for Element_id in ifcsurfaceid_list:
            loop_string += (str((Element_id)).strip("'"))
            loop_string += (" ,")
            loop_string2 = loop_string[:-1]
            print(loop_string2, " ) ,",ifcbuildingid," );")

#Added material when needed by commenting the below back in

    #assign material #115 to all walls
    print("#"+str(next(counter))," = IFCRELASSOCIATESMATERIAL
(",guid()," ,#102,$,$, (" , end=' ' )
    loop_string = ""
    for Element_id in wall_id_list:
        loop_string += (str((Element_id)).strip("'"))
        loop_string += (" ,")
        loop_string2 = loop_string[:-1]
        print(loop_string2, " ) ,#115);")

    #assign material #116 to all roofs
    print("#"+str(next(counter))," = IFCRELASSOCIATESMATERIAL
(",guid()," ,#102,$,$, (" , end=' ' )
    loop_string = ""
    for Element_id in roof_id_list:
        loop_string += (str((Element_id)).strip("'"))
        loop_string += (" ,")
        loop_string2 = loop_string[:-1]
        print(loop_string2, " ) ,#116);")

    #assign material #117 to all floors
    print("#"+str(next(counter))," = IFCRELASSOCIATESMATERIAL
(",guid()," ,#102,$,$, (" , end=' ' )
    loop_string = ""
    for Element_id in floor_id_list:

```

```

        loop_string += (str((Element_id)).strip("'"))
        loop_string += (",")
        loop_string2 = loop_string[:-1]
        print(loop_string2, ")",#117);")

    #assign material #118 to all ground
    print("#"+str(next(counter))," = IFCRELASSOCIATESMATERIAL
(",guid(),",#102,$,$,(", end=' ')
    loop_string = ""
    for Element_id in ground_id_list:
        loop_string += (str((Element_id)).strip("'"))
        loop_string += (",")
        loop_string2 = loop_string[:-1]
        print(loop_string2, ")",#118);")

#pipes work
for pipe in generic:
    pl = 0
    pos = pipe.findall('.//{%s}posList' % ns_gml)
    for polygon in pos:
        ifc_id_list=[]
        pl=0
        points_list= [float(val) for val in (polygon.text).strip().split(' ')]
        #points list is the list of the bounding points eg. (5 points for
rectangle) every three points have the value of x and y and z
        points_list_chunks=chunks(points_list,3)
        #reference_point=find_reference_point(points_list_chunks)
        bounding_points=move_to_local(reference_point,points_list_chunks)

        while pl<len(bounding_points):
            #iterate over every point in the boundary and create an elemetn from
this point and store the id in ifc_id_list
            ifc_id="#"+str(next(counter))
            print((ifc_id)," = IFCCARTESIANPOINT ((",(str(bounding_points
[pl])).strip("[ ]"),");")
            ifc_id_list.append(ifc_id)
            pl=pl+1
        ifcpolyloopid="#"+str(next(counter))
        print(ifcpolyloopid," = IFCPOLYLOOP ((", end=' ')
        #note; end= is used to identify what tp print after printstatment have
ended

        loop_string = ""
        for Element_id in ifc_id_list:
            loop_string += (str((Element_id)).strip("'"))
            loop_string += (",")
            loop_string2 = loop_string[:-1]
            print(loop_string2, "));")

        ifcfaceouterboundid="#"+str(next(counter))
        print(ifcfaceouterboundid," = IFCFACEOUTERBOUND (",ifcpolyloopid,"
.T.);")

        ifcfaceid="#"+str(next(counter))
        print(ifcfaceid," = IFCFACE ((",ifcfaceouterboundid,");")

        ifcopenshellid="#"+str(next(counter))
        print(ifcopenshellid," = IFCOPENSHELL ((",ifcfaceid,");")

        ifcshellbasedsurfacedmodelid="#"+str(next(counter))
        print(ifcshellbasedsurfacedmodelid," = IFCSHELLBASEDSURFACEMODEL
((",ifcopenshellid,");")

```



```

        ifcshaperepresentationid="#" + str(next(counter))
        print(ifcshaperepresentationid, " = IFCSHAPEREPRESENTATION
($, 'Body', 'SurfaceModel', (", ifcshellbasedsurfacemodelid, "));")

        ifcproductdefiniteshapeid="#" + str(next(counter))
        print(ifcproductdefiniteshapeid, " = IFCPRODUCTDEFINITIONSHAPE ($, $,
(", ifcshaperepresentationid, "));")

        print("#" +
str(next(counter)), "=IFCBUILDINGELEMENTPROXY(", guid(), ", #102, 'Pipe', $, $, #119, ", ifcpr
oductdefiniteshapeid, ", $, $);")
        print("\n"
"\nENDSEC;"
"\nEND-ISO-10303-21;")

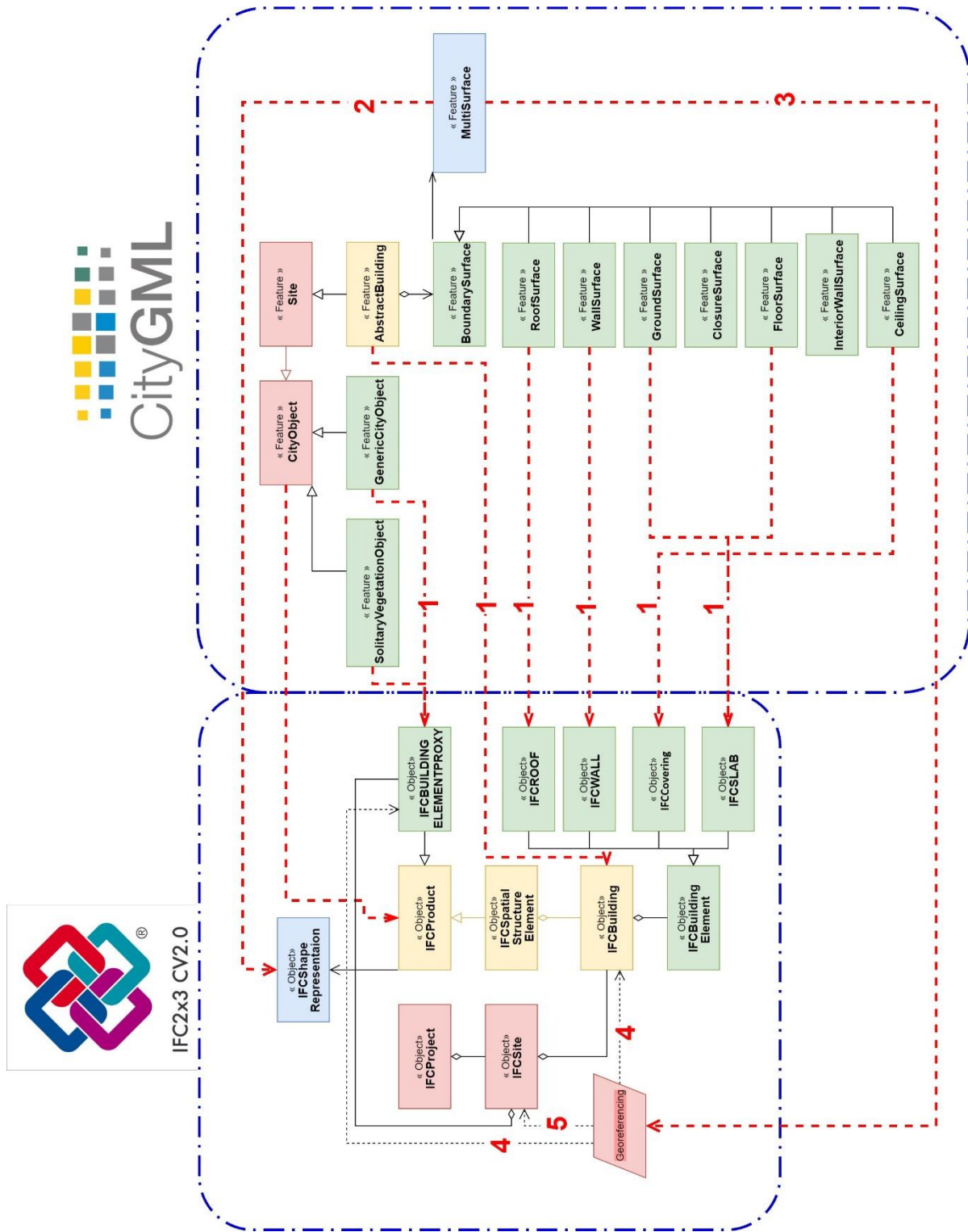
        FILE.write("")
        FILE.close()

#path="complete_city_mpdel_with_pipes_reprojected.gml"
path="Source.gml"
#path="new.gml"
#path="1.gml"
#path="complete_city_mpdel_with_pipes_reprojected.gml"
#path="small_pipe_reprojected.gml"
#path="new_ground_solid_removed.gml"
#path="Ground.gml"
dst="Result.ifc"

CityGML2IFC(path, dst)

```

C. METHODOLOGY RESULTING DATA MODEL



9 REFERENCES

3DCityDB | cesiumjs.org. (n.d.). Retrieved September 26, 2019, from

<https://cesiumjs.org/demos/3DCityDB/>

Autodesk. (2018). Revit 2018 New Feature—Geo-reference Coordinates | Revit Products 2018 |

Autodesk Knowledge Network. Retrieved April 13, 2019, from

<https://knowledge.autodesk.com/support/revit-products/learn-explore/caas/video/youtube/watch-v-Xij-oWzL1f4.html>

Borrmann, A., Kolbe, T. H., Donaubauer, A., Steuer, H., & Jubierre, J. R. (2013).

TRANSFERRING MULTI-SCALE APPROACHES FROM 3D CITY MODELING TO IFC-BASED TUNNEL MODELING. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-2/W1, 75–85. <https://doi.org/10.5194/isprsannals-II-2-W1-75-2013>

buildingsmart. (2019a). IfcSite. Retrieved April 13, 2019, from [http://www.buildingsmart-](http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/ifcproductextension/lexical/ifcsite.htm)

[tech.org/ifc/IFC2x3/TC1/html/ifcproductextension/lexical/ifcsite.htm](http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/ifcproductextension/lexical/ifcsite.htm)

buildingsmart. (2019b). Start Page of IFC2x3 Final Documentation. Retrieved April 16, 2019,

from <http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/index.htm>

Diakite, A. (2018). *About the Geo-referencing of BIM models*. 13.

Donkers, S. (2013). *Automatic generation of CityGML LoD3 building models from IFC models*.

127.

- El-Mekawy, M., Östman, A., & Hijazi, I. (2012). An Evaluation of IFC-CityGML Unidirectional Conversion. *International Journal of Advanced Computer Science and Applications*, 3(5).
<https://doi.org/10.14569/IJACSA.2012.030525>
- GIS and BIM Integration Will Transform Infrastructure Design. (2018, July 17). Retrieved January 21, 2019, from Redshift EN website: <https://www.autodesk.com/redshift/gis-and-bim-integration/>
- Gröger, G., Kolbe, T. H., Nagel, C., & Häfele, K.-H. (2012). *OGC City Geography Markup Language (CityGML) Encoding Standard*. 344.
- Gröger, G., & Plümer, L. (2012). CityGML – Interoperable semantic 3D city models. *ISPRS Journal of Photogrammetry and Remote Sensing*, 71, 12–33.
<https://doi.org/10.1016/j.isprsjprs.2012.04.004>
- Hütter, J. (2016, July 6). KIT - IAI - FZKViewer [Text]. Retrieved September 26, 2019, from <https://www.iai.kit.edu/1648.php>
- IfcOpenShell / Forums / Help: Creating profile definition with coordinates. (2019). Retrieved September 12, 2019, from <https://sourceforge.net/p/ifcopenshell/discussion/1782717/thread/efd956d7/>
- IfcSpatialStructureElement. (2019). Retrieved September 19, 2019, from <http://standards.buildingsmart.org/IFC/RELEASE/IFC2x3/TC1/HTML/ifcproductextension/lexical/ifcspatialstructureelement.htm>
- Kavisha. (2015). *CityGML based Interoperability for the Transformation of 3D Data Models*. 74.
- Kolbe, T. H. (2009). Representing and Exchanging 3D City Models with CityGML. In J. Lee & S. Zlatanova (Eds.), *3D Geo-Information Sciences* (pp. 15–31).
https://doi.org/10.1007/978-3-540-87395-2_2

Kolbe—2009—Representing and Exchanging 3D City Models with Ci.pdf. (n.d.).

Lee, N., & Hollar, D. A. (2013). *Probing BIM Education in Construction Engineering and Management Programs Using Industry Perceptions.* 8.

Ohori, K. A., Diakité, A., Ledoux, H., Stoter, J., & Krijnen, T. (2018). *Final report 10 January, 2018.* 30.

STEP-file, ISO 10303-21 [Web page]. (2017, January 3). Retrieved September 11, 2019, from <https://www.loc.gov/preservation/digital/formats/fdd/fdd000448.shtml>

