Student thesis series INES nr 672

Linked geodata: CityGML represented as a virtual knowledge graph

Felix Hansson

2024
Department of
Physical Geography and Ecosystem Science
Lund University
Sölvegatan 12
S-223 62 Lund
Sweden

Felix Hansson (2024).

Linked geodata: CityGML represented as a virtual knowledge graph

Master degree thesis, 30 credits in Physical Geography and Ecosystem Analysis
Department of Physical Geography and Ecosystem Science, Lund University

Level: Master of Science (MSc)

Course duration: *January* 2024 until *June* 2024

Linked geodata: CityGML represented as a virtual knowledge graph

---

Felix Hansson

Master thesis, 30 credits, in *GIS and Remote Sensing*

Lars Harrie
Department of Physical Geography and Ecosystem Science

Rachid Oucheikh
Department of Physical Geography and Ecosystem Science

Acknowledgements

I would like to thank my supervisors Lars Harrie and Rachid Oucheikh for the invaluable feedback and support throughout the writing and technical implementation of the thesis. I would also like to give a thanks to Lars for providing the topic idea and the opportunity to work on a project that I discovered to be something I truly enjoyed working on and can hopefully continue to work on in the future.

A special thanks goes out to my friend Maximilian for being an invaluable sounding board and discussion partner in my efforts to learn and apply all the syntax and query languages I had to learn in order to see this thesis to the end.

In support of open-source developers and the invaluable work they do in order to make projects like this one possible, I would like to mention that this work was done using Protégé.

Lastly, I would also like to extend thanks to all my friends around the world and here in Lund that have listened to my endless struggles and rants throughout this period. Thanks for all the fun and for motivating me to push through all the challenges.

Abstract

CityGML is an important standard to present 3D geometry, topology, semantics and appearance that together with 3D city models, which have had increased use within analysis and applications, such as emergency response, energy consumption and occupancy measurement. However, the standard's querying and integration capabilities can still be further explored. One way to do this is through the Semantic Web technology knowledge graphs (KG). Proof-of-concept studies have been made using this approach but remains to be properly applied outside of studies as ad-hoc and data conversion approaches are still the most prevalent in practice. The approach can potentially be applied to fields such as the building permit process and transport infrastructure management as means to further help the digitalization in these fields and improve the workflows by allowing seamless linking and integration of other data while being queriable. This study provides a demonstration in how CityGML data can be presented as a virtual knowledge graph (VKG) in an effective manner through the commonly used tools 3DCityDB and Protégé with the Ontop plugin. This provides a means to improve interoperability between different types of data as well as a method to better manage semantical 3D city models. A framework is described in this study to populate the commonly used CityGML 2.0 ontology with CityGML data containing buildings from an area in Malmö, Sweden. This is primarily done through the use of R2RML mapping that retrieves the CityGML data from a 3DCityDB relational database to create virtual instances of data based on the CityGML ontology, thus exposing the data as a VKG through the Ontop system. To ensure that the data is effectively represented and demonstrated, SPARQL queries were performed to validate and test the KG. Seven queries were made in total to test different parts of the KG and to demonstrate some practical implications of the approach. The resulting KG constructed from the mapping retrieves the expected results through the queries, based on comparisons with the original data. As a result, it can be concluded that the CityGML data is effectively represented in the KG. Most notably, the showcasing of the mapping for LoD3 buildings provides a novel description of the implementation process. The framework described is sufficient for the representation despite the limitations imposed by the 3DCityDB database schema and Protégé suite while also demonstrating the potential use of the approach for urban planning processes.

Table of Contents

1. Introduction

3D city models have been increasingly applied for various analysis and applications, such as emergency response, energy consumption, occupancy measurement, building type classification, propagation of noise, 3D cadastre as well as different environmental simulation results, such as noise, urban energy, daylight and wind comfort (Eriksson et al., 2020; Uggla et al., 2023). The applications of 3D city models is diverse and leads to increasingly complex models as the applications of 3D city models increase. In particular, analysis and simulation applications requires semantically richer models and even further might require connections to external databases or systems (Uggla et al., 2023). As the applications of 3D city models become increasingly more complex, so does the approaches and implementations of those models.

Some cities in Sweden and in other parts of the world have identified increasing needs for more efficient handling of 3D geodata and digital twins of the respective cities. In Sweden, Stockholm, Gothenburg and Malmö identified those needs. All three Swedish cities already uses 3D geodata in the form of 3D models for some of their application needs. One example from Stockholm is a 3D model application for the building planning process. The application entails showing a proposed project imported into a 3D model and producing 360 degree images for visualization purposes (Uggla et al., 2023). While the application is useful as is, additional analysis and data retrieval is not possible through the model. This can be enabled if the model is constructed as a semantic 3D city model. This would for example enable analyses on how a project contributes to an area in terms of overall housing.

Gothenburg also currently employs 3D models based on geodata. The city aims to construct a full digital twin with metaverse applications by 2030 (Uggla et al., 2023). While the current model is a parametric and semantic model, the city faces challenges in linking external databases, such as building models and assets, to allow for objects to be connected to external thematic data. This would further enhance analysis and application capabilities through the use of the model.

Malmö's primary interest is a 3D base map for visualisations purposes that could also potentially be connected to external databases and systems (Uggla et al., 2023). This is similar to the needs faced with Gothenburg's digital twin. While Malmö does not currently have any publicly available full digital twin, a 3D base map aiming to be linked to external databases is still in need of semantic city models as a means to implement linked 3D geodata.

In all of the above mentioned Swedish cities, there is an interest in methods for better linking of 3D geodata in city models. The cities have concluded that their current geospatial products in the form of 2D maps and current 3D models are not enough to meet the future needs. All three cities expressed needs for better methods of linking 3D geodata with building. The exact application needs of the cities is hard to predict but the requirement of linking data present in the needs of all three cities.

In order to specify the needs of the cities there is an ongoing innovation program in Sweden. The program is a collaboration between different actors in the urban planning sector to digitalize more parts of the processes in said sector. The program has been named Smart Built Environment and one part of the program focuses on the interoperability between different types of digital data and becoming better at sharing the data (Smart Built Environment, 2024). The program contains several projects that works towards developing new standards and workflows for digital data. Workflows for achieving linked data has been the topic in multiple of these projects, with the report on the recently developed Swedish national model for 3D city models, 3CIM, being the most relevant one in this context.

The program contains another project aimed at strengthening the urban planning sector's ability to cooperate digitally for more effective construction and management. The concept of interoperability is synonymous with this project and in order to effectively achieve this, systems has to be developed that are better at sharing and understanding information and data between different organizations. The project specifically aims to implement and demonstrate a technical framework based on Linked Data and Semantic Web. These technologies has been highlighted as good standardised methods for creating an internationally standardised interoperable platform (Smart Built Environment, 2023b).

Helsinki, Singapore and Zurich were cities in other countries that identified the same needs as the cities in Sweden and have developed digital twin models of their own as well. These are currently used primarily for analysis and visualization applications within urban planning. The city of Zurich highlights that the digital twin model is used for zoning development, high rise planning, climate simulations and visualization for architectural competitions. The city of Helsinki emphasizes that their digital twin models primarily serves as a tool for scenario analysis and prediction (Lehtola et al., 2022).

CityGML is a widely used and important standard for this type of 3D city model that can define 3D geometry, topology, semantics and appearance of objects (Ohori et al., 2018). Implementation-wise, CityGML is defined as a GML application schema, consisting of text files that represents a select part of a dataset, region, object or Level of Detail (LoD). The presences of topology and semantics makes it possible to query data for analysis or other purposes. The standard has some capabilities of answering queries through external database software packages such as 3DCityDB (Ding et al., 2023; Eriksson et al., 2020; Uggla et al., 2023). However, the querying potential of CityGML still remains to be further explored. Furthermore, CityGML is a complex and hierarchical structure covering a wide field of representations. This leads to linking and interoperability issues when trying to map other types of data to it (Ding et al., 2024; Vilgertshofer et al., 2017). One way of dealing with the issues presented is through the use of Semantic Web technologies and more specifically knowledge graphs (KGs) (Ding et al., 2024).

The approach of using KGs is gaining more interest as a means to make 3D city models fully quarriable and integrated with other types of data. Proof-of-concept studies has been done for the approach but remains to be properly applied outside of studies as ad-hoc and format conversion approaches are more prevalent as solutions in practice. Newer, state-of-the-art ontologies has also been refined and developed that has yet to be applied in practical studies, presenting new opportunity to investigate the approach further (Chadzynski et al., 2021; Ding et al., 2023; Ding et al., 2024). The multiple fields and areas mentioned above are applications where a KG approach would provide benefit for workflows. Being able to more seamlessly query and link many types of data is what has been asked for by the cities in Sweden. Current conversion methods are difficult to implement and do not fully enable seamless interoperability like the KG approach would provide. Furthermore it is a step towards improved interoperability that will improve workflows and processes for many more applications and organizations (Ding et al., 2024).

1.1. Problem statement

The increasing popularity of 3D city models for application and analysis purposes within urban planning have led to cities both in Sweden and internationally to expressing needs for better methods to manage 3D city models. Stockholm, Gothenburg, Malmö, Helsinki, Singapore and Zurich are cities that recognize the need to manage increasingly complex 3D city models and while some of the mentioned cities employs models, the inability to connect and link external databases to 3D city models limits the query and analysis capabilities. Employing semantically richer models together with a knowledge graph approach is one method for enabling such linkage and connection.

The standard CityGML is a commonly used data format to represent 3D city models. Industry Foundation Class (IFC) is another standard that is used to represent highly detailed models of buildings and infrastructure. The two standards complement each other naturally in a 3D city model environment and are therefore interesting to link in a model. These standards still remain challenging to seamlessly achieve interoperability between as well as linking them to other data formats because of their difference in focus and model paradigm. Semantic technologies such as knowledge graphs presents a possible solution for representing multiple data formats as one, comprehensive and fully semantically linked model, enabling seamless interoperability between models. However, the practical implementation still remains limited as ad-hoc approaches and conversions of data formats are still the most prevalent method for creating interoperability. This creates opportunity and an interest in investigating the method, as it provides a solution to the challenges faced by the aforementioned cities and therefore warrants an investigation of the approach. Furthermore, investigating the knowledge graph approach is in the interest of the Smart Built Environment program. Conducting a study on the method can serve to help future developments in the program by providing insight and reference material for the approach.

The hypothesis is that the transformation process from CityGML data to a knowledge graph does not result in significant loss of information and that potential loss can be controlled. Investigating which available ontology best supports development of a KG can serve to control potential loss as well as potential information loss when transitioning from relational tables to a KG. Subsequently, the manual handling of inconsistencies in semantics after transition will also be a means to investigate loss. Validating and testing the result will give an indication of how well scalability, adaptability and structures are maintained to conclude if the approach is an improvement over existing methods.

1.2. Aim

The purpose of this master's thesis is to investigate an approach for how CityGML building data can be delivered as a knowledge graph to further aid and improve processes and workflows where interoperability between 3D city models and other model structures such as Building Information Model (BIM) is necessary.

More specifically, the aim can be specified as a number of research questions:

1) How can CityGML building data be effectively modelled and transformed into a knowledge graph on top of relational databases for urban planning applications?
2) What are the limitations of the commonly used 3DCityDB database schema and CityGML ontology when used for the purpose of representing CityGML data as a virtual knowledge graph?
3) How can a knowledge graph representation of CityGML data be used to effectively carry out queries within potential common use cases, such as urban applications?

1.3. Limitations

A significant limitation of scope in this study is the implementation of CityGML only. The study aims to focus on the workflow and process of representing CityGML data as a VKG rather than integrating it with other types of data. While interoperability and integration is introduced as a motivation for studying the process, it is not practically included in the study as it is beyond the scope in terms of time available. Furthermore, no thorough study of query efficiency will be done besides simpler discussion regarding the execution times of the tested queries. The study also limits itself to only testing one commonly used database schema for the proposed system.

1.4. Disposition

The report outlines the related work regarding the topics introduced through the introduction and problem statement in section 2. The sections cover applications of 3D city models and interoperability issues, the CityGML standard, ontologies together with knowledge graphs, the available ontologies for CityGML and tools used to represent data as knowledge graphs. The process and workflow of representing CityGML 2.0 data as a VKG is presented in section 3, covering the system used to expose the data as a VKG as well as the practical workflow for both mapping and validating the VKG. Section 4 outlines the results of the data mappings as well as the query results from the validation and testing of the resulting VKG. Section 5 discusses the outcomes and their inferences on the research questions presented. Lastly, section 6 includes the conclusions made throughout the report.

2. Related work

This section gives an overview to the different topics related to the problem statement. It provides a general description of the potential applications, specifically the building permit process and infrastructure management, providing context as to why the knowledge graph approach can be beneficial for the processes. The relevant structure details of CityGML is also explained to provide context into the data format that is of interest for the study and the cities in question. An understanding of knowledge graphs and related semantic technologies, such as ontologies, are central to the implementation process and a section is therefore dedicated to providing explanation of those concepts. CityGML ontologies are required to represent the data as a virtual knowledge graph. A review of CityGML ontologies used in previous studies is therefore presented in this section as well.

2.1. Potential Applications: Supporting the need for a knowledge graph approach.

The building permit process is highlighted as one of the most promising cases for automation through digital processing and digital data about buildings, infrastructure and other similar construction. Much of the national and international legislation is moving in a direction that promotes digital transition and the building permit process is one such field that is currently lagging behind in that transition, as manual processing of permits is still the primary method. The challenge naturally lies in the difficulty of harmonizing complex workflows in a way that retains all the used information in the manual processing, information that sometimes can be relatively informal such as local knowledge. Both BIM and GIS models are integral pieces of information to be used in such a process and therefore interoperability between them is a key (Noardo et al., 2022).

The use of BIM and GIS in the building permit process has been studied where methods for increasing automation of the process through BIM and geospatial data was investigated (Olsson et al., 2018). BIM and geospatial data was used together with local planning regulations and construction standards in order to create an automated process for checking compliance with regulations regarding building height, building area and general maintenance of the character in a built-up area. The method used for checking building height compliance involved transforming a BIM model to a CityGML surface geometry model where all the wall elements and attributes specific for the height checking was extracted and transformed. The building model and detailed planning map, which contains the height rules, were imported into FME where polygons and lines were constructed to represent the ground surface and building height. The ground surface was constructed based on a DEM while the building height was calculated as the difference between the ground surface and the intersection line between the roof surface and façade of a building. Once this calculation had been done, it was checked against the regulation breakpoint to see if the building was in compliance with it. The method for checking building area involved extracting and transforming the building footprints from BIM models and loading them into FME, as the footprint in the majority of cases is equivalent to the building area. From the footprints the area of the building can then be calculated and checked against the regulations for compliance. The general maintenance of the character in a built-up area revolved around having a particular design for buildings in an area. Such criteria are difficult to develop automated checking for but the study integrated BIM and geospatial data as a means to visualize a built-up area and this method was mainly tested as a feasibility study due to the shortcomings of the models lacking information regarding, for example, the colour of a building. The study shows that there is potential in developing automated checks in the building permit process through the use and integration of BIM and GIS data (Olsson et al., 2018).

BIM models focuses on the representation of buildings and their components, providing a detailed platform for information sharing and management in engineering projects. GIS models on the other hand largely represents spatial information and relationships between objects such as buildings but inherently lacks detailed semantic information about buildings. Integration of BIM and GIS complements the models as a means to meet the needs of 3D city model applications. The GIS data structure CityGML is widely used across Europe with all capitals providing CityGML models. Germany in particular provides LoD2 models for almost every large city in the country (Tan et al., 2023). CityGML is very complex and quite difficult to interpret because of its very wide scope (Noardo et al., 2022). The presence of semantics in the data structure as well as the widespread prevalence of CityGML makes it the preferred standard over other model structures based on geometric models only. Conversion and integration between BIM and CityGML specifically can therefore be considered essential in promoting interoperability between model structures and formats, allowing designers to consider the suitable references for their workflow process (Noardo et al., 2022).

The infrastructure sector is another field with possible opportunities to interoperate between BIM and GIS data. The infrastructure sector commonly stores 3D geodata in CityGML as well as BIM models in specific IFC standards to model infrastructure facilities, such as tunnels, road and railways. However, CityGML and IFC cannot directly be mapped onto each other as a means to have all data working together without loss of data (Vilgertshofer et al., 2017).

IFC is a well-established open file format for exchange of data and information as models for use in the BIM field, commonly applied in the Architecture, Engineering and Construction (AEC) industry. The format enables exchange of data and its associated geometrical properties such as walls, beams and columns. It also allows for exchange of associated heterogeneous attributes such as mechanical properties, costs and construction work time (Gerbino et al., 2021). Interoperability between BIM models and Geospatial Information Models (GIM) such as CityGML is often required in the urban planning process. For example, buildings and infrastructure planning can require geospatial data in order to perform shadow, visibility and emission analyses. Interoperability allows for buildings and infrastructure to be rendered in detail in GIS, enabling a general view and impact of planned construction. However, solutions for enabling interoperability is not trivial due to the two model paradigms being different in purpose and modelling perspective (Herle et al., 2020).

IFC typically allows for modelling of specific physical objects to describe various types of installation in buildings, roads, railways and tunnels with semantic relationships. CityGML contains corresponding semantic entities to the aforementioned physical objects. To allow for direct mapping of CityGML and IFC onto each other, the entities in both models are required to be semantically and spatially aggregated otherwise data is lost in the process. This is likely not the case as IFC focuses on providing a finer coarseness of the data than CityGML. Attempting to directly map the IFC to CityGML would therefore result in a loss of information as CityGML is not designed to model information at the finer levels that IFC provides (Vilgertshofer et al., 2017). Attempting the reverse process of mapping CityGML to IFC is also problematic (Tan et al., 2023). It would be necessary to convert surface models in CityGML to solid models, which are used in IFC. This can only be done through extrusion of the surface models that are primarily 3D planar objects. However, objects such as walls and roofs that are required in IFC do not have defined precise thickness in CityGML since it does not contain this type of information (Chognard et al., 2018). Geometric representations poses another issue for the mapping. IFC supports three different geometric representations, specifically Constructive Solid Geometry (CSG), sweep solid and Boundary Representation (B-rep). On the other hand, GIS and CityGML typically only support B-rep, meaning that a choice has to be made for which representation corresponds the best to the B-rep in CityGML (Zhu et al., 2018).

One proposed solution to the previously mentioned problem of interoperability is to utilize the concept of Linked Data. The concept has been successfully used to link geodata and BIM-data. One example of this is the linkage of CityGML 2.0 and IFC Tunnel data for a shield tunnel project through applying semantic web technology (Vilgertshofer et al., 2017). The linking of the two models was done by first converting the EXPRESS (IFC Tunnel) schema and CityGML Extensible Markup Language (XML) schema to Web Ontology Language (OWL) descriptions. Both models also had to be converted to Resource Description Framework (RDF) instance files in order for the data to be RDF representations that could be used to link the models. Lastly, the two OWL descriptions was manually inspected to find class similarities between the descriptions. These similarities were then used to create a mapping which defines the link elements between the two models at instance level. This served as a proof of concept for the approach at the time and shows that there is potential for this kind of application within the infrastructure sector as well. Other approaches have been successfully applied within the infrastructure sector to facilitate interoperability between data. CityGML and IFC data have been integrated to facilitate better asset management of railway infrastructure. The approach used to enable this differs from the previous case in that the CityGML data was imported and converted to work in an IFC environment. The approach is different from linking data but successfully achieve an acceptable level of interoperability. A BIM/GIS framework can be used to improve management of railways by supporting the decision-making process both in operation and maintenance. Regardless of approach taken to facilitate interoperability, there is clear opportunity and improvement to be had from further work on interoperability between GIS and BIM data in the infrastructure field (Garramone et al., 2022).

2.2. CityGML

CityGML is an international standard for semantic 3D city models. The standard presents four different characteristics used in virtual 3D city modelling: semantics, geometry, topology and appearance (Kolbe, 2009). Currently there are two versions of CityGML that see use in studies and practical application, denominated as CityGML 2.0 and CityGML 3.0. The most notable difference between the two versions is the revised LoD concept. In the 2.0 version, LoD is defined as five different levels while the 3.0 version contains four different levels, removing the most detailed level used for representing the interior of objects. Instead, the 3.0 version allows user to define interior or exterior LoD on all four levels. Furthermore, the newer version provides a revised space concept within the models. The version allows for defining space as different volumetric entities such as a physical space or a logical space. Physical space can in turn also be defined as occupied or unoccupied space. This new space concept in CityGML 3.0 further enhances the data structure by enabling new semantic relationships and expressions of topology, geometry and thematic relations between spaces (Kutzner et al., 2020). However, despite CityGML 3.0 being the most current standard, CityGML 2.0 remains the most used version by the cities of Sweden. This is largely due to the lack of technical implementation of the newer version by the cities and municipalities in Sweden (Smart Built Environment, 2023a). For this reason, only the 2.0 version is used and the term CityGML specifically refers to the 2.0 version going forward in this study.

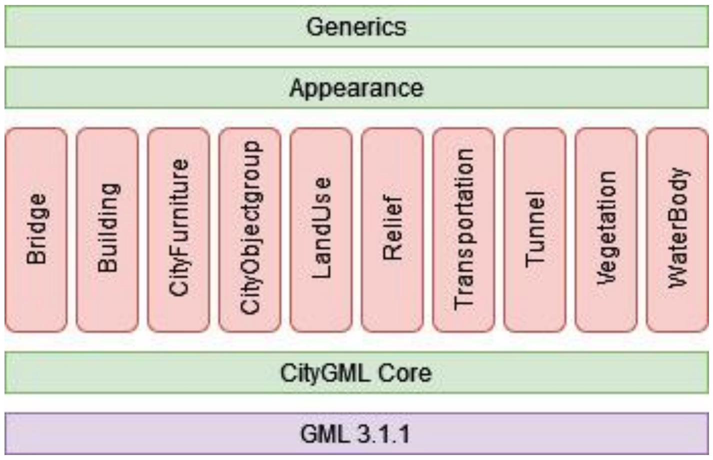


Figure 2.1: Modularization of CityGML 2.0. Updated version based on Kolbe, 2009

Modelling in CityGML 2.0 can be done on 5 different well-established LoDs, ranging from LoD0 to LoD4, increasing in complexity and accuracy of the model. The first level, LoD0 is 2.5D digital terrain model. For the building models, LoD1 is a simple block model without any kind of roof structure. LoD2 presents buildings as expanded block models with distinct roof structure and larger installations such as stairs. LoD3 represents architectural models with detailed window, door, wall and roof structures. LoD4 adds interior structure such as rooms, stairs and furniture on top of the other details defined by LoD3 (Kolbe, 2009). CityGML 2.0 data is partitioned into different modules (see Figure 2.1). The horizontal modules define structures that are relevant and applied to all the vertical modules. Such structures can for example be appearance in the model, geometry defined in the Core module and generic data attributes about objects. The vertical modules represent the different thematic models that CityGML includes. The thematic models available in CityGML is bridge, building, city furniture, city object group, land use, relief, transportation, tunnel, vegetation and water body (Kolbe, 2009).

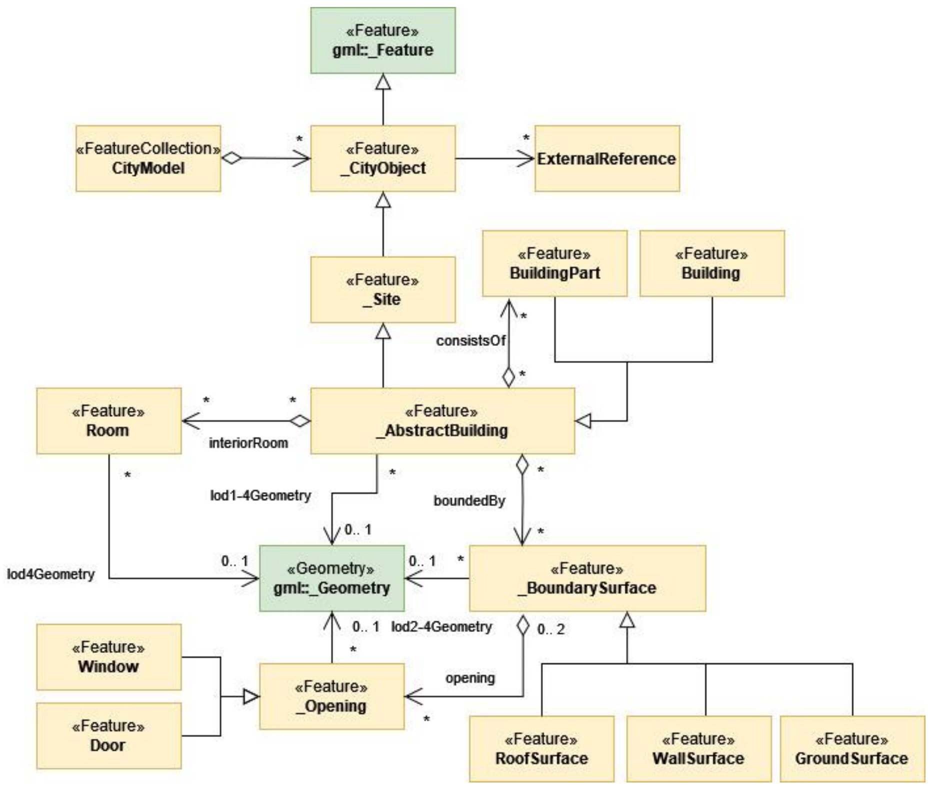


Figure 2.2: A Unified Model Language (UML) diagram showing a simplified part of the CityGML Building model. Attributes have been excluded from classes for readability. Based on the CityGML UML diagrams by OGC, 2012.

The semantic structure of CityGML is based on the ISO 19100 framework for modelling geographic features. The structure is formally modelled as classes through Unified Modeling Language (UML) notation (see Figure 2.2). The classes can contain any number of spatial and non-spatial attributes. The semantics are provided for all geographic features within the CityGML structure. The top level class of CityGML is the *CityObject* abstract class, which inherits name, description and *gml:id* from the superclass provided by the Geography Markup Language (GML) base. A *CityObject* may be linked to feature in other datasets through *ExternalReferences* and also aggregated to construct a *CityModel*, which is subclass of the GML *FeatureCollection* superclass. A *CityObject* have multiple subclasses in the form of different thematic areas defined in the corresponding modules (see Figure 2.1).

One of the more important and commonly used subclasses is *AbstractBuilding*, which represents a single building model. From this class two additional non-abstract classes are derived: *Building* and *BuildingPart*. These have a composite design where a *Building* may contain *BuildingParts* The additional detail features in LoD2 and higher become semantically represented through the classes *RoofSurface*, *WallSurface*, *GroundSurface, Window* and *Door* (see Figure 2.2). These classes represents the individual composite surface geometries and their structures for roofs, walls, footprints (ground), windows and doors. They inherit attributes and relations from their respective *CityObject* as they are derived from it. *RoofSurface*, *WallSurface* and *GroundSurface* are subclasses of the *BoundarySurface* class that is a subclass of both the *Room* and *AbstractBuilding* class. These classes are central for more detailed descriptions of buildings and their surfaces. *Window* and *Door* are in turn subclasses of the *Opening* class, derived from the *BoundarySurface* class (Kolbe, 2009).

CityGML becomes a very powerful format for 3D city modelling with rich semantic information. However, the format comes with certain challenges through its complexity, hierarchical structure and interoperability issues with other data formats such as IFC of the BIM domain. The interoperability issue stems from technical issue related to the difference in data structure and alignment of use cases (Noardo et al., 2022). These issues cause CityGML to be difficult when transforming and decoding for different applications and visualisation purposes (Ding et al., 2024).

3CIM

3CIM was developed as a means to standardise 3D geodata and meet the increasingly complex applications and analyses faced in the current urban planning processes (Uggla et al., 2023). Multiple cities and actors in Sweden as well as in other countries expressed need for methods of handling 3D geodata that is not tied to a supplier specific format. Based on this it was decided to develop a shared information model built upon CityGML. The project was successful in delivering a model based on CityGML 2.0, extended through an Application Domain Extension (ADE), that had the key features of allowing multiple types of external data sources to be linked to the model. The 3CIM model data could be imported into 3DCityDB, a software used to store CityGML data as relational tables. However, 3DCityDB is not natively capable of importing an ADE without extending the software. Instead of developing a plugin in Java to facilitate importing the ADE, Feature Manipulation Engine (FME) was used to develop a workflow that correctly link and translate attributes to facilitate the importing of the ADE (Uggla et al., 2023).

The 3CIM model was tested in different applications such as visualization, flood simulations, daylight simulations and noise simulations. For the visualization test, the model data was used in multiple different visualization tools. The tested tools were Unreal Engine and Twinmotion, CityEngine, OpenCitiesPlanner and Digital Twin City Viewer. The model was able to be visualized in all of the tools but conversion of the model into different formats was necessary as none of the tools natively reads GML-files. For the flood simulations the model was used to extract LoD2 building data and hard surface data. The building and surface data was used together with an external DEM to provide the necessary information to perform a flood simulation. Although the results of the simulation were acceptable, the model is lacking in high resolution height data. Using only the model for the simulation would therefore prove challenging. The daylight simulations were important to test as current day methods are mostly manual (Smart Built Environment, 2023a). The model was used to extract window geometries and data from LoD3 buildings to be used for the simulation and then use the model to store the simulation results. The simulations were successful and the results could be stored in the model in the CityJSON format. However, the storing method proved somewhat difficult and easier methods should be investigated. Lastly, in the noise simulations the 3CIM model was linked to external databases containing road attribute data. The transport theme in 3CIM was also used to represent the geometries for the roads. Additionally, the Building, Vegetation and City Furniture themes were used as input for the simulation. The results showed that the model has potentially to be used for this kind of simulation in the future as well. The results using the model were acceptable in all of the mentioned applications with some highlighted difficulties and challenges for specific applications, such as the conversion needs and the lacking resolution of height data. The 3CIM model was therefore deemed to meet the expressed needs of the cities for a standard better suited for managing increasingly complex 3D geodata (Smart Built Environment, 2023a; Uggla et al., 2023).

2.3. Knowledge Graphs and ontologies

Interoperability is a technical issue that involves many other dimensions, which leads to several challenges that can be tackled and many ways to improve those processes. The ISO standard 11364-1 (ISO/TC 184 2012) defines a framework for model-driven interoperability. The framework contextualizes concerns, barriers and approaches to interoperability. Different approaches can indicate solutions to how barriers can be removed through integration where a common format for all models is found. One approach suggested is linked models as a solution to interoperability for data models, including GIS and BIM models. The approach corresponds to what the standard has contextualized. In practice, the data in each individual model is kept independent and in the original storage form. Another link model is used to connect the topic-specific models, which is either done through the use of Semantic Web technologies or web services. The KG approach is a technology that can be used to achieve models capable of linking as described. A Semantic Web technology stack is needed in order to facilitate such an approach. Such a stack include languages such as RDF and OWL, which are further described below (Herle et al., 2020).

Semantic Web technologies and ontology-based methods have in recent years garnered increased interest within the field of urban planning and Geographic Information Science through the City Information Model concept (Shi et al., 2023). KG is an approach for structuring data in the form of graphs with edges encoded as object properties. Ontologies are used to provide semantics to the data and as such is integral when transforming data to KGs. An ontology refers to a concrete conceptualization of a domain of things (Gruber, 1993). CityGML being based on XML means that it adheres to an XML schema containing specific elements defined by the CityGML standard, such as elements defining specific building or part of one. The specific building element can in turn contain GML elements that for example defines the geometry. An ontology for CityGML can be generated from said XML schema but requires significant work and has already been done in previous studies (Vinasco-Alvarez et al., 2020). Alternatively, it is also possible to derive CityGML ontologies directly from UML diagrams of CityGML (Métral et al., 2010).

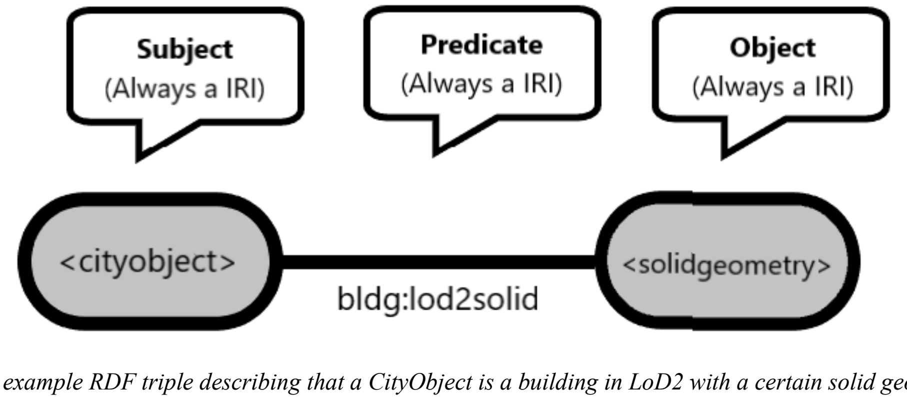


Figure 2.3: An example RDF triple describing that a CityObject is a building in LoD2 with a certain solid geometry. Each of the different components contained in a triple is also displayed.

At the core of KG solutions are typically ontologies. “Ontology” in the computer science field defines and conceptualizes a domain of interest and allow for viewing information that is only relevant for a given domain in a sharable way. To facilitate creation and sharing of ontologies, W3C has defined some standard languages such as Resource Description Framework (RDF) and Web Ontology Language (OWL). These provides a rich language used to encode data as ontologies. A knowledge graph can be modelled and represented in multiple ways, with RDF being a common way to do this. RDF is a method for modelling a knowledge graph as a directed edge-labelled graph (del graph). Del graph modelling consists of defining a set of nodes with directed labelled edges, representing binary relations between nodes. RDF defines these types of nodes referred to as Internationalised Resource Identifiers (IRIs) and are used to denote the existence of entities and its relation to between them. RDF nodes can also be referred to as being constructed by subject, predicate and object (see Figure 2.3) because of the similarities to how languages form sentences. One full set of subject, predicate and object is commonly referred to as an RDF triple. These entities can contain a wide range of datatypes such as integers, dates, strings etc. (Hogan et al., 2021). OWL is a language that is designed to represent rich and complex knowledge. Through the language things and their relations can be described. It is the most commonly used language to use to create documents describing whole ontologies (OWL Working Group 2012).

To convert conventional geospatial data to a geospatial KG a systematic approach is often needed to minimize loss of information. One such approach is the ontology-based data access (OBDA) paradigm. This approach allows users to access data through a given domain ontology and becomes semantically linked through a mapping utilizing the R2RML language, which is a language for expressing mappings from relational databases to RDF datasets (Das et al., 2012). An OBDA can be realized in two different fashions: materialized or virtual. A materialized knowledge graph (MKG) involves materializing the original data as RDF graphs through the use of specialized systems, such as Ontop. These can then be loaded into RDF storing systems that support geospatial KGs. A virtual knowledge graph (VKG) never generates any RDF graphs from the original data but keeps it virtual. Together with ontologies and mappings the data can be exposed as a virtual RDF graph. This is also achievable through the Ontop system (Ding et al., 2024). Both approaches have their advantages and are commonly used when presenting geospatial KGs. The MKG approach has the advantage of faster query-answering time, both for spatial semantic queries and non-spatial queries. VKG has the advantage of being the most storage efficient approach, as the data can be kept in its original form without any need to materialize it as an RDF graph (Li et al., 2022).

In general, KG approaches has advantages such as maintaining original data structures. Scalability and adaption on a case basis are also additional advantages. The drawbacks of KGs is that the implementation requires an intelligent approach and that the linked model is maintained and updated when the original structures are (Herle et al., 2020).

In order be able to to define RDF triples in an efficient manner, syntax through a serialization language is needed. There are multiple commonly used serialization languages for defining RDF, some examples being Turtle, TriG, N-Triples, N-Quads, JSON-LD, N3, RDF/XML and RDF/JSON (Beckett et al., 2014). Out of these languages, Turtle is the most interesting with regards to the KG approach as it is compatible with the N-Triples format and SPARQL. Coincidentally, Turtle is also the syntax used for writing R2RML. SPARQL is a query language used for querying RDF and is capable of querying a diverse set of data sources. It also contains capabilities for querying graph patterns, which is an essential functionality for practical use of KGs (Harris & Seaborne, 2013). Turtle provides syntax for writing RDF triples in a compact and natural text format that is easily readable by humans.

2.4. CityGML ontologies

The ontologies for CityGML are important as means to sufficiently describe the relation of things contained in a CityGML model. Methods to present the data as relational databases served to fix some of the problems pertaining to the static nature of XML files but still limits the implementation of semantic interoperability. The official XML encoding of CityGML was mainly intended as an exchange format and linking data requires a standard mechanism, which is only implicit in XML-based encodings. The implementation of ontologies in this regard servers to fill the remaining gap by enabling easier explicit linking of data (Heath & Bizer, 2011).

The University of Geneva have developed a CityGML ontology through applying Extensible Stylesheet Language Transformations (XLST) to CityGML 2.0 data. The resulting ontology contains a one to one matching of concepts that can bring the standard and semantic interoperability together (Chadzynski et al., 2021). It contains 185 classes, 281 object properties, 92 data properties and 1254 axioms. The ontology has been frequently applied with success in works and studies that utilizes or present CityGML data as a knowledge graph (Ding et al., 2023; Ding et al., 2024; Zalamea et al., 2016). The ontology is publicly accessible and therefore easy to apply and use for this purpose. However, the ontology has been more closely inspected in recent years and has revealed to have quality issues that requires fixing to comply with semantic web standards (Chadzynski et al., 2021). Through evaluation of the ontology it was found that the ontology failed to pass an accuracy test conducted through the Protégé ontology editor. It was also found to contain entities that were redundant within upper ontologies the would cover the CityGML ontology it. Conversely, the ontology was also found to lack certain entities that would allow it to comply with larger, upper ontology domains. Besides the limitations of the ontology, there is lack of tools to allow for populating the ontology with data.

While the ontology created by the University of Geneva describes the CityGML in a sufficient manner, the inconsistencies and other issues hinder compliance with larger domain spaces in which CityGML is contained within or can be linked to. In order to facilitate this interoperability with other knowledge graphs, the ontology created by the University of Geneva was refined in order to fix the inconsistencies and the new ontology created from this process was coined OntoCityGML (Chadzynski et al., 2021). It contains 344 classes, 272 object properties, 78 data properties and 3363 axioms. The ontology showed none of the errors that was present in the previous ontology during validation. Additionally, the ontology successfully manages to make CityGML compliant with World Wide Web Consortium (W3C) standards. This leads to triple stores constructed using this ontology to allow integration of geospatial data with other general knowledge graphs. The ontology has only seen a handful of applications but has proved to be viable as a schema for CityGML. As of 2024, at least two studies have been made where the ontology was used. One study used the ontology in the development of an autonomous intelligent software agent for instantiation, visualization and analysis of City Information Models (CIM) based on geospatial knowledge graphs. The agent was tested through creating semantic model of Berlin in LoD2 that was compliant with CityGML 2.0 through the use of the OntoCityGML (Chadzynski et al., 2022). The second study deals with the future work mentioned in the previous study by developing and integrating additional interface features such as a geospatial processor for transforming SQL to SPARQL (Agarwal et al., 2023). Other complementary agents were also developed, such as a thematic surface discovery agent, a city information agent, a city energy analyst agent and a distance agent. These agents intend to create a set of interfaces that allows for building scalable information systems enabling large city models based on knowledge graphs. The work natural applies the OntoCityGML since the logical foundations and the base work is reliant upon it (Agarwal et al., 2023).

Linked models can be presented using the prominent knowledge graph technology within the Semantic Web, which has been mentioned and successfully used in other studies to present CityGML data as knowledge graphs. A proof-of-concept study was done where 161 R2RML mappings were made to link CityGML data containing buildings to its corresponding ontology, presenting the CityGML data as a virtual knowledge graph. Queries were then performed over the knowledge graph to validate the result and to showcase some interesting and practical queries (Ding et al., 2023). This study was then further worked upon by incorporating other datasets into the knowledge graph. OpenStreetMaps (OSM) data was linked to the previously constructed CityGML knowledge graph. By implementing a supplementary ontology called LinkedGeoData (LGD), OSM data could successfully be implemented and linked to the existing graph. The integration of OSM data enabled even more complex queries to be done, such as finding all hotels over 30 meters in height (Ding et al., 2024). A third study looked into the using an RDF graph approach for integrating GIS and BIM data. Two RDF graphs were constructed by transforming the GIS data in the form of CityGML and BIM data in the form of IFC respectively (Hor et al., 2016). Semantic Web technology is then used to integrate the two graphs through the newly introduced Integrated Geospatial Information Model approach, achieving linkage between the two datasets. Using the Graph Matching for Ontologies (GMO) algorithm to measure similarities between the two graphs, the correct links could be found for corresponding concepts between the graphs. The integrated model could then be queried in a unified and fully integrated manner (Hor et al., 2016).

3. Methodology

The following section describes the process adopted for creation and validation of a KG based on CityGML data. The whole process can more practically be seen as divided into to two parts: (1) construction of the VKG, and (2) validation of the VKG. The construction of the VKG refers to the process of initializing a system consisting of a physical database, ontology and mapping to support the creation of a knowledge graph structure as well as the actual steps and processes used to map the data to a graph. The validation of the VKG refers to the process of testing and validating the resulting graph from the first part to ensure that no significant loss of data or information occurred and that the data is correctly linked according to semantics. The first part, construction of the KG, can furthermore be described as two individual steps: *Initialization of data and system* and *KG mapping*. These two steps are then followed by the *validation and testing* step described earlier. In order to successfully create a KG, a system architecture was created to facilitate the processes and tools that was needed based on the described steps (see Figure 3.1). Through the first two steps, the resulting KG could either be created as a VKG or MKG, depending on preference. The creation of a VKG requires three components: data in a relational database, an ontology and a mapping used to create the needed RDF triples that will represent the graph. A VKG has the benefit of requiring less storage for data, as the data populating the graph is kept virtual and retrieved on-demand. Large data sets such as 3D city models may have high space requirement and be cumbersome to be kept materialized as an MKG. The virtual pipeline required to maintain a VKG can also be beneficial when working with evolving and changing data such as city models, as it maintains consistency without the need to update multiple environments. The resulting KG was therefore created as a VKG to better suit the nature of the data and potential applications. The study limits itself to just constructing a VKG as the described maintenance and update need can be seen as inherent in potential application such as the building permit process, where frequent update and maintenance are commonplace for the system.

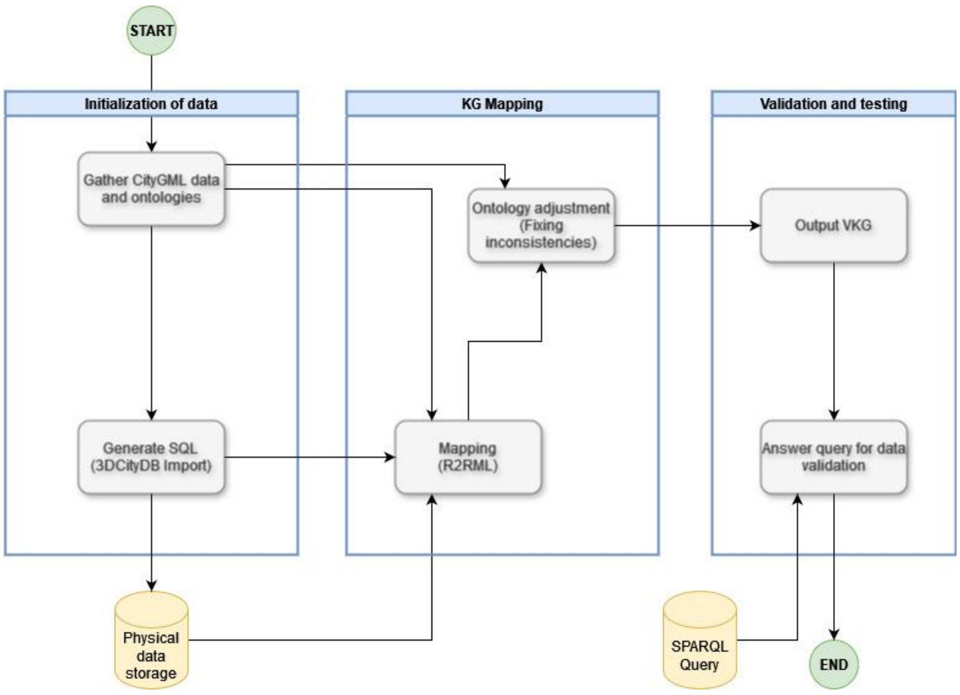

START
Initialization of data
KG Mapping
Validation and testing
Gather CityGML data and ontologies
Ontology adjustment (Fixing inconsistencies)
Output VKG
Generate SQL (3DCityDB Import)
Mapping (R2RML)
Answer query for data validation
Physical data storage
SPARQL Query
END

Figure 3.1: An overview of the system architecture used for the creation of the CityGML VKG. The architecture contains three primary modules: Initialization of data, KG mapping and validation and testing. Each module contain the individual steps and processes in the workflow.

The knowledge graph approach has general advantages such as maintaining original data structures. Scalability and adaption on a case basis are also additional advantages. The drawbacks of such an approach is that the implementation needs to be done intelligently and that the linked model is maintained and updated when the original structures are in the case of MKG (Herle et al., 2020). Those drawbacks are easily negated through the needs of the building permit process itself and VKG therefore make the KG and by extension the VKG approach appropriate as the primary focus of the study. However, the VKG approach does suffer drawbacks such as slower query times when performing integrated queries over multiple models. This is mostly due to the large number of UNION clauses that are needed to perform matching of the data models. An example of this could be the need for multiple UNION clauses in order to match overlaps in ground surfaces between two models. One model might contain multiple surface geometries that only overlaps with one in the other model, which would require multiple clauses to complete the matching. In such scenarios, MKG approaches appear to perform better (Ding et al., 2024).

3.1. Initialization of data and system

The initial starting point of the created system was the *Initialization of data and system.* This meant inputting CityGML data and generating a physical storage for said data. A physical storage was needed to be created in order to be able to reference the data needed to populate the KG. This was accomplished by embedding the CityGML data into a relational database, essentially restructuring and mapping the data to rows and columns in order to make the data queriable while also enabling updates to be done. Multiple solutions are available for performing this step and 3DCityDB was the solution chosen for this study. 3DCityDB is a commonly used tool for importing CityGML data into a SQL database. It provides both a software tool for importing and exporting data as well as a predefined SQL schema, making for an easy and streamlined process. 3DCityDB has been used in previous studies of the same nature as this and in studies adapting more ad-hoc approaches by incorporating different types of data rather than just CityGML, proving the software to be useful in use cases such as this study (Ding et al., 2024; Uggla et al., 2023).

3.1.1. 3DCityDB

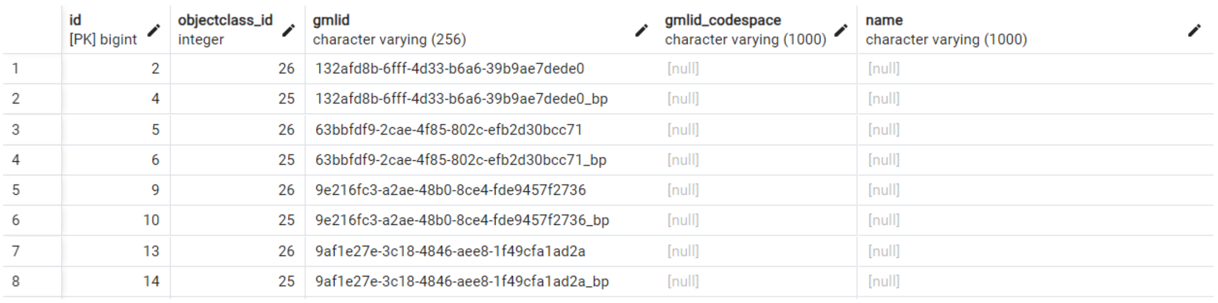
With the increased use and application complexity of CityGML data by cities around the world, a need for better storage solutions of CityGML became apparent. Due to the potential complexity and size of 3D geospatial data, no well-eastablished GIS vendor had created solutions for providing efficient storage, analysis, management, interaction and visualization of 3D city models. 3DCityDB was created as a means to provide these functionalities (Yao et al., 2018).

3DCityDB is an Open Source package that contains a set of tools for importing, exporting, managing, analysing and visualizing CityGML data as well as a database schema developed specifically for storing the CityGML standard in a relational database. The database schema maps the objects in the CityGML data structure to the structure of a spatially-enhanced relational database management system (SRDBMS). The schema support Oracle and PostGIS as the dedicated SRDMBS choice, enabling 3DCityDB to utilize the spatial data representation and processing capabilities of SRDBMS. 3DCityDB sees application in practical production in a number of cities in the world, such as Berlin, Potsdam, Munich, Frankfurt, Zurich, Rotterdam and Singapore (3DCityDB Team, 2019).

3DCityDB supports database creation for both CityGML 1.0 and 2.0 currently. The database defines classes and relations for objects as well as their respective geometrical, topological, semantical and appearance properties. Vector and grid representation is provided through the chosen SRDBMS as to avoid special solutions, allowing direct read and write access for other systems such as BIM. Furthermore, the software package includes an ADE manager, allowing for dynamic extension of the 3DCityDB schema using ADEs. Implementing an ADE requires a dedicated Java library for a given ADE (3DCityDB Team, 2019).

The database schema provided by 3DCityDB contains 15 different models that in total consists of 66 tables in the physical database. The 15 different models in the provided schema are: *Metadata, Core, geometry representation, Appearance, Building, Bridge, City Furniture, Generic Objects and Attributes, LandUse, Digital Terrain, Transportation, Tunnel, Vegetation* and *WaterBody* (3DCityDB Team, 2019). The relevant models for this study is Core, Building and geometry representation. The tables of most interest in those models are the "*cityobject*", "*building*", "*thematic_surface*" and "*surface_geometry*" tables.

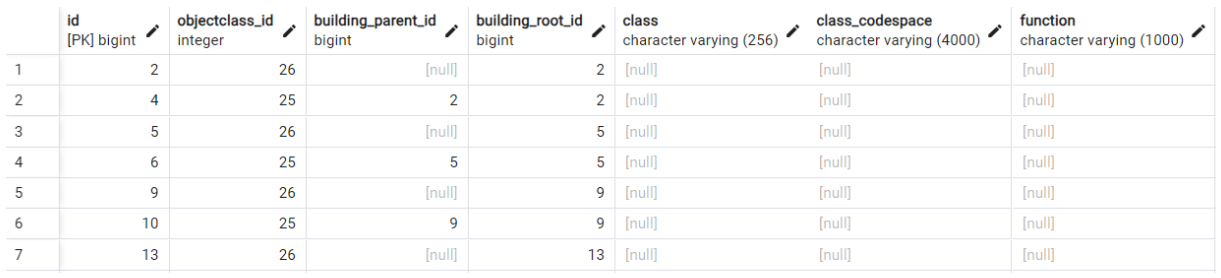
The "*cityobject*" table contains records representing all the objects present in the data regardless of their respective theme (see Figure 3.2). The table includes some attributes defining class, function and usage for the given object in the data. The primary function of the table is to provide a unique identifier for each object, which is done through the "*gmlid*" column. The table also contains an "*id*" column which is used as the primary key, used for joining other tables and referencing data records in those tables (3DCityDB Team, 2019).



	id [PK] bigint	objectclass_id integer	gmlid character varying (256)	gmlid_codespace character varying (1000)	name character varying (1000)
1	2	26	132afd8b-6fff-4d33-b6a6-39b9ae7dede0	[null]	[null]
2	4	25	132afd8b-6fff-4d33-b6a6-39b9ae7dede0_bp	[null]	[null]
3	5	26	63bbfdf9-2cae-4f85-802c-efb2d30bcc71	[null]	[null]
4	6	25	63bbfdf9-2cae-4f85-802c-efb2d30bcc71_bp	[null]	[null]
5	9	26	9e216fc3-a2ae-48b0-8ce4-fde9457f2736	[null]	[null]
6	10	25	9e216fc3-a2ae-48b0-8ce4-fde9457f2736_bp	[null]	[null]
7	13	26	9af1e27e-3c18-4846-aee8-1f49cfa1ad2a	[null]	[null]
8	14	25	9af1e27e-3c18-4846-aee8-1f49cfa1ad2a_bp	[null]	[null]

Figure 3.2: An excerpt displaying the "cityobject" table in the 3DCityDB database schema. The excerpt only displays some of the columns in the table.

The "building" table is a merging of the three CityGML classes *AbstractBuilding*, *Building* and *BuildingPart* (see Figure 3.3). Figure 2.2 showed that all three classes contained in the table are subclasses of *CityObject* and subsequently can be joined with the "*cityobject*" table. This is done through referencing the ID column as the id is shared between the tables. The table contains information on if the object is a *Building* (class ID 26) or *Building Part* (class ID 25). The table also stores arbitrary attribute data pertaining to a building such as roof type, measured height and construction date. Additionally, the table contains several columns representing geometry IDs as foreign keys. The columns represent each of the different LoDs (0-4) and each of the different geometry aggregations (solid, multi-surface, footprint and roof footprint). Each of these IDs refers to entries in the "*surface_geometry*" table (3DCityDB Team, 2019).



	id [PK] bigint	objectclass_id integer	building_parent_id bigint	building_root_id bigint	class character varying (256)	class_codespace character varying (4000)	function character varying (1000)
1	2	26	[null]	2	[null]	[null]	[null]
2	4	25	2	2	[null]	[null]	[null]
3	5	26	[null]	5	[null]	[null]	[null]
4	6	25	5	5	[null]	[null]	[null]
5	9	26	[null]	9	[null]	[null]	[null]
6	10	25	9	9	[null]	[null]	[null]
7	13	26	[null]	13	[null]	[null]	[null]

Figure 3.3: An excerpt displaying the "building" table in the 3DCityDB database schema. The excerpt only displays some of the columns in the table.

The "*surface_geometry*" and "*thematic_surface*" are two tables containing information relating to the planar surfaces that makes up the geometry of buildings. All geometry is explicitly stored in the "*surface_geometry*" table as either geometry, implicit geometry or solid geometry. The geometry column is used to store surface-based geometry as exactly one polygon, potentially containing holes. The implicit geometry stores the geometry of prototype objects used such as trees or traffic lights used for commonly reoccurring objects. The solid geometry column is used to store the volumetric geometry. This volumetric geometry is comprised of the boundary surfaces that are stored in the geometry column. Each of these geometries can be referenced using previously mentioned geometry IDs that are present in other tables. The "*thematic_surface*" table is the table that contains attribute and class information pertaining to the boundary surfaces of an object. The multi-surface ID foreign keys are located here and can be used to references the boundary surfaces in the geometry column in the "*surface_geometry*" table (3DCityDB Team, 2019).

3DCityDB supports Oracle and PostgreSQL as backend for the database. The provided SQL schema together with PostgreSQL was chosen for the SQL generation step. PostgreSQL was the chosen option because of its common usage and renown in the geospatial domain through the PostGIS extension. During the import process of the CityGML data, all features were filtered out except for *Building* and *BuildingPart*, resulting in the relational database only containing features corresponding to the Building module in the CityGML structure. This was done since the building permit process was highlighted as a potential application area and buildings are therefore the most relevant feature in CityGML data to use for testing the feasibility of the KG approach. Important to note is that the import process creates indexing in the database. Most importantly, the import process creates Generalized Search Tree (GiST) indexing for geometries. GiST is a tree-structured access method that works as a base template in which multiple indexing schemes such as B-tree and R-tree can be implemented. B-trees are self-balancing, multi-level tree structures that enable search and sequential access through a leaf node structure. R-tree share a similar structure to B-trees but are considered multi-dimensional and typically useful for spatial data querying and storing. This makes for flexible access methods that can be customized to suit different data types like geometric objects (PostgreSQL Global Development Group, 2024). No additional customization of the database was done as it was not needed to proceed with the second step in the process, that being *KG mapping*.

3.1.2. Study data

The CityGML data used in this study is of an area in Malmö. The dataset contains a total of 14262 *CityObjects* where 420 of those are *Buildings* and *BuildingParts* respectively. The buildings in the data is a mix of both LoD2 and LoD3, with 51 of them being LoD2 and 369 being LoD3. During the data import process with 3DCityDB the remaining *CityObjects* not associated with the building module was excluded from the import. This resulted in 11213 *CityObjects* being imported into the database and used in the study. A majority of these *CityObjects* were objects associated with the buildings. In the case of this dataset, those objects were roof, wall and ground surfaces. In total, 2117 *RoofSurfaces*, 7887 *WallSurfaces*, 369 *GroundSurfaces*, 420 *Buildings* and 420 *BuildingParts* were imported and used throughout this study. The data is originally a test dataset used for developing an ADE to serve as the Swedish national profile of CityGML (Uggla et al., 2023). The additional data provided by the ADE is removed from the dataset through the initialization process and only the base CityGML 2.0 data was used throughout the remaining processes. The *Building* and *BuildingParts* objects contain no additional attribute data besides their respective geometries and class information. In order to have queries retrieve meaningful and practical results, some arbitrary attribute data needs to be present in the dataset. The dataset contains a number of empty attribute fields meant to hold information regarding function, usage, year of construction, year of demolition, roof type, measured height, storeys above ground, storeys below ground, storey height above ground and storey height below ground. Measured height and storeys above ground are types of information that can be relevant in the urban planning process, since this is characteristics that are regulated in detail development plans (Lantmäteriet, 2024). Values for these fields were created by manually updating the fields using SQL queries and necessary functions after having imported the data to a relational database using 3DCityDB. The solid geometries of each building were used to calculate the height and update the measured height field for each building object accordingly. The storeys above ground values were then derived from these calculations by dividing the measured height by 2.8 to calculate how many floors each respective building would fit.

3.2. Knowledge graph mapping

With the choice of only focusing on VKG construction, the primary activity in the second step became mapping and adjusting the ontologies. To accomplish this, additional systems was needed to be able to manage ontologies as well as creating mappings to represent the knowledge graph. Ontop is a system that support creation of both MKG and VKG. It provides all the necessary support for languages such as R2RML and SPARQL that is needed for construction of a VKG based on CityGML data. It is generally a popular system that sees widespread use for the purpose of implementing VKG and OBDA approaches (Xiao et al., 2020). It has also seen previous use in other studies where CityGML data is represented as knowledge graphs (Chadzynski et al., 2021; Ding et al., 2023; Ding et al., 2024; Vilgertshofer et al., 2017).

3.2.1. Ontop and Protégé

In order to implement a VKG and OBDA approach, dedicated systems are required to free end-users from dealing with low-level data organizations and details. This allows user to concentrate on more practical high-level tasks of implementation and analysis (Xiao et al., 2020). Ontop is one such popular system for constructing VKGs. It does so by providing a reasoning engine translating SPARQL queries expressed over a knowledge graph into SQL queries which retrieves data from a relational database that populates the graph. This translation allows for Ontop to leverage functionality present in the relational databases. In the case of this study, this meant being able to utilizes spatial functions and the indexing that was created during the data import process to make querying more efficient. The construction of knowledge graphs in Ontop relies on the R2RML mapping language. Ontop can be used as a plugin in the ontology editor Protégé, which allows for simultaneous management and adjustment of ontologies during the mapping process (Ontop, 2024). The system supports many features in commonly used languages and protocols related to Linked Data and RDF stores, such as SPARQL 1.1, R2RML, OWL 2 QL, SPARQL entailment regime and the SPARQL 1.1 HTTP protocol. In particular, the SPARQL entailment regime is what enables the system to define a specific evaluation of basic graph patterns based on SPARQL logic (Glimm et al., 2013). Ontop is published under the liberal Apache 2 license, meaning that system can potentially have more uses cases and applications outside of its already known popularity (Xiao et al., 2020).

Protégé is a free, open source and wide spread software platform used for building and managing ontologies as well as knowledge-based systems. It is provided in many different frameworks, such as desktop clients and a web-based system. The desktop client supports advanced features for construction and management of OWL ontologies and RDF triples while the web-based system is more lightweight and simpler in features (Musen, 2015). The desktop client provides tools for import, export, visualization, editing and construction of ontologies. The software's plug-in architecture enables extension and customization to it through the development of plugins. Ontop is one such plugin that can be used directly in the Protégé desktop client (Musen, 2015).

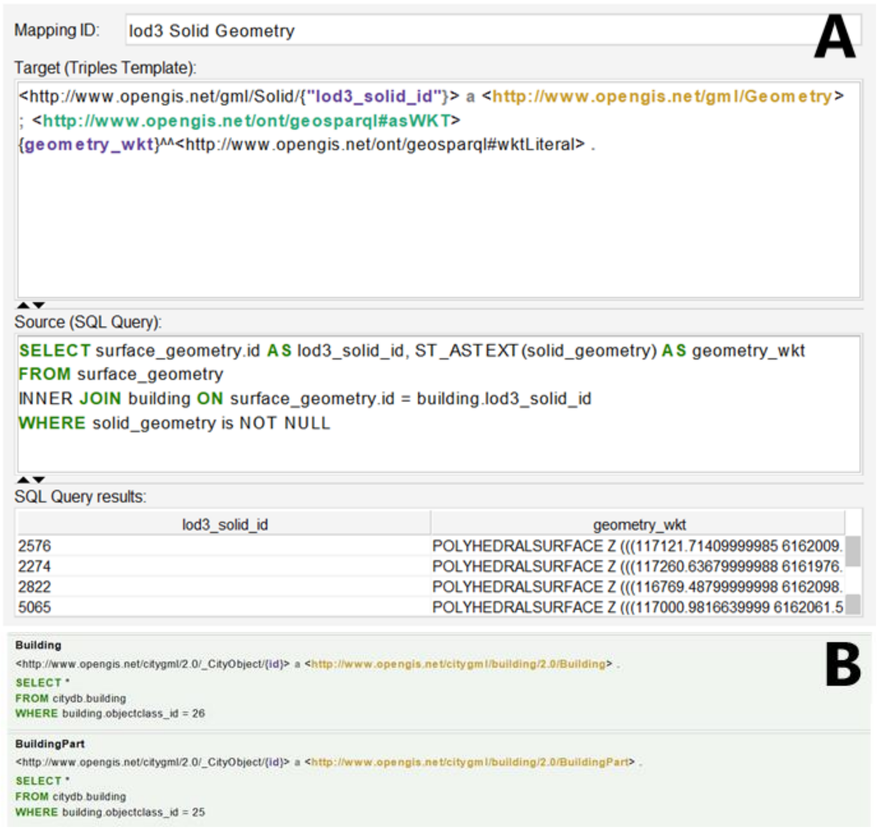
3.2.2. R2RML mapping

The primary CityGML 2.0 ontology used in this study is the one developed by the University of Geneva. It has seen prominent use in many studies as the primary ontology to describe the CityGML domain. However, the ontology is known to have several inconsistencies where object and data properties share the same IRIs, making the ontology unusable unless fixed. These inconsistencies have been seen and discovered in other studies using the same ontology (Chadzynski et al., 2021). This was resolved by removing either one of the properties sharing the same IRIs, removing the one that made the least logical sense within the context of the ontology. For example, the property "measured height" was both an object and data property in the ontology. Logically, it makes more sense for "measured height" to be a data property rather than an object property and therefore the object property was removed. This same logic was applied for all identified inconsistencies in the ontology. Additionally, some auxiliary ontologies were used alongside the primary CityGML ontology. While not necessary to do so, they provided properties that helped in defining certain semantics and constraints within the CityGML ontology. For example, the GeoSPARQL ontology was notably used to allow for defining different geometrical classes and formats that complies with OGC standards. Another auxiliary ontology that was used was the RDF schema ontology. This was added to provide constraints for labels used to populate some parts of the graph.

Together with the ontologies, the most central and user-centric step of the process could be started, that being the R2RML mapping of the knowledge graph. The mappings allowed for constructing a knowledge graph that uses the data from the physical database to populate the graph dynamically through defining SQL queries tied to each mapping, retrieving specific data. These queries are written separately from the RDFs being constructed, requiring that each individual attribute in the CityGML dataset is queried to populate their respective ontological concepts. The data used in this study contains both LoD2 and LoD3 buildings, meaning that much of the database contains empty tables that otherwise would contain data for other feature types. Furthermore, the buildings in the dataset are defined as LoD2 and LoD3 but the LoD3 buildings do not contain all LoD3 specific attributes, such as openings and windows. However, it is still possible to classify the individual surface geometries in the LoD3 buildings. Through a more detailed mapping, it is possible incorporate more semantic detail in the KG for the LoD3 buildings. With this in mind, the mapping intended to retrieve and represent all the crucial and useful data that is expected to be present in building features.

The primary criteria for all mappings are to ensure that all objects mapped to the KG have appropriate relationships described and adhere to the available linkage in the database. For example, everything in the dataset is a *CityObject* but can belong to different classes that have specified relationships according to the UML structure for CityGML. A criteria of the mapping is to ensure those are maintained and that it is functional to do so with the database schema used. Furthermore, this can be described more generally for the process. The general criteria are to ensure that semantical equivalence is achieved according to the original structure, adhering to ontological structure, modelling considerations in line with the database structure, ensuring interoperability with used standards and adhere to domain specific standards like regulations in the building permit process.

The mappings consisted of three different components: an ID, a target and a source. The ID refers to an arbitrary ID which identifies each unique mapping. The target is a RDF triple defined through Turtle syntax that incorporates placeholder variables which are populated through the source SQL query. Each triple defines a specific semantic relationship between objects of different classes in the CityGML ontology. The source is in turn an SQL query written to retrieve the specific information that is to populate the target triples. Figure 3.4 illustrates three different mappings that were written for constructing the knowledge graph. In image A the previously mentioned mapping ID, target and source can be seen. The given example shows the mapping for solid geometry of a LoD3 building. More specifically, the mapping displays the linking of a *CityObject's* LoD3 solid geometry ID in the database to the ontology class *"Geometry"* as well as linking the same ID to data property which stores the solid geometry as Well-Known Text (WKT) through the use of the GeoSPARQL ontology. The first target line in the first example in Figure 3.4 describes a solid geometry (represented through the IRI for solid geometry IDs) as the subject, belonging to the class *"Geometry"*, which is the object of this triple. The two nodes in the triple is the first IRI representing solid geometries and the yellow (colour code for classes) IRI representing the *Geometry* class. The predicate for the triple is the *"a"*, which is a shorthand for describing objects as a certain RDF type and class. The semicolon in the second line denotes another triple with the subject being the same as in the first triple. Another predicate was then defined that described solid geometries as having a data attribute. Data properties is displayed as green IRIs and used to store actual data values to objects. The third line defines the object for this triple which is the value that is stored through the data property, in this case geometry stored in the WKT format. The SQL query was written to retrieve the solid ID and the geometry from a database table called *"surface_geometry"*. This is the table that stores all geometries and relevant information associated with them. It is the standard table provided by the 3DCityDB SQL schema that was used and is a generalisation of the CityGML geometry module. Because of this, the data in the table is not directly intuitive to map but requires users to have an understanding of the data to correctly map geometries to the corresponding classes in the ontology. In the example, the solid geometry was mapped to the general *"Geometry"* class in the ontology. This was one possible solution as it is the most general geometry class in the ontology and provides necessary properties for the mapped data. It is possible to map the geometries to more specific subclasses, for example the *"Solid"* class, which is a subclass of geometry. However, it was not necessary for this example as the subject is data that was previously mapped to the *"LoD3 solid"* object property. The object property *"lod3Solid"* is used within the *"_AbstractBuilding"* and *"Solid"* class, enabling the inference of attributes and characteristics associated with the class onto the linked data. This inference process is enabled by the reasoning engine provided by Ontop. Image B in Figure 3.4 displays two additional mappings that links *CityObjects* to the *Building* and *BuildingPart* class in the CityGML ontology respectively. All the mappings that were made follows the same structure as the examples, linking different classes, object and data properties in the CityGML ontology.



Mapping ID: lod3 Solid Geometry

Target (Triples Template):

```
<http://www.opengis.net/gml/Solid/{"lod3_solid_id"}> a <http://www.opengis.net/gml/Geometry>
; <http://www.opengis.net/ont/geosparql#asWKT>
{geometry_wkt}^^<http://www.opengis.net/ont/geosparql#wktLiteral> .
```

Source (SQL Query):

```
SELECT surface_geometry.id AS lod3_solid_id, ST_ASTEXT(solid_geometry) AS geometry_wkt
FROM surface_geometry
INNER JOIN building ON surface_geometry.id = building.lod3_solid_id
WHERE solid_geometry is NOT NULL
```

SQL Query results:

lod3_solid_id	geometry_wkt
2576	POLYHEDRALSURFACE Z (((117121.71409999985 6162009.
2274	POLYHEDRALSURFACE Z (((117260.63679999988 6161976.
2822	POLYHEDRALSURFACE Z (((116769.48799999998 6162098.
5065	POLYHEDRALSURFACE Z (((117000.9816639999 6162061.5

Building

```
<http://www.opengis.net/citygml/2.0/_CityObject/{id}> a <http://www.opengis.net/citygml/building/2.0/Building> .
SELECT *
FROM citydb.building
WHERE building.objectclass_id = 26
```

BuildingPart

```
<http://www.opengis.net/citygml/2.0/_CityObject/{id}> a <http://www.opengis.net/citygml/building/2.0/BuildingPart> .
SELECT *
FROM citydb.building
WHERE building.objectclass_id = 25
```

Figure 3.4: Three example mappings used in the construction of the VKG. Image A shows the mapping ID, target and source structure used in Ontop to define solid geometry for LoD3 buildings. Image B shows two mappings used to link CityObjects to their respective subclass of Building or BuildingPart.

The multi-surface mapping process for LoD3 buildings was the most detailed mapping process and differed from the solid geometry mapping process. In order to correctly map surface geometries to the correct buildings, an additional mapping had to be written to correctly link *Building* and *BuildingPart* together. The *BuildingPart* objects contain the necessary attributes needed to link surface geometries to the *Building* class. In the database, *BuildingParts* were linked with other *CityObjects* that represented the individual multi-surface compositions of a building. In the used dataset three different multi-surface classes was present: *RoofSurface, WallSurface* and *GroundSurface.* Since these classes are subclasses of *BoundarySurface,* it is appropriate to consider what kind of ID or object that should be classified this way in the mappings. This is important as there are many alternatives that would resulting in a functioning VKG but would fail the criteria of semantical equivalence, as certain relationship might be omitted in certain cases. Considering that the class intends to describe the individual bounding surfaces that make up a complete building, the most appropriate ID to store here was deemed to be multi-surface IDs, as that aligned the most with the semantical equivalence criteria by not omitting any predicate present in the UML diagrams for CityGML 2.0.

A mapping was written in order to describe *BuildingParts* as being bounded by the multi-surface *CityObjects*. The object property used as predicate for this mapping was *boundedBy*, as this property is the term describing the relationship in the UML diagram. The *CityObjects* representing multi-surface objects are also present in another table in the database called *"thematic_surface"*. This table contains attributes and data specifically pertaining to these surface *CityObjects*, such as the multi-surface ID that can be referenced in the *"surface_geometry"* table. Utilizing the multi-surface ID attribute, a mapping was made to classify the ID as a *BoundarySurface*, linking the *CityObjects* and *BoundarySurface* nodes through the predicate *lod2MultiSurface*. All of the multi-surfaces can now be correctly referenced and classified as their respective class in the KG. This was done through three similar mappings, one for each of the surface classes. In the same mappings, the multi-surfaces IDs were also mapped to their respective individual surfaces that comprises the multi-surface, similarly to how *Building* and *BuildingPart* was mapped to their respective classes in Figure 3.4. This was defined using the predicate *surfaceMember* in order to link *BoundarySurface* nodes and *Surface* nodes. Lastly, three mappings were made again in order to map the actual geometries of each surface as WKT to the KG in the same manner as was shown in Figure 3.4, completing the mappings for the different surfaces in LoD3 buildings.

A VKG can be constructed to a functional degree with just a few mappings and can be further improved and developed with more mappings. With a functional mapping and manually fixed ontology, it is possible to query the VKG directly in Ontop using SPARQL. A quarriable VKG allows for validation through test queries to investigate the consistency and robustness of the graph.

3.3. Knowledge graph validation

To test and validate the constructed knowledge graph and its representation of CityGML data, queries are done over the graph utilizing Ontop and its provided SPARQL query editor and manager. SPARQL is the standard query language for Linked Data and RDF databases and thus the primary query language to use for validating and testing the knowledge graph (W3C RDF

Data Access Working Group, 2008). By performing queries the VKG can be tested on how well it would answer legitimate semantical queries that could be used within the potential areas of use for this approach. The testing would allow for determining the level of completeness and robustness of the knowledge graph. A complete and robust knowledge graph representation can be considered to be an effective implementation. These metrics were validated through manual cross-examination between the original data and the results obtained through the SPARQL queries. If significant loss would have occurred in the construction of the VKG the test queries would reveal if that is the case through incomplete or missing data retrieval.

A total of seven queries were written as a means to test the coherence of the constructed VKG. All seven queries were written as means to validate the mappings by incorporating most of the data mapped to the VKG. Query 1 servers to test some validity and completeness of the VKG. Queries 2-6 demonstrate some potentially practical queries that can be done to filter different buildings of interest and to test the validity of those parts in the VKG. Query 6 and 7 was designed to test the potential use of spatial querying made possible by the more detailed information for LoD3 buildings (see Appendix A for more details regarding the used SPARQL vocabular). The queries were designed with increasing complexity, incorporating more predicates from query to query. The completeness of the VKG was be determined through the queries ability to return results from the implemented mappings. Therefore, the queries were designed to retrieve information from the different parts of the KG. If the queries were successful in retrieving the correct data, that part of the KG is deemed sufficient in its completeness. The robustness of the VKG was determined through the queries ability to return the expected results. Using the database containing the original data, equivalent SQL queries can be performed to retrieve the expected results that the SPARQL queries should retrieve. If the queries returned results equivalent to those given by the SQL queries over the relational database, the targeted part of the KG was deemed sufficiently robust. Below is the aforementioned queries demonstrated as well as the intention behind each query.

Query 1 was designed to retrieve all the relevant information linked to each unique *CityObject*. It retrieves class name, the solid geometry ID as strings and the solid geometry, which is defined as WKT. Retrieving basic information about a building and its geometry is fundamental to potential application cases and is therefore an important first test for validating the VKG.

The query retrieves the wanted information by specifying variables (marked through the "?" prefix) that are queried to follow the specific predicates that leads to objects and RDF resource that is supposed the be retrieved by the query. Each variable can then be used in the selection statement to be retrieved. The functions BIND and STRAFTER are used to recreate variables, in this case to remove and recreate some selected variables without the full IRI.

SPARQL query 1: Find all buildings assigned as LoD3 and their corresponding CityObjects ID, class name, LoD3 solid ID and solid geometry.

```
PREFIX building: <http://www.opengis.net/citygml/building/2.0/>
PREFIX geosparql: <http://www.opengis.net/ont/geosparql#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?cityObject ?classname ?lod3_solid_id ?geometry_wkt
```

```
WHERE {
  ?cityObject building:lod3Solid ?lod3_solid_iri .
  ?lod3_solid_iri geosparql:asWKT ?geometry_wkt .

   BIND(STRAFTER(STR(?lod3_solid_iri), "http://www.opengis.net/gml/Solid/")
AS ?lod3_solid_id) .

  ?cityObject building:class ?class_id .
  ?class_id rdfs:label ?classname .
}
```

Query 2 was designed to be an example of a filtering query, intended to find some specific buildings fulfilling the filtering requirement. The filtering is based on the mapped data property measuredHeight, representing the height of each respective building. The query intends to retrieve all building IDs for buildings that are taller than 10 meters. A query like this is can be practical for real application and exemplify a simple filtering query. Filtering queries in general are useful with a wide variety of attributes to find specific buildings or determine if certain breakpoints are meet.

The design of this query is similar to Query 1. The measured height attribute is linked to the *CityObjects* and therefore is designed to find RDF resources of the type "*_CityObject*". Simultaneously, the predicate used to map data to the "measuredHeight" data property is queried to retrieve each respective building's height. The FILTER function is introduced here as a means to filter the *CityObjects* retrieved so that only results above 10 meters are displayed.

SPARQL query 2: Find all buildings with a measured height above 10 meters.

```
PREFIX citygml: <http://www.opengis.net/citygml/2.0/>
PREFIX building: <http://www.opengis.net/citygml/building/2.0/>

SELECT ?cityObjectId ?measuredHeight
WHERE {
  ?cityObject a citygml:_CityObject ;
  building:measuredHeight ?measuredHeight .

  BIND (URI(?cityObject) AS ?cityObjectIdUri)
  BIND (xsd:integer(REPLACE(STR(?cityObjectIdUri),
"http://www.opengis.net/citygml/2.0/_CityObject/", "")) AS ?cityObjectId)

  FILTER (?measuredHeight > 10)
}
```

Query 3 is another example of a filtering query where additional information is retrieved alongside the filter criteria. The query retrieves information such as measured height, the number of floors above ground and the solid geometry about buildings with 4 to 6 floors. The intention behind the query is to validate that the attribute data (height and number of floors) has been implemented as intended. Retrieving multiple types of attribute data like this can be useful in a building permit scenario to retrieve relevant information for existing buildings.

Query 3 is structured similar to Query 2 with no new functions. It simply extends the previous query to retrieve more data through different predicates. The FILTER function now employs a range compared to Query 2 and filters based on the number of floors rather than building height.

SPARQL query 3: Find all buildings with 4 to 6 floors above ground. Retrieve the height and solid geometry of each building fulfilling the requirement.

```
PREFIX citygml: <http://www.opengis.net/citygml/2.0/>
PREFIX building: <http://www.opengis.net/citygml/building/2.0/>
PREFIX geosparql: <http://www.opengis.net/ont/geosparql#>

SELECT ?cityObjectId ?measuredHeight ?storeysAboveGround ?geometry_wkt
WHERE {
  ?cityObject a citygml:_CityObject ;
  building:storeysAboveGround ?storeysAboveGround ;
  building:measuredHeight ?measuredHeight .

  ?cityObject building:lod3Solid ?lod3_solid_iri .
  ?lod3_solid_iri geosparql:asWKT ?geometry_wkt .

  BIND (URI(?cityObject) AS ?cityObjectIdUri)
  BIND (xsd:integer(REPLACE(STR(?cityObjectIdUri),
"http://www.opengis.net/citygml/2.0/_CityObject/", "")) AS ?cityObjectId)

  FILTER (?storeysAboveGround >= 4 && ?storeysAboveGround <= 6)
}
```

Query 4 is an example of utilizing the more detailed semantics available for LoD3 buildings. LoD3 provides more detailed information and semantics regarding the individual surfaces that comprise the solid geometry. It therefore becomes possible to query specific surface types in LoD3 buildings and this query is an example of how that can be done. In this case, the query retrieves all the ground surfaces for every building in the dataset. It retrieves the surface geometry alongside its specific ID as well as what building this surface is associated with. The practical implications of a query such as this is further exemplified in Query 5 and 6 which extends this type of query.

Query 4 utilizes many new predicates to create multiple variables. Many of the variables written into the query is not selected to be retrieved but could easily be done by adding the variable name to the selection statement. The whole query is essentially one long chain of querying through multiple links until the surface geometry is reached, which is the last object in the chain.

SPARQL query 4: Find all ground surfaces associated with LoD3 buildings and their respective surface IDs.

```
PREFIX citygml: <http://www.opengis.net/citygml/2.0/>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX cityobject: <http://www.opengis.net/citygml/2.0/_CityObject/>
PREFIX building: <http://www.opengis.net/citygml/building/2.0/>
PREFIX geom: <http://www.opengis.net/gml/>
```

```
SELECT ?buildingId ?lod3_surface_id ?geometry_wkt
WHERE {
  ?boundaryObject building:lod3MultiSurface ?lod3_multi_id.
  ?lod3_multi_id a building:GroundSurface ;
          geom:surfaceMember ?lod3_surface_id .
  ?lod3_surface_id geo:asWKT ?geometry_wkt .
  ?building building:consistsOfBuildingPart ?buildingPart .
  ?buildingPart building:boundedBy ?boundaryObject

  BIND (URI(?building) AS ?cityObjectIdUri)
  BIND (xsd:integer(REPLACE(STR(?cityObjectIdUri),
"http://www.opengis.net/citygml/2.0/_CityObject/", "")) AS ?buildingId)
}
```

Query 5 is an example of retrieving surfaces geometries and other information from a specific building. The query is designed to retrieve the roof surface geometry of a specific *CityObjects*, in this case the building with the *CityObject* ID 7748. The URI of this *CityObject* is bound to a specific variable in the query ("*?building*"). The rest of the query is similar to Query 4, with the only difference being that the height of the building is retrieved as well as the roof surface being retrieved rather than the ground surface. Finding information and specific geometries can be useful within the urban planning context. The building permit process involves a step where a proposed building project will be checked if it is in compliance with the ruling detail plan for the area, or if outside of a detail development plan, checked if it is suitable for the area and as a building (Boverket, 2020). In such a context retrieving information like what this query does can be of use.

SPARQL query 5: Find the height and roof geometry of the specific building with the CityObject ID 7748.

```
PREFIX citygml: <http://www.opengis.net/citygml/2.0/>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX cityobject: <http://www.opengis.net/citygml/2.0/_CityObject/>
PREFIX building: <http://www.opengis.net/citygml/building/2.0/>
PREFIX geom: <http://www.opengis.net/gml/>

SELECT ?buildingId ?lod3_surface_id ?roof_geometry ?measured_height
WHERE {
  ?boundaryObject building:lod3MultiSurface ?lod3_multi_id.
  ?lod3_multi_id a building:RoofSurface ;
          geom:surfaceMember ?lod3_surface_id .
  ?lod3_surface_id geo:asWKT ?roof_geometry .

  BIND(cityobject:7748 AS ?building)
  ?building building:consistsOfBuildingPart ?buildingPart ;
          building:measuredHeight ?measured_height .
  ?buildingPart building:boundedBy ?boundaryObject .

  BIND (URI(?building) AS ?cityObjectIdUri)
  BIND (xsd:integer(REPLACE(STR(?cityObjectIdUri),
"http://www.opengis.net/citygml/2.0/_CityObject/", "")) AS ?buildingId)
}
```

Query 6 extends what was done in Query 4 by incorporating spatial querying functions. The intention behind the query is to allow for measuring of distance between buildings and filter based on it. The GeoSPARQL ontology provides the function for calculating distance in the Ontop SPARQL endpoint. Important to note is that the distance function does not support 3D geometries such as a polyhedron and can therefore not use the solid geometries to answer such queries. Ground surfaces can be used to aggregate footprints of buildings, providing a practical surface polygon that can be used for distance calculations. Finding distances to nearby existing buildings can be of importance when considering where to build new buildings and the design of them.

The design of the query is similar to Query 4 in that it retrieves the ground surface for a specific target building (in this case *CityObject* 8833). The distance is then calculated from the specific ground surface associated with that building to all other ground surfaces that does not belong to the target building. A FILTER operation is used to ensure that the query does not retrieve the distance for the target building to itself. The query uses an additional FILTER operation to only retrieve results that has a distance of 5 meters or less.

SPARQL query 6: Find buildings with a 5 meter or less distance from the building with CityObject ID 8833.

```
PREFIX citygml: <http://www.opengis.net/citygml/2.0/>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX cityobject: <http://www.opengis.net/citygml/2.0/_CityObject/>
PREFIX building: <http://www.opengis.net/citygml/building/2.0/>
PREFIX geom: <http://www.opengis.net/gml/>
PREFIX surface: <http://www.opengis.net/gml/Surface/>

SELECT ?targetBuilding ?distance ?building
WHERE {
  ?boundaryObject building:lod3MultiSurface ?lod3_multi_id.
  ?lod3_multi_id a building:GroundSurface ;
           geom:surfaceMember ?lod3_surface_id .
  ?lod3_surface_id geo:asWKT ?geometry_wkt .

  surface:168419 geo:asWKT ?targetGeometry .
  BIND(cityobject:8833 AS ?targetBuilding)

  ?building building:consistsOfBuildingPart ?buildingPart .
  ?buildingPart building:boundedBy ?boundaryObject .

  BIND(geof:distance(?targetGeometry, ?geometry_wkt,
<http://www.opengis.net/def/uom/OGC/1.0/metre>) AS ?distance)
  FILTER(?building != ?targetBuilding)
  FILTER(?distance <= 5)
}
ORDER BY ASC(?distance)
```

Query 7 adds more spatial functions on top of what was showcased in Query 6. This query utilizes another GeoSPARQL function (*geom:sfTouches*) to check for surfaces that touches the ground surface of the building with the ID 8833. Some additional BIND functions are also used in order to make the result more readable by shortening IRIs. The intention behind this query is less practical as there are significantly simpler methods for retrieving and finding surface geometries associated with a specific building. The query was designed this way to test Ontop's efficiency with leveraging spatial operations. The query response time is of interest as the potential uses cases highlighted in the study would make prominent use of complex queries utilizing spatial operations. Investigating the systems performance in terms of response time is therefore important as a way to validate the approaches effectiveness in practice.

SPARQL query 7: Retrieve the distance between CityObject 8833 and all other surfaces. Also find the surfaces that touches the ground surface in CityObject 8833.

```
PREFIX citygml: <http://www.opengis.net/citygml/2.0/>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX cityobject: <http://www.opengis.net/citygml/2.0/_CityObject/>
PREFIX building: <http://www.opengis.net/citygml/building/2.0/>
PREFIX geom: <http://www.opengis.net/gml/>
PREFIX surface: <http://www.opengis.net/gml/Surface/>

SELECT ?buildingId ?surfaceType ?distance ?touching
WHERE {
  ?boundaryObject building:lod3MultiSurface ?lod3_multi_id.
  ?lod3_multi_id a ?surface_type ;
    geom:surfaceMember ?lod3_surface_id .
  ?lod3_surface_id geo:asWKT ?geometry_wkt .
  ?building building:consistsOfBuildingPart ?buildingPart .
  ?buildingPart building:boundedBy ?boundaryObject .

  surface:168419 geo:asWKT ?targetGeometry .
  BIND(cityobject:8833 AS ?targetBuilding)

  BIND(geof:sfTouches(?targetGeometry, ?geometry_wkt) AS ?touching)
  BIND(geof:distance(?targetGeometry, ?geometry_wkt,
<http://www.opengis.net/def/uom/OGC/1.0/metre>) AS ?distance)

  FILTER(?surface_type = building:GroundSurface || ?surface_type =
building:WallSurface || ?surface_type = building:RoofSurface)

  BIND (URI(?building) AS ?cityObjectIdUri)
  BIND (xsd:integer(REPLACE(str(?cityObjectIdUri),
"http://www.opengis.net/citygml/2.0/_CityObject/", "")) AS ?buildingId)

  BIND (URI(?targetBuilding) AS ?TargetUri)
  BIND (xsd:integer(REPLACE(str(?TargetUri),
"http://www.opengis.net/citygml/2.0/_CityObject/", "")) AS
?targetBuildingId)

  BIND (SUBSTR(str(?surface_type),
STRLEN(str("http://www.opengis.net/citygml/building/2.0/")) + 1) AS
?surfaceType)
}
ORDER BY DESC(?touching)
```

4. Results

The results is divided up into two subsections. The first describes the results attained through the VKG mapping by providing a simplified overview of the whole constructed VKG and later showcasing more detail regarding the individual mappings. The second section describes and interprets the results of the queries and compares it to the expected results based on the original database, if the method is applicable for the given query.

4.1. VKG mapping

The constructed VKG contains selected data as a means to display the general concept and important semantics to map when intending to create a VKG representing CityGML data. Figure 4.1 illustrates a diagram of the VKG design and the implemented data in it. In total, 20 mappings were done to achieve the VKG, resulting in a total of 224212 triples. Each triple in the VKG represents the semantic relationship between two objects or an object and its data attribute. The previously mentioned criteria is present in all the incorporated mappings with each mapping using appropriate predicates within the used CityGML ontology. The mappings covers the necessary concepts in the ontology and the correlating data in the database to represent buildings and their geometries in a sufficient manner. The mappings for LoD3 and LoD2 multi-surface geometries can been seen in Figure 4.2 The two classes relevant for the study, *Building* and *BuildingPart*, has been mapped together with appropriate class name labels from the *CityObjects* in the relational database. Solid geometries for both LoD2 and LoD3 buildings has been mapped to the geometry class as well as to a separate data property storing the solid geometries as WKT. Composite geometries, referred to as multi-surface geometries within CityGML has also been mapped to represent the boundary surfaces for the buildings in the dataset. The mapping between LoD2 and LoD3 multi-surfaces differs due to the difference in available semantic information. LoD3 has more descriptive semantics for each surfaces available, allowing for classification of each surface as either roof, wall or ground. These semantic details are not available for LoD2 buildings in the dataset and the multi-surfaces associated with those buildings has therefore been directly mapped as boundary surfaces without classification. Each *CityObject* in the data has also been mapped to have arbitrary attributes regarding function, usage, year of construction, year of demolition, roof type, measured height, storeys above ground, storeys below ground, storey height above ground and storey height below ground. However, due to the limited presence of attribute data in the original dataset only measured height and storeys above ground have data values. The semantic relationship between RDF resources are represented by blue and green arrows. Blue arrows represent a relationship between entities as object properties while green represents a type assignment of a RDF resource e.g. a *CityObject* being assigned the class *BuildingPart*. Data properties of the different classified RDF resources are displayed as text inside the container boxes in the diagram.

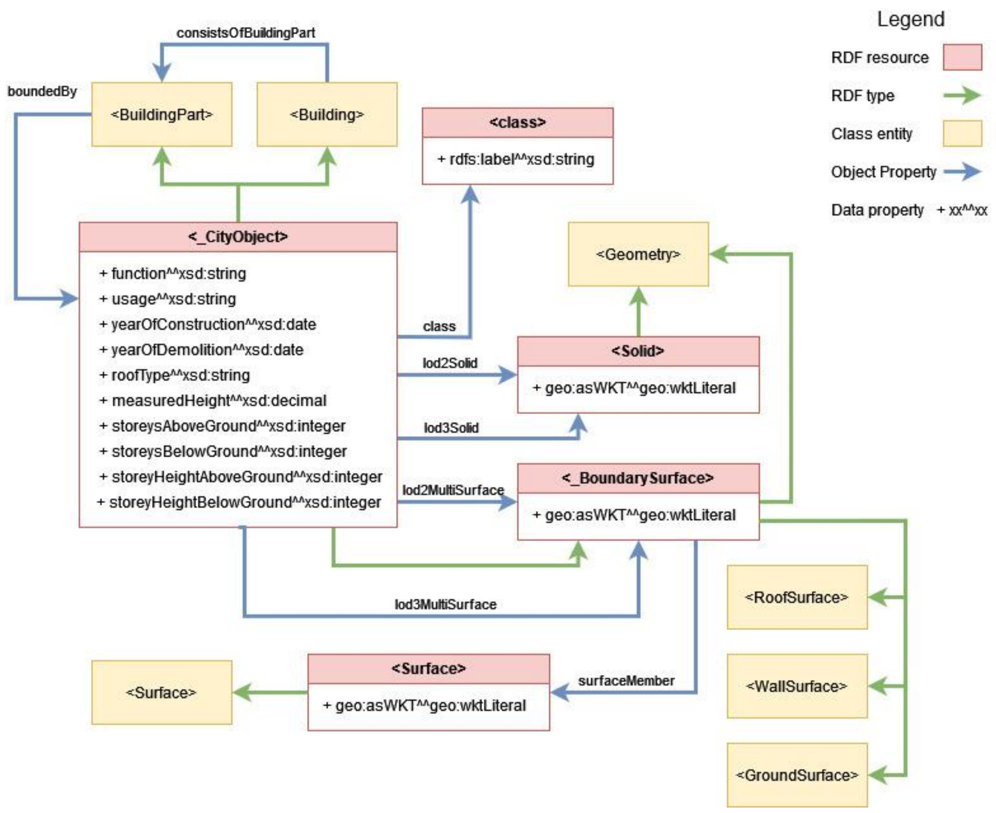
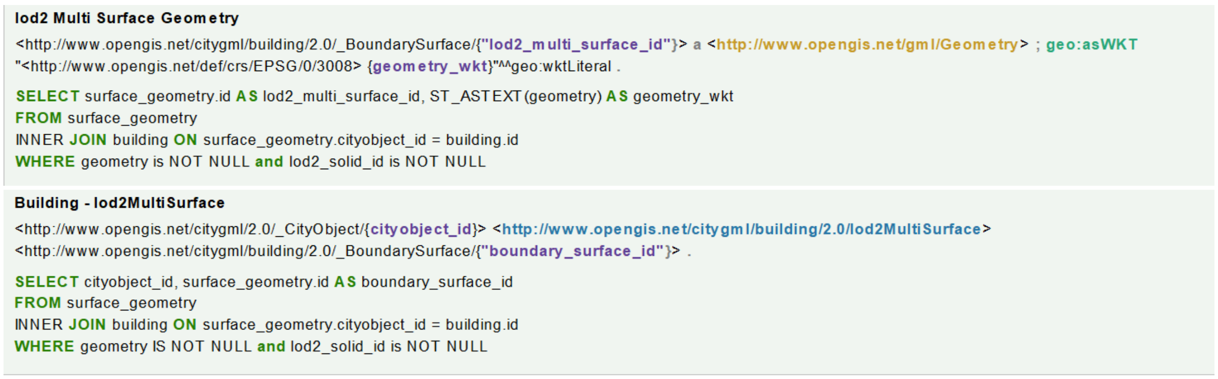


Figure 4.1: A diagram of the VKG mappings and the implemented data. The VKG features the building theme in CityGML with both LoD2 and LoD3 buildings.



lod2 Multi Surface Geometry

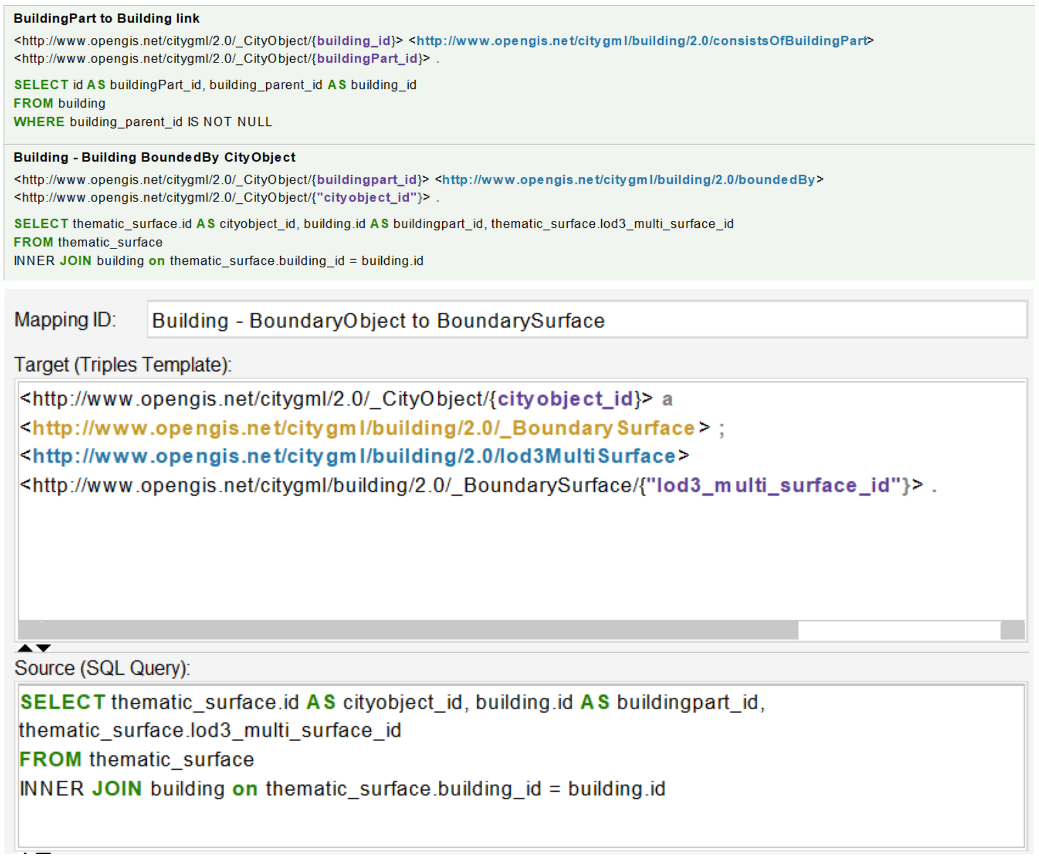
```
<http://www.opengis.net/citygml/building/2.0/_BoundarySurface/{"lod2_multi_surface_id"}> a <http://www.opengis.net/gml/Geometry> ; geo:asWKT
"<http://www.opengis.net/def/crs/EPSG/0/3008> {geometry_wkt}"^^geo:wktLiteral .

SELECT surface_geometry.id AS lod2_multi_surface_id, ST_ASTEXT(geometry) AS geometry_wkt
FROM surface_geometry
INNER JOIN building ON surface_geometry.cityobject_id = building.id
WHERE geometry is NOT NULL and lod2_solid_id is NOT NULL
```

Building - lod2MultiSurface

```
<http://www.opengis.net/citygml/2.0/_CityObject/{cityobject_id}> <http://www.opengis.net/citygml/building/2.0/lod2MultiSurface>
<http://www.opengis.net/citygml/building/2.0/_BoundarySurface/{"boundary_surface_id"}> .

SELECT cityobject_id, surface_geometry.id AS boundary_surface_id
FROM surface_geometry
INNER JOIN building ON surface_geometry.cityobject_id = building.id
WHERE geometry IS NOT NULL and lod2_solid_id is NOT NULL
```

Figure 4.2: The mappings made for representing LoD2 multi-surface geometries. LoD2 lacks the necessary details to individually identify and map the different surfaces as different classes.

To provide more detail about the mappings summarized in Figure 4.1, some additional mappings are shown in detail in Figure 4.2, 4.3 and 4.4. The multi-surface mapping for LoD2 buildings (see Figure 4.2) share similarities with the mapping shown in Figure 3.4 but due to the structure of the database schema and the data used in the study the SQL queries used to retrieve the data differs. The data does not contain any specific IDs for LoD2 multi-surfaces and therefore cannot directly be retrieved and referenced using the ID. Instead, the retrieval method used revolves around joining tables by *CityObject* IDs and retrieve the "*surface_geometry"* IDs as IDs representing boundary surfaces for buildings. This way, all the geometries associated with a specific building gets retrieved. This can then be filtered by only retrieving records where surface geometries are present and from LoD2 buildings. Surface geometries are geometries found in the *"geometry"* column. This method for mapping allows for the surface geometries to still be mapped using alternative IDs to still be able to retrieve geometries without further details.



BuildingPart to Building link

```
<http://www.opengis.net/citygml/2.0/_CityObject/{building_id}> <http://www.opengis.net/citygml/building/2.0/consistsOfBuildingPart>
<http://www.opengis.net/citygml/2.0/_CityObject/{buildingPart_id}> .

SELECT id AS buildingPart_id, building_parent_id AS building_id
FROM building
WHERE building_parent_id IS NOT NULL
```

Building - Building BoundedBy CityObject

```
<http://www.opengis.net/citygml/2.0/_CityObject/{buildingpart_id}> <http://www.opengis.net/citygml/building/2.0/boundedBy>
<http://www.opengis.net/citygml/2.0/_CityObject/{"cityobject_id"}> .

SELECT thematic_surface.id AS cityobject_id, building.id AS buildingpart_id, thematic_surface.lod3_multi_surface_id
FROM thematic_surface
INNER JOIN building on thematic_surface.building_id = building.id
```

Mapping ID: Building - BoundaryObject to BoundarySurface

Target (Triples Template):

```
<http://www.opengis.net/citygml/2.0/_CityObject/{cityobject_id}> a
<http://www.opengis.net/citygml/building/2.0/_BoundarySurface> ;
<http://www.opengis.net/citygml/building/2.0/lod3MultiSurface>
<http://www.opengis.net/citygml/building/2.0/_BoundarySurface/{"lod3_multi_surface_id"}> .
```

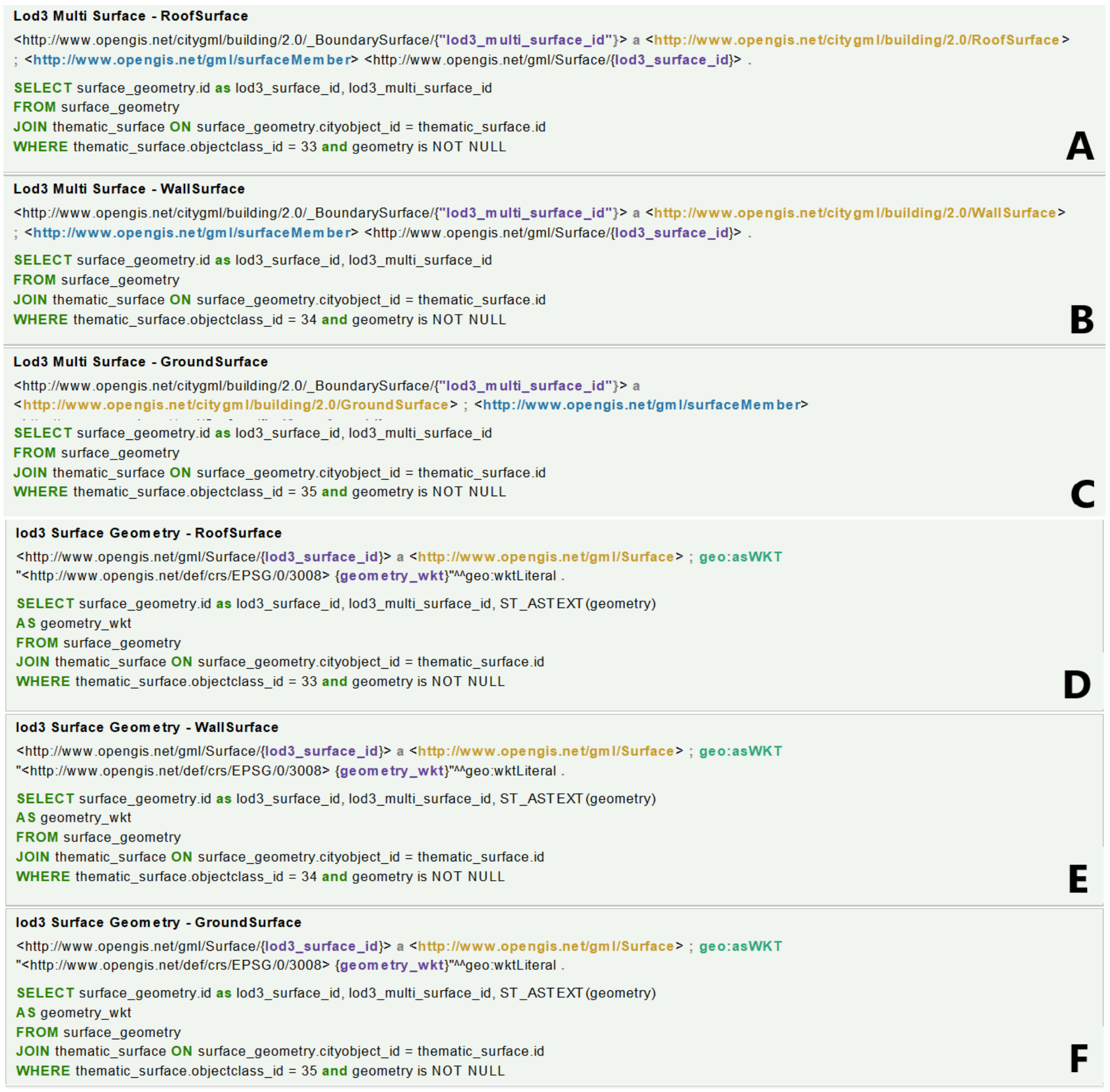
Source (SQL Query):

```
SELECT thematic_surface.id AS cityobject_id, building.id AS buildingpart_id,
thematic_surface.lod3_multi_surface_id
FROM thematic_surface
INNER JOIN building on thematic_surface.building_id = building.id
```

Figure 4.3: The mappings made to describe the relationship between Building and BuildingPart as well as BuildingParts bounded by other CityObjects. This mappings were necessary to correctly link surfaces to specific LoD3 buildings.

For mapping multi-surfaces in LoD3 buildings, the amount of mappings are substantially more in order to recreate the correct relationships in the KG. Figure 4.3 displays some of the mappings created to recreate the needed relationships describing how surfaces related to building objects and populate the graph with the appropriate data. The first mapping is made to link the *Building* and *BuildingPart* classes. This is necessary as the actual multi-surface are linked and referenced through the *BuildingPart* class in the database rather than *Building*. The SQL used to retrieve the necessary data is trivial in this case. The second mapping links the *BuildingPart* with the *CityObjects* representing the specific boundary objects for each building. These *CityObjects* are the actual objects representing the multi-surfaces and from these it is possible to derive their individual classes and surface geometries. The SQL used to retrieve the data is once again trivial, only joining two tables together.

The third mapping shown in Figure 4.3 is the mapping created to classify the previously mentioned *CityObjects* as *BoundarySurface* and to assign the objects with the *lod3MultiSurface* object property. The LoD3 multi-surface ID attribute associated with the *CityObjects* are used as object in the RDF triple and stored in the *BoundarySurface* IRI. This now populates the KG with unclassified composite surfaces that make up the boundary surfaces of each respective building in the dataset.



```
Lod3 Multi Surface - RoofSurface
<http://www.opengis.net/citygml/building/2.0/_BoundarySurface/{"lod3_multi_surface_id"}> a <http://www.opengis.net/citygml/building/2.0/RoofSurface>
; <http://www.opengis.net/gml/surfaceMember> <http://www.opengis.net/gml/Surface/{lod3_surface_id}> .

SELECT surface_geometry.id as lod3_surface_id, lod3_multi_surface_id
FROM surface_geometry
JOIN thematic_surface ON surface_geometry.cityobject_id = thematic_surface.id
WHERE thematic_surface.objectclass_id = 33 and geometry is NOT NULL
```

A

```
Lod3 Multi Surface - WallSurface
<http://www.opengis.net/citygml/building/2.0/_BoundarySurface/{"lod3_multi_surface_id"}> a <http://www.opengis.net/citygml/building/2.0/WallSurface>
; <http://www.opengis.net/gml/surfaceMember> <http://www.opengis.net/gml/Surface/{lod3_surface_id}> .

SELECT surface_geometry.id as lod3_surface_id, lod3_multi_surface_id
FROM surface_geometry
JOIN thematic_surface ON surface_geometry.cityobject_id = thematic_surface.id
WHERE thematic_surface.objectclass_id = 34 and geometry is NOT NULL
```

B

```
Lod3 Multi Surface - GroundSurface
<http://www.opengis.net/citygml/building/2.0/_BoundarySurface/{"lod3_multi_surface_id"}> a
<http://www.opengis.net/citygml/building/2.0/GroundSurface> ; <http://www.opengis.net/gml/surfaceMember>

SELECT surface_geometry.id as lod3_surface_id, lod3_multi_surface_id
FROM surface_geometry
JOIN thematic_surface ON surface_geometry.cityobject_id = thematic_surface.id
WHERE thematic_surface.objectclass_id = 35 and geometry is NOT NULL
```

C

```
lod3 Surface Geometry - RoofSurface
<http://www.opengis.net/gml/Surface/{lod3_surface_id}> a <http://www.opengis.net/gml/Surface> ; geo:asWKT
"<http://www.opengis.net/def/crs/EPSG/0/3008> {geometry_wkt}"^^geo:wktLiteral .

SELECT surface_geometry.id as lod3_surface_id, lod3_multi_surface_id, ST_ASTEXT(geometry)
AS geometry_wkt
FROM surface_geometry
JOIN thematic_surface ON surface_geometry.cityobject_id = thematic_surface.id
WHERE thematic_surface.objectclass_id = 33 and geometry is NOT NULL
```

D

```
lod3 Surface Geometry - WallSurface
<http://www.opengis.net/gml/Surface/{lod3_surface_id}> a <http://www.opengis.net/gml/Surface> ; geo:asWKT
"<http://www.opengis.net/def/crs/EPSG/0/3008> {geometry_wkt}"^^geo:wktLiteral .

SELECT surface_geometry.id as lod3_surface_id, lod3_multi_surface_id, ST_ASTEXT(geometry)
AS geometry_wkt
FROM surface_geometry
JOIN thematic_surface ON surface_geometry.cityobject_id = thematic_surface.id
WHERE thematic_surface.objectclass_id = 34 and geometry is NOT NULL
```

E

```
lod3 Surface Geometry - GroundSurface
<http://www.opengis.net/gml/Surface/{lod3_surface_id}> a <http://www.opengis.net/gml/Surface> ; geo:asWKT
"<http://www.opengis.net/def/crs/EPSG/0/3008> {geometry_wkt}"^^geo:wktLiteral .

SELECT surface_geometry.id as lod3_surface_id, lod3_multi_surface_id, ST_ASTEXT(geometry)
AS geometry_wkt
FROM surface_geometry
JOIN thematic_surface ON surface_geometry.cityobject_id = thematic_surface.id
WHERE thematic_surface.objectclass_id = 35 and geometry is NOT NULL
```

F

Figure 4.4: The mappings made to describe relationships from the CityObjects bounding the BuildingPart leading all the way to individual surface geometries for LoD3 buildings.

The next mappings done in the process of representing LoD3 geometry can be seen in Figure 4.4. The first three mappings (A, B and C) in the figure displays the different classifications done for the composite surfaces and the simultaneous linking of the individual surfaces' ID. To ensure that surfaces was classified as the correct one, three different mappings had to be written where the SQL queries were designed to filter the specific class and just retrieve those objects. The filtering is primarily controlled by filtering on *objectclass_id* in the *"thematic_surface"* table. The three classes filtered for were roof surfaces (class ID 33), wall surfaces (class ID 34) and ground surfaces (class ID 35). The individual surfaces that comprise the composite multi-surfaces are also mapped as *surfaceMembers* of their respective multi-surface object. This is enabled by joining the two tables *"surface_geometry"* and *"thematic_surface"* and retrieving the individual surface IDs through the SQL query.

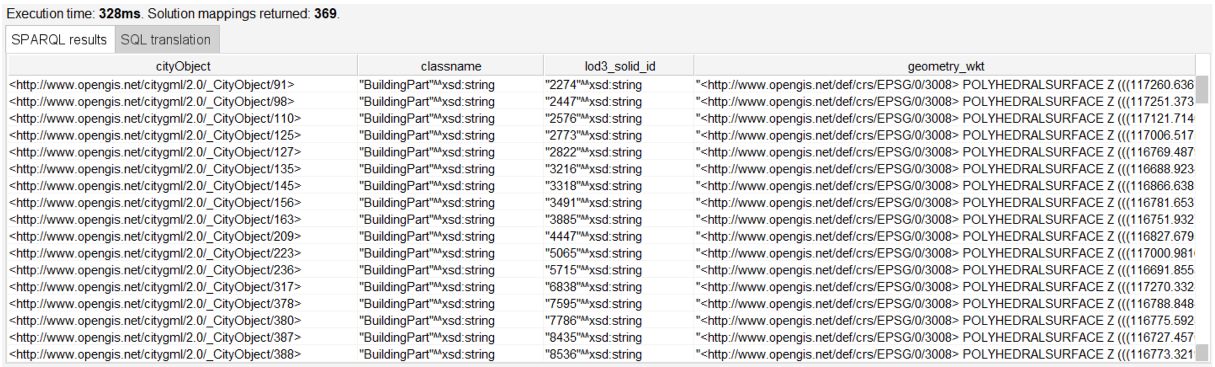
The following three mappings (D, E and F) follow a similar structure and method. They classify the individual surfaces as their respective class as well as defines the link between surface ID and the surface geometries as WKT. The SQL queries used for these mappings are similar to the previous three in Figure 4.4. The same filtering operations and joining are used with the primary difference being the retrieval of the actual geometries and casting them as WKT. With the these final three mappings, representation of the individual surfaces in LoD3 buildings has been achieved.

4.2. VKG evaluation

The VKG was validated and tested using the queries shown in section 3.3. The resulting table retrieved through SPARQL query 1 can be seen in Figure 4.5. The table shows an excerpt of all the retrieved *CityObjects* defined as having LoD3 geometry. In total 369 records were retrieved. All of the *CityObjects* belonged to the class *BuildingPart* and their respective LoD3 solid ID and the respective solid geometry was correctly retrieved. The *CityObjects* are all of the *BuildingPart* class because they are the *CityObjects* that are directly linked to the solid geometries. By cross-examining the resulting table with the results from an equal SQL query on the relational database it was concluded that SPARQL query 1 successfully retrieves all the expected records.

SPARQL query 1: Find all buildings assigned as LoD3 and their corresponding CityObject ID, class name, LoD3 solid ID and solid geometry. The prefix definitions are omitted for readability.

```
SELECT DISTINCT ?cityObject ?classname ?lod3_solid_id ?geometry_wkt
WHERE {
  ?cityObject building:lod3Solid ?lod3_solid_iri .
  ?lod3_solid_iri geosparql:asWKT ?geometry_wkt .

   BIND(STRAFTER(STR(?lod3_solid_iri), "http://www.opengis.net/gml/Solid/")
AS ?lod3_solid_id) .

  ?cityObject building:class ?class_id .
  ?class_id rdfs:label ?classname .
}
```


Execution time: **328ms**. Solution mappings returned: **369**.

SPARQL results | SQL translation

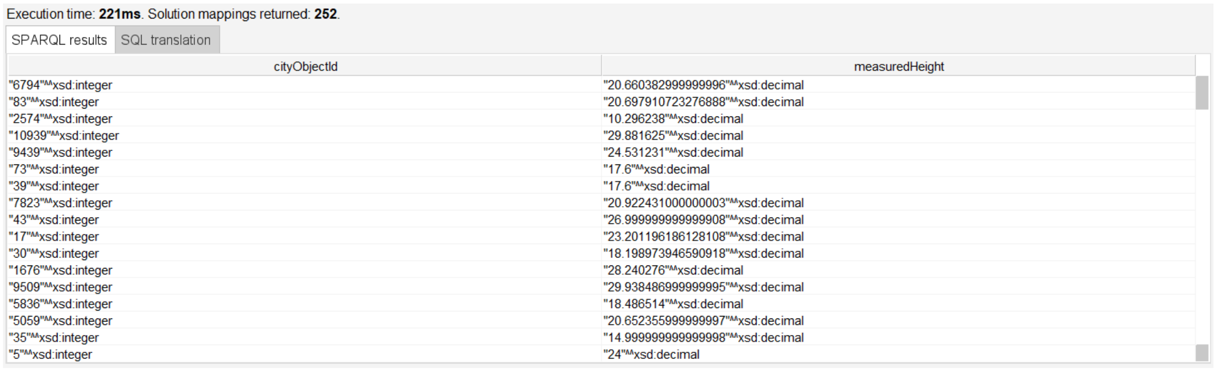
cityObject	classname	lod3_solid_id	geometry_wkt
<http://www.opengis.net/citygml/2.0/_CityObject/91>	"BuildingPart"^^xsd:string	"2274"^^xsd:string	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((117260.636
<http://www.opengis.net/citygml/2.0/_CityObject/98>	"BuildingPart"^^xsd:string	"2447"^^xsd:string	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((117251.373
<http://www.opengis.net/citygml/2.0/_CityObject/110>	"BuildingPart"^^xsd:string	"2576"^^xsd:string	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((117121.714
<http://www.opengis.net/citygml/2.0/_CityObject/125>	"BuildingPart"^^xsd:string	"2773"^^xsd:string	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((117006.517
<http://www.opengis.net/citygml/2.0/_CityObject/127>	"BuildingPart"^^xsd:string	"2822"^^xsd:string	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116769.487
<http://www.opengis.net/citygml/2.0/_CityObject/135>	"BuildingPart"^^xsd:string	"3216"^^xsd:string	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116688.923
<http://www.opengis.net/citygml/2.0/_CityObject/145>	"BuildingPart"^^xsd:string	"3318"^^xsd:string	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116866.638
<http://www.opengis.net/citygml/2.0/_CityObject/156>	"BuildingPart"^^xsd:string	"3491"^^xsd:string	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116781.653
<http://www.opengis.net/citygml/2.0/_CityObject/163>	"BuildingPart"^^xsd:string	"3885"^^xsd:string	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116751.932
<http://www.opengis.net/citygml/2.0/_CityObject/209>	"BuildingPart"^^xsd:string	"4447"^^xsd:string	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116827.679
<http://www.opengis.net/citygml/2.0/_CityObject/223>	"BuildingPart"^^xsd:string	"5065"^^xsd:string	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((117000.981
<http://www.opengis.net/citygml/2.0/_CityObject/236>	"BuildingPart"^^xsd:string	"5715"^^xsd:string	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116691.855
<http://www.opengis.net/citygml/2.0/_CityObject/317>	"BuildingPart"^^xsd:string	"6838"^^xsd:string	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((117270.332
<http://www.opengis.net/citygml/2.0/_CityObject/378>	"BuildingPart"^^xsd:string	"7595"^^xsd:string	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116788.848
<http://www.opengis.net/citygml/2.0/_CityObject/380>	"BuildingPart"^^xsd:string	"7786"^^xsd:string	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116775.592
<http://www.opengis.net/citygml/2.0/_CityObject/387>	"BuildingPart"^^xsd:string	"8435"^^xsd:string	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116727.457
<http://www.opengis.net/citygml/2.0/_CityObject/388>	"BuildingPart"^^xsd:string	"8536"^^xsd:string	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116773.321

Figure 4.5: Excerpt results from SPARQL query 1. The query intended to find all buildings assigned as LoD3 and their corresponding CityObject ID, class name, LoD3 solid ID and solid geometry.

The results from SPARQL query 2 can be seen in Figure 4.6. 252 records were retrieved by the query with similar attributes as the previous query. Each *CityObject* and the measured height of it can be seen in the table if the height is above 10 meters. The results in the figure were once again cross-examined against an equal SQL query in the relational database with the same results for both queries, concluding that the query works as intended and the graph is mapped correctly.

SPARQL query 2: Find all buildings with a measured height above 10 meters. The prefix definitions are omitted for readability.

```
SELECT ?cityObjectId ?measuredHeight
WHERE {
  ?cityObject a citygml:_CityObject ;
  building:measuredHeight ?measuredHeight .

  BIND (URI(?cityObject) AS ?cityObjectIdUri)
  BIND (xsd:integer(REPLACE(STR(?cityObjectIdUri),
"http://www.opengis.net/citygml/2.0/_CityObject/", "")) AS ?cityObjectId)

  FILTER (?measuredHeight > 10)
}
```


Execution time: **221ms**. Solution mappings returned: **252**.

SPARQL results | SQL translation

cityObjectId	measuredHeight
"6794"^^xsd:integer	"20.660382999999996"^^xsd:decimal
"83"^^xsd:integer	"20.697910723276888"^^xsd:decimal
"2574"^^xsd:integer	"10.296238"^^xsd:decimal
"10939"^^xsd:integer	"29.881625"^^xsd:decimal
"9439"^^xsd:integer	"24.531231"^^xsd:decimal
"73"^^xsd:integer	"17.6"^^xsd:decimal
"39"^^xsd:integer	"17.6"^^xsd:decimal
"7823"^^xsd:integer	"20.922431000000003"^^xsd:decimal
"43"^^xsd:integer	"26.999999999999908"^^xsd:decimal
"17"^^xsd:integer	"23.201196186128108"^^xsd:decimal
"30"^^xsd:integer	"18.198973946590918"^^xsd:decimal
"1676"^^xsd:integer	"28.240276"^^xsd:decimal
"9509"^^xsd:integer	"29.938486999999995"^^xsd:decimal
"5836"^^xsd:integer	"18.486514"^^xsd:decimal
"5059"^^xsd:integer	"20.652355999999997"^^xsd:decimal
"35"^^xsd:integer	"14.999999999999998"^^xsd:decimal
"5"^^xsd:integer	"24"^^xsd:decimal

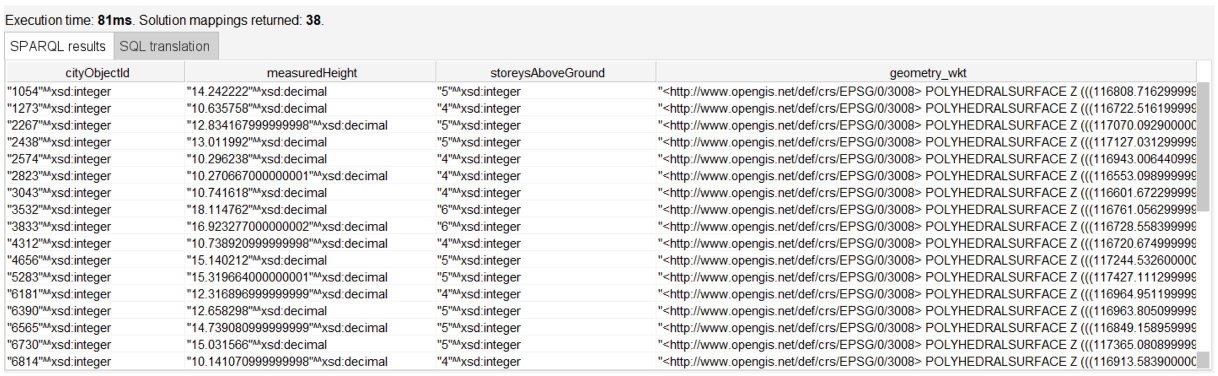
Figure 4.6: Excerpt results from SPARQL query 2. The query intended to find all buildings with a measured height above 10 meters.

The results from SPARQL query 3 can be seen in Figure 4.7. The results include the ID of each *CityObject* and their corresponding height, number of floors and solid geometry that fulfilled the requirement of being a LoD3 building with the number of floors being in the range of 4 to 6. The cross-examination against the relational database showed an equal number of *CityObjects* retrieved like the previous two queries.

SPARQL query 3: Find all buildings with 4 to 6 floors above ground. Retrieve the height and solid geometry of each building fulfilling the requirement. The prefix definitions are omitted for readability.

```
SELECT ?cityObjectId ?measuredHeight ?storeysAboveGround ?geometry_wkt
WHERE {
  ?cityObject a citygml:_CityObject ;
  building:storeysAboveGround ?storeysAboveGround ;
  building:measuredHeight ?measuredHeight .

  ?cityObject building:lod3Solid ?lod3_solid_iri .
  ?lod3_solid_iri geosparql:asWKT ?geometry_wkt .

  BIND (URI(?cityObject) AS ?cityObjectIdUri)
  BIND (xsd:integer(REPLACE(STR(?cityObjectIdUri),
"http://www.opengis.net/citygml/2.0/_CityObject/", "")) AS ?cityObjectId)
```

```
  FILTER (?storeysAboveGround >= 4 && ?storeysAboveGround <= 6)
}
```



Execution time: **81ms**. Solution mappings returned: **38**.

SPARQL results | SQL translation

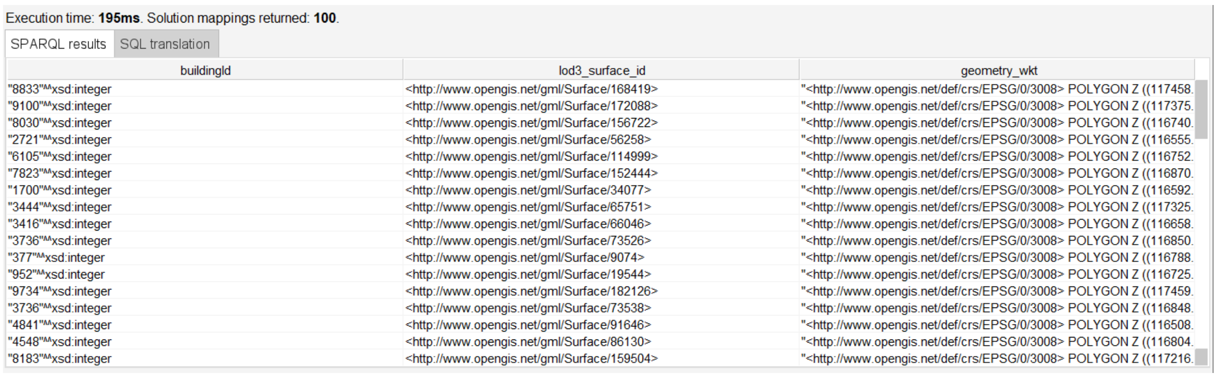
cityObjectId	measuredHeight	storeysAboveGround	geometry_wkt
"1054"^^xsd:integer	"14.242222"^^xsd:decimal	"5"^^xsd:integer	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116808.716299999
"1273"^^xsd:integer	"10.635758"^^xsd:decimal	"4"^^xsd:integer	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116722.516199999
"2267"^^xsd:integer	"12.834167999999998"^^xsd:decimal	"5"^^xsd:integer	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((117070.092900000
"2438"^^xsd:integer	"13.011992"^^xsd:decimal	"5"^^xsd:integer	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((117127.031299999
"2574"^^xsd:integer	"10.296238"^^xsd:decimal	"4"^^xsd:integer	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116943.006440999
"2823"^^xsd:integer	"10.270667000000001"^^xsd:decimal	"4"^^xsd:integer	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116553.098999999
"3043"^^xsd:integer	"10.741618"^^xsd:decimal	"4"^^xsd:integer	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116601.672299999
"3532"^^xsd:integer	"18.114762"^^xsd:decimal	"6"^^xsd:integer	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116761.056299999
"3833"^^xsd:integer	"16.923277000000002"^^xsd:decimal	"6"^^xsd:integer	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116728.558399999
"4312"^^xsd:integer	"10.738920999999998"^^xsd:decimal	"4"^^xsd:integer	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116720.674999999
"4656"^^xsd:integer	"15.140212"^^xsd:decimal	"5"^^xsd:integer	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((117244.532600000
"5283"^^xsd:integer	"15.319664000000001"^^xsd:decimal	"5"^^xsd:integer	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((117427.111299999
"6181"^^xsd:integer	"12.316896999999999"^^xsd:decimal	"4"^^xsd:integer	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116964.951199999
"6390"^^xsd:integer	"12.658298"^^xsd:decimal	"5"^^xsd:integer	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116963.805099999
"6565"^^xsd:integer	"14.739080999999999"^^xsd:decimal	"5"^^xsd:integer	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116849.158959999
"6730"^^xsd:integer	"15.031566"^^xsd:decimal	"5"^^xsd:integer	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((117365.080899999
"6814"^^xsd:integer	"10.141070999999998"^^xsd:decimal	"4"^^xsd:integer	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYHEDRALSURFACE Z (((116913.583900000

Figure 4.7: Results from SPARQL query 3. The query intended to find all buildings with 4 to 6 floors above ground while retrieving the height and solid geometry of each building fulfilling the requirement.

The resulting table for SPARQL query 4 is displayed in Figure 4.8. The query intended to test the long chain of predicates that was mapped to retrieve a specific class of surfaces from LoD3 buildings. The resulting table shows each respective building's *CityObject* ID along with the surface IDs and geometries associated with the building. The amount of retrieved records have been limited to 100. Although it is not apparent in the result shown in the figure each of the buildings can be associated with multiple ground surfaces, since they are designed to be multi-surface composite geometries. This results was not cross-examined against the original data structure as just testing that the mappings work were deemed enough to consider the graph validated.

SPARQL query 4: Find all ground surfaces associated with LoD3 buildings and their respective surface IDs. The prefix definitions are omitted for readability.

```
SELECT ?buildingId ?lod3_surface_id ?geometry_wkt
WHERE {
  ?boundaryObject building:lod3MultiSurface ?lod3_multi_id.
  ?lod3_multi_id a building:GroundSurface ;
          geom:surfaceMember ?lod3_surface_id .
  ?lod3_surface_id geo:asWKT ?geometry_wkt .
  ?building building:consistsOfBuildingPart ?buildingPart .
  ?buildingPart building:boundedBy ?boundaryObject

  BIND (URI(?building) AS ?cityObjectIdUri)
  BIND (xsd:integer(REPLACE(STR(?cityObjectIdUri),
"http://www.opengis.net/citygml/2.0/_CityObject/", "")) AS ?buildingId)
}
```


Execution time: **195ms**. Solution mappings returned: **100**.

SPARQL results | SQL translation

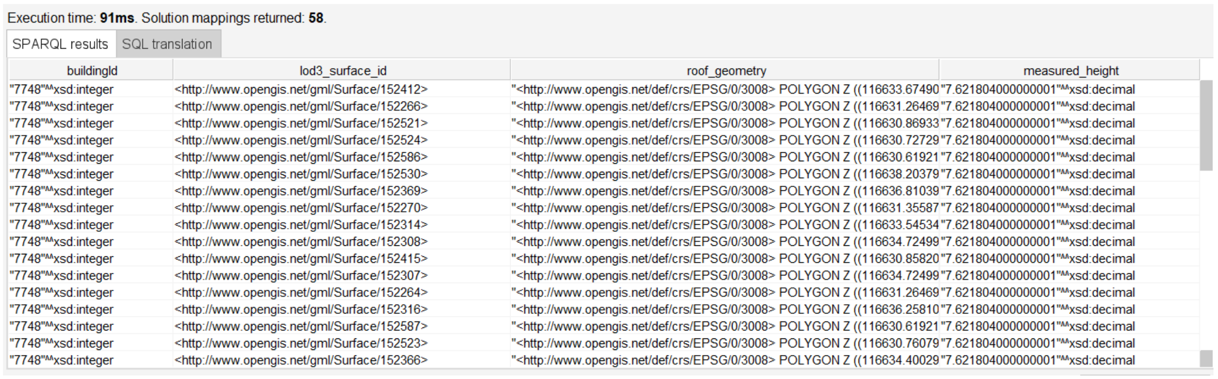
buildingId	lod3_surface_id	geometry_wkt
"8833"^^xsd:integer	<http://www.opengis.net/gml/Surface/168419>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((117458.
"9100"^^xsd:integer	<http://www.opengis.net/gml/Surface/172088>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((117375.
"8030"^^xsd:integer	<http://www.opengis.net/gml/Surface/156722>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116740.
"2721"^^xsd:integer	<http://www.opengis.net/gml/Surface/56258>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116555.
"6105"^^xsd:integer	<http://www.opengis.net/gml/Surface/114999>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116752.
"7823"^^xsd:integer	<http://www.opengis.net/gml/Surface/152444>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116870.
"1700"^^xsd:integer	<http://www.opengis.net/gml/Surface/34077>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116592.
"3444"^^xsd:integer	<http://www.opengis.net/gml/Surface/65751>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((117325.
"3416"^^xsd:integer	<http://www.opengis.net/gml/Surface/66046>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116658.
"3736"^^xsd:integer	<http://www.opengis.net/gml/Surface/73526>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116850.
"377"^^xsd:integer	<http://www.opengis.net/gml/Surface/9074>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116788.
"952"^^xsd:integer	<http://www.opengis.net/gml/Surface/19544>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116725.
"9734"^^xsd:integer	<http://www.opengis.net/gml/Surface/182126>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((117459.
"3736"^^xsd:integer	<http://www.opengis.net/gml/Surface/73538>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116848.
"4841"^^xsd:integer	<http://www.opengis.net/gml/Surface/91646>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116508.
"4548"^^xsd:integer	<http://www.opengis.net/gml/Surface/86130>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116804.
"8183"^^xsd:integer	<http://www.opengis.net/gml/Surface/159504>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((117216.

Figure 4.8: Excerpt results from SPARQL query 4. The query intended to find all ground surfaces associated with LoD3 buildings and their respective surface IDs.

The results from SPARQL query 5 (see figure 4.9) is similar to what was displayed in query 4. Instead of ground surfaces the query now retrieves all the roof surfaces of a select building, in this case the building with the *CityObject* ID of 7748. 58 records are retrieved indicating that the building in question consists of 58 different surfaces that comprises the whole roof of the building. The measured height is also retrieved and shown to be the same for all of the surfaces, which is correct given that they are all associated with the same building. The resulting table from the query was cross-examined against the relational database which retrieved the same results using an equivalent SQL query.

SPARQL query 5: Find the height and roof geometry of the specific building with the CityObject ID 7748. The prefix definitions are omitted for readability.

```
SELECT ?buildingId ?lod3_surface_id ?roof_geometry ?measured_height
WHERE {
  ?boundaryObject building:lod3MultiSurface ?lod3_multi_id.
  ?lod3_multi_id a building:RoofSurface ;
           geom:surfaceMember ?lod3_surface_id .
  ?lod3_surface_id geo:asWKT ?roof_geometry .

  BIND(cityobject:7748 AS ?building)
  ?building building:consistsOfBuildingPart ?buildingPart ;
           building:measuredHeight ?measured_height .
  ?buildingPart building:boundedBy ?boundaryObject .

  BIND (URI(?building) AS ?cityObjectIdUri)
  BIND (xsd:integer(REPLACE(STR(?cityObjectIdUri),
"http://www.opengis.net/citygml/2.0/_CityObject/", "")) AS ?buildingId)
}
```


Execution time: **91ms**. Solution mappings returned: **58**.

SPARQL results | SQL translation

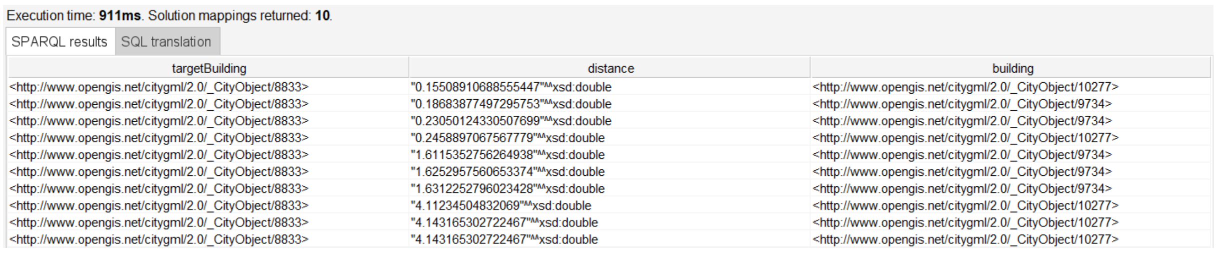
buildingId	lod3_surface_id	roof_geometry	measured_height
"7748"^^xsd:integer	<http://www.opengis.net/gml/Surface/152412>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116633.67490	"7.621804000000001"^^xsd:decimal
"7748"^^xsd:integer	<http://www.opengis.net/gml/Surface/152266>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116631.26469	"7.621804000000001"^^xsd:decimal
"7748"^^xsd:integer	<http://www.opengis.net/gml/Surface/152521>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116630.86933	"7.621804000000001"^^xsd:decimal
"7748"^^xsd:integer	<http://www.opengis.net/gml/Surface/152524>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116630.72729	"7.621804000000001"^^xsd:decimal
"7748"^^xsd:integer	<http://www.opengis.net/gml/Surface/152586>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116630.61921	"7.621804000000001"^^xsd:decimal
"7748"^^xsd:integer	<http://www.opengis.net/gml/Surface/152530>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116638.20379	"7.621804000000001"^^xsd:decimal
"7748"^^xsd:integer	<http://www.opengis.net/gml/Surface/152369>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116636.81039	"7.621804000000001"^^xsd:decimal
"7748"^^xsd:integer	<http://www.opengis.net/gml/Surface/152270>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116631.35587	"7.621804000000001"^^xsd:decimal
"7748"^^xsd:integer	<http://www.opengis.net/gml/Surface/152314>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116633.54534	"7.621804000000001"^^xsd:decimal
"7748"^^xsd:integer	<http://www.opengis.net/gml/Surface/152308>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116634.72499	"7.621804000000001"^^xsd:decimal
"7748"^^xsd:integer	<http://www.opengis.net/gml/Surface/152415>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116630.85820	"7.621804000000001"^^xsd:decimal
"7748"^^xsd:integer	<http://www.opengis.net/gml/Surface/152307>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116634.72499	"7.621804000000001"^^xsd:decimal
"7748"^^xsd:integer	<http://www.opengis.net/gml/Surface/152264>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116631.26469	"7.621804000000001"^^xsd:decimal
"7748"^^xsd:integer	<http://www.opengis.net/gml/Surface/152316>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116636.25810	"7.621804000000001"^^xsd:decimal
"7748"^^xsd:integer	<http://www.opengis.net/gml/Surface/152587>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116630.61921	"7.621804000000001"^^xsd:decimal
"7748"^^xsd:integer	<http://www.opengis.net/gml/Surface/152523>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116630.76079	"7.621804000000001"^^xsd:decimal
"7748"^^xsd:integer	<http://www.opengis.net/gml/Surface/152366>	"<http://www.opengis.net/def/crs/EPSG/0/3008> POLYGON Z ((116634.40029	"7.621804000000001"^^xsd:decimal

Figure 4.9: Results from SPARQL query 5. The query intended to find the height and roof geometry of the specific building with the CityObject ID 7748.

The result from the SPARQL query 6 (see Figure 4.10) is an example of what information can be retrieved utilizing spatial functions present in GeoSPARQL. The query returns 10 records in total which displays the target building that was used in the query in the first column. The second column of the table shows the distance from the target building and the specific ground surface to ground surfaces in other buildings. The query only retrieves cases where the distance calculated was 5 meters or less. In reality, the number of buildings that fulfilled this requirement was two, as the third column that shows the respective nearby buildings only include two unique *CityObject* IDs.

SPARQL query 6: Find buildings with a 5 meter or less distance from the building with CityObject ID 8833. The prefix definitions are omitted for readability.

```
SELECT ?targetBuilding ?distance ?building
WHERE {
  ?boundaryObject building:lod3MultiSurface ?lod3_multi_id.
  ?lod3_multi_id a building:GroundSurface ;
          geom:surfaceMember ?lod3_surface_id .
  ?lod3_surface_id geo:asWKT ?geometry_wkt .

  surface:168419 geo:asWKT ?targetGeometry .
  BIND(cityobject:8833 AS ?targetBuilding)

  ?building building:consistsOfBuildingPart ?buildingPart .
  ?buildingPart building:boundedBy ?boundaryObject .

  BIND(geof:distance(?targetGeometry, ?geometry_wkt,
<http://www.opengis.net/def/uom/OGC/1.0/metre>) AS ?distance)
  FILTER(?building != ?targetBuilding)
  FILTER(?distance <= 5)
}
ORDER BY ASC(?distance)
```


Execution time: **911ms**. Solution mappings returned: **10**.

SPARQL results | SQL translation

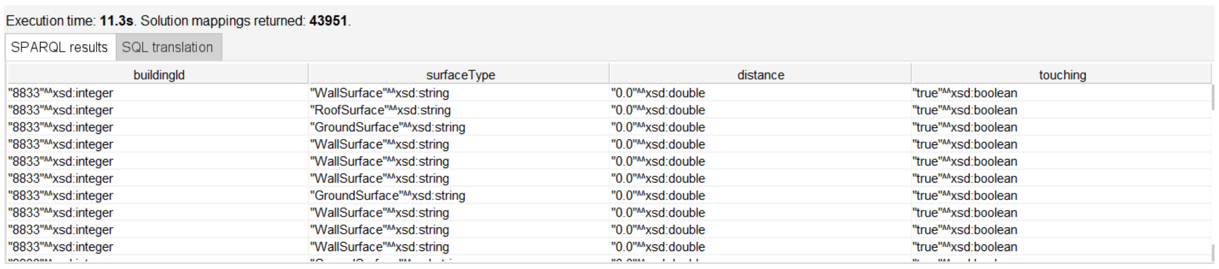
targetBuilding	distance	building
<http://www.opengis.net/citygml/2.0/_CityObject/8833>	"0.15508910688555447"^^xsd:double	<http://www.opengis.net/citygml/2.0/_CityObject/10277>
<http://www.opengis.net/citygml/2.0/_CityObject/8833>	"0.18683877497295753"^^xsd:double	<http://www.opengis.net/citygml/2.0/_CityObject/9734>
<http://www.opengis.net/citygml/2.0/_CityObject/8833>	"0.23050124330507699"^^xsd:double	<http://www.opengis.net/citygml/2.0/_CityObject/9734>
<http://www.opengis.net/citygml/2.0/_CityObject/8833>	"0.2458897067567779"^^xsd:double	<http://www.opengis.net/citygml/2.0/_CityObject/10277>
<http://www.opengis.net/citygml/2.0/_CityObject/8833>	"1.6115352756264938"^^xsd:double	<http://www.opengis.net/citygml/2.0/_CityObject/9734>
<http://www.opengis.net/citygml/2.0/_CityObject/8833>	"1.6252957560653374"^^xsd:double	<http://www.opengis.net/citygml/2.0/_CityObject/9734>
<http://www.opengis.net/citygml/2.0/_CityObject/8833>	"1.6312252796023428"^^xsd:double	<http://www.opengis.net/citygml/2.0/_CityObject/9734>
<http://www.opengis.net/citygml/2.0/_CityObject/8833>	"4.11234504832069"^^xsd:double	<http://www.opengis.net/citygml/2.0/_CityObject/10277>
<http://www.opengis.net/citygml/2.0/_CityObject/8833>	"4.143165302722467"^^xsd:double	<http://www.opengis.net/citygml/2.0/_CityObject/10277>
<http://www.opengis.net/citygml/2.0/_CityObject/8833>	"4.143165302722467"^^xsd:double	<http://www.opengis.net/citygml/2.0/_CityObject/10277>

Figure: 4.10: Excerpt results from SPARQL query 6. The query intended to find buildings with a 5 meter or less distance from the building with CityObject ID 8833.

The results from the SPARQL query 7 is shown in Figure 4.11. Besides the calculated distance, the results now also provide boolean values for if surfaces touch the ground surface of the target building. The results also retrieve all other surfaces and calculates the distance from the target surface to those. The type of surface is also retrieved and shown in the results. For this specific query, the execution time for completing the query was 11.3 seconds. When performing the equivalent query in SQL over the relational database it retrieves the same number of records, that being 43951. The execution time for the query when performed over the relational database was 5.9 seconds.

SPARQL query 7: Retrieve the distance between CityObject 8833 and all other surfaces. Also find the surfaces that touches the ground surface in CityObject 8833.

```
SELECT ?buildingId ?surfaceType ?distance ?touching
WHERE {
  ?boundaryObject building:lod3MultiSurface ?lod3_multi_id.
  ?lod3_multi_id a ?surface_type ;
    geom:surfaceMember ?lod3_surface_id .
  ?lod3_surface_id geo:asWKT ?geometry_wkt .
  ?building building:consistsOfBuildingPart ?buildingPart .
  ?buildingPart building:boundedBy ?boundaryObject .

  surface:168419 geo:asWKT ?targetGeometry .
  BIND(cityobject:8833 AS ?targetBuilding)

  BIND(geof:sfTouches(?targetGeometry, ?geometry_wkt) AS ?touching)
  BIND(geof:distance(?targetGeometry, ?geometry_wkt,
<http://www.opengis.net/def/uom/OGC/1.0/metre>) AS ?distance)

  FILTER(?surface_type = building:GroundSurface || ?surface_type =
building:WallSurface || ?surface_type = building:RoofSurface)

  BIND (URI(?building) AS ?cityObjectIdUri)
  BIND (xsd:integer(REPLACE(str(?cityObjectIdUri),
"http://www.opengis.net/citygml/2.0/_CityObject/", "")) AS ?buildingId)

  BIND (URI(?targetBuilding) AS ?TargetUri)
  BIND (xsd:integer(REPLACE(str(?TargetUri),
"http://www.opengis.net/citygml/2.0/_CityObject/", "")) AS
?targetBuildingId)

  BIND (SUBSTR(str(?surface_type),
STRLEN(str("http://www.opengis.net/citygml/building/2.0/")) + 1) AS
?surfaceType)
}
ORDER BY DESC(?touching)
```



Execution time: **11.3s**. Solution mappings returned: **43951**.

SPARQL results | SQL translation

buildingId	surfaceType	distance	touching
"8833"^^xsd:integer	"WallSurface"^^xsd:string	"0.0"^^xsd:double	"true"^^xsd:boolean
"8833"^^xsd:integer	"RoofSurface"^^xsd:string	"0.0"^^xsd:double	"true"^^xsd:boolean
"8833"^^xsd:integer	"GroundSurface"^^xsd:string	"0.0"^^xsd:double	"true"^^xsd:boolean
"8833"^^xsd:integer	"WallSurface"^^xsd:string	"0.0"^^xsd:double	"true"^^xsd:boolean
"8833"^^xsd:integer	"WallSurface"^^xsd:string	"0.0"^^xsd:double	"true"^^xsd:boolean
"8833"^^xsd:integer	"WallSurface"^^xsd:string	"0.0"^^xsd:double	"true"^^xsd:boolean
"8833"^^xsd:integer	"GroundSurface"^^xsd:string	"0.0"^^xsd:double	"true"^^xsd:boolean
"8833"^^xsd:integer	"WallSurface"^^xsd:string	"0.0"^^xsd:double	"true"^^xsd:boolean
"8833"^^xsd:integer	"WallSurface"^^xsd:string	"0.0"^^xsd:double	"true"^^xsd:boolean
"8833"^^xsd:integer	"WallSurface"^^xsd:string	"0.0"^^xsd:double	"true"^^xsd:boolean

Figure 4.11: Excerpt results from SPARQL query 7. The query intended to find surfaces touching the ground surface of the building with CityObject ID 8833. It also retrieved the distance to all surface geometries.

5. Discussion

The representation of CityGML data as a VKG was deemed successful in presenting building information through the mappings provided. The provided framework and implementation through the method is easy to use from a user perspective due to the relatively few mappings needed in order to present basic geometry and attribute data for buildings. The queries performed confirms that the representation is complete and robust by retrieving the expected information and that query expression used over the KG functions as intended. While the results show successful implementation, below discussions gives some insight into challenges and limitations discovered during the implementation process as well as future possibilities for the approach.

5.1. Evaluation of the VKG representation

While the implementation of the VKG was successful, the approach does present some user-centred challenges that are interesting to highlight. The most challenging part of creating a VKG representation is the correlation of the 3DCityDB database schema and ontological concepts. The database provides a large number of *CityObjects* associated with different classes. The need to occasionally map building *CityObjects* to surface *CityObjects* through different object property predicates requires good understanding of the data and the CityGML standard to make the correct and appropriate choices of ontological concepts for the RDF definitions. Identifying the proper class to assign to a *CityObject* is also a challenging prospect in the process due to the way the database structures and relates the CityGML data across different tables. The user will most likely make decisions regarding classification of *CityObjects* and predicates utilizing the additional information, such as the provided CityGML object classes. For example, some *CityObjects* represent thematic surfaces in the form of walls, roofs and ground and can be identified through the object class ID. This level of detail is only available in LoD3 buildings in the database schema used. LoD2 buildings does contain a level of detail that does include modelling of roof structures. However, the individual roof surface geometry is not available for induvial mapping like it is for LoD3 buildings. The roof structure is available in the solid geometry but not as an identifiable individual geometry which limits potential queries and analysis with buildings in LoD2. The thematic surfaces available in LoD3 can easily be classified in the mapping through this knowledge but the user also has to consider what predicates would be appropriate as a means to semantically link the *CityObjects*. A wall surface is a boundary surface and the UML for CityGML specifies the relationship between them as the aggregation described as *"boundedBy"* (Open Geospatial Consortium, 2012). However, mapping the corresponding *CityObject* ID to the *"_BoundarySurface"* IRI would result in misleading information as while the object is a boundary surface, the ID does not necessarily correspond to the surface ID. While this deduction can be considered somewhat trivial, the thought process behind correlating different *CityObjects* can be unintuitive depending on the structure of the used database.

The queries were performed as a means to test and validate the representation and displayed sufficient completeness and robustness. The completeness can see seen through the successful querying the relevant semantical relationships detailing geometry and attributes for LoD3 and LoD2 buildings. The showcased detail for LoD3 buildings by being able to query different surfaces is especially interesting, as previous studies have highlighted a paucity in representing this LoD in KG approaches (Ding et al., 2024). This paper provides novelty in the framework by showcasing the implementation of LoD3 *CityObjects* in a KG context. The robustness is evident through the fact that the queries over the VKG returned the expected results in all the tested queries.

Furthermore, the queries were intended as a practical showcase of how information can be retrieved for practical analysis purposes. However, the queries are not without limitations. While they are inspired by what information is commonly useful in the building permit process, such as building height and distance to nearby buildings, the queries themselves are not equal to the analysis done for building permits. In detail development planned areas, constructions and renovations are required to adhere to the plans and are tested on if the changes are suitable with regards to purpose, shape and material (Boverket, 2020). The queries can be seen as a step in retrieving the information needed to conduct the necessary analysis for that purpose.

One limitation of the framework presented is the fact that the framework is specifically implemented based on the CityGML data that was provided for the study. The types of information stored and how it is encoded in CityGML data varies based on the producer and country (Uggla et al., 2023). The mappings that were made only mapped the data that was available in the actual dataset, which means that certain classes and properties within the CityGML ontology was never mapped. For example, only LoD2 and LoD3 buildings were present in the dataset so only those two LoDs were mapped, leaving LoD0, LoD1 and LoD4 unmapped. This impacts the reusability of the specific framework provided in this study. However, the proposed framework can be expanded upon in future works in order to increase the reusability. The presented framework is intended to showcase the general process and give enough details to be able to apply the method to expand the mappings and to map other themes in the CityGML structure.

5.2. Effects of the database schema and ontology

3DCityDB and Protégé together with the Ontop plugin are central components of the system and workflow presented in this paper. While the tools could successfully be used to construct a VKG representing CityGML data to an adequate degree, it is worthwhile to evaluate the effectiveness of these tools. 3DcityDB is especially interesting to evaluate within the context of the presented workflow due to it prevalence in the urban planning environment in Sweden (Smart Built Environment, 2023a).

The database schema provided by 3DCityDB provided a total of 66 tables where very few were actually used for the mapping process. This is due to many of the tables being associated with the different themes and modules that are present in the CityGML 2.0 data structure as well as the fact that only the building theme was used in this study. It could be argued that many of the tables become redundant in the cases where just one or very few themes are imported into the database. The tables *"building"*, *"thematic_surface"* and *"surface_geometry"* are the three tables that are central to mapping the geometries correctly and therefore important to understand their structure and relation. While the documentation provides enough insight as to how these are related, the separation of IDs for different geometries between the two tables can appear unintuitive for the user. The table *"thematic_surface"* is used to store the ID and information regarding all the geometries that comprise the thematic boundary features of a building. This table provides the necessary information to be able to classify a geometry as one of the thematic surfaces within the CityGML structure. However, the actual geometries of these surfaces are stored in the *"surface_geometry"* table together with all other geometries, such as solid and implicit. Creating the full representation through the mapping from a base *CityObject* representing a building to the individual thematic surfaces could be argued as being inefficient. This is because of the need to join all three tables together in the SQL queries to retrieve the needed data. From a perspective of mapping the data to a knowledge graph, it would be more efficient to create tables requiring less joining of the tables and lessen the number of total tables needed.

It is also worthwhile to highlight the "surface_geometry" table, as the table presents geometry data in an aggregated structure that is significantly different from the structure in the ontology. While the implementation does not significantly increase the complexity of the mappings, it does create a need for the user to consider the correct correspondence and translation from database to ontology. This in combination with the aforementioned structure of the "thematic_surface" table results in the logical linking and relations between the tables and the data they hold to be the most difficult and user-centric parts of the representation process.

Protégé and Ontop provided the necessary platform for managing the CityGML 2.0 ontology and mapping. A tool for managing ontologies is required in order to handle the known inconsistencies in the CityGML 2.0 ontology from the University of Geneva. From a practical perspective when considering the scalability of the workflow presented, it is important to alter the ontology as little as possible and employ unified data standards. Countries have been found to have different standards for encoding CityGML data. For example, Sweden is currently working with the 3CIM ADE standard and implementing it on national level (Uggla et al., 2023). Another example is Estonia that encodes addresses together with geometries and links buildings to its corresponding surface geometries rather than solid geometries (Ding et al., 2024). Adding properties to the ontology to better fit how CityGML data is structured in 3DCityDB or any other system would lead to inconsistencies between the original data structure and the KG representation. Developing new ontologies based on ADE is likely a better solution in the cases where adding properties and classes to an ontology is of interest.

The used ontology itself is also another limitation of the study. The known inconsistencies in the ontology had to be handled in order for the reasoner engine to function, which enables the mappings. The chosen solution in the case of the study was to remove illogical object and data properties. This is not the only way to handle inconsistencies. It is possible to also handle these inconsistencies by renaming the IRIs, ensuring that each of them is unique. This would mean that certain concepts in the ontology would no longer align with the UML diagrams but this could also be the case when removing properties completely. The best method for handling inconsistency could vary depending on the use case and database schema and it is difficult to say if the approach used in this study is better than others. The approach used in this study is acceptable within the context of the data used and use case but it might not be the case in general.

Besides the mapping tools that Ontop provides, it also includes the necessary SPARQL endpoint allowing for querying of the VKG. For query 1-6, the performance and execution time were good with all of them retrieving the correct results under 1 second. The design of query 7 was however intended to push the complexity of the query by incorporating multiple spatial operations while also retrieving a significantly higher amount of results than the other queries. The execution time for query 7 was significantly slower than the other 6 queries. Compared to the execution time of the equivalent SQL query over the relational database, the SPARQL query took double the amount of time to execute. While the execution time is relatively slow and slower than when querying over a relational database, the data retrieval efficiency is still acceptable. This is likely due to Ontop's ability to leverage underlying indexing functionality in the original database through SPARQL to SQL translations. The database utilized GiST indexing for geometries which is a very flexible indexing method that adapts different indexes depending on the data type (Rogov, 2019). For example, in the case of geometric objects, GiST can adapt an R-tree index since it supports relative position locators rather than just the "greater", "less" and "equal" operators that B-tree is capable of. The flexibility of GiST does not necessarily mean that it is a spatial index but it is able to adapt spatial indexing to better support querying of geometric objects. With this in mind, it becomes apparent that adapting indexing in the underlying relational database is important for the VKG approach as a means to increase the efficiency of querying. The study proves that increasingly complex queries utilizing spatial operation increases execution time significantly even when spatial indexing is used. To ensure that querying is as efficient as possible, spatial indexing should be used in the relational database when implementing a VKG approach.

5.3. Applications and future research

The results of the study show that it is possible to develop complete and robust knowledge graph representations of CityGML building objects in a simple and effective manner. The implementation can be considered effective because of the relatively straightforward and simple implementation process showcased through the results. The queries are able to retrieve basic information regarding building data and while this does contribute to the prospects of more efficient interoperability, there is more practical use cases that can be tested and more complex queries can be performed. Future work should look into the possibilities of performing more complex queries, especially incorporating more spatial functionalities over 3D geometries in the queries. The implementation of querying 3D geometries such as polyhedral surfaces is still limited in Ontop despite being able to leverage relational spatial databases. Linking and integrating different types of data and performing spatial queries over a KG representation is also subject for future research within the KG representation approach. The highlighted Swedish cities all expressed a need for methods to integrate more data with 3D city models. This study limits itself to developing a framework for part of the process that can be required for uses cases such as improving interoperability between data structures, where this would be a first step in such a case. Future research should look into the specifics of the interoperability needs and how this can be implemented to take the next step for the KG representation approach.

The building theme within CityGML was the focus of this study which leaves opportunity to further study the same approach for the other remaining themes in CityGML, such as the transportation theme. The transportation theme would be of particular interest to implement using the KG approach as the ability to perform more complex analysis and management for infrastructure is something that is sought after by the cities that currently have and maintain digital twin models (Lehtola et al., 2022).

Furthermore, with CityGML 3.0 significantly altering the established CityGML paradigm by reworking the current LoD structure there will be a future need to implement the KG approach for that version of CityGML. A new ontology have been developed for this version which naturally introduces significant change in implementation framework for this approach (Vinasco-Alvarez et al., 2022). As of the writing of this thesis, support for CityGML 3.0 by the tools used in the proposed system and framework has yet to be implemented. 3DCityDB is undergoing development with version 5 to implement support for CityGML 3.0 together with a new and improved database schema (3DCityDB Team, 2019). A new database schema also means that the necessary mappings to expose the CityGML data needs to be revised with respect to the new schema and ontology.

6. Conclusions

The aim and intention for this thesis was to investigate how CityGML could be represented as a virtual knowledge graph in an effective manner for urban planning applications and how queries can be carried out using the representation for urban planning purposes. The framework used in this thesis proves to be effective in modelling CityGML data as knowledge graphs. The 20 mappings created were sufficient in creating a VKG representation of the data. The VKG was constructed to present all relevant geometry, attributes and semantic relationships of building objects in both LoD2 and LoD3. For LoD2, both solid and surface geometries were mapped as well as buildings attributes, in this case measured height and number of floors. The same respective data was mapped for LoD3 as well, albeit in a more detailed manner for surface geometries given the higher amount of semantic detail available for LoD3. The mappings done to present LoD3 geometry is especially interesting as it provides more detailed geometry and allows for more complex queries to be performed. Through the results, the framework can be considered as simple to use and sufficient for modelling a knowledge graph representation. The low amount of mappings provided and the relative simplicity of them makes for an arguably efficient and effective implementation of the method.

The thesis also aimed at discussing the currently common software and tools used for knowledge graph representation, those being 3DCityDB and ontologies. Through the method used it is clear that both software (3DCityDB and Ontop) are suited for this kind of transformation process. However, the 3DCityDB database schema presents redundancy to the framework and system with its large amount of tables. The tables *"thematic_surface", "surface_geometry"* and *"building"* are the primary tables being used out of the 66 provided in the schema. This, alongside the need for the tables to be joined in many mapping cases causes redundancy when the schema is used for this purpose. A less redundant database schema could serve to make the proposed framework even more effective for modelling knowledge graphs. Efficient management of ontologies through tools such as Ontop is important due to the known inconsistencies in the CityGML 2.0 ontology used. Furthermore, adapting the approach at a larger practical scale is likely to require developing national standards ontologies in the form of CityGML ADEs. In such cases, ontology management tools are essential.

The results shown for the performed queries further confirms and shows that the knowledge graph works as intended, is complete and robust by returning what is expected of the queries. The seven queries tested different parts of the constructed VKG in order to validate the completeness and robustness of all the parts. The first three queries tests simple building information retrieval and filtering. The first query retrieves *CityObject* IDs, their class and geometry information while the second and third query filters *CityObjects* based on measured height and number of floors respectively. Query 4 and 5 tests the different surface classes available in LoD3 by retrieving ground and roof surfaces respectively, with the latter also including filtering operations. Query 6 and 7 introduces spatial operations over the surface geometries available for LoD3 buildings. This is used as a base to conduct filtering with different distance conditions. The last four queries utilizes the higher level of detail available in LoD3 buildings by querying over the surface geometry parts of the VKG. These surface geometries are then used for some real-life inspired analysis queries, such as finding buildings within a minimum distance of a given building. While the queries performed has some practical use in urban planning, much remains to be explored in terms of incorporation spatial operations over the representation. Being able to use spatial operations over 3D geometries would increase the flexibility and use of the KG approach. By answering the research questions this thesis has shown that while there is room for improvement in many regards with the framework and method, it is still possible to effectively implement CityGML representation as VKGs for the urban planning process.

References

3DCityDB Team. (2019). *3DCityDB Documentation*. Retrieved 2024-04-08 from https://3dcitydb-docs.readthedocs.io/en/release-v4.1/index.html

Agarwal, M., Akroyd, J., Chadzynski, A., Chua, J., Grišiūtė, A., Herthogs, P., Hofmeister, M., Kraft, M., Li, S., Lloyd, E., Tai, H. Y., Tsai, Y. K., & Yan, J. (2023). Semantic 3D city interfaces—Intelligent interactions on dynamic geospatial knowledge graphs. *Data-Centric Engineering*, *4*, e20, Article e20. https://doi.org/10.1017/dce.2023.14

Beckett, D., Berners-Lee, T., Prud'hommeaux, E., & Carothers, G. (2014). *RDF 1.1 Turtle*. World Wide Web Consortium (W3C). Retrieved 2024-05-20 from https://www.w3.org/TR/turtle/#bib-N-TRIPLES

Boverket. (2020). *Guide för Bygglov och bygglovsprocessen*. Boverket. Retrieved 2024-04-18 from https://www.boverket.se/sv/om-boverket/guider/guide-for-bygglov-och-byggprocessen/

Chadzynski, A., Krdzavac, N., Farazi, F., Lim, M. Q., Li, S., Grisiute, A., Herthogs, P., von Richthofen, A., Cairns, S., & Kraft, M. (2021). Semantic 3D City Database — An enabler for a dynamic geospatial knowledge graph. *Energy and AI*, *6*, 100106. https://doi.org/https://doi.org/10.1016/j.egyai.2021.100106

Chadzynski, A., Li, S., Grisiute, A., Farazi, F., Lindberg, C., Mosbach, S., Herthogs, P., & Kraft, M. (2022). Semantic 3D City Agents—An intelligent automation for dynamic geospatial knowledge graphs. *Energy and AI*, *8*, 100137. https://doi.org/https://doi.org/10.1016/j.egyai.2022.100137

Chognard, S., Dubois, A., Benmansour, Y., Torri, E., & Domer, B. (2018, 2018//). Digital Construction Permit: A Round Trip Between GIS and IFC. Advanced Computing Strategies for Engineering, Cham.

Das, S., Sundara, S., & Cyganiak, R. (2012). *R2RML: RDB to RDF Mapping Language*. World Wide Web Consortium (W3C). Retrieved 2024-02-26 from https://www.w3.org/TR/r2rml/

Ding, L., Xiao, G., Pano, A., Fan, H., Calvanese, D., & Meng, L. (2023). Querying CityGML Data through Virtual Knowledge Graphs. *Abstr. Int. Cartogr. Assoc.*, *6*, 53. https://doi.org/10.5194/ica-abs-6-53-2023

Ding, L., Xiao, G., Pano, A., Fumagalli, M., Chen, D., Feng, Y., Calvanese, D., Fan, H., & Meng, L. (2024). Integrating 3D city data through knowledge graphs. *Geo-spatial Information Science*, 1-20. https://doi.org/10.1080/10095020.2024.2337360

Eriksson, H., Johansson, T., Olsson, P.-O., Andersson, M., Engvall, J., Hast, I., & Harrie, L. (2020). Requirements, Development, and Evaluation of A National Building Standard—A Swedish Case Study. *ISPRS International Journal of Geo-Information*, *9*(2). https://doi.org/https://doi.org/10.3390/ijgi9020078

Garramone, M., Tonelli, E., & Scaioni, M. (2022). A MULTI-SCALE BIM/GIS FRAMEWORK FOR RAILWAYS ASSET MANAGEMENT. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, *XLVI-5/W1-2022*, 95-102. https://doi.org/10.5194/isprs-archives-XLVI-5-W1-2022-95-2022

Gerbino, S., Cieri, L., Rainieri, C., & Fabbrocino, G. (2021). On BIM Interoperability via the IFC Standard: An Assessment from the Structural Engineering and Design Viewpoint. *Applied Sciences*, *11*(23). https://doi.org/https://doi.org/10.3390/app112311430

Glimm, B., Ogbuji, C., Hawke, S., Herman, I., Parsia, B., Polleres, A., & Seaborne, A. (2013). *SPARQL 1.1 Entailment Regimes*. World Wide Web Consortium (W3C). Retrieved 2024-04-09 from https://www.w3.org/TR/sparql11-entailment/

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, *5*(2), 199-220. https://doi.org/https://doi.org/10.1006/knac.1993.1008

Harris, S., & Seaborne, A. (2013). *SPARQL 1.1 Query Language*. World Wide Web Consortium (W3C). Retrieved 2024-05-20 from https://www.w3.org/TR/sparql11-query/

Heath, T., & Bizer, C. (2011). *Linked data: evolving the web into a global data space* (1. ed.). Morgan & Claypool. http://linkeddatabook.com/book

Herle, S., Becker, R., Wollenberg, R., & Blankenbach, J. (2020). GIM and BIM. *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, *88*(1), 33-42. https://doi.org/https://doi.org/10.1007/s41064-020-00090-4

Hogan, A., Blomqvist, E., Cochez, M., D'amato, C., Melo, G. D., Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Neumaier, S., Ngomo, A.-C. N., Polleres, A., Rashid, S. M., Rula, A., Schmelzeisen, L., Sequeda, J., Staab, S., & Zimmermann, A. (2021). Knowledge Graphs. *ACM Comput. Surv.*, *54*(4), Article 71. https://doi.org/10.1145/3447772

Hor, A. H., Jadidi, A., & Sohn, G. (2016). BIM-GIS INTEGRATED GEOSPATIAL INFORMATION MODEL USING SEMANTIC WEB AND RDF GRAPHS. *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.*, *III-4*, 73-79. https://doi.org/10.5194/isprs-annals-III-4-73-2016

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). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-87395-2_2

Kutzner, T., Chaturvedi, K., & Kolbe, T. (2020). CityGML 3.0: New Functions Open Up New Applications. *PFG – Journal of Photogrammetry Remote Sensing and Geoinformation Science*, *88*. https://doi.org/http://dx.doi.org/10.1007/s41064-020-00095-z

Lantmäteriet. (2024). *Detailed Development Plans*. Lantmäteriet. Retrieved 2024-04-19 from https://www.lantmateriet.se/en/maps/our-map-services/detailed-development-plans/

Lehtola, V. V., Koeva, M., Elberink, S. O., Raposo, P., Virtanen, J.-P., Vahdatikhaki, F., & Borsci, S. (2022). Digital twin of a city: Review of technology serving city needs. *International Journal of Applied Earth Observation and Geoinformation*, *114*, 102915. https://doi.org/https://doi.org/10.1016/j.jag.2022.102915

Li, W., Wang, S., Wu, S., Gu, Z., & Tian, Y. (2022). Performance benchmark on semantic web repositories for spatially explicit knowledge graph applications. *Computers, Environment and Urban Systems*, *98*, 101884. https://doi.org/https://doi.org/10.1016/j.compenvurbsys.2022.101884

Métral, C., Billen, R., Cutting-Decelle, A.-F., & van Ruymbeke, M. (2010). Ontology-based approaches for improving the interoperability between 3D urban models [Journal Article]. *Electronic Journal of Information Technology in Construction*, *15*, 169-184. https://doi.org/https://www.itcon.org/paper/2010/14

Musen, M. A. (2015). The Protégé Project: A Look Back and a Look Forward. *AI Matters*, *1*(4), 4-12. https://doi.org/10.1145/2757001.2757003

Noardo, F., Guler, D., Fauth, J., Malacarne, G., Mastrolembo Ventura, S., Azenha, M., Olsson, P.-O., & Senger, L. (2022). Unveiling the actual progress of Digital Building Permit: Getting awareness through a critical state of the art review. *Building and Environment*, *213*, 108854. https://doi.org/https://doi.org/10.1016/j.buildenv.2022.108854

Ohori, K., Biljecki, F., Kumar, K., Ledoux, H., & Stoter, J. (2018). Modeling Cities and Landscapes in 3D with CityGML. In *Building Information Modeling* (pp. 199-215). https://doi.org/10.1007/978-3-319-92862-3_11

Olsson, P.-O., Axelsson, J., Hooper, M., & Harrie, L. (2018). Automation of Building Permission by Integration of BIM and Geospatial Data. *ISPRS International Journal of Geo-Information*, *7*(8).

Ontop. (2024, 3/12/2024). *Ontop Guide*. https://ontop-vkg.org/guide/

Open Geospatial Consortium, O. (2012). *CityGML UML diagrams*. Open Geospatial Consortium. Retrieved 2014-04-23 from https://www.citygmlwiki.org/images/8/8d/CityGML_1_0_0_UML_diagrams.pdf

OWL Working Group (2012). *Web Ontology Language (OWL)*. World Wide Web Consortium (W3C). Retrieved 2024-02-23 from https://www.w3.org/OWL/

PostgreSQL Global Development Group. (2024). *PostgreSQL 16 Documentation - GiST Indexes*. Retrieved 2024-05-22 from https://www.postgresql.org/docs/current/gist.html

Rogov, E. (2019). *Indexes in PostgreSQL - 5 (GiST)*. Retrieved 2024-05-23 from https://medium.com/postgres-professional/indexes-in-postgresql-5-gist-86e19781b5db

Shi, J., Pan, Z., Jiang, L., & Zhai, X. (2023). An ontology-based methodology to establish city information model of digital twin city by merging BIM, GIS and IoT. *Advanced Engineering Informatics*, *57*, 102114. https://doi.org/https://doi.org/10.1016/j.aei.2023.102114

Smart Built Environment. (2023a). *3CIM - Smart Built Environemnt projekt 2020-2022*. S. B. Environment. https://www.smartbuilt.se/media/hobe5pn5/3cim_slutrapport.pdf

Smart Built Environment. (2023b). *Interoperabilitet - Digital samverkan för den byggda miljön*. Retrieved 2024-05-02 from https://www.smartbuilt.se/projekt/informationsinfrastruktur/interoperabilitet-digital-samverkan-for-den-byggda-miljon/

Smart Built Environment. (2024). *Smart Built Enivornment*. https://www.smartbuilt.se/in-english/

Tan, Y., Liang, Y., & Zhu, J. (2023). CityGML in the Integration of BIM and the GIS: Challenges and Opportunities. *Buildings*, *13*(7). https://doi.org/https://doi.org/10.3390/buildings13071758

Uggla, M., Olsson, P., Abdi, B., Axelsson, B., Calvert, M., Christensen, U., Gardevärn, D., Hirsch, G., Jeansson, E., Kadric, Z., Lord, J., Loreman, A., Persson, A., Setterby, O., Sjöberger, M., Stewart, P., Rudenå, A., Ahlström, A., Bauner, M., . . . Harrie, L. (2023). Future Swedish 3D City Models—Specifications, Test Data, and Evaluation. *ISPRS International Journal of Geo-Information*, *12*(2). https://doi.org/https://doi.org/10.3390/ijgi12020047

Vilgertshofer, S., Amann, J., Willenborg, B., Borrmann, A., & Kolbe, T. H. (2017). *Linking BIM and GIS Models in Infrastructure by Example of IFC and CityGML* COMPUTING IN CIVIL ENGINEERING 2017: INFORMATION MODELLING AND DATA ANALYTICS,

Vinasco-Alvarez, D., Samuel, J., Servigne, S., & Gesquière, G. (2020). *From CityGML to OWL*. https://hal.science/hal-02948955

Vinasco-Alvarez, D., Samuel, J., Servigne, S., & Gesquière, G. (2022). *RDF/OWL URBAN DATA ONTOLOGIES - VERSION1*. University of Lyon. Retrieved 2024-05-21 from https://datasets.liris.cnrs.fr/rdfowl-urban-data-ontologies-version1

W3C RDF Data Access Working Group. (2008). *SPARQL Protocol for RDF*. World Wide Web Consortium (W3C). Retrieved 2024-03-15 from https://www.w3.org/TR/rdf-sparql-protocol/

Xiao, G., Lanti, D., Kontchakov, R., Komla-Ebri, S., Güzel-Kalaycı, E., Ding, L., Corman, J., Cogrel, B., Calvanese, D., & Botoeva, E. (2020). The virtual knowledge graph system ontop. *DL 2020. Description Logics 2020 CEUR Workshop proceedings*. https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=edsswe&AN=edsswe.oai.DiVA.org.umu.197975&site=eds-live&scope=site

Yao, Z., Nagel, C., Kunde, F., Hudra, G., Willkomm, P., Donaubauer, A., Adolphi, T., & Kolbe, T. H. (2018). 3DCityDB - a 3D geodatabase solution for the management, analysis, and visualization of semantic 3D city models based on CityGML. *Open Geospatial Data, Software and Standards*, *3*(1), 5. https://doi.org/10.1186/s40965-018-0046-7

Zalamea, O., Orshoven, J., & Thérèse, S. (2016). *From a CityGML to an ontology-based approach to support preventive conservation of built cultural heritage*. https://agile-gi.eu/images/conferences/2016/documents/shortpapers/165_Paper_in_PDF.pdf

Zhu, J., Wright, G., Wang, J., & Wang, X. (2018). A Critical Review of the Integration of Geographic Information System and Building Information Modelling at the Data Level. *ISPRS International Journal of Geo-Information*, *7*(2). https://doi.org/https://doi.org/10.3390/ijgi7020066

Appendices

Appendix A: SPARQL Vocabulary

A.1. Introduction

This appendix presents the basics of the SPARQL vocabulary used in the queries presented in this report in order to retrieve and manipulate RDF data. The vocabulary provides explanations and examples of essential operations and functions used.

A.2. Prefixes

The following prefixes was used for the SPARQL queries presented in order to shorten and make the query writing more efficient.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX citygml: <http://www.opengis.net/citygml/2.0/>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX cityobject: <http://www.opengis.net/citygml/2.0/_CityObject/>
PREFIX building: <http://www.opengis.net/citygml/building/2.0/>
PREFIX geom: <http://www.opengis.net/gml/>
PREFIX surface: <http://www.opengis.net/gml/Surface/>
```

A.3. Sample queries

Basic instance retrieval

The following query retrieves all instances of the class *"Building"*.

```
PREFIX building: <http://www.opengis.net/citygml/building/2.0/>

SELECT ?cityobject
WHERE {
  ?cityobject a building:Building .
}
```

Instance property retrieval

The following query retrieves all properties and their values for the specific instance *"CityObject/93"*.

```
PREFIX cityobject: <http://www.opengis.net/citygml/2.0/_CityObject/>

SELECT ?property ?value
WHERE {
   cityobject:93 ?property ?value .
}
```

BIND operation showcase

The following query showcases the use of the BIND operation available in SPARQL and how it is used for the queries in this report. BIND allows a value to assigned to a variable from a basic graph pattern or property path. It is used in the queries as a means to make retrieved results easier to read by shortening IRIs down to just ID numbers.

```
PREFIX building: <http://www.opengis.net/citygml/building/2.0/>

SELECT ?cityobjectID
WHERE {
  ?cityobject a building:Building .
  BIND (URI(?cityobject) AS ?cityobjectURI)
  BIND (xsd:integer(REPLACE(str(?cityobjectURI),
"http://www.opengis.net/citygml/2.0/_CityObject/", "")) AS ?cityobjectID)
}
```

FILTER showcase

The following query showcases the use of the FILTER operation available in SPARQL and how it is used in the queries in this report. FILTER allows for filtering out a selection of the retrieved results based on select property values. It is used to conduct analysis by finding certain building in compliance with a set height threshold.

```
PREFIX building: <http://www.opengis.net/citygml/building/2.0/>

SELECT ?cityobject ?measuredHeight
WHERE {
  ?cityobject building:measuredHeight ?measuredHeight .
  FILTER(?measuredHeight > 10)
}
```

GeoSPARQL distance function showcase

The following query provides an example of how the distance calculation function provided by the GeoSPARQL ontology is used in the queries. The distance function can be used to calculate the distance between two 2D geometries. In the queries it is used to calculate the distance between two different surface polygons in different buildings in order to estimate the distance between any two buildings.

```
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX building: <http://www.opengis.net/citygml/building/2.0/>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geom: <http://www.opengis.net/gml/>

SELECT ?startSurface ?distance ?endSurface
WHERE {
  {
    SELECT ?otherSurface
    WHERE {
      ?lod3_surface_id geo:asWKT ?otherSurface .
    }
    LIMIT 1
  }
  BIND(?otherSurface AS ?endSurface)
  ?lod3_surface_id geo:asWKT ?startSurface .
  BIND(geof:distance(?startSurface, ?endSurface,
<http://www.opengis.net/def/uom/OGC/1.0/metre>) AS ?distance)
}
```