# A feasibility study on CityGML for cadastral purposes

Dsilva, M.G.

*Award date:*
2009

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computer Science

# A Feasibility Study on CityGML for Cadastral Purposes

McEnroe Gifford Dsilva
(0666625)
Master's Thesis

Supervisors:
dr. B. Speckmann
dr. M. A. Westenberg

Referee:
prof. dr. K. M. van Hee

Eindhoven, The Netherlands
July 2009

## Abstract

*The cadastral organization in the Netherlands, called the* Kadaster*, takes responsibility for the registration of land and property. In the current system used by the Kadaster, a piece of land, called a parcel is associated with legal information such as ownership right and ownership boundaries. A parcel may be associated either with a single ownership right or multiple ownership rights. For example, a parcel having a house owned by a single person indicates single ownership right and a parcel having a building with several apartments, each owned by a different person indicate multiple ownership rights. On selecting a parcel, its legal information is produced to the user. However in the case of multiple ownership rights, it is difficult to associate which part of the parcel is owned by which person. To address this issue, the current system produces a scanned image of the floor plans of the given building. These floor plans indicate the ownership rights of each floors. But the information from the scanned images of the floor plans cannot be interpreted by a computer for queries. Hence, there is a need for an alternative method to deal with parcels having multiple ownership rights.*

*This thesis proposes the use of* CityGML *to visualize and to store cadastral information into buildings having single as well as multiple ownership rights in 3D. CityGML is an XML based, common semantic information model for the representation of 3D city objects. In order to represent legal information into CityGML, this thesis proposes an extension to CityGML for cadastral purposes. It also provides an automated process to represent legal information in extended CityGML. As a result, this thesis presents a* proof of concept *for a way to represent the multiple ownership rights associated with a given parcel.*

# Acknowledgments

First and foremost, I would like to express my sincere gratitude to my project advisor dr. B. Speckmann for her continuous support and guidance for my thesis. Without her encouragement this project would not have reached completion. I am grateful to my co-advisor dr. M. A. Westenberg for his valued inputs, advice and patience throughout the project, and in particular, the immense support in helping me write the thesis report. I am immensely thankful to prof. dr. K. M. van Hee for his interest in the project and being a part of the defence committee. I would like to thank the Kadaster for their input and motivation, especially ir. M. Sterenberg for his co-operation and his enthusiasm towards this project. This project would not have been possible without the support from my colleague Mr. P. N. Jain. I thank Mr. X. L. Ernest Mithun for his help throughout the thesis. Finally, I thank my family and all my friends for their support.

McEnroe Gifford Dsilva
Eindhoven
July 28, 2009

# Nomenclature

| | |
|---|---|
| 2D | 2-Dimensional |
| 3D | 3-Dimensional |
| ADE | Application Domain Extension |
| DTM | Digital Terrain Model |
| GDI NRW | Geodata Infrastructure North Rhine-Westphalia |
| GIS | Geographic Information System |
| GML | Geography Markup Language |
| LOD | Level of Detail |
| OGC | Open Geospatial Consortium |
| SIG 3D | Special Interest Group 3D |
| UML | Unified Modeling Language |
| URI | Uniform Resource Identifier |
| XML | Extensible Markup Language |
| XSD | XML Schema Definition |

# Contents

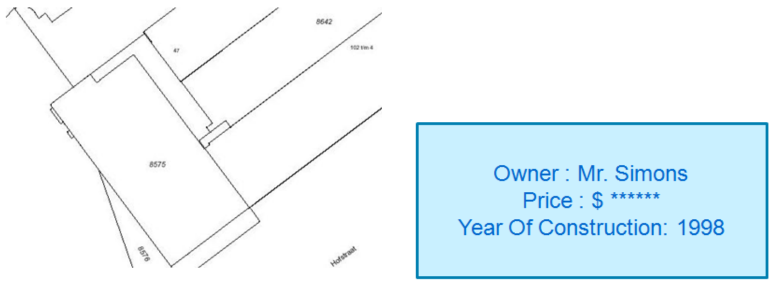
# Chapter 1

# Introduction

A correct, consistent and complete registration of land and property plays a very important role for legal purposes. A *cadastre* is the organization that takes care of the land and property registration. A cadastre [8] is defined by FIG (International Federation of Surveyors) as *"Cadastre is normally a parcel based, and up-to-date land information system containing a record of interests in land (e.g. rights, restrictions and responsibilities). It usually includes a geometric description of land parcels linked to other records describing the nature of the interests, the ownership or control of those interests, and often the value of the parcel and its improvements. It may be established for fiscal purposes (e.g. valuation and equitable taxation), legal purposes (conveyancing), to assist in the management of land and land use (e.g. for planning and other administrative purposes), and enables sustainable development and environmental protection."* In the Netherlands, the *Kadaster* takes the responsibility of the registrations of the land and property ownership.

The Kadaster describes information regarding land use in cadastral maps. These maps are *parcel* based where a parcel is an individual lot of land with its own legal description. Each parcel is associated with a unique identifier known as *parcel number*. A parcel may also be associated with legal information such as, ownership right, ownership type, address etc. An *ownership right* is the real right entitled to a person (or persons) for a parcel. A parcel may have a single ownership right or multiple ownership right. For example, a parcel having a house owned by a single person indicates single ownership right and a parcel having a building with several apartments, each owned by a different person indicate multiple ownership rights.

The cadastral map of a city used by the Kadaster comprises all parcels of that city. Generally, a parcel has a single ownership right associated to it. On selecting one such parcel, information on legal details about that parcel is provided to the user. Figure 1.1 shows one such example of a parcel with single ownership right where details about that parcel are provided. In case of a parcel with multiple ownership rights, the current system at the Kadaster provides a scanned image of the floor plans containing the ownership rights of that building. The building having multiple ownership rights may be either apartments of a building or stores of a commercial building,



Owner : Mr. Simons
Price : $ ******
Year Of Construction: 1998

(a) A parcel. (b) Legal information about the parcel.

**Figure 1.1**: *A parcel with single ownership right.*

each with many owners. We generalize both these cases as apartment buildings. The floor plans associate each apartment of that building with an ownership right. Figure 1.2 shows one such scanned image of a floor plan. In the figure we observe that there are multiple floors. Each floor has a floor plan associated to it. The numbers in the floor plans indicate the ownership right. The thick lines in the image indicate the ownership boundary. The thin lines indicate the interior partitions of the ownership boundary. The texts in the image give details about the usage type of the regions. For example, the text ’kamer’ indicates room, ’badkamer’ indicates bath room, etc.

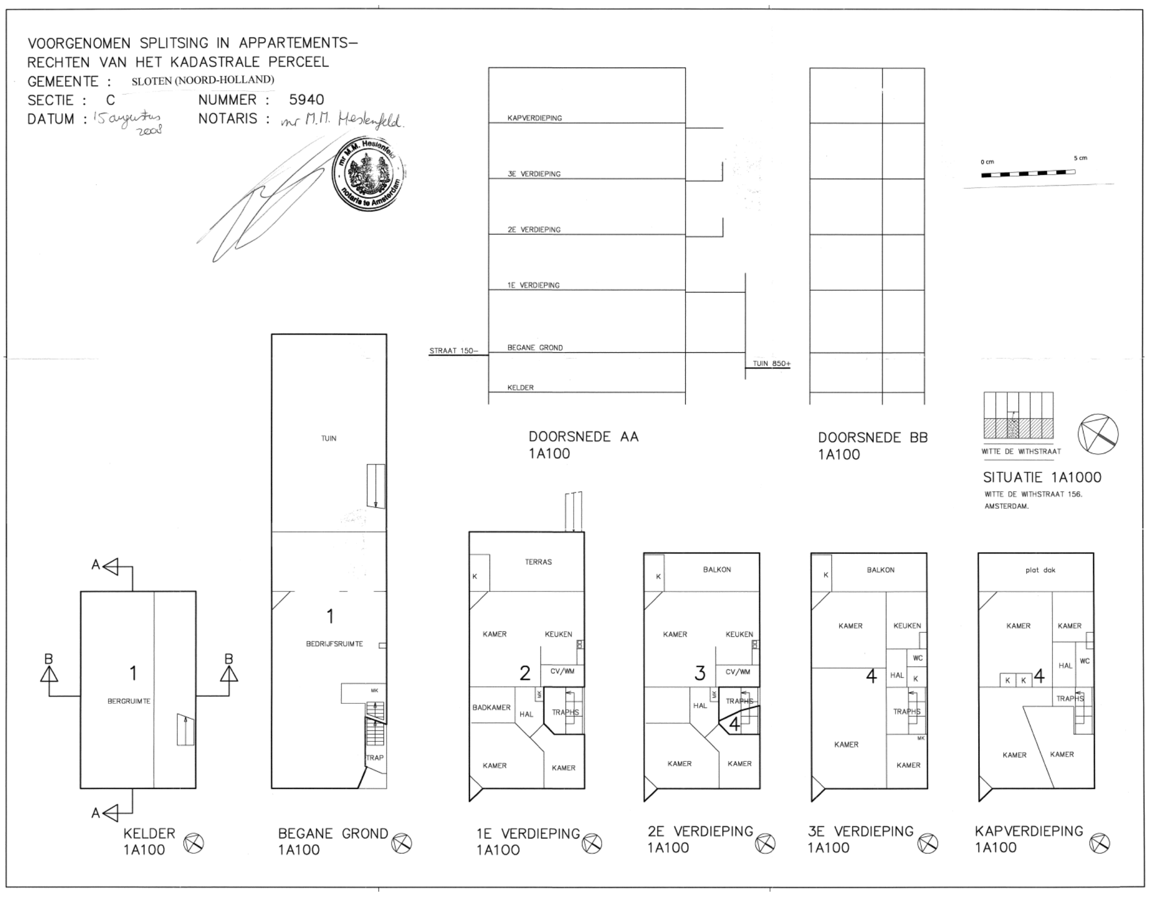


**Figure 1.2**: *Scanned image of a building containing floor plans including ownership rights (shown as large number) of each floor. The ownership boundaries are indicated with thick lines.*

The method used by the current system to deal with parcels having multiple ownership rights has a limitation. A scanned image of the floor plans is only human readable. Queries such as, ’who owns the 4th floor in a building with parcel number 5940?’, or ’who is the neighbour of Mr. Simons?’ cannot be answered by a computer, as the system will have to interpret the information from the scanned floor plans.

A solution to the above problem is by extracting the information from these scanned images. Information such as ownership boundary, ownership rights, floor number etc. are extracted. In order to extract the information from the scanned images, an automatic recognition system should be developed. This method recognizes and then extracts the information from the scanned images and stores it in a computer processable format. This will enable it to answer all the related queries.

Once the information is extracted from the scanned images, it has to be visualized. A possible solution is to visualize the extracted information in 2D. However, while using 2D, the visualization is cluttered when used for parcels with multiple ownership rights. Therefore, instead of visualizing in 2D, a 3D visualization could help to avoid the clutter. Figure 1.3 shows the 3D visualization of the data. Here, the floors are stacked on top of each other and placed on the parcel. This way, the information of each ownership right is visualized separately.

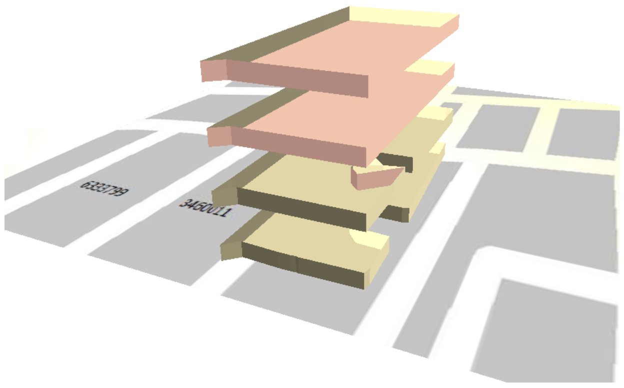


**Figure 1.3**: *3D modeled building with distinguishable floors, placed on a parcel.*

In order to visualize in 3D, the data must be represented in a suitable format. There are several formats available that can be used to visualize the extracted information. But most of the formats define models purely based on the geometric aspects (use of lines and polygons to represent the shape of a desired part). A format that adds semantic (meaning and relationship between objects) and topological (spatial properties) aspects into the models, along with geometrical aspects is *CityGML*. CityGML [3] is a common semantic information model for the representation of 3D urban objects that can be shared over different applications. It is an XML-based format for the storage and exchange of virtual 3D city models. By representing the data in CityGML format, a 3D city model is obtained, which can be visualized. CityGML is also extendable to adapt to the requirements of specific applications. Hence CityGML is a potential format to represent ownership rights information from the scanned images of floor plans.

## 1.1 Aim of the project

The aim of the project is to:

- Conduct a feasibility study on CityGML to represent legal information like ownership rights, ownership boundaries.
- Develop a method to automatically extract the information from building floor plans and represent it in computer processable format.

Figure 1.4 shows an overview of the project.

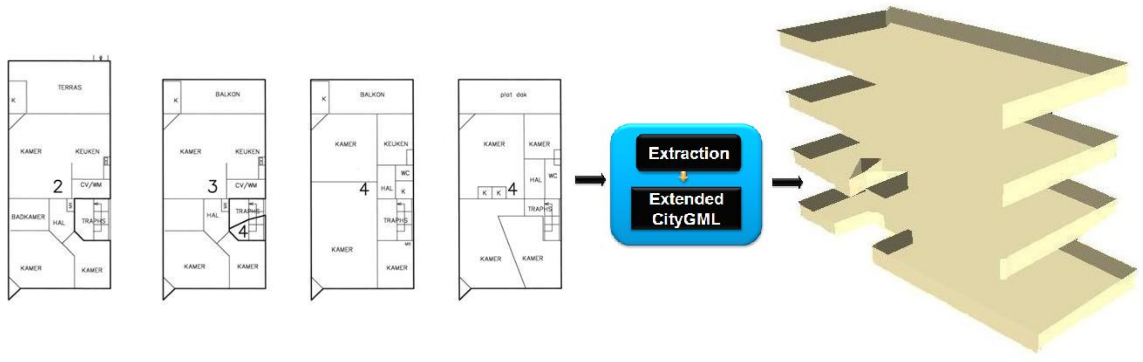


**Figure 1.4**: *Overview of the project, which shows a scanned image of the floor plans as an input to the system and an extended CityGML model as an output.*

This project is divided into two parts. This thesis contains the study on the feasibility of CityGML for cadastral purposes. The development of a method to automatically extract the information from building floor plans [4] is conducted as another part of the project.

## 1.2 Problem Definition

The aim of the project is to conduct a feasibility study on CityGML for cadastral purposes. CityGML has been developed for the representation of 3D city objects. In order to integrate the legal aspects associated with city objects like buildings, an extension to CityGML is required. This extension should incorporate all legal aspects such as ownership rights, etc to be used for cadastral purposes. Following this extension, a method for automatic representation of legal information in this extended CityGML needs to be developed.

## 1.3 Related work

CityGML has been developed very recently. Since 2002, the members of the Special Interest Group 3D (SIG 3D) [7] of the initiative Geodata Infrastructure North Rhine-Westphalia (GDI NRW) are working on the development of 3D models and geovisualization [3]. The SIG 3D is an open group which consists of more than 70 companies, research institutes and municipalities. This group also works on commercializing and popularizing the development of 3D models.

Several companies and institutes have worked on various aspects to encourage the growth of CityGML. Many universities like the Technical University of Bonn, the Technical University of Berlin and the Hasso-Plattner-Institute for IT Systems Engineering at University of Potsdam have provided CityGML viewers and other extensions to CityGML. In 2005, a project $'Pilot3D'$ of the GDI NRW was successfully implemented and evaluated which was a subset of CityGML. Currently, the official 3D models of several important cities have been modelled in CityGML. Among the recent developments, in August 2008, the CityGML specification version 1.0.0 was adopted as official Open Geospatial Consortium (OGC) standard [7], by the members of OGC. More details on CityGML can be found in the official web pages of CityGML [1, 2]. The schema of CityGML [5] is also provided to the end users for reference.

There are several extensions developed for CityGML by various organizations and universities. OpenGIS CityGML Encoding Standard [3] provides an example of an extension for Noise Immission Simulation. Another extension is developed by Schulte [6] for a web based 3D flood information service. These two extensions were used as a reference to come up with an application specific extension. More examples of extensions to CityGML can be found in the official web pages of CityGML [1, 2].

As far as the cadastre is concerned, Stoter [8] has worked extensively on the study of cadastres and focuses on how to record 3D situations in cadastral registration. Her thesis proposes a full 3D cadastre, in order to improve the method of registration for 3D property situations. This thesis was used in identifying the properties that need to be associated with a building for cadastral purposes.

## 1.4 Results

As a result of our study, we propose an extension to CityGML, developed for cadastral purpose. This extension shows ownership rights and other details associated to each apartment building, which provides a solution for the representation of multiple ownership rights. We have also developed a method for automatic extraction of information from scanned floor plans of the building. This method extracts relevant information such as ownership rights, ownership boundaries, floor numbers from these images. The extracted information from these scanned images can be represented in the proposed extension to CityGML. The project presents a *proof of concept* for an image processing pipeline [4] and a way to represent the multiple ownership rights associated with a parcel.

## 1.5 Organization

This thesis focuses on the feasibility study on CityGML. It provides details on CityGML, the study made on the various models of CityGML, and the proposal for an extension to CityGML for cadastral purposes. It also provides a method to automatically represent the extracted information from the scanned images in the extended CityGML. An introduction to CityGML is provided in Chapter 2. This chapter highlights the various features of CityGML that are extensively used in this thesis. It briefly discusses the CityGML models that are studied in this project, and also describes the structure of the Building model, that is used in CityGML to model a building. It provides a brief description on the two popular CityGML viewers. In Chapter 3, a description on the two methods of extension to CityGML is given. This is followed by a proposed extension to CityGML for associating legal properties with a building. The extension highlights the new attributes to be added to the building class and a new class to represent legal information for an apartment. This chapter concludes by showing the overall design structure of the proposed extended CityGML with an example of a building that is modelled with this extension. Chapter 4 describes an automated process to represent legal information into extended CityGML. It briefly describes the various steps involved in this process. In the end, it showcases a final outlook of a modelled building using this process. The entire thesis is summarized in Chapter 5. This chapter also contains a section on some future work related to this project.

Chapter 2

# An introduction to CityGML

Building virtual 3D city models is the most recent and the most popular method in urban planning. Several companies, universities, municipalities and cities are also building virtual 3D city models for other purposes such as, navigation planning, environment simulations, architectural design, disaster management, homeland security, tourism and facility management, to name a few. But most of the city models have been defined as purely geometrical models (use of lines and polygons to represent the shape of a desired part). This means that these models can be used mainly for visualization purposes. But, adding semantic (meaning and relationship between objects) and topological (spatial properties) aspects into the models, enables them to perform spatial data mining, queries, analysis tasks, etc. A format that provides facilities to model a city in a manner to enable it to perform all these tasks is *CityGML*.

CityGML [3] is a semantic information model for the representation of 3D urban objects. It is designed as an open data model and XML-based format for the storage and exchange of virtual 3D city models. It is implemented as an application schema (that describes the data elements and their interrelationships) of the Geography Markup Language 3 (GML3). GML3 is an extendible international standard for spatial data exchange and encoding, issued by the Open Geospatial Consortium (OGC) and the ISO TC211 (Standardization in the field of digital geographic information given by International Organization for Standardization). CityGML was developed with the aim to have a common definition of the basic entities, attributes, and relations of a 3D city model. In the aim of coming up with a single standard, the OGC has introduced an extensible CityGML standard, where we can extend CityGML to adapt to the requirements of specific applications, countries or organizations. CityGML also has several CityGML viewers that are used to visualize the 3D city models.

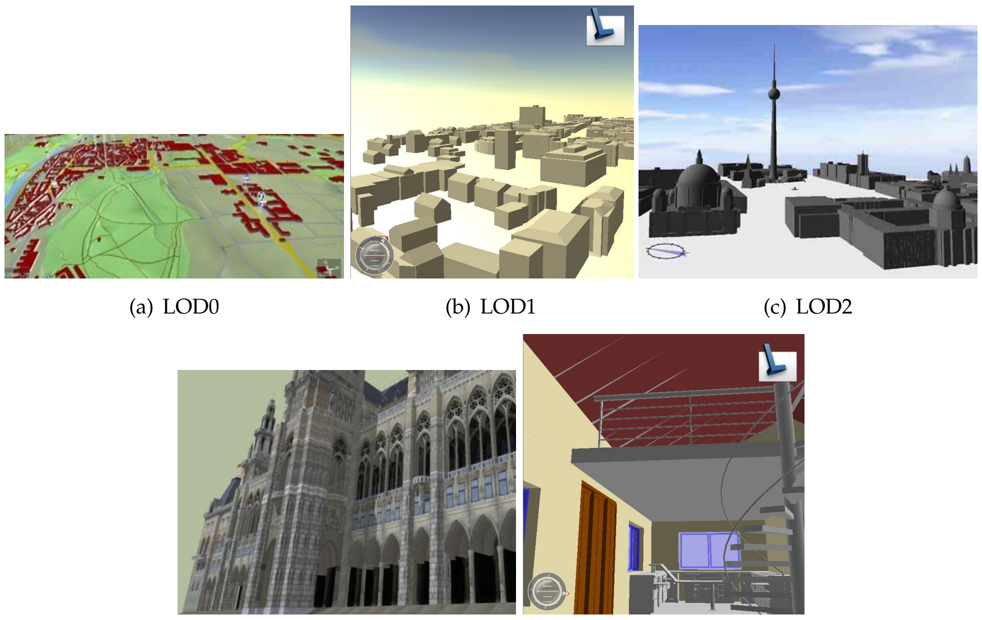
The aim of this project is to conduct a feasibility study on CityGML, to be able to model buildings with legal information. To fulfill this, we have to study the properties of a building provided in CityGML. If the provided properties are insufficient to associate legal information, we need to propose a suitable solution to address this issue. This chapter contains a description on the basic concepts in CityGML, and highlights some of its features that are used in this project. A building model in CityGML is a sub-class of the *thematic model* (see section 2.2.1). Therefore, a short description on the thematic model followed by a brief description on the building model is provided in this chapter. In Section 2.3 and 2.4, the code structure of the building model is provided with an example, which will help the reader to understand the way a building can be modelled and visualized in CityGML. In the end, the chapter presents details on the various CityGML viewers that were studied, in the survey conducted in the initial phase of the project.

## 2.1 Features of CityGML

CityGML provides several features that are used in various aspects while modelling a 3D city model. Some of the features are extensively used in this thesis. This section explains these features in brief.

### 2.1.1 5 Levels of Details

There are five different Levels of Details (LOD) supported by CityGML. Figure 2.1 gives an example to each of the five levels of details available in CityGML. LOD0 is the coarsest level consisting of two and a half dimensions. It basically consists of a terrain over which a map or an image can be draped. LOD1 consists of a block model, comprising buildings with flat roofs as seen in Figure 2.1(b). These block models can be mapped with textures. LOD2 defines buildings with definite roof structures and thematically (based on a certain theme) differentiated surfaces. It may also contain vegetation objects like trees. LOD3 defines models with detailed wall and roof structures, balconies etc. Textures can also be mapped onto these structures. In addition, detailed vegetation and transportation objects (eg: cars, bikes) can be modelled. LOD4 is a higher level of LOD3. It comprises of adding interior structures to 3D objects. For example, a building may be composed of rooms, interior doors, stairs and furniture, which can be shown in LOD4. The figure shows the interior of a house modelled in LOD4. Usually one of the five LODs are used to model a city. But it is also possible that the same object may be represented in different LOD simultaneously. Figure 2.2 gives an example where the same CityGML object (here, buildings) is visualized in 3 different LODs. This enables the analysis and visualisation of the same object with regard to different degrees of resolution.



(a) LOD0 (b) LOD1 (c) LOD2

(d) LOD3 (e) LOD4

**Figure 2.1**: *The five levels of details(LOD).*

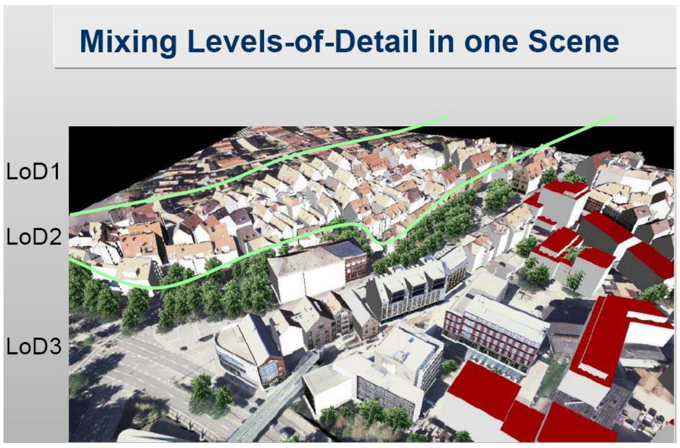


**Figure 2.2**: *Multiple representations in different LODs simultaneously (source:[3]).*

### 2.1.2 Dictionaries and external code lists for enumerative attributes

Objects usually have attributes that describe them. These attributes often have a restricted number of discrete values. An example is the attribute *BuildingClass type*, whose attribute values usually are habitation, sanitation, administration, trade, healthcare, security, etc. If attributes are misspelled, then it interferes with interoperability. In CityGML attributes and their possible values are specified by *external code lists* and implemented using *simple dictionaries*. Having such a structure ensures that the same name is used for the same notion. Simple dictionaries and external code lists may be extended or redefined by users, or can have references to existing models. In this thesis, the dictionaries and external code lists are used in the Application Domain Extension (ADE) in our proposed extension to CityGML, which is explained in the next chapter.

### 2.1.3 Modularisation

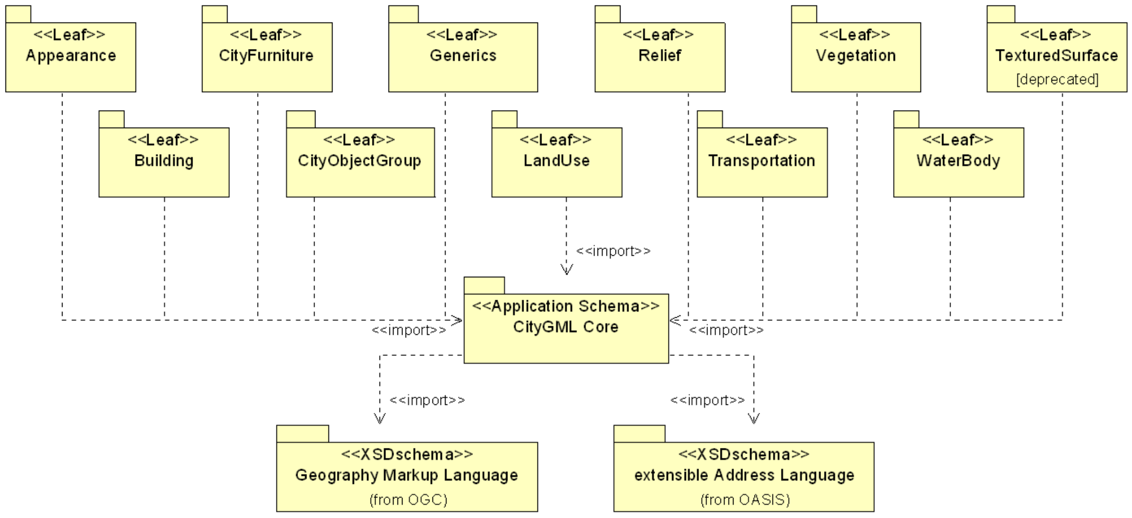


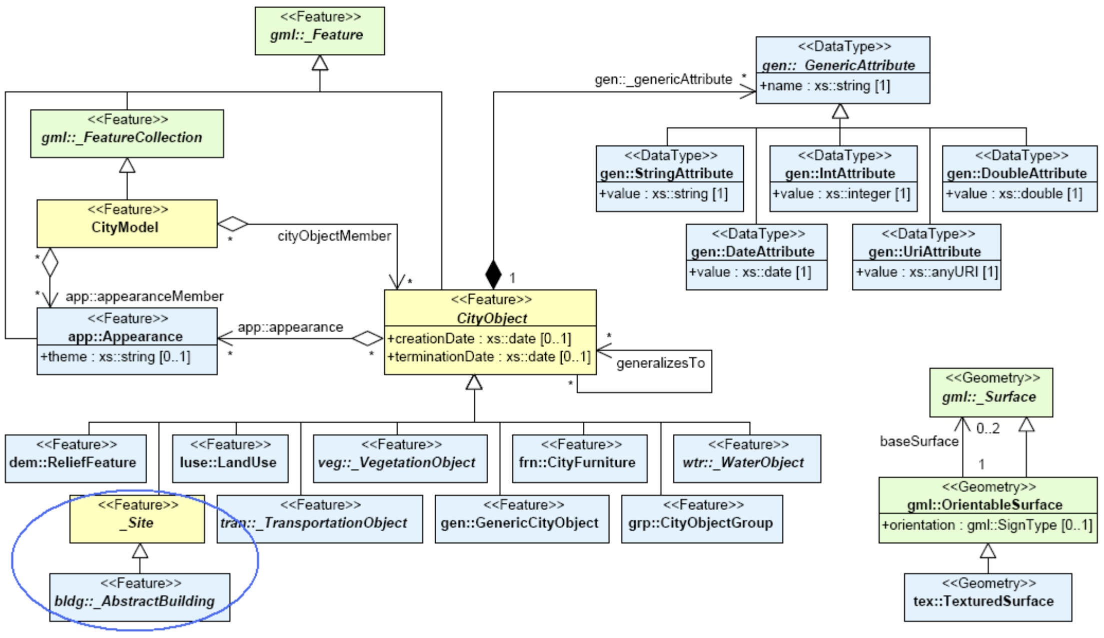
**Figure 2.3**: *Various modules in CityGML and their schema dependencies shown as a UML package diagram (source:[3]).*

The CityGML structure is divided into modules. It consists of class definitions for the important types of objects that can be found in a city. Only the classes that are identified to be relevant or required in many different application areas are used. The CityGML is decomposed into a *core module* and *thematic extension modules*. The core module consists of the basic concepts and components of the CityGML data model. Each extension is based on the core module, and covers a specific area of a virtual 3D city model. All extension modules are independent from each other, but has a dependency over the CityGML core module. There are eleven thematic extension modules that are currently used in CityGML. They are the following: *Appearance, Building, CityFurniture, CityObjectGroup, Generics, LandUse, Relief, Transportation, Vegetation, Water-Body,* and *TexturedSurface*. Figure 2.3 gives an idea on the various modules under CityGML. In the figure, we see that each module is represented by a package. Each extension module (indicated by the leaf packages) imports the core module. The core module also imports XSD (XML Schema Definition) schemas such as GML and extensible Address Language.

## 2.2 Models in CityGML

CityGML covers properties regarding geometrical, topological, appearance and sematical aspects for each object. Although it is necessary to study the models of each property, detailed description on each model is beyond the scope of this project. A description on each model can be found in [3]. The following section discusses only the model that is required for the study of extension of CityGML for cadastral purposes.

### 2.2.1 Thematic model



<<Feature>> gml::_Feature
<<Feature>> gml::_FeatureCollection
<<Feature>> CityModel
cityObjectMember
app::appearanceMember
<<Feature>> app::Appearance +theme : xs::string [0..1]
app::appearance
<<Feature>> CityObject +creationDate : xs::date [0..1] +terminationDate : xs::date [0..1]
generalizesTo
gen::_genericAttribute
<<DataType>> gen::_GenericAttribute +name : xs::string [1]
<<DataType>> gen::StringAttribute +value : xs::string [1]
<<DataType>> gen::IntAttribute +value : xs::integer [1]
<<DataType>> gen::DoubleAttribute +value : xs::double [1]
<<DataType>> gen::DateAttribute +value : xs::date [1]
<<DataType>> gen::UriAttribute +value : xs::anyURI [1]
<<Feature>> dem::ReliefFeature
<<Feature>> luse::LandUse
<<Feature>> veg::_VegetationObject
<<Feature>> frn::CityFurniture
<<Feature>> wtr::_WaterObject
<<Feature>> _Site
<<Feature>> tran::_TransportationObject
<<Feature>> gen::GenericCityObject
<<Feature>> grp::CityObjectGroup
<<Feature>> bldg::_AbstractBuilding
<<Geometry>> gml::_Surface
baseSurface
0..2
1
<<Geometry>> gml::OrientableSurface +orientation : gml::SignType [0..1]
<<Geometry>> tex::TexturedSurface

**Figure 2.4**: *UML diagram of CityGML's core module (source:[3]).*

The thematic model is defined based on the modularisation of classes. It consists of a core module. All extension modules are based on and dependent on the CityGML core module. Based on the application domain, CityGML extension modules may be combined with the core module.

The UML diagram of the top level class hierarchy of the thematic model in CityGML is presented in Figure 2.4. As seen in the figure, _CityObject is the core class. The core class is an abstract base class (a parent class , or a class that cannot be instantiated) from which the extension modules are derived. The subclasses of _CityObject comprise the different thematic fields of a city model, which are covered by separate CityGML extension modules. Since one of the aims of the project is to add legal properties to buildings or apartments, we restrict our study to *building module* which is a sub-class of one of the eleven thematic extension modules, _Site. The encircled area in the figure highlights on the model that is interesting for this thesis.

### 2.2.2 Building model

The building model is the most important and the most detailed thematic concept of CityGML. Thematic and spatial aspects of buildings, building parts and installations can be represented in four levels of detail, LOD1 to LOD4 (LOD0 is too coarse to model a building). The UML diagram of the building model is shown in Figure 2.9, at the end of the chapter. The main class of the model is _AbstractBuilding, which is a subclass of the thematic class _Site. _AbstractBuilding is specialized to either a Building or a BuildingPart. The _AbstractBuilding inherits all properties from _CityObject since it is a subclass of the root class _CityObject. An example for a CityGML schema file is given in *Appendix A* for _AbstractBuildingType.

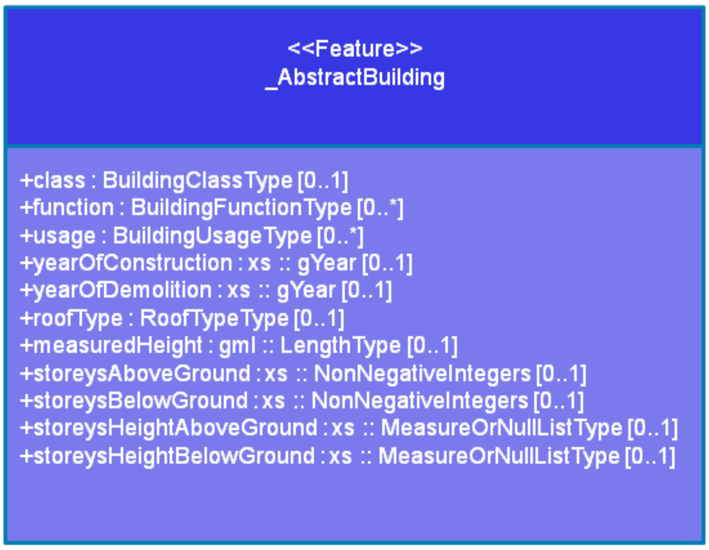

<<Feature>>
_AbstractBuilding
+class : BuildingClassType [0..1]
+function : BuildingFunctionType [0..*]
+usage : BuildingUsageType [0..*]
+yearOfConstruction : xs :: gYear [0..1]
+yearOfDemolition : xs :: gYear [0..1]
+roofType : RoofTypeType [0..1]
+measuredHeight : gml :: LengthType [0..1]
+storeysAboveGround : xs :: NonNegativeIntegers [0..1]
+storeysBelowGround : xs :: NonNegativeIntegers [0..1]
+storeysHeightAboveGround : xs :: MeasureOrNullListType [0..1]
+storeysHeightBelowGround : xs :: MeasureOrNullListType [0..1]

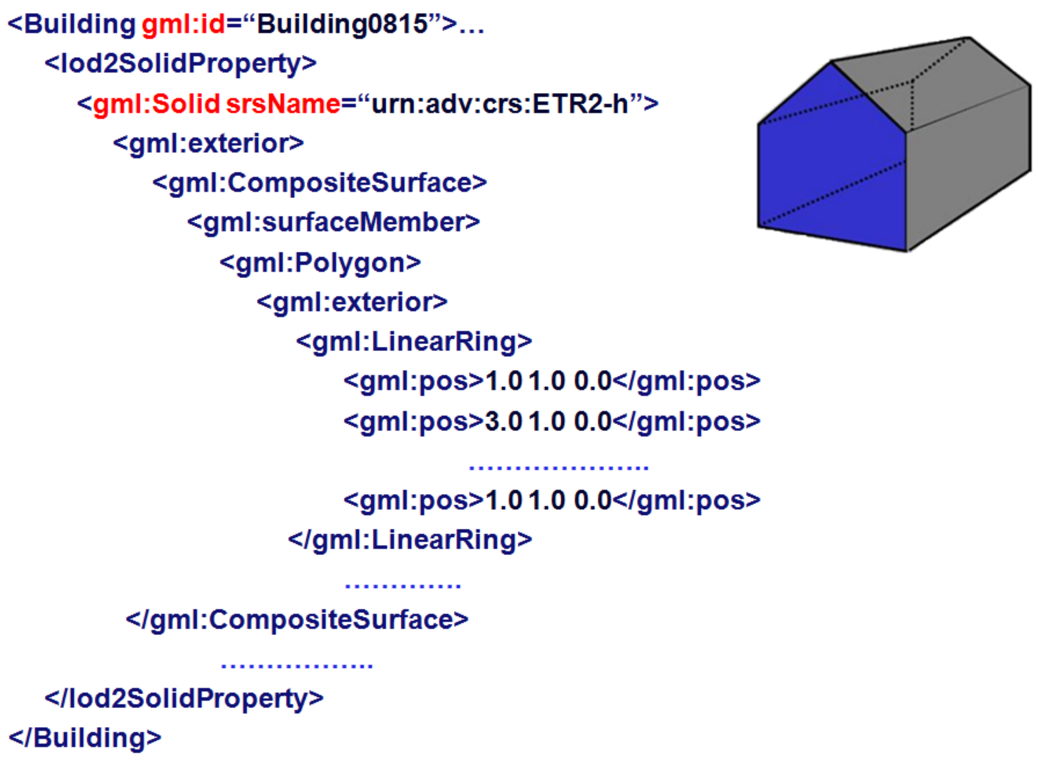
**Figure 2.5**: *Abstract Building class and its attributes.*

Among the eleven thematic extension modules, the Building model is studied extensively since this model has more relevance to the project in hand. The idea is to use the Building class of the CityGML to generate a 3D city model, to visualize the buildings with cadastral information. Figure 2.5 shows the attributes present in the _AbstractBuilding. Hence a building may consist of all or some of the properties given as its attributes.

## 2.3 Structure of Building in CityGML

When building a 3D model, each tag in the code has a special significance. The root tag for any 3D city model is *CityModel*. This means that every model built in CityGML will have this root tag as the base tag. Each object in CityGML is a *cityObjectMember*. That is, a cityObjectMember could be a building, road, etc., which are the objects found in a city. If a building is modelled, then the tag *bldg:Building* is used to represent the building structure. A building is created from many surfaces, which are made of polygons. Hence, each surface in a building should have a corresponding *surfaceMember*. Figures 2.6 and 2.7 give an example that shows the structure of CityGML to model a simple building.

The most basic element in CityGML is the *gml:pos*. It contains the positional information of each vertex or corner of a surface. Any given surface will have a set of corners. The positional values of the corners with respect to the 3-Dimensional co-ordinates are stored in this tag. Figure 2.6 shows an example which maps the position of the end points of the front surface of the building. The values 1.0 1.0 and 0.0 are the x, y and z co-ordinate values of one of the end points of the front surface. Instead of having a *gml:pos* for a single end point, a different tag called *gml:posList* could be used. A *gml:posList* is a list of the positions of all corners of a surface. Similarly several surfaces can be represented. Figure 2.7 shows the front surface, back surface, roof etc, each being represented by a surfaceMember.



```
<Building gml:id="Building0815">...
   <lod2SolidProperty>
      <gml:Solid srsName="urn:adv:crs:ETR2-h">
         <gml:exterior>
            <gml:CompositeSurface>
               <gml:surfaceMember>
                  <gml:Polygon>
                     <gml:exterior>
                        <gml:LinearRing>
                           <gml:pos>1.0 1.0 0.0</gml:pos>
                           <gml:pos>3.0 1.0 0.0</gml:pos>
                                  ....................
                           <gml:pos>1.0 1.0 0.0</gml:pos>
                        </gml:LinearRing>
                           .............
          </gml:CompositeSurface>
                  .................
   </lod2SolidProperty>
</Building>
```

**Figure 2.6**: *Structure of Building in CityGML 1.*

```
<Building gml:id="Building0815">...
   <lod2SolidProperty>
      <gml:Solid srsName="urn:adv:crs:ETR2-h">
         <gml:exterior>
            <gml:CompositeSurface>
               <gml:surfaceMember>
                        //front surface
               </gml:surfaceMember>
               <gml:surfaceMember>
                        //side surface
               </gml:surfaceMember>
                        //here comes side, back, roof and ground surfaces
            </gml:CompositeSurface>
         </gml:exterior>
       </gml:Solid>
      </lod2SolidProperty>
   </Building>
```

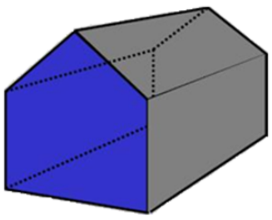


**Figure 2.7**: *Structure of Building in CityGML 2.*

## 2.4 A case study: a building in LOD1

A building in LOD1 is modelled as a block. It is not very detailed and has no definite roof structure. This section gives an example of a single building modelled in LOD1. This example gives an idea of the overall structure used while modeling a building. LOD1 is used to model the buildings throughout the thesis and also used to present the extended properties (explained in the next chapter). The code of the modelled building is given in *Appendix B*. The visualization of the building for the given code in CityGML is shown in Figure 2.8. Similarly, several buildings with different shapes and sizes at various positions can be used to model a city in LOD1.

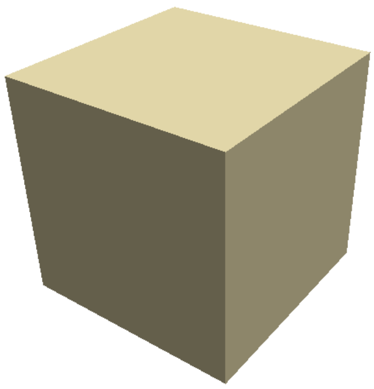


**Figure 2.8**: *LOD1 building in CityGML.*

## 2.5 Survey on CityGML viewers

As a part of the survey that was conducted in the initial phase of the project, several available CityGML viewers [1, 2] were studied and analyzed. These viewers are used to visualize the city models. Most of the viewers that were analyzed were either open source or free products. Among them, two products are used in this thesis to visualize the modelled buildings and to view its properties (and extended properties, explained in the next chapter). A brief description on each viewer is presented here.

### 2.5.1 LandXplorer

LandXplorer is a commercial tool, a product by Autodesk. It is one of the most popular tool available in the market. It can be used to edit, explore and load 3D city models. But for the freely available product, the edit and explore options are disabled. It is easy to use and has a good navigation system. It also provides facilities to view the properties of the objects (components) of the models.

### 2.5.2 Aristotele

Aristotele is an open source software developed by the University of Bonn. It provides facilities to select components and also to view and edit (modify) the properties. There are facilities to add plug-ins. The navigation system provided by Aristotele is not very user friendly.

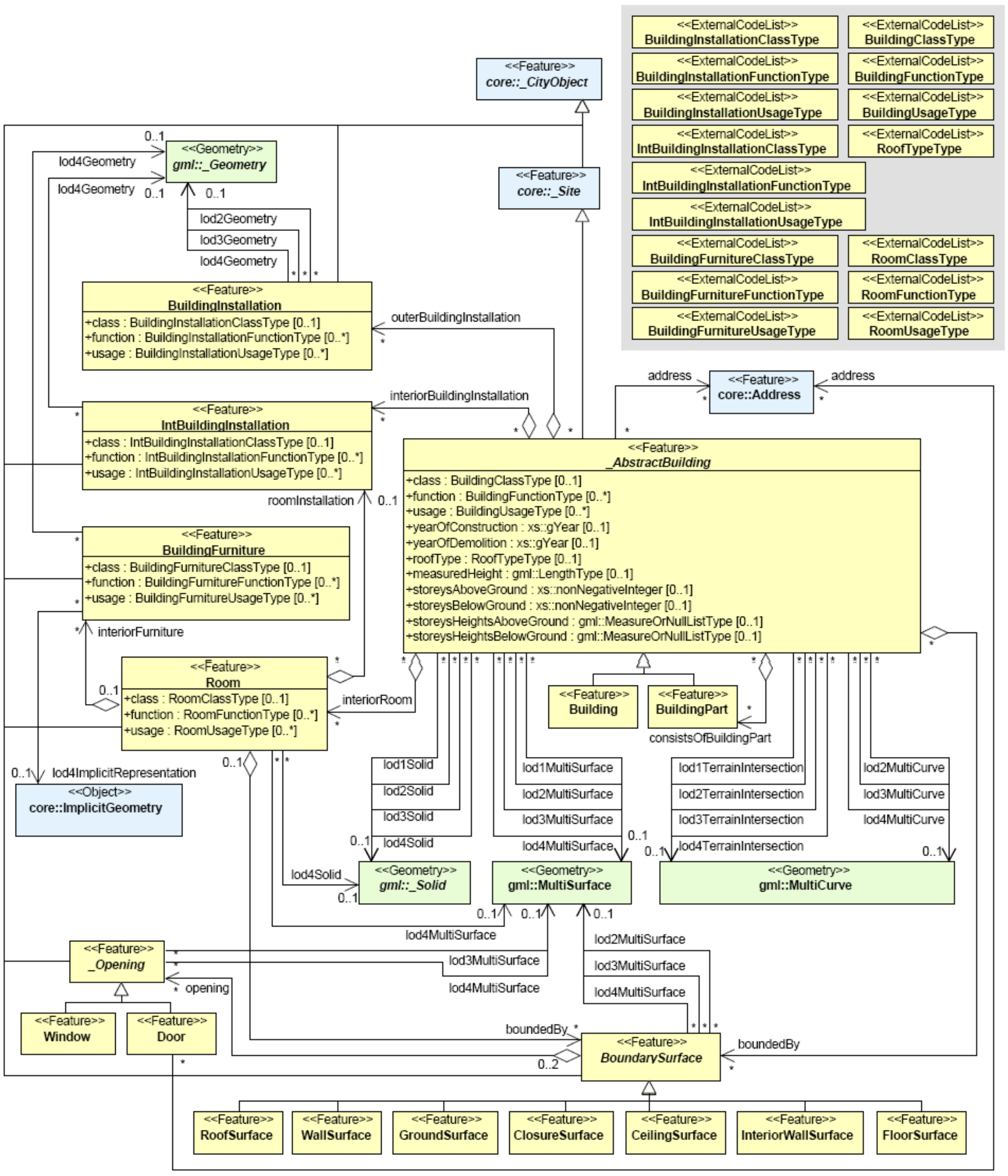


**Figure 2.9**: *UML diagram of CityGML's building model (source:[3]).*

Chapter 3

# An extension to CityGML

CityGML, with all its powerful features is a suitable format to build virtual 3D city model. It covers most of the important types of object within a city. It also has many attributes to model a building. But until this day there are no facilities or attributes in CityGML that describe legal information about the building. This means that most of the information on cadastral issues cannot be modelled into a building class. The Kadaster requires some means to be able to add additional information while modelling a building, so that it can use CityGML to visualize ownership rights that belong to a building. Just like adding legal information into building for cadastral purpose, there will also be cases in other practical applications, where the attributes defined for the objects modelled in CityGML will not be sufficient. Or there might be 3D objects which are not covered by the thematic classes of CityGML. In order to add such features, attributes and classes, there are facilities provided to extend CityGML.

This chapter concentrates on extending the CityGML for cadastral purposes. It provides details on the two methods that are available in CityGML for extension. Then an extension is proposed for cadastral purposes. This extension tries to incorporate all possible legal data into the existing building model and also proposes another class to represent apartments in the building. Later, the overall design structure of the building model of CityGML with this extension is provided, followed by an example of a building modelled with this extension.

## 3.1 Methods of extension of CityGML

CityGML provides two methods that can be used to extend CityGML.

- Generic city objects and attributes
- Application Domain Extensions

### 3.1.1 Generic city objects and attributes

The concept of generic objects and attributes allows for the extension of CityGML applications during runtime. This allows any CityObject to have additional attributes, whose names, data types, and values can be provided by a running application without any change of the CityGML XML schema. By using generic objects, features that are not represented by the predefined classes of the CityGML data model may be modelled and exchanged.

Generic city objects and attributes have some disadvantages.

- There is only a limited number of predefined data types like string, integer, double, date and URI that can be used for generic attributes.

- An XML parser cannot validate the layout and occurrence of generic objects and attributes . This may reduce semantic interoperability. For example, if two generic objects have the same variables, then they cannot be validated.
- Naming conflicts of generic attributes or objects can occur.

### 3.1.2 Application Domain Extensions

*Application Domain Extensions (ADE)* specify additions such that new properties can be introduced to existing CityGML classes. It can also be used to define new object types. For example, adding an attribute *parcel number* to a building. The difference between ADEs and generic objects and attributes is, that an ADE has to be defined in an extra XML schema definition file with its own namespace. The XML Schema definition of the extended CityGML modules have to be explicitly imported into this file.

ADEs have advantages over generic objects and attributes.

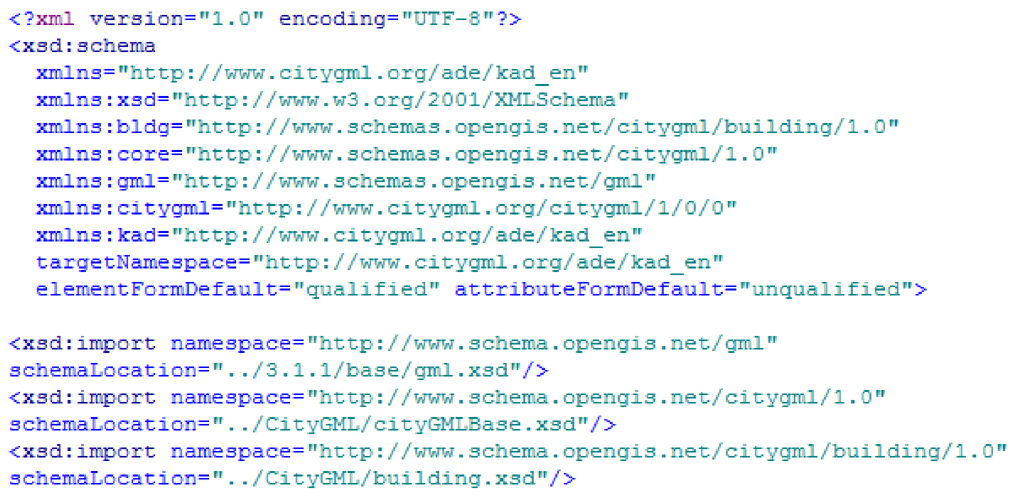
- ADEs can be formally specified.
- Extended CityGML instance documents can be validated against the CityGML and its ADE schema.
- More than one ADE can be used in the same dataset.
- ADEs may be defined for one or more CityGML modules which provides a high flexibility.
- ADEs can be defined and standardised by communities that are interested in specific application domains.

## 3.2 Extension for cadastral purposes

To have legal information embedded into the building class, the CityGML has to be extended. For this, one of the two methods have to be used. Because of the clear advantages of Application Domain Extensions (ADE), this method is used to extend CityGML. Now the question that arises is, "what are the properties or attributes that could be added in order to have all legal details into this class"? It is necessary to identify all the properties that have to be added. Figure 2.5 in Section 2.2.2 of the previous chapter shows the properties that are predefined in the Building class. Some of the properties that are needed to describe legal information are identified to be building owners, the building registration number or the building number, the parcel number of the parcel to which the building belongs to, and the type of the building. These properties are added to _AbstractBuilding class. This thesis also proposes a new class, that is specifically related to apartments, called _KadasterApartment. Before the development of the ADE to add all the identified properties and the new class, it is necessary to learn the header of the XML schema.

### 3.2.1 XML schema header

In this thesis, based on the initial survey conducted, a few attributes are identified to be missing or necessary in the building class that could be added to store legal information. ADEs are basically specific XML schema files (XSD - XML Schema Definition) containing application specific XML element or attribute definitions. A XSD specifies how to formally describe the elements in an XML document. With the help of the XML schema, the hierarchical structure of an XML document is defined unambiguously and can be read by tools. These tools are used to validate the XML instances against the defined XML schema.



```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  xmlns="http://www.citygml.org/ade/kad_en"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:bldg="http://www.schemas.opengis.net/citygml/building/1.0"
  xmlns:core="http://www.schemas.opengis.net/citygml/1.0"
  xmlns:gml="http://www.schemas.opengis.net/gml"
  xmlns:citygml="http://www.citygml.org/citygml/1/0/0"
  xmlns:kad="http://www.citygml.org/ade/kad_en"
  targetNamespace="http://www.citygml.org/ade/kad_en"
  elementFormDefault="qualified" attributeFormDefault="unqualified">

<xsd:import namespace="http://www.schema.opengis.net/gml"
schemaLocation="../3.1.1/base/gml.xsd"/>
<xsd:import namespace="http://www.schema.opengis.net/citygml/1.0"
schemaLocation="../CityGML/cityGMLBase.xsd"/>
<xsd:import namespace="http://www.schema.opengis.net/citygml/building/1.0"
schemaLocation="../CityGML/building.xsd"/>
```

**Figure 3.1**: *Root tag of an XML schema file.*

Figure 3.1 shows the root tag of an XML schema. For each newly created application schema, a *namespace* must be declared. Thereafter, using this namespace, all the created objects can be associated to the ADE. The root tag of this file is $< schema >$ and the definition of the namespace is stored within the attribute *targetNamespace* as shown in the figure. Therefore any application can identify the location of the schema by reading from this object, which makes sure that all objects are distinct even if the same object name was used before. More details on extension with an example of an ADE is given in [6]. The ADE schemas, whose objects are referenced or extended within an ADE, have to be imported using $< xsd : import >$, as shown in the figure.

### 3.2.2 Extension of the class _AbstractBuilding

Figure 3.2 shows some of the attributes that are identified to be necessary in order to add legal information to a building. These attributes store details about the various legal aspects found in a building. Most of the attributes use predefined data types. But based on the specific application, user defined data types can be used. As a part of this extension, an application specific data type is defined called the *BuildingType* which tells about the purpose of the building, i.e. if it is a commercial building, a private building, etc. The external code list for defining BuildingType is given in Appendix C. The building type could be one of the following values: private, commercial, residential, others, none. An external code list, as explained in Section 2.1.3 is used to define the discrete values that a BuildingType can possess. Appendix D shows the structure of the ADE that is designed to add the additional properties to the class _AbstractBuilding.

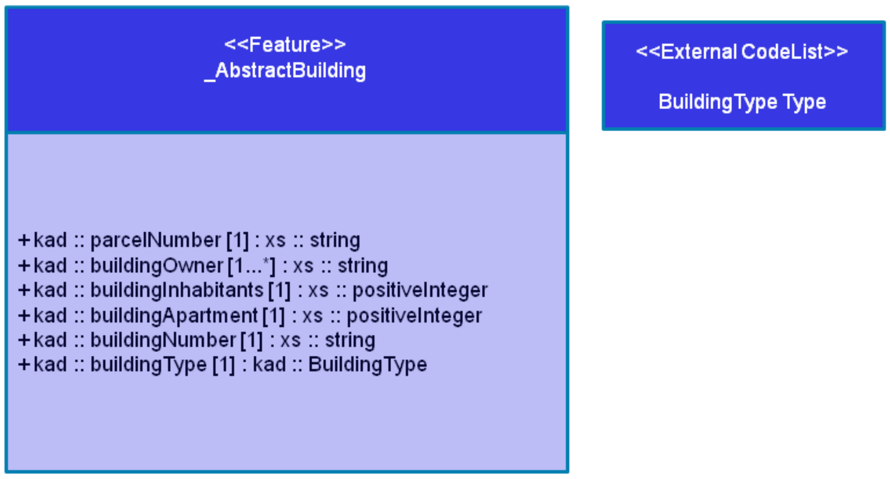

<<Feature>>
_AbstractBuilding
+kad :: parcelNumber [1] : xs :: string
+kad :: buildingOwner [1...*] : xs :: string
+kad :: buildingInhabitants [1] : xs :: positiveInteger
+kad :: buildingApartment [1] : xs :: positiveInteger
+kad :: buildingNumber [1] : xs :: string
+kad :: buildingType [1] : kad :: BuildingType
<<External CodeList>>
BuildingType Type

**Figure 3.2**: *Identified properties to extend the class _AbstractBuilding.*

### 3.2.3 New class definition

The structure of building model in CityGML has an abstract class called _AbstractBuilding. A building may have other objects along with it. Some of the common objects seen are as follows: building furniture such as mail box, building installations such as water pipes, interior building installations such as mirrors, etc. A building may even be sub divided into rooms, but CityGML has no method to identify an apartment in a building. An apartment may consist of many rooms and a building may or may not have many apartments.

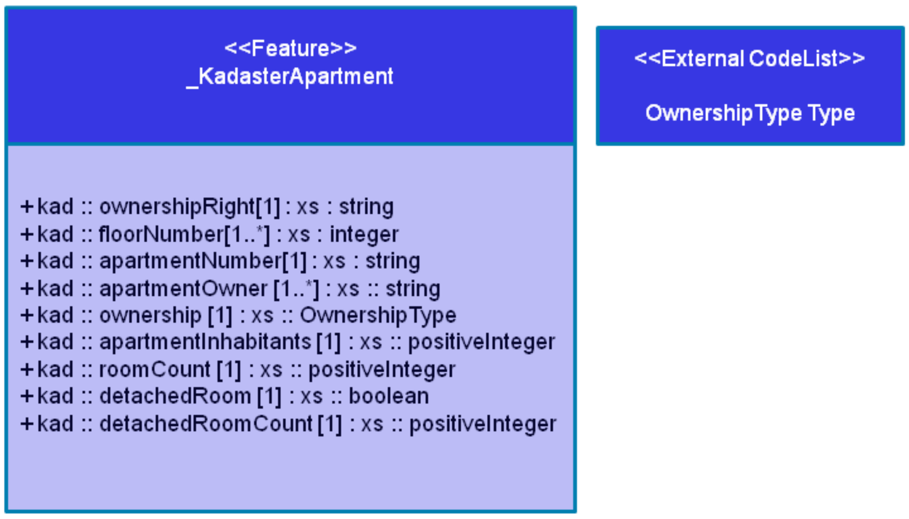

<<Feature>>
_KadasterApartment
+kad :: ownershipRight[1] : xs : string
+kad :: floorNumber[1..*] : xs : integer
+kad :: apartmentNumber[1] : xs : string
+kad :: apartmentOwner [1..*] : xs :: string
+kad :: ownership [1] : xs :: OwnershipType
+kad :: apartmentInhabitants [1] : xs :: positiveInteger
+kad :: roomCount [1] : xs :: positiveInteger
+kad :: detachedRoom [1] : xs :: boolean
+kad :: detachedRoomCount [1] : xs :: positiveInteger
<<External CodeList>>
OwnershipType Type

**Figure 3.3**: *Identified new class KadasterApartment with its properties.*

In this thesis, a design of a new class called *KadasterApartment* is proposed in order to store legal information on ownership rights. Figure 3.3 shows the attributes that are defined for this new class. Some of the properties of the class identify the ownership right, the floor number of the apartment, the apartment number, the owner of the apartment, the ownership type, the number of inhabitants in the apartment, the room count. It also has provisions to identify if the apartment also has detached rooms with the same ownership rights. This is necessary to know that there are other regions such as a garage or a storage along with the apartment, that belong to the same person. Appendix E shows the structure of the ADE that is designed to add the properties to the new class KadasterApartment. The *OwnershipType* another data type that is defined to suit this application. The OwnershipType tells about the ownership type of the apartment, i.e. if it is an apartment on partnership, or a private apartment, etc. Other identified properties can easily be added into the ADE in a similar manner. The external code list for defining OwnershipType is given in Appendix F.

## 3.3 The overall design structure

The new class KadasterApartment and the legal properties that are added to _AbstractBuilding class have to be integrated to the CityGML Building model. Figure 3.4 shows the simple version of the building model before the extension. After the integration of the new class and the properties, by importing the designed ADE, the building model looks as shown in Figure 3.5. The KadasterApartment class becomes the subclass of _AbstractBuilding.

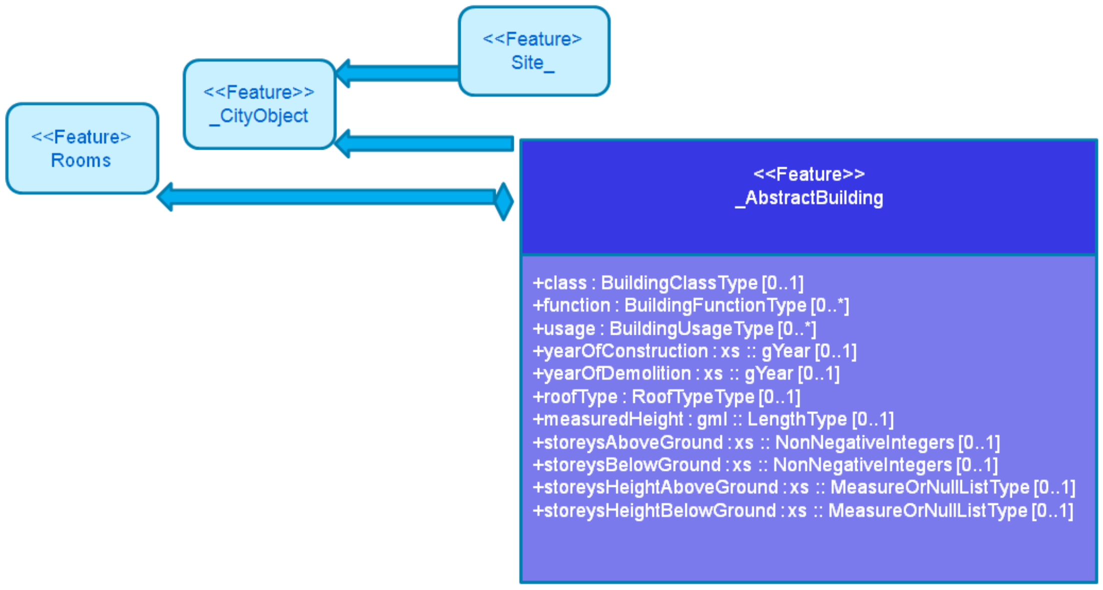

<<Feature>
Site_
<<Feature>>
_CityObject
<<Feature>
Rooms
<<Feature>>
_AbstractBuilding
+class : BuildingClassType [0..1]
+function : BuildingFunctionType [0..*]
+usage : BuildingUsageType [0..*]
+yearOfConstruction : xs :: gYear [0..1]
+yearOfDemolition : xs :: gYear [0..1]
+roofType : RoofTypeType [0..1]
+measuredHeight : gml :: LengthType [0..1]
+storeysAboveGround : xs :: NonNegativeIntegers [0..1]
+storeysBelowGround : xs :: NonNegativeIntegers [0..1]
+storeysHeightAboveGround : xs :: MeasureOrNullListType [0..1]
+storeysHeightBelowGround : xs :: MeasureOrNullListType [0..1]

**Figure 3.4**: *The CityGML Building model before the extension.*

## 3.4 ADE modelled onto a building

The new extended CityGML model with features to represent legal information can be used to model buildings. By calling the ADE by referring to its namespace in the root tag, there legal information of the building can be added and can also be visualized by a CityGML viewer. These viewers provide facilities to navigate through the model or zoom towards one of the objects in the model. An object from a building model or a city model can also be selected and its properties

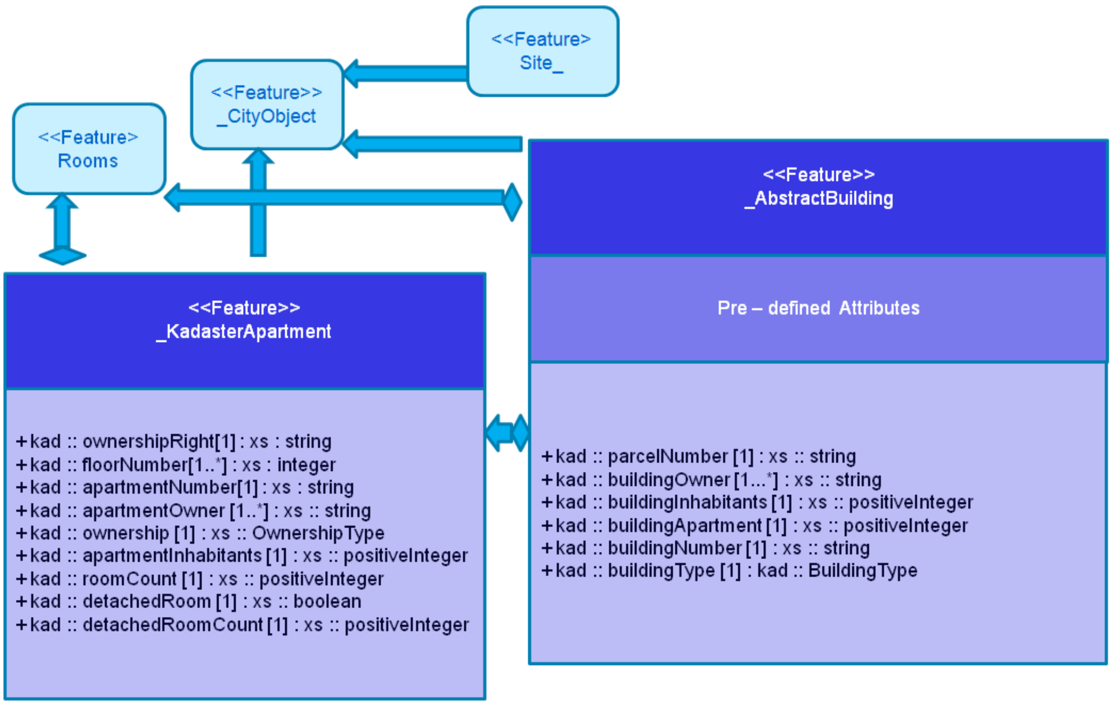
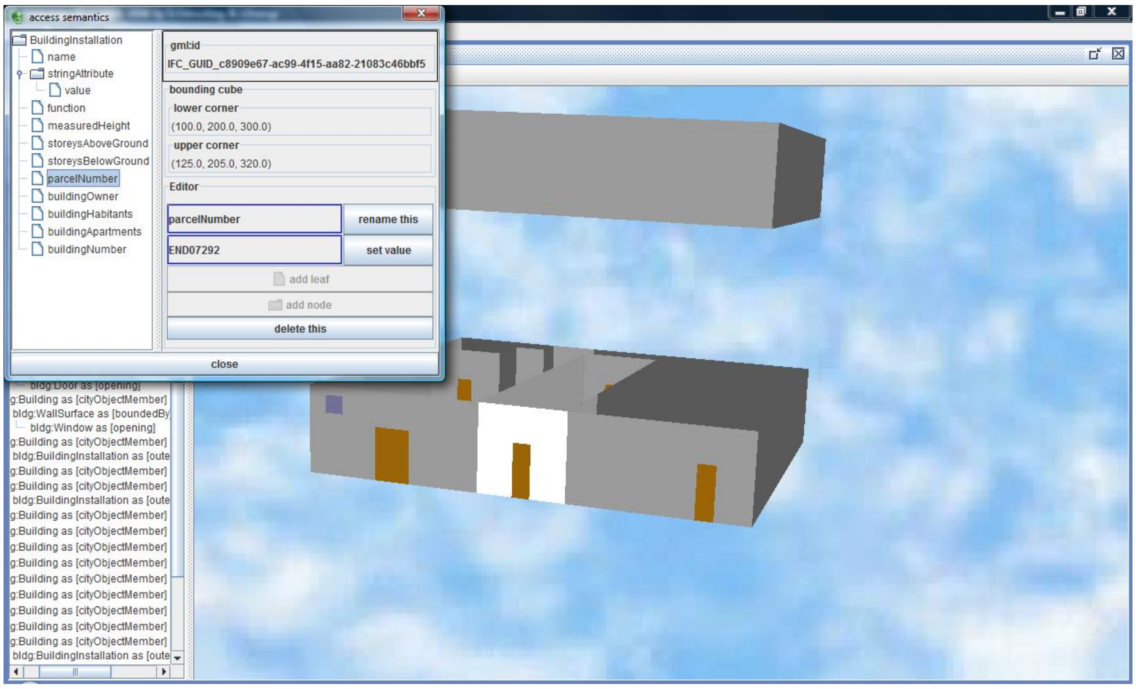

<<Feature>
Site_
<<Feature>>
_CityObject
<<Feature>
Rooms
<<Feature>>
_AbstractBuilding
Pre – defined Attributes
<<Feature>>
_KadasterApartment
+kad :: ownershipRight[1] : xs : string
+kad :: floorNumber[1..*] : xs : integer
+kad :: apartmentNumber[1] : xs : string
+kad :: apartmentOwner [1..*] : xs :: string
+kad :: ownership [1] : xs :: OwnershipType
+kad :: apartmentInhabitants [1] : xs :: positiveInteger
+kad :: roomCount [1] : xs :: positiveInteger
+kad :: detachedRoom [1] : xs :: boolean
+kad :: detachedRoomCount [1] : xs :: positiveInteger
+kad :: parcelNumber [1] : xs :: string
+kad :: buildingOwner [1...*] : xs :: string
+kad :: buildingInhabitants [1] : xs :: positiveInteger
+kad :: buildingApartment [1] : xs :: positiveInteger
+kad :: buildingNumber [1] : xs :: string
+kad :: buildingType [1] : kad :: BuildingType

**Figure 3.5**: *The new design of CityGML Building model with legal properties.*

can be viewed. Figure 3.6 provides an example of a building with legal information associated to it. Figures 3.7 and 3.8 give a clear picture of the extended properties of _AbstractBuilding class and the new class KadasterApartment respectively shown in one of the viewers.

The next chapter provides details on the process used to automate the representation of the extracted information on legal issues from scanned images of floor plans in extended CityGML.


access semantics

BuildingInstallation
- name
- stringAttribute
  - value
- function
- measuredHeight
- storeysAboveGround
- storeysBelowGround
- parcelNumber
- buildingOwner
- buildingHabitants
- buildingApartments
- buildingNumber

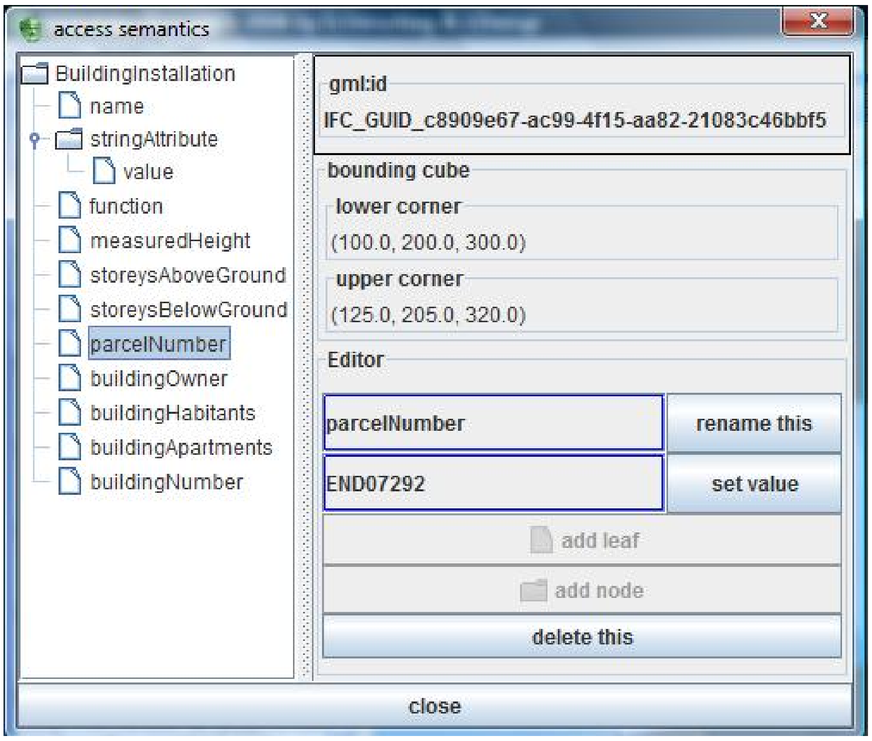
gml:id
IFC_GUID_c8909e67-ac99-4f15-aa82-21083c46bbf5

bounding cube
lower corner
(100.0, 200.0, 300.0)
upper corner
(125.0, 205.0, 320.0)

Editor
parcelNumber | rename this
END07292 | set value
add leaf
add node
delete this
close

```
bldg:Door as [opening]
g:Building as [cityObjectMember]
bldg:WallSurface as [boundedBy
 bldg:Window as [opening]
g:Building as [cityObjectMember]
bldg:BuildingInstallation as [oute
g:Building as [cityObjectMember]
g:Building as [cityObjectMember]
bldg:BuildingInstallation as [oute
g:Building as [cityObjectMember]
g:Building as [cityObjectMember]
g:Building as [cityObjectMember]
g:Building as [cityObjectMember]
g:Building as [cityObjectMember]
g:Building as [cityObjectMember]
g:Building as [cityObjectMember]
g:Building as [cityObjectMember]
g:Building as [cityObjectMember]
bldg:BuildingInstallation as [oute
```

**Figure 3.6**: *A building with legal properties.*


access semantics

BuildingInstallation
- name
- stringAttribute
  - value
- function
- measuredHeight
- storeysAboveGround
- storeysBelowGround
- parcelNumber
- buildingOwner
- buildingHabitants
- buildingApartments
- buildingNumber

gml:id
IFC_GUID_c8909e67-ac99-4f15-aa82-21083c46bbf5

bounding cube
lower corner
(100.0, 200.0, 300.0)
upper corner
(125.0, 205.0, 320.0)

Editor
parcelNumber | rename this
END07292 | set value
add leaf
add node
delete this
close

**Figure 3.7**: *Extended properties of _AbstractBuilding.*

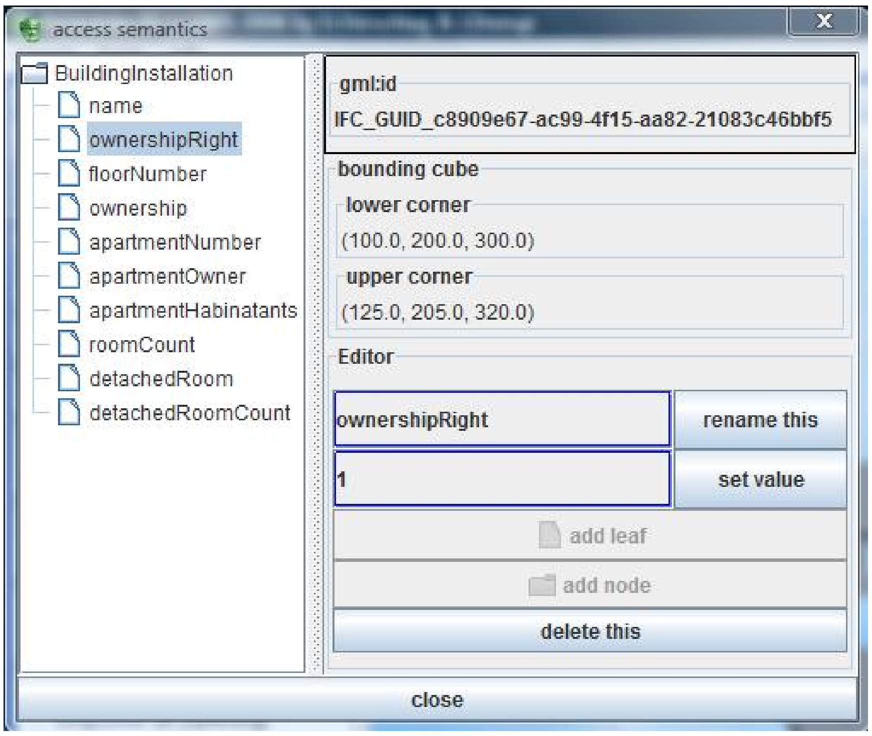


**Figure 3.8**: *Attributes of the class KadasterApartment.*

# Chapter 4
# Representation of extracted information in CityGML

The previous chapter presented an extension of CityGML for cadastral purposes. As mentioned in Chapter 1, the scanned images of the floor plans of the buildings have legal information. This information can be extracted from the floor plans [4]. This chapter highlights the process that has been developed to automatically represent the extracted data from the floor plans in extended CityGML for cadastral purposes.

This process needs an input file with legal information in it. In this case, it is obtained by extracting the information from scanned images. This file is taken and processed, in order to get an output file in an extended CityGML format. This output file can be viewed using any CityGML viewer. Figure 4.1 shows the overall process that takes place to achieve this.

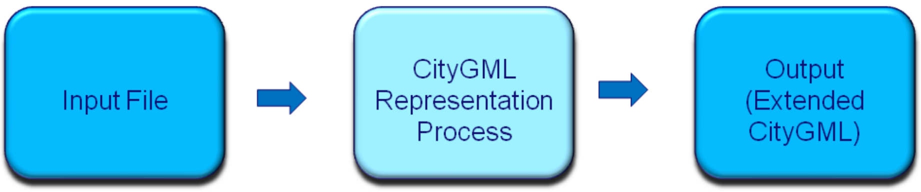


**Figure 4.1**: *Overview of CityGML representation process.*

## 4.1 Outline on the input file

Before focusing the attention on the process of representing the data in CityGML, it is helpful to gain knowledge on the input file. This section gives a brief detail on the structure of the input file and also on the process that generates this file.

As explained in Chapter 1, the details about the legal rights of a parcel or a building are stored in a scanned format. The data from these scanned images cannot be queried since a computer cannot automatically interpret the information. Therefore any information from such scanned formats has to be extracted and stored in a form such that a computer can read it. The process of extracting the information from scanned images of the floor plans of the buildings takes the floor plans as the input file. Since Kadaster is only interested in the ownership rights and ownership boundaries of the buildings, other minor details such as text and thin lines can be eliminated. Only the image with the ownership boundaries and ownership rights associated to them is considered. This stage is reached by using the techniques of image processing. By converting the image into a graph, details about the scanned image can be stored in a file in the form of a simple data structure as shown in Figure 4.2.

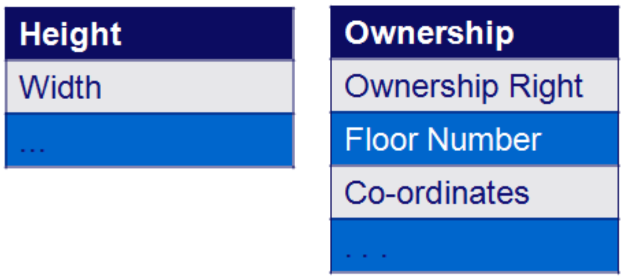


**Figure 4.2:** *Extracted information from floor plans.*

The output file that is generated by extracting the information from the scanned images is the input file for representing the extracted information in extended CityGML. The *height-width* defines the overall height and width of the scanned image of the floor plan that is taken into consideration. Based on this, we can further compute the coordinates with respect to x, y, or z axis for each region. Each given region has a set of attributes associated with it. It contains the *ownership rights*, the *floor number* and the *coordinates* of the ownership boundary or the region. The ownership rights determines the owner of a given region. The floor number tells about the floor to which the given region belongs to. The coordinates of the ownership boundary are the position of the pixel in the image (pixel coordinates) of each corner of the region. This means that the coordinates given are not with respect to x, y and z axis. This is calculated for each corner of the region, by taking its pixel coordinates and the height-width value from the scanned image. Figure 4.3 shows two regions with each having an ownership right added to them. And the dots indicate the corners of each region. The inner region has an ownership right value of ”1”, and the outer region has an ownership right value of ”2”.

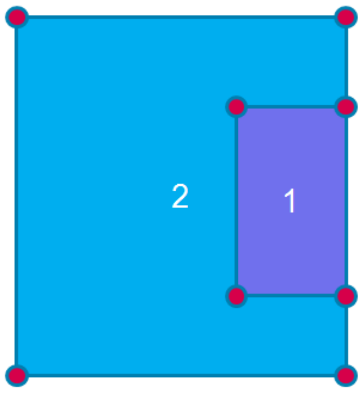


**Figure 4.3:** *Region with Ownership Rights.*

## 4.2 Steps performed to represent the extracted information in extended CityGML

There are various steps involved to convert the input file into CityGML format. Figure 4.4 shows the various processes that take place in achieving this goal. Each step has a special significance and can be easily distinguished in the process. Most of the steps are interdependent on each other and require data from the previous steps. The steps are briefly explained below.

### 4.2.1 Separation of objects into ownership rights, floor number and regions

The data is taken in a single file which contains information regarding ownership rights, floor number and coordinates of each region. These various aspects have to be distinguishable. Therefore all ownership rights, floor numbers and the coordinates of each region are grouped. The result of this step is used throughout the process. Most of the decisions made later in the process are based on either the ownership rights of the region or the floor number of the region. Therefore it is necessary to group them in the initial step.

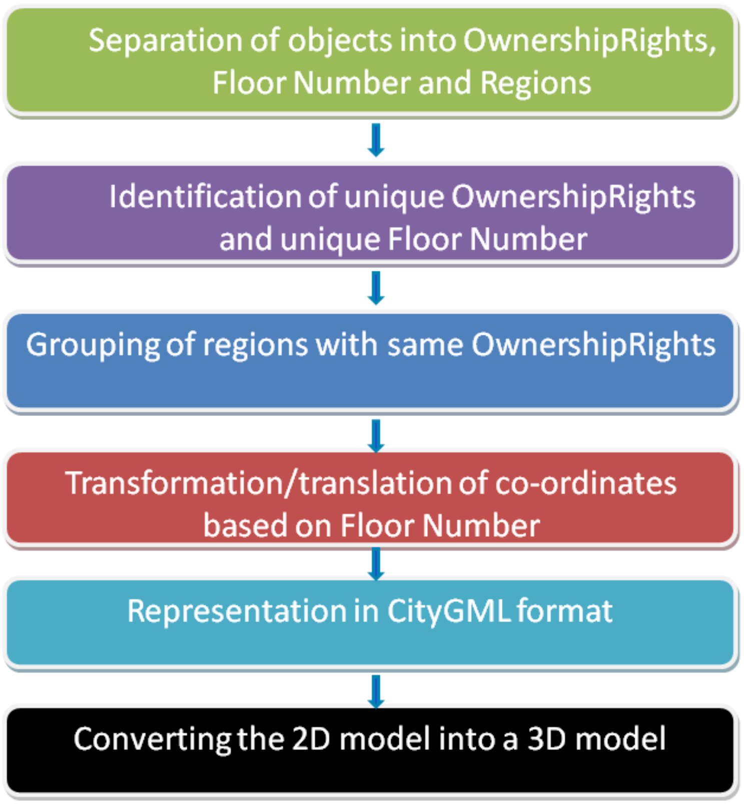

Separation of objects into OwnershipRights, Floor Number and Regions
Identification of unique OwnershipRights and unique Floor Number
Grouping of regions with same OwnershipRights
Transformation/translation of co-ordinates based on Floor Number
Representation in CityGML format
Converting the 2D model into a 3D model

**Figure 4.4**: *Steps performed to represent the extracted information in extended CityGML.*

### 4.2.2 Identification of unique ownership rights and unique floor number

As explained in the beginning of this chapter, during the process of extracting the information from the scanned images, in a given floor, only the regions with the same ownership rights are interesting cases. Therefore, the sub-region information is discarded. Consider two cases as shown in Figure 4.5. Case one, where a floor in a building is owned by two different people. This means that there will be two regions having the same floor number, but different ownership rights. Case two, where a single owner owns regions in different floors. So, there will also be regions with different floor numbers having the same ownership right.

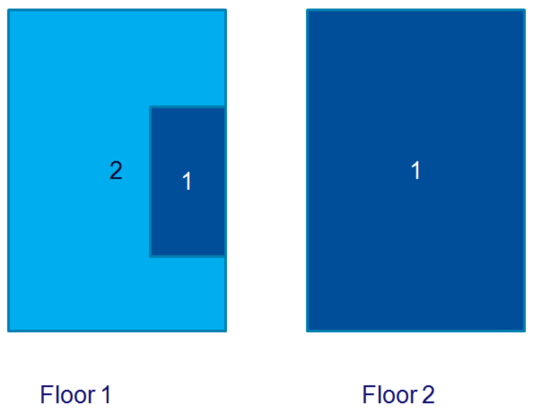


**Figure 4.5**: *A floor plan of a building with two floors and two ownership rights.*

### 4.2.3 Grouping of regions with same ownership rights

A person may own property in different areas. For example, the garage may not be attached to the house or it may not even be on the same parcel. This means that several regions can have the same ownership rights. The desire of Kadaster is to be able to view all the regions owned by a single owner at a time. That is, all the regions with the same ownership rights must be highlighted/visualized simultaneously. The objective behind having this step, is to find a method to be able to perform this task.

In the initial phase of the project, a survey was carried out in order to study various CityGML viewers. Several viewers were analyzed based on their usability, output structure, existing properties, the cost, etc. All the viewers provided a facility to select an object in the city model, and some of them even allowed the user to view the properties of the objects. The viewers were built in such a way that only the selected objects are highlighted at a time. In order to achieve the goal of being able to select all the objects that belong to the same ownership rights, some steps have to be taken. To achieve this goal, a *plug-in* (hardware or software module that adds a specific feature or service to a larger system) may be added to the viewer. Adding a plug-in means that this plug-in should be able to stop the viewer from doing its default action of selecting a single object at a time, but be able to select many based on a certain condition. Another option is to have an application specific viewer built. But, instead of developing a new viewer, a simpler and easy solution to address this issue is adapted. The objects that belong to the same ownership rights can be grouped together. This means that every region having the same ownership right will be considered as a single object by the viewer. By doing this, no change occurs to the geometric structure of the regions, but semantically, they are considered as a single object. So, based on the condition that they have the same ownership rights, a CityGML viewer will be able to select all the regions of the building. By doing so, the objects may be disconnected to each other, and yet be selected at the same time. This step is the main highlight of the process since it achieves the required need without having to do any external work.

This step has three advantages.

- No additional plug-ins needed.
- Any viewer can be used.
- Works well even for disconnected objects.
- Does not change the geometric properties of the region.

For example, in Figure 4.5, the regions with ownership right as ″1″ are put together as a single object even though they belong to different floors. The region with ownership right as ″2″ will be take as another object.

### 4.2.4 Transformation/translation of coordinates based on floor number

Once the regions are grouped based on ownership rights, the next task is to take care of the floor numbers. Consider a huge building having subsequently large number of floors. Suppose in each floor, there are rooms that belong to different people, then it means that this building has an enormous number of ownership rights associated to it. In order to be able to view the regions with different ownership rights of this building, these regions have to be kept in 3D space. By identifying the floor number of each region, the height for each region is computed and the region is placed in 3D space. This will help in a clear visualization of each region and also help to easily identify an ownership right, and all the regions that belong to it.

The pixel coordinates of the corners of the regions are converted into $x$, $y$ or $z$ coordinates before the transformation of regions based on the floor numbers is completed. For this conversion, the pixel coordinates of the corners of the regions are taken along with the values of height-width. Based on the floor number of that region, the height of the region is changed (that is, the z axis value is changed). The result of this step transforms the regions into 3D space depending on the floor to which the region belongs to. The Figure 4.6 shows an example where the regions are transformed into 3D space.

Once the regions are placed on their respective height based on the floor numbers, they have to be stacked on top of each other. This is done so that it gives a replica of the actual building. Now, since the floors have to be stacked, it also means that the $x$ and $y$ coordinates of each region have to be translated with respect to the ground floor.

A simple approach is employed to solve the problem of floor stacking. This method is based on choosing an *anchor point* for each region in a floor by selecting the corner with the minimum x-axis value. The difference between this anchor point and the *point of translation* (say, towards the ground floor) is calculated. Based on this difference, the coordinates of the remaining corners of that region is computed. This way the new coordinates of every corner is obtained in terms

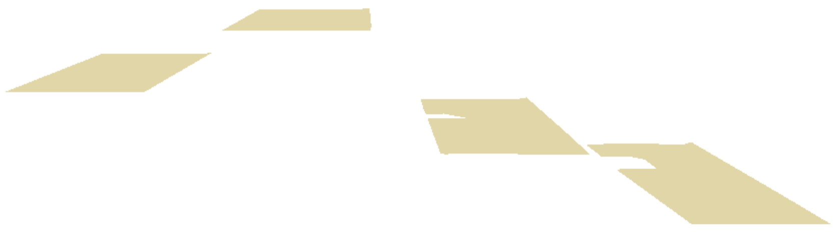
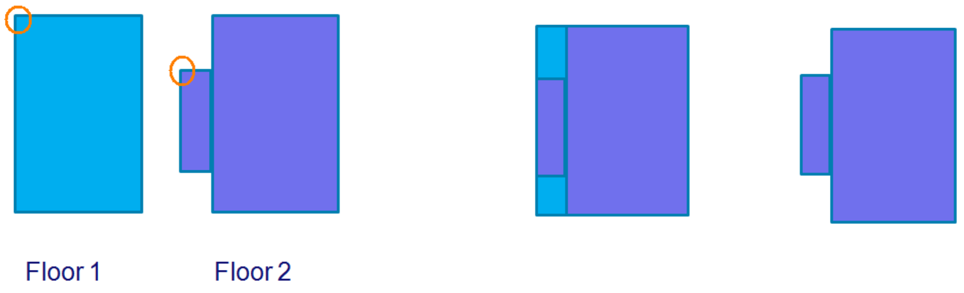


**Figure 4.6**: *The output of the end result of changing the z-axis based on floor number.*



(a) Two floors before stacking. (b) Floors after stacking using the simple approach. (c) Expected result after stacking.

**Figure 4.7**: *Example of an exception that does not work as expected while translating the coordinates of the region based on floor number.*

of x, y and z coordinates. This stacks the floors on top of each other. This method works well for many cases that could be seen. An exception is shown in Figure 4.7. Suppose a floor has an external region, like a balcony, which is built outside the boundary of the floor below it, then it is possible that this simple method may not work as expected. Figure 4.7(a) shows the top view of 2 floors of an apartment before this step of the process is done. The circles to each floors indicate the selected anchor point for each region. Figure 4.7(b) shows the result of the translation of the floors based on floor number using this method and Figure 4.7(c) shows the expected result for this case. A dedicated algorithm that stacks floors correctly in more complicated situations is left for future work.

Since most of the buildings are usually well structured, and oddly structured buildings form only a small percentage of the total number of buildings, this method provides the required proof of concept in stacking the floors. Figure 4.9 shows the result, a 3D view of the end product of this step. Here the floors of the building are properly stacked as expected. The floors are placed on top of each other with a noticeable gap so that the boundaries of each region in every floor can be viewed easily. It is also easy to view and select smaller regions within a floor.

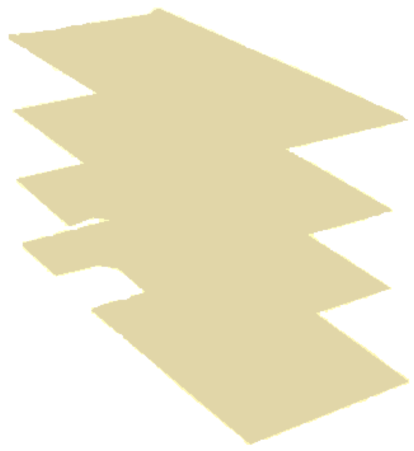


**Figure 4.8**: *Output after translating the coordinates based on Floor Number.*

### 4.2.5 Representation in CityGML format

After translating the co-ordinate values of the region based on the floor number, and grouping each region based on the same ownership right, the regions have to be represented in CityGML. Similar regions are grouped together and represented in CityGML format. The format of CityGML is explained in Chapter 2 with an example. The data provided here is represented in a similar manner. Additional information like ownership rights and floor number are also added into the structure in a similar manner as explained in Chapter 3. Hence the result of this process is a file in Extended CityGML.

### 4.2.6 Converting the 2D model into a 3D model

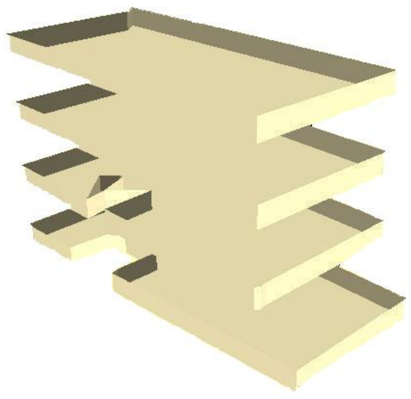


**Figure 4.9**: *The final output of the Building after representing it in Extended City GML.*

Though we have floors floating on top of each other in 3D space, the floors by themselves do not really look 3D. By converting every region into 3D, by raising its walls, the building model looks much more pleasant and the users too can relate the model to the respective buildings in reality. Figure 4.10 shows the final output of a building from the floor plans which was taken as an input. This is the resulting file which is a 3D model in extended CityGML.

## 4.3 The final outlook

The result of all the steps in the process is an extended CityGML file. The information in the file can be queried, used for data mining and analyzed. The buildings can also be mapped onto any terrain. Figure 4.11 shows the modeled building being mapped on a terrain with several parcels separated by roads. This building is mounted on one of the parcels. Figure 4.12 shows the extracted properties that is associated to each apartment of the building. This method can be used by the Kadaster to model any situation which normally can not be easily viewed in a 2D cadastral map.

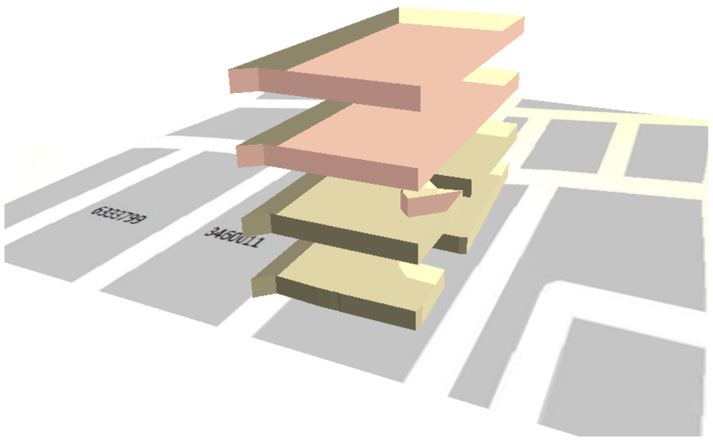
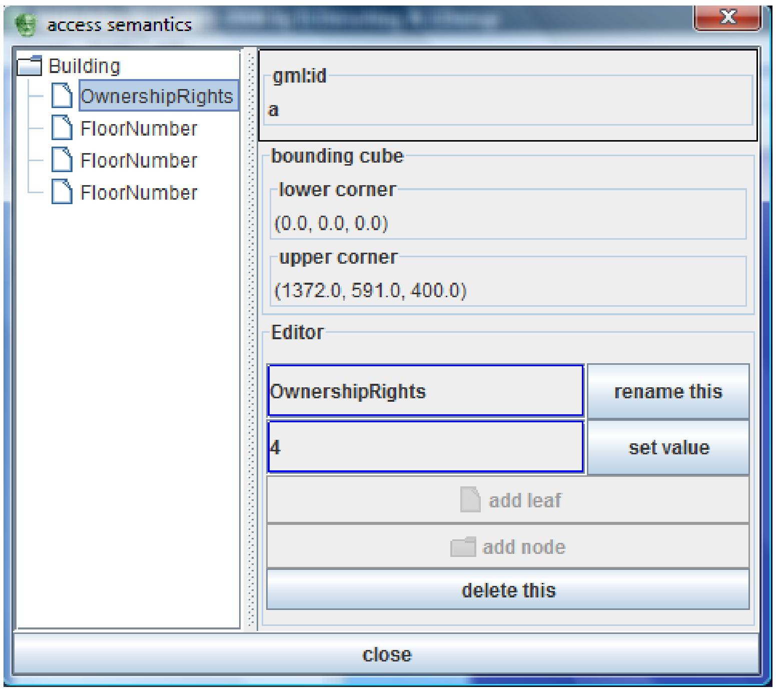


**Figure 4.10**: *The 3D model of an apartment building placed over a parcel. The regions owned by a single ownership right is highlighted in pink.*



gml:id
a

bounding cube
lower corner
(0.0, 0.0, 0.0)
upper corner
(1372.0, 591.0, 400.0)

Editor
OwnershipRights
4

**Figure 4.11**: *The properties of the 3D model of an apartment building. The ownership right '4' has ownership boundaries in 3 floors.*

# Chapter 5

# Summary and future work

## 5.1 Summary

This thesis presents a feasibility study on CityGML for cadastral purpose. For parcels with multiple ownership rights, the current system provides a scanned image of the floor plans of the building which is not computer processable for queries. This thesis proposes the use of CityGML to deal with this issue. In order to accomplish this, an extension to CityGML has been developed which enables it to visualize and represent legal information associated to buildings and apartments. As a result, the thesis presents a *proof of concept* for a way to represent the multiple ownership rights associated with a parcel.

The legal information from the scanned images of floor plans of the buildings can be extracted [4]. This extracted information can be represented in this proposed extended CityGML. This thesis also presents a method that is developed to automatically represent the extracted information in extended CityGML.

## 5.2 Future work

There is a lot of work that can be done to have a system specifically built for cadastre related applications. As far as the field of CityGML and related topics are concerned, more work is yet to be done on the following ideas.

- Customized viewer for web applications
  3D city models in CityGML format have many commercial and open source viewers that can be used to visualize these models. But a customized viewer for web applications for cadastral purpose can be built so that it is easy for the users to interact with the viewer. Another reason to have a viewer built for web applications is that the cadastre cannot expect the users to download the viewer to use the system. Any updates on the viewer means that the user needs to update his viewer. Therefore, it is not feasible to use the available viewers and a specific web application viewer needs to be built.

- A detailed extension
  The CityGML can be extended in a suitable way to cover all registration related issues that are not covered in this thesis. This thesis does not consider the registration details on each parcel. This can also be incorporated into the process.

- Global format for all registration types
  The ADE can be verified by the CityGML communities and a format that can be used by all registration units can be developed and standardized.

While automating the process of representing the extracted information from the scanned images for the floor plans in extended CityGML, the following topics need more work.

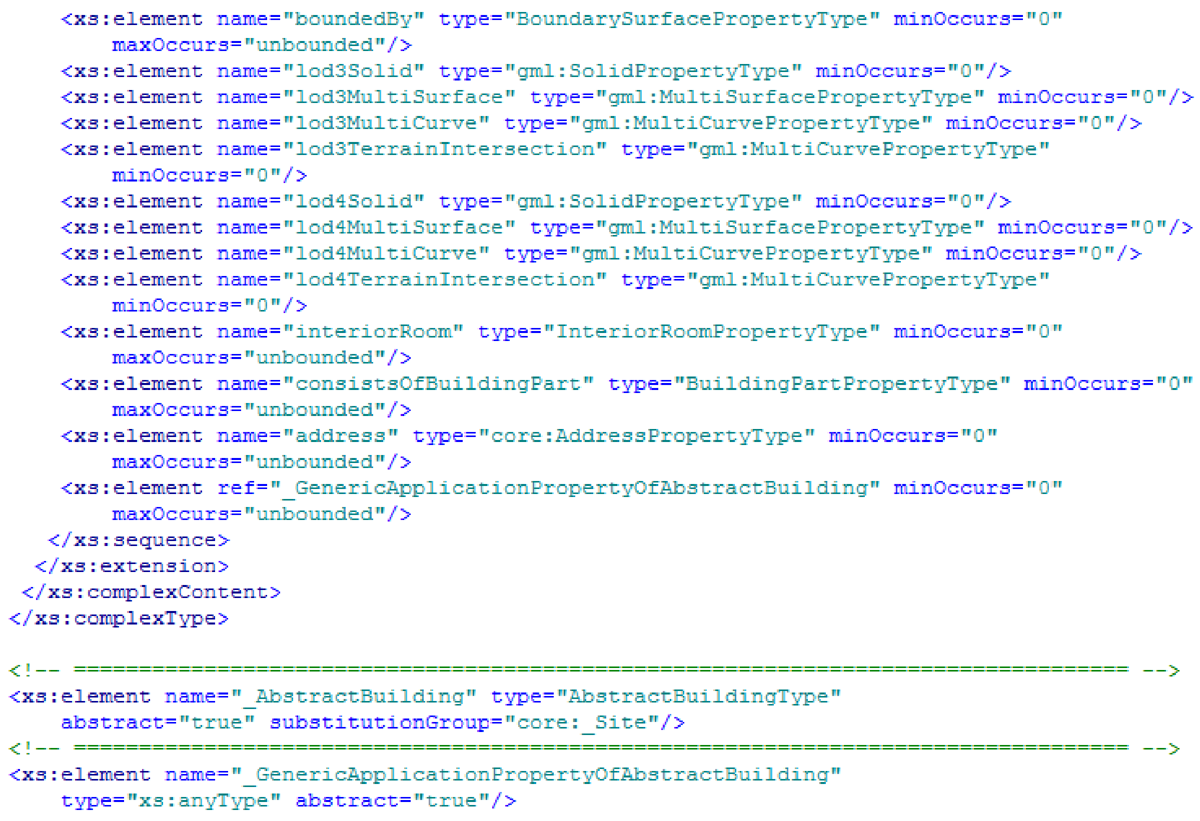
- A better method to stack the floors
  The current method has some exceptions where it may fail to stack the floors on the exact positions (section 4.2.4). The new method could take shape matching and other ideas into consideration while stacking the floors, which will give a better result.

- Considering the real height of the floors
  The current system does not consider the real height of each floor. There may be buildings where different floors may have different heights.

# Appendix A

## A-1 Appendix A

```
<xs:complexType name="AbstractBuildingType" abstract="true">
 <xs:complexContent>
  <xs:extension base="core:AbstractSiteType">
   <xs:sequence>
    <xs:element name="class" type="BuildingClassType" minOccurs="0"/>
    <xs:element name="function" type="BuildingFunctionType" minOccurs="0"
        maxOccurs="unbounded"/>
    <xs:element name="usage" type="BuildingUsageType" minOccurs="0"
        maxOccurs="unbounded"/>
    <xs:element name="yearOfConstruction" type="xs:gYear" minOccurs="0"/>
    <xs:element name="yearOfDemolition" type="xs:gYear" minOccurs="0"/>
    <xs:element name="roofType" type="RoofTypeType" minOccurs="0"/>
    <xs:element name="measuredHeight" type="gml:LengthType" minOccurs="0"/>
    <xs:element name="storeysAboveGround" type="xs:nonNegativeInteger" minOccurs="0"/>
    <xs:element name="storeysBelowGround" type="xs:nonNegativeInteger" minOccurs="0"/>
    <xs:element name="storeyHeightsAboveGround" type="gml:MeasureOrNullListType"
        minOccurs="0"/>
    <xs:element name="storeyHeightsBelowGround" type="gml:MeasureOrNullListType"
        minOccurs="0"/>
    <xs:element name="lod1Solid" type="gml:SolidPropertyType" minOccurs="0"/>
    <xs:element name="lod1MultiSurface" type="gml:MultiSurfacePropertyType"
        minOccurs="0"/>
    <xs:element name="lod1TerrainIntersection" type="gml:MultiCurvePropertyType"
        minOccurs="0"/>
    <xs:element name="lod2Solid" type="gml:SolidPropertyType" minOccurs="0"/>
    <xs:element name="lod2MultiSurface" type="gml:MultiSurfacePropertyType"
        minOccurs="0"/>
    <xs:element name="lod2MultiCurve" type="gml:MultiCurvePropertyType" minOccurs="0"/>
    <xs:element name="lod2TerrainIntersection" type="gml:MultiCurvePropertyType"
        minOccurs="0"/>
    <xs:element name="outerBuildingInstallation" type="BuildingInstallationPropertyType"
        minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="interiorBuildingInstallation" type="IntBuildingInstallationPropertyType"
        minOccurs="0" maxOccurs="unbounded"/>
```



```
    <xs:element name="boundedBy" type="BoundarySurfacePropertyType" minOccurs="0"
        maxOccurs="unbounded"/>
    <xs:element name="lod3Solid" type="gml:SolidPropertyType" minOccurs="0"/>
    <xs:element name="lod3MultiSurface" type="gml:MultiSurfacePropertyType" minOccurs="0"/>
    <xs:element name="lod3MultiCurve" type="gml:MultiCurvePropertyType" minOccurs="0"/>
    <xs:element name="lod3TerrainIntersection" type="gml:MultiCurvePropertyType"
        minOccurs="0"/>
    <xs:element name="lod4Solid" type="gml:SolidPropertyType" minOccurs="0"/>
    <xs:element name="lod4MultiSurface" type="gml:MultiSurfacePropertyType" minOccurs="0"/>
    <xs:element name="lod4MultiCurve" type="gml:MultiCurvePropertyType" minOccurs="0"/>
    <xs:element name="lod4TerrainIntersection" type="gml:MultiCurvePropertyType"
        minOccurs="0"/>
    <xs:element name="interiorRoom" type="InteriorRoomPropertyType" minOccurs="0"
        maxOccurs="unbounded"/>
    <xs:element name="consistsOfBuildingPart" type="BuildingPartPropertyType" minOccurs="0"
        maxOccurs="unbounded"/>
    <xs:element name="address" type="core:AddressPropertyType" minOccurs="0"
        maxOccurs="unbounded"/>
    <xs:element ref="_GenericApplicationPropertyOfAbstractBuilding" minOccurs="0"
        maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- ================================================================================= -->
<xs:element name="_AbstractBuilding" type="AbstractBuildingType"
    abstract="true" substitutionGroup="core:_Site"/>
<!-- ================================================================================= -->
<xs:element name="_GenericApplicationPropertyOfAbstractBuilding"
    type="xs:anyType" abstract="true"/>
```

**Figure A.1**: *CityGML schema file for _AbstractBuildingType*

## A-2 Appendix B

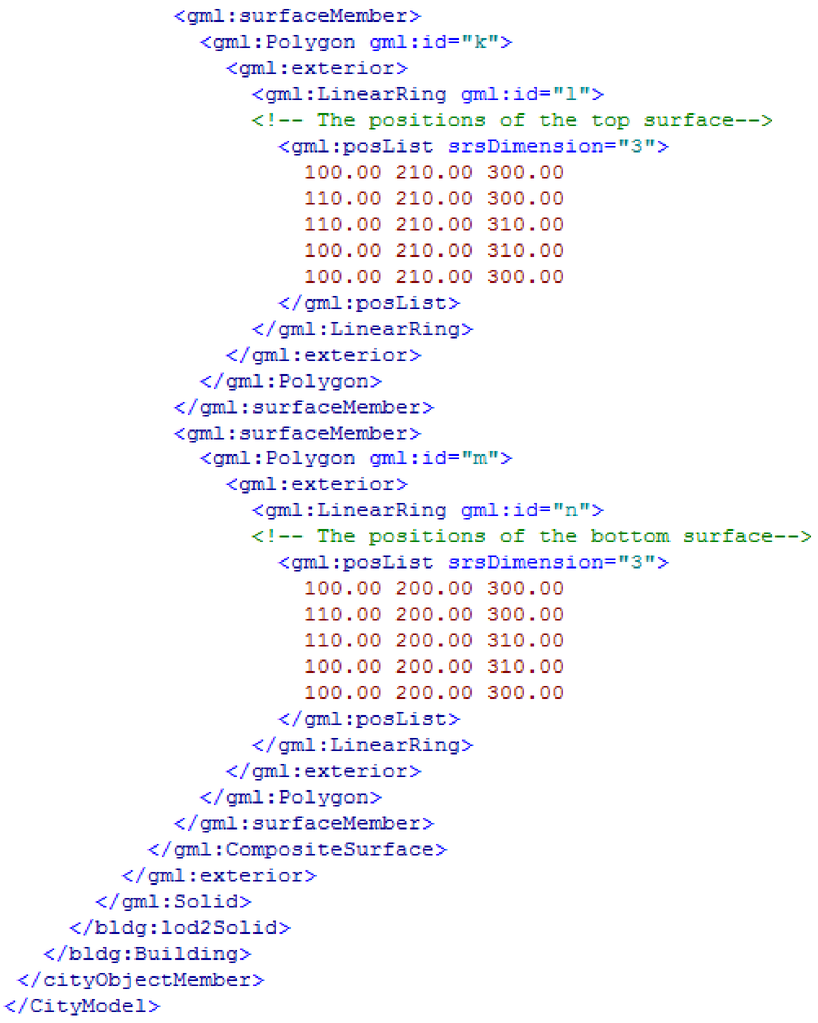
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Written by McEnroe Gifford Dsilva, version "1.0" -->
<!-- Technical University of Eindhoven, the Netherlands -->
<CityModel xmlns="http://www.opengis.net/citygml/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:gml="http://www.opengis.net/gml"
xmlns:bldg="http://www.opengis.net/citygml/building/1.0"
xsi:schemaLocation="http://www.opengis.net/citygml/building/1.0
http://schemas.opengis.net/citygml/building/1.0/building.xsd">

<gml:description>This is a simple example to show
how to model a building in LOD1</gml:description>

<gml:name>LOD1 Building</gml:name>

  <cityObjectMember>
    <bldg:Building gml:id="a">
      <gml:boundedBy>
        <gml:Envelope srsDimension="3" srsName="b">
        <!-- The positions of the lower and the upper corners
        of the model-->
          <gml:lowerCorner>100.00 200.00 300.00</gml:lowerCorner>
          <gml:upperCorner>125.00 210.00 320.00</gml:upperCorner>
        </gml:Envelope>
      </gml:boundedBy>
      <bldg:lod2Solid>
        <gml:Solid gml:id="a">
          <gml:exterior>
            <gml:CompositeSurface gml:id="b">
              <gml:surfaceMember>
                <gml:Polygon gml:id="c">
                  <gml:exterior>
                    <gml:LinearRing gml:id="d">
                    <!-- The positions of the front surface-->
                      <gml:posList srsDimension="3">
                        100.00 200.00 300.00
                        110.00 200.00 300.00
                        110.00 210.00 300.00
                        100.00 210.00 300.00
                        100.00 200.00 300.00
                      </gml:posList>
                    </gml:LinearRing>
                  </gml:exterior>
                </gml:Polygon>
              </gml:surfaceMember>
```

```
<gml:surfaceMember>
  <gml:Polygon gml:id="e">
    <gml:exterior>
      <gml:LinearRing gml:id="f">
      <!-- The positions of the side1 surface-->
        <gml:posList srsDimension="3">
          100.00 210.00 300.00
          100.00 210.00 310.00
          100.00 200.00 310.00
          100.00 200.00 300.00
          100.00 210.00 300.00
        </gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:Polygon>
</gml:surfaceMember>
<gml:surfaceMember>
  <gml:Polygon gml:id="g">
    <gml:exterior>
      <gml:LinearRing gml:id="h">
      <!-- The positions of the side2 surface-->
        <gml:posList srsDimension="3">
          100.00 210.00 310.00
          100.00 200.00 310.00
          110.00 200.00 310.00
          110.00 210.00 310.00
          100.00 210.00 310.00
        </gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:Polygon>
</gml:surfaceMember>
<gml:surfaceMember>
  <gml:Polygon gml:id="i">
    <gml:exterior>
      <gml:LinearRing gml:id="j">
      <!-- The positions of the back surface-->
        <gml:posList srsDimension="3">
          110.00 210.00 310.00
          110.00 200.00 310.00
          110.00 200.00 300.00
          110.00 210.00 300.00
          110.00 210.00 310.00
        </gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:Polygon>
</gml:surfaceMember>
```



```
              <gml:surfaceMember>
                <gml:Polygon gml:id="k">
                  <gml:exterior>
                    <gml:LinearRing gml:id="l">
                    <!-- The positions of the top surface-->
                      <gml:posList srsDimension="3">
                        100.00 210.00 300.00
                        110.00 210.00 300.00
                        110.00 210.00 310.00
                        100.00 210.00 310.00
                        100.00 210.00 300.00
                      </gml:posList>
                    </gml:LinearRing>
                  </gml:exterior>
                </gml:Polygon>
              </gml:surfaceMember>
              <gml:surfaceMember>
                <gml:Polygon gml:id="m">
                  <gml:exterior>
                    <gml:LinearRing gml:id="n">
                    <!-- The positions of the bottom surface-->
                      <gml:posList srsDimension="3">
                        100.00 200.00 300.00
                        110.00 200.00 300.00
                        110.00 200.00 310.00
                        100.00 200.00 310.00
                        100.00 200.00 300.00
                      </gml:posList>
                    </gml:LinearRing>
                  </gml:exterior>
                </gml:Polygon>
              </gml:surfaceMember>
            </gml:CompositeSurface>
          </gml:exterior>
        </gml:Solid>
      </bldg:lod2Solid>
    </bldg:Building>
  </cityObjectMember>
</CityModel>
```

**Figure A.2**: *Code of the building modelled in LOD1*

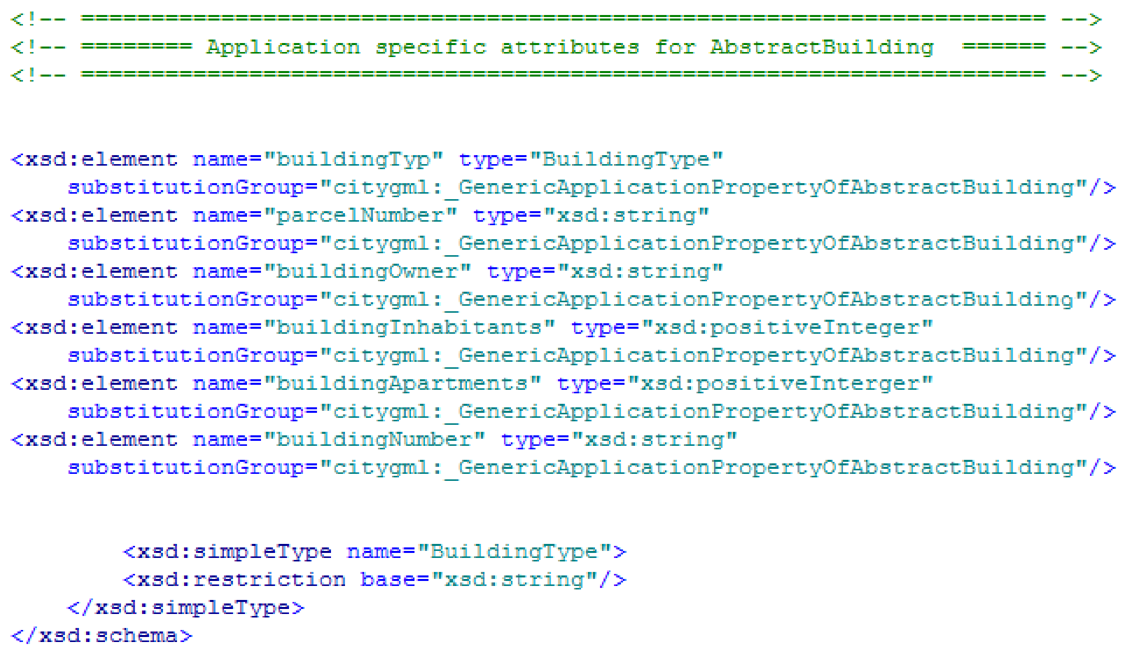
## A-3 Appendix C

```
<?xml version="1.0" encoding="UTF-8"?>
<gml:Dictionary
  xmlns="http://www.opengis.net/gml"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/gml ../../3.1.1/profiles/
  simpleDictionary/1.0.0/gmlSimpleDictionaryProfile.xsd" gml:id="BuildingType">

  <name>BuildingType</name>
  <dictionaryEntry>
    <gml:Definition gml:id="id1">
      <gml:description></gml:description>
      <gml:name codeSpace="urn:d_nrw_sig3/ade/kad_de">1</gml:name>
      <gml:name>none</gml:name>
    </gml:Definition>
  </dictionaryEntry>
  <dictionaryEntry>
    <gml:Definition gml:id="id2">
      <gml:description></gml:description>
      <gml:name codeSpace="urn:d_nrw_sig3d/ade/kad_de">2</gml:name>
      <gml:name>commercial</gml:name>
    </gml:Definition>
  </dictionaryEntry>
  <dictionaryEntry>
    <gml:Definition gml:id="id3">
      <gml:description></gml:description>
      <gml:name codeSpace="urn:d_nrw_sig3d/ade/kad_de">3</gml:name>
      <gml:name>residential</gml:name>
    </gml:Definition>
  </dictionaryEntry>
  <dictionaryEntry>
    <gml:Definition gml:id="id4">
      <gml:description></gml:description>
      <gml:name codeSpace="urn:d_nrw_sig3d/ade/kad_de">4</gml:name>
      <gml:name>private</gml:name>
    </gml:Definition>
  </dictionaryEntry>
  <dictionaryEntry>
    <gml:Definition gml:id="id5">
      <gml:description></gml:description>
      <gml:name codeSpace="urn:d_nrw_sig3d/ade/kad_de">5</gml:name>
      <gml:name>others</gml:name>
    </gml:Definition>
  </dictionaryEntry>
</gml:Dictionary>
```

**Figure A.3**: *External code list for BuildingType*

## A-4 Appendix D



```
<!-- ================================================================= -->
<!-- ======== Application specific attributes for AbstractBuilding  ====== -->
<!-- ================================================================= -->


<xsd:element name="buildingTyp" type="BuildingType"
    substitutionGroup="citygml:_GenericApplicationPropertyOfAbstractBuilding"/>
<xsd:element name="parcelNumber" type="xsd:string"
    substitutionGroup="citygml:_GenericApplicationPropertyOfAbstractBuilding"/>
<xsd:element name="buildingOwner" type="xsd:string"
    substitutionGroup="citygml:_GenericApplicationPropertyOfAbstractBuilding"/>
<xsd:element name="buildingInhabitants" type="xsd:positiveInteger"
    substitutionGroup="citygml:_GenericApplicationPropertyOfAbstractBuilding"/>
<xsd:element name="buildingApartments" type="xsd:positiveInterger"
    substitutionGroup="citygml:_GenericApplicationPropertyOfAbstractBuilding"/>
<xsd:element name="buildingNumber" type="xsd:string"
    substitutionGroup="citygml:_GenericApplicationPropertyOfAbstractBuilding"/>


        <xsd:simpleType name="BuildingType">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
</xsd:schema>
```

**Figure A.4**: *ADE for identified legal properties for the class _AbstractBuilding*

## A-5 Appendix E

```
<!-- ==================================================================== -->
<!-- ======================= Kadaster Apartment  ======================== -->
<!-- ==================================================================== -->

<xsd:element name="kadasterApartmentProperty"
    type="KadasterApartmentPropertyType"
    substitutionGroup="kad:_GenericApplicationPropertyOfAbstractBuilding"/>

<xsd:complexType name="KadasterApartmentPropertyType">
    <xsd:sequence minOccurs="0">
        <xsd:element ref="KadasterApartment" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="gml:AssociationAttributeGroup"/>
</xsd:complexType>
<xsd:complexType name="KadasterApartmentType">
  <xsd:complexContent>
    <xsd:extension base="core:AbstractBuildingType">
      <xsd:sequence>
        <xsd:element name="ownershipRight" type="string" minOccurs="0"/>
        <xsd:element name="floorNumber" type="integer"
                minOccurs="0"maxOccurs="unbounded"/>
        <xsd:element name="ownership" type="OwnershipType" minOccurs="0"/>
        <xsd:element name="apartmentNumber" type="xsd:string" minOccurs="0"/>
        <xsd:element name="apartmentowner" type="xsd:string"
                minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="apartmentInhabitants" type="xsd:positiveInteger"
                minOccurs="0"/>
        <xsd:element name="roomCount" type="xsd:positiveInteger" minOccurs="0"/>
        <xsd:element name="detachedRoom" type="xsd:boolean" minOccurs="0"/>
        <xsd:element name="detachedRoomCount" type="xsd:integer" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:simpleType name="OwnershipType">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:element name="KadasterApartment" type="KadasterApartmentType"
substitutionGroup="core:_CityObject"/>
```

**Figure A.5:** *ADE for identified legal properties for the new class _KadasterApartment*

## A-6 Appendix F

```xml
<?xml version="1.0" encoding="UTF-8"?>
<gml:Dictionary
 xmlns="http://www.opengis.net/gml"
 xmlns:gml="http://www.opengis.net/gml"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.opengis.net/gml ../../3.1.1/profiles/
 simpleDictionary/1.0.0/gmlSimpleDictionaryProfile.xsd" gml:id="OwnershipType">

  <name>OwnershipType</name>
  <dictionaryEntry>
    <gml:Definition gml:id="id1">
      <gml:description></gml:description>
      <gml:name codeSpace="urn:d_nrw_sig3/ade/kad_de">1</gml:name>
      <gml:name>none</gml:name>
    </gml:Definition>
  </dictionaryEntry>
  <dictionaryEntry>
    <gml:Definition gml:id="id2">
      <gml:description></gml:description>
      <gml:name codeSpace="urn:d_nrw_sig3d/ade/kad_de">2</gml:name>
      <gml:name>partnership</gml:name>
    </gml:Definition>
  </dictionaryEntry>
  <dictionaryEntry>
    <gml:Definition gml:id="id3">
      <gml:description></gml:description>
      <gml:name codeSpace="urn:d_nrw_sig3d/ade/kad_de">3</gml:name>
      <gml:name>private</gml:name>
    </gml:Definition>
  </dictionaryEntry>
  <dictionaryEntry>
    <gml:Definition gml:id="id4">
      <gml:description></gml:description>
      <gml:name codeSpace="urn:d_nrw_sig3d/ade/kad_de">4</gml:name>
      <gml:name>others</gml:name>
    </gml:Definition>
  </dictionaryEntry>
</gml:Dictionary>
```

**Figure A.6**: *External code list for OwnershipType*

# Bibliography

[1] CityGML. Official homepage of CityGML. `http://www.citygml.org/`, 2007.

[2] CityGML Wiki. CityGML-City Geography Markup Language. `http://www.citygmlwiki.org/index.php/Main_Page`, 2005.

[3] G. Gröger, T.H. Kolbe, A. Czerwinski, and C. Nagel. OpenGIS City Geography Markup Language (CityGML) Encoding Standard. Open Geospatial Consortium Inc., 2008.

[4] P.N. Jain. An automatic recognition method for building floor plans. Master's thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, 2009.

[5] OpenGIS schemas. CityGML schema. `http://schemas.opengis.net/citygml/`, 1994.

[6] C. Schulte. Concept of CityGML Application Domain Extension for a web based 3D flood information service. Master's thesis, University of Applied Science, Stuttgart, Germany, 2008.

[7] SIG 3D. Special Interest Group 3D. `http://www.ikg.uni-bonn.de/sig3d`, 1999.

[8] J.E Stoter. *3D Cadastre*. PhD thesis, TU Delft, the Netherlands, 2004.