Improving the Interoperability Between City and Road Semantics

An Integration of CityGML and OKSTRA Data Based On Semantic Web Technologies

Yuan Zheng

Chairman	Prof. Bauke de Vries
First Supervisor	Ir. Wiet Mazairac
Second Supervisor	Prof. Jacob Beetz

Eindhoven, 12-2017

Colophon

Master thesis
Title:Improving the Interoperability of Between City and Infrastructure Information
Subtitle:An Integration of CityGML and OKSTRA Data Based On Semantic Web and Linked Data Technology
Version:Final Version

Author:
Name: Yuan Zheng 郑远
ID: 0981516
Email: y.zheng@student.tue.nl

University:
University: Eindhoven University of Technology
Faculty: Architecture, Building and Planning
Department (main): Built Environment
Department (secondary): Construction Management and Engineering
Chair: Information Systems in the Built Environment (ISBE)

Graduation committee:
Academic supervisors:
Chairman: Prof. dr. ir. B. (Bauke) De Vries
First supervisor: Ir. L.A.J. (Wiet) Mazairac
Second supervisor: Prof.dr. Dipl-Ing. J. (Jakob) Beetz (RWTH Aachen)

Date: Monday 4th December, 2017

Technische Universiteit
Eindhoven
University of Technology

Preface

This report is the result of my two-year education period and my graduation research carried out at Eindhoven University of Technology. In the 2015, I came from China to start my master program of Construction Management and Engineering at here with my ambition. I truly obtained a vast amount of novel knowledge about system engineering in built environment, Building Information Modeling, Geographical Information System and smart city and urban environment, which has shaped my academic orientation and could help me to make contribution for my motherland in the future.

The formulation of my research was not easy. The lack of knowledge of semantic web and programming were big problems for me. In the beginning of the graduation project, I paid a lot of time to study semantic web relevant knowledge and begin to learn scripting from almost zero basis. After the study, the research went smoothly and my knowledge and skill grew. This research shines me a light for my future steps.

Here, I would like to thank all my families, teachers and friends who support and stimulate me during my entire master period. First, I really appreciate my parents who founded me to study here in the Netherlands. Meanwhile, the academic and technical support from my supervisors Prof.Jacob Beetz and Dr.Wiet are greatly appreciated. Furthermore, I also feel thankful to the support from my department Prof.Burke de Vries, Ir.Aant van der Zee, Dr.Qi Han and Dr.Peter van der Waerden. I would also show acknowledgement to Mr.Stefan Wick for providing the OKSTRA model of Nordrhein-Westfalen state and CityGML technical support from Dr. Claudine Metral in University of Geneva. Furthermore, I would also like to appreciate Qifan Dai, who encouraged me every time when I was facing obstacles. Also my friend Dalei Li and Qiyue Wang for tutoring me some scripting skills. What's more, I would also like to thank to my cousin Jingyue Ma, a 16-year-old boy that is currently fighting with serious illness, your courage is also my courage to keep fighting with my ambition, hope you can recover soon.

I hope you will enjoy reading and could be learning from this research as I had.

King Regards,

Yuan Zheng, in Eindhoven, the Netherlands

郑远,于荷兰埃因霍温

Contents

List of Figures

List of Tables

Summary

Recent years, 3D city models are increasingly being adopted throughout the world. The current trend of the 3D city model is appending semantics to the geometrical objects describing the city and relevant information in order to enlarge the possible applications of such models. CityGML is known as the most widely implemented 3D semantic city data model which is gradually being accepted and implemented in worldwide as the city modelling source.

However, although CityGML claims to contain not only the geometric information of city objects but also their semantics, there are some problems about the language to represent the semantics of urban information. First is CityGML is insufficient to contain all the city objects and their semantics. Therefore, the CityGML should consider to integrate with other heterogeneous city semantic data sources to enrich the volume of city information, but the CityGML itself has limitation of data source expansion, this is known as the second problem. Moreover, there is also the third problem which is the insufficient utilities of semantic information inside the CityGML model.

In the most of the heterogeneous city data sources, the infrastructure data one of the most necessary data source to be involved, especially for road data. Although current CityGML expanded several modules to contain road and relevant infrastructure information, but which is not as comprehensive and in detailed as existing road data platform such as OKSTRA. Therefore, from another perspective with expanding the CityGML data standard, to integrate OKSTRA with CityGML will promote the profundity of CityGML's semantics and enhance the utility of existed infrastructure information source since the isolated road information can be incorporated in the entire city environment.

Therefore, this research is focused on integrating CityGML and OKSTRA models. The main challenge of this research is to solve the integration issue of CityGML with road information data sources and respond three latent problems of CityGML. Especially, to solve the data sharing between CityGML and road data domains. Moreover, developing the further data querying and analysis. Therefore, it is required an effective data integration approach to ensure successfully and consequentially incorporate these two information sources.

In this research, the Semantic Web technologies is implemented as an integrating platform for integrating the OKSTRA road information model and CityGML model. The research main objective is to gain insight in how semantic web technology can be utilized as an integration approach in order to enhance the interoperability of CityGML and OKSTRA semantics. This goal can be divided into two sub-goals

1. Develop the methodology of using Semantic Web technology in order to integrate CityGML and OKSTRA models.
2. Realize the integration process based on the developed method using existed tools and conduct an experimental implementation case. Furthermore, query the integration product in order to verify the integration process and obtain useful semantic query result.

In order to achieve the research objective, this research is conducted in two phases, the theoretical

research and experimental implementation. The theoretical research starts with the literature study and then establishes the method of integrating OKSTRA and CityGML models and implementation process map. The integration methodology is divided into five parts: constructing ontologies, transform CityGML and OKSTRA models into RDF graphs, create links between the constructed ontologies, link the RDF graphs, and query the integrated RDF set. The designed methodology is examined through the experimental implementation.

In the experimental implementation, after the raw data process and ontology development, the instance data models of CityGML and OKSTRA are transformed to RDF graphs corresponded with the developed ontologies. Then, the ontology mapping process is conducted to determine the link between two ontologies and RDF graphs. In this research, the data linking between CityGML and OKSTRA is achieved to create link based the spatial relationship of buildings in CityGML and road sections in OKSTRA. This spatial relationship is determined by employing GeoSPARQL technology to calculate the shortest distance between each building and road section to find the closest road section to each building. Using this relationship, two RDF graphs are integrated into one RDF set. Theoretically, the integrated RDF set is able to be synthetically processed and queried across two data domains. Thus, several thematic queries are conducted later in order to verify if the integration is successful and prove the integration product has additional value of utilizing the semantics in RDF set from two data sources. Meanwhile, the query results are also illustrated.

The thesis is constructed into six chapters to clarify and verify the integration process. Chapter 1 introduces the research background and the research process. The basic knowledge of CityGML, OKSTRA, Semantic Web technologies is given in Chapter 2, together with the literature summary of introducing the current state of Semantic Web implementation research in the AECO/FM industry. Chapter 3 describes the methodology that integrates the CityGML and OKSTRA model based on Semantic Web technologies. In Chapter 4 and 5 this methodology is testified through an experimental implementation. In the end, chapter 6 gives some reflections of this integration method and provides some future research recommendations.

Overall, from this research, it can be proved that Semantic Web technology is a suitable approach to integrate CityGML and OKSTRA data source. As a conclusion, the main research question can be answered. The implementation of the integration of CityGML and OKSTRA based on Semantic Web technologies can be achieved through a framework including several steps: 1) Collect the necessary semantics data resource from CityGML and OKSTRA platforms 2) Analyze the data schema of both platform and develop their ontologies 3)Transform the data into RDF graphs under the developed ontologies, 4) Map the developed ontologies to determine the links between RDF resources 5) Merge all the triples and link the determined related resources 6) develop SPARQL query or use other RDF processing methods to apply the integrated product.

Abstract

Known as one of the most widely implemented 3D semantic city data models, CityGML has already being adopted throughout the world, which is able to contain a large amount of city semantic data along with 3D geometries. However, the infrastructure information in the CityGML is limited and should be enriched by considering to integrate with other infrastructure data source in order to expand the comprehensiveness of city semantics content and create further semantic information usage.

This research which aims to develop the methodology of using Semantic Web technologies in order to integrate CityGML and German road information system OKSTRA and realize experimental implementation of developed methodology. Meanwhile, a paradigm of applying Semantic Web SPARQL query approach is set up to realize spatial quires based on the integrated CityGML and OKSTRA RDF set.

Glossary

CityGML	An open standardised data model and exchange format to store digital 3D models of cities and landscape
Smart City	An urban area that uses different types of electronic data collection sensors to supply information used to manage assets and resources efficiently
AECO/FM	The abbreviation of Architecture, Engineering, Construction, Operations and Facility Management
UML	The abbreviation of Unified Modeling Language
KML	The abbreviation of Keyhole Markup Language
Collada	An interchange file format for interactive 3D applications
ESRI	An powerful mapping and spatial data analytics software
OGC	The abbreviation of Open Geospatial Consortium
GIS	The abbreviation of Geographic Information System
BIMs	The abbreviation of Building Information Models
Semantic Web	A framework allowing data to be shared and reused across application, enterprise, and community boudaries
OKSTRA	The abbreviation of Objektkatalog fur das Strassen- und Verkehrswesen = Object catalog for road and traffic related data
XML	The abbreviation of Extensible Markup Language
RDF	The abbreviation of Resource Description Framework
Turtle	The abbreviation of A format for expressing data in the RDF data model
GML3	The abbreviation of Geography Markup Language 3
LOD	The abbreviation of Levels of Detail

SIG 3D	The abbreviation of Special Interest Group 3D
ISO	The abbreviation of International Organization for Standardization
IT	The abbreviation of Information Technology
CTE	The abbreviation of common table expression
EXPRESS	A standard modeling language for product data
XSD	The abbreviation of XML Schema Definition
EXPRESS-G	A standard graphical notation for information models
STEP	The abbreviation of Standard for the Exchange of Product data
BASt	The abbreviation of German Federal Highway Research Institute
OWL	A semantic Web language designed to represent knowledge about things and their relations
SPARQL	SPARQL Protocol and RDF Query Language
RDFS	A data-modelling vocabulary for RDF data
HTTP	The abbreviation of Hypertext Transfer Protocol
BIM	A process to create and manage digital representations of buildings.
LOD	The abbreviation of Linked Open Data
IoT	The abbreviation of Internet of things
IFC	The abbreviation of Industry Foundation Classes
XSLT	The abbreviation of Extensible Style sheet Language Transformations
DTD	The abbreviation of Document Type Definitions
JSON	The abbreviation of JavaScript Object Notation
BPMN	The abbreviation of Business Process Model and Notation
GeoSPARQL	The abbreviation of A Geographic Query Language for RDF Data
CSV	The abbreviation of comma-separated values
WKT	The abbreviation of Well-known text
URI	The abbreviation of A string of characters used to identify a resource
3D-PDF	A PDF file that happens to contain 3D data

LiDAR	The abbreviation of Light Detection and Ranging

Chapter 1

Introduction

1.1 Research Background

3D city models are increasingly being adopted throughout the world. Facing the challenges of increasing complexity of construction, urbanization and management, the 3D city models could provide further value and additional utility for built environment relevant effective decision-making support and tremendously improve the process of facility management, urban planning, infrastructure designing, simulation, emergency and etc [18]. Moreover, 3D city models could also be an essential element for constructing Smart City, which could bring advantages of data visualization and analysis support, even could be a bridge to link with the ICT equipment of smart city and other data sources for integrating city information within the Smart-Cities context [52]. Therefore, the benefits of the 3D city models have successfully attracted the interest of stakeholders from the Architecture, Engineering, Construction, Operations and Facility Management (AECO/FM) industry.

In the past, most of the existing 3D city models focused on representing graphical and geometrical information, while the use of semantic was neglected in the 3D city modelling domain[16]. In fact, the models without semantic data can only be considered as a visualization mean of 3D geometry but not be able to realize extra functionality such as thematic queries, analysis tasks, or even spatial data mining for domains of facility management, urban planning, infrastructure designing, city simulation and emergency management [42][29], because these applications require not only 3D geometry visualization but also rely on complex semantic information in different scales and domains [62]. Sticking to the models without semantic models could become a bottleneck for further 3D city model implementation since it limits the utility and potential of 3D city models. Therefore, a strong demand emerged that the current 3D city models should also contain more city semantic information in order to achieve different engineering and planning applications that enable complex queries and analysis for facilitating sophisticated decision tasks [56].

Therefore, the current trend of the 3D city model is appending semantics to the geometrical objects describing the city and relevant information in order to enlarge the possible applications of such models. Known as the most widely implemented 3D semantic city data model, CityGML is a common semantic information model for the representation of 3D urban objects, which has already added the semantic parts along with the geometrical information (shown as fig. 1.1) and is gradually being accepted and implemented in worldwide as the modelling sources. Compared with CityGML, other 3D city model data representations such as KML, Collada and ESRI Shape files that are mainly focusing on the geometrical and geographical visualization aspects and lack semantic information. Moreover, the CityGML standard laid the foundation for the storage and application of semantics, which stimulates the progress of semantic 3D city modeling[64]. Currently, some of the cities in the worldwide has already created their own 3D city model based on

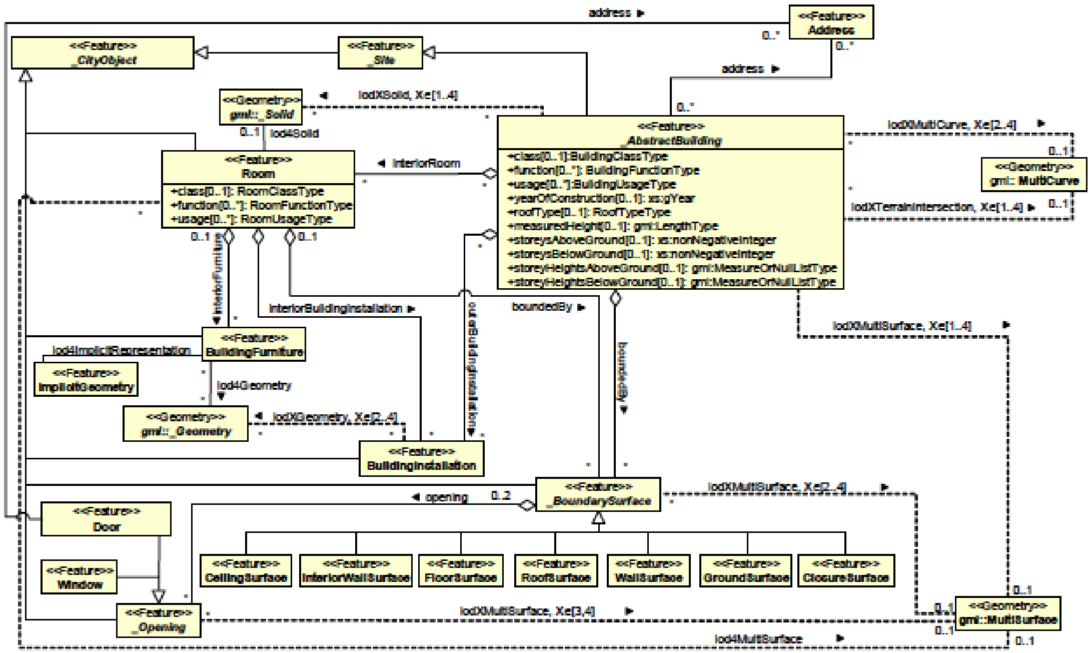

address ▸
<<Feature>> Address
0..*
<<Feature>> _CityObject
<<Feature>> _Site
address ▸
0..*
<<Geometry>> gml::_Solid
◂ lodXSolid, Xe[1..4]
0..1 lod4Solid
<<Feature>> _AbstractBuilding
+class[0..1]:BuildingClassType
+function[0..*]: BuildingFunctionType
+usage[0..*]:BuildingUsageType
+yearOfConstruction[0..1]: xs:gYear
+roofType[0..1]: RoofTypeType
+measuredHeight[0..1]: gml:LengthType
+storeysAboveGround[0..1]: xs:nonNegativeInteger
+storeysBelowGround[0..1]: xs:nonNegativeInteger
+storeyHeightsAboveGround[0..1]: gml:MeasureOrNullListType
+storeyHeightsBelowGround[0..1]: gml:MeasureOrNullListType
lodXMultiCurve, Xe[2..4] ▸
<<Geometry>> gml:: MultiCurve
lodXTerrainIntersection, Xe[1..4] ▸
0..1
<<Feature>> Room
+class[0..1]: RoomClassType
+function[0..*]: RoomFunctionType
+usage[0..*]: RoomUsageType
◂ interiorRoom
interiorFurniture
interiorBuildingInstallation ▸
outerBuildingInstallation
boundedBy ▸
boundedBy
lodXMultiSurface, Xe[1..4] ▸
<<Feature>> BuildingFurniture
lod4ImplicitRepresentation
<<Feature>> ImplicitGeometry
lod4Geometry
<<Geometry>> gml::_Geometry
◂ lodXGeometry, Xe[2..4]
<<Feature>> BuildingInstallation
<<Feature>> _BoundarySurface
◂ opening 0..2
lodXMultiSurface, Xe[2..4] ▸
<<Feature>> Door
<<Feature>> Window
<<Feature>> _Opening
<<Feature>> CeilingSurface
<<Feature>> InteriorWallSurface
<<Feature>> FloorSurface
<<Feature>> RoofSurface
<<Feature>> WallSurface
<<Feature>> GroundSurface
<<Feature>> ClosureSurface
lodXMultiSurface, Xe[3,4] ▸
<<Geometry>> gml::MultiSurface
lod4MultiSurface ▸
0..1

Figure 1.1: UML diagram of CityGML's building model, contains both semantic and geometrical information Source:OGC

CityGML standard and published as open data sources (shown as Appendix A).

1.2 Problem Statement

Although CityGML is able to represent various city objects in 3D geometry and contain their relevant semantics, several latent problems could impact its future development. First is CityGML is not able to contain all the city objects and their semantics. Practically, there are miscellaneous semantic data sources to describe various city objects and relevant city information, however CityGML is not possible to cover them all. Even the CityGML could expand its data schema and modules continuously in order to cover more city objects and their related semantics, but in terms of both integrity and details of data, the CityGML itself is still insufficient. For example, one of the most important city objects are the buildings in the CityGML. However, from the comprehensiveness perspective, the building models in the CityGML has less detailed information if compared with BIMs of these buildings, only the most detailed CityGML LOD4 is closest to BIM, in which buildings are portrayed as architectural models with their surfaces,openings and details[58]. Therefore, in order to tackle this comprehensiveness problem, the CityGML should consider to integrate with other heterogeneous city semantic data sources, but the CityGML itself has limitation of data source expansion, known as the second problem. This leads a gap for CityGML to integrate with these data sources in various format representation. The third problem is the insufficient utilities of semantic information inside the CityGML model. Although CityGML has been recognized as 3D modeling platform within the GIS domain, currently it has still been mainly used as 3D model resource for visualization purpose and the semantics in the CityGML model is limited to be directly used as urban or spatial data source for querying and analysis just like data processing in conventional GIS tools. All these problems have to be tackled in order to build a comprehensive 3D semantic city information model applications based on CityGML.

In the most of the heterogeneous city data sources, one essential aspect should be mentioned is the infrastructure data. Moreover, in the infrastructure domain, road is one of the most important city component that known as the vessel of city's transportation and has an impact on urban lives. Therefore, reliable information of road on the city level is needed and should be linked with other city semantics to support policy making and then to assess impact of decisions on the performance of the city in order to improve future decisions. Although current CityGML expanded several modules to contain road and relevant infrastructure information, but which is not comprehensive and in detailed to compare with existing road data platform. In fact, there are already several systematic and comprehensive road and infrastructure information standards such as OKSTRA and LandXML that have been implemented in different countries. Therefore, from another perspective with expanding the data standard for different modules, a possible way to improve the CityGML semantic data model is to integrate the existing infrastructure data with CityGML data. Most importantly, integrating road information and CityGML will promote the profundity of CityGML's semantics and enhance the utility of existed infrastructure information sourcesince the isolated road information can be incorporated in the entire city environment.

Therefore, the main challenge of this research is to solve the integration issue of CityGML with road information data sources and respond three latent problems of CityGML. Especially, to solve the data integration and interoperability between CityGML and road data domains. Moreover, developing the further data querying and analysis. Therefore, it is required an effective data integration approach to ensure successfully and consequentially incorporate these two information sources.

Despite the importance and omnipresence of the road information in the city level, the topic of integrating CityGML and road data has not been investigated extensively, and there has been a few holistic research that focusing on schema expansion but less data integration that encompass the complete pipeline as described. The current heterogeneous city semantic data sources of integrating with CityGML among the scholars is primarily focusing on Building Information Models (BIMs), which are not directly related with this research but still valuable for finding the appropriate integration approaches.

The Semantic Web technologies could be such an integrating platform for integrating the road information data source and CityGML model. The Semantic Web is known as an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation[11]. The Semantic Web applications has been tested in the integration of heterogeneous data model such as BIMs and Geo-spatial data sources like CityGML and other GIS data source among the scholars, these current works will be minutely discussed in the Chapter 2. According to the researches of integrating BIMs and CityGML, the application of Semantic Web technologies has obvious advantages that meet current integration requirement of miscellaneous data and is able to create links between different information data sets from different domains and to provide further query that could make comparison with other approaches. Thus, in this research, the Semantic Web technologies are employed to realize the integration of CityGML semantics and road information data. A detailed description of the process will be presented in the following chapters.

1.3 Research Questions

The research problem has been stated in the last section. In this research, the road data model is the German OKSTRA (Objektkatalog fur das Strassen- und Verkehrswesen = Object catalog for road and traffic related data) model, which will be integrate with the CityGML model. Moreover, the further utilization of integrated model is achieved as query applications based on the Semantic Web query language SPARQL and GeoSPARQL. Based on the problem area, this section outlines the research questions of the proposed research to conduct the research process and solve the research problem. The main research question is:

How to realize the data integration of CityGML and OKSTRA based on Semantic Web technologies in order to enhance the city semantic information interoperation meanwhile to develop query applications based on the integration product?

This main research question can be divided into a number of sub-questions in two parts:

The first part is **'How the semantic web technology can be utilized to integrate CityGML and OKSTRA models?'**, including several following sub-questions:

1. How Semantic Web technology can enhance the semantic information sharing during the integration process?

2. How to use semantic web technology to integrate CityGML and OKSTRA from theoretical aspect?

The second part is **'how the integration result can be used for developing query applications based on a specific implementation'**, including several following sub-questions:

3.What are detailed process to transform the CityGML and OKSTRA model into RDF graphs?

4. What kind of linking between CityGML data and OKSTRA data can be created?
5. How the linked CityGML-OKSTRA model could be utilized by querying integrated product?

6. What kinds of topics of query can be achieved within the research scope?
All the question and sub-questions will be answered in Chapter 6, as the conclusions part of this thesis.

1.4 Research Goals

The aims of this project is to propose a research that studies and develops approach and methodology for integrating CityGML and OKSTRA data model based on Semantic Web technologies and provide rather specific guidance for further researches in the field of information management for city scale. Meanwhile, develop series of general approaches, tools and programs to realize the integration of CityGML and OKSTRA models in specific German regions. Moreover, using the semantic web query technology to realize the semantic queries of integration product. Therefore, the main research goal is to gain insight in how semantic web technology can be utilized as an integration approach in order to enhance the interoperability of CityGML and OKSTRA semantic. This goal can be divided into two sub-goals which are corresponding with two parts of the research questions:

1. Develop the methodology of using Semantic Web technology in order to integrate CityGML and OKSTRA models

2. Realize the integration process based on the developed method using existed tools and conduct an experimental implementation case. Furthermore, query the integration product in order to verify the integration process and obtain useful semantic query result

1.5 Research Design

This research is conducted in two phases, the theoretical research and experimental implementation, shown as fig. 1.2.

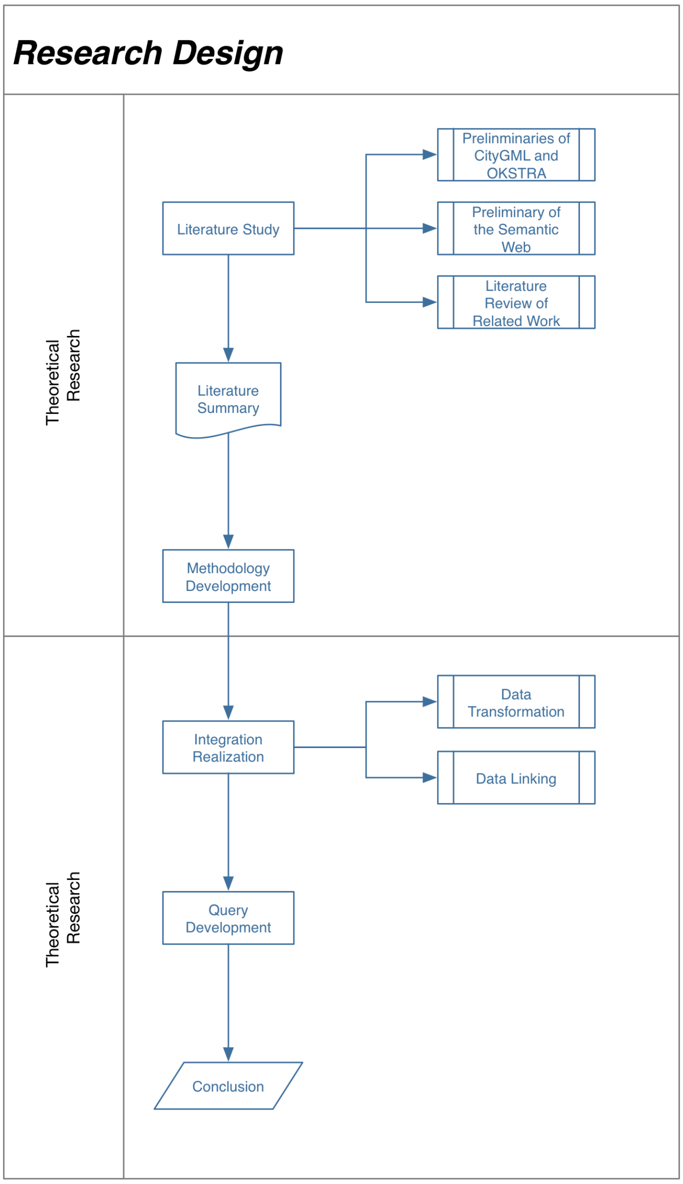


Figure 1.2: Research Design

1.5.1 Theoretical Research

The theoretical research starts with the literature study which consists of three parts: First is to acquire the preliminaries about the CityGML and OKSTRA data model standards. This step will obtain the knowledge of basic data structure of both CityGML and OKSTRA. After the general knowledge about OKSTRA and CityGML is acquired, the focus turns to Semantic Web technologies including Linked data approach, RDF study and Semantic Web querying technology. When all the preliminaries have been studied, the third parse is to start the literature study that focusing on the current researches of the roles and applications of Semantic Web technologies in AECO/FM industry.

After the literature study, the second part for theoretical research is to establish the method of integrating OKSTRA and CityGML models and implementation process map. The result will be used to conduct the experimental implementation phase. Therefore, a design of the integration method is needed to guide the data transformation and linking procedures in further experimental implementation.

1.5.2 Experimental Implementation

The experimental implementation aims to realize the integration of CityGML and OKSTRA instance models based on the integration method that enables Semantic Web technologies. Several steps constitute the entire implementation process. The first step is the data processing in order to generation the target CityGML and OKSTRA instance model for the experimental implementation. After the data processing has been done, the integration based on the principle of the semantic web technology is conducted contains following steps: 1)the ontologies development, 2)transformation of CityGML and OKSTRA data-XML file into RDF format, and 3)the data linking process. At last, queries of the linked data RDF file is executed to verify if the integration is successful.

1.6 Report Outline

The first chapter is concluded by presenting the general layout of this report (see in fig. 1.3). The basic knowledge of CityGML, OKSTRA, Semantic Web technologies is given in Chapter 2, together with the literature summary of introducing the current state of Semantic Web implementation research in the AECO/FM industry. Chapter 3 describes the methodology of integrating

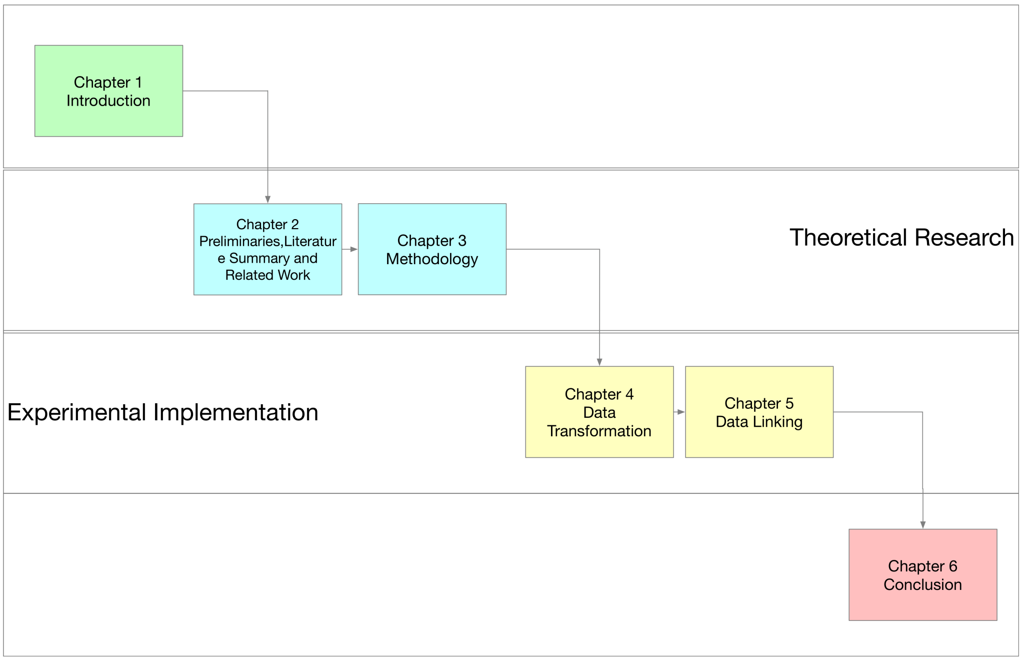


Figure 1.3: Report Layout

CityGML model and OKSTRA data, together with the process mapping of the experimental implementation including the data transformation, data linking and query development. The details of experimental implementation is described in Chapter 4 and 5. Finally in Chapter 6 provides the conclusions, recommendations and future work from this research.

Chapter 2

Preliminaries,Literature Summary and Related Work

Currently, Semantic Web technologies have been increasingly developed and applied in different industries. Simultaneously, the research of using semantic web technologies in the AECO/FM industries started in the early 2000s[50] and has been increasingly adopted. In this chapter, firstly the preliminaries about relevant fields of CityGML, OKSTRA as well as the Semantic Web technologies are briefly discussed. In Section 2.1 and Section 2.2 relevant knowledge of CityGML and OKSTRA data standards is provided. In the following Section 2.3, the knowledge of relevant Semantic Web technologies are introduced. Following with the literature study of this research about the role and advantages of Semantic Web technology and related works of data integration in AECO/FM industry are briefly summarized, which aims to gain an insight about what kind of role that Semantic Web is playing in the AECO domain to enhance the semantic information sharing and be utilized for integration process of city and infrastructure information.

2.1 CityGML

CityGML is a common semantic information model for the representation of 3D urban objects that can be shared over different applications, which is known as an open standardized data model and exchange format to store digital 3D models and semantics of cities and landscapes in the XML-based format for virtual 3D city models[34]. It is an application schema for the Geography Markup Language 3 (GML3), the extendable international standard for spatial data exchange issued by the Open Geospatial Consortium (OGC) and the ISO TC211[35]. Known as a generic information standard for 3D city models, CityGML represents city elements and objects with modulates information[44]. The whole CityGML file consists of 12 modules, including different city elements (shown as fig. 2.1).
The latest version of CityGML is CityGML 2.0.0, which is designed based on a number of standards from the ISO 191xx family and is also implemented as an application schema for Geography Markup Language (GML 3.1.1) from OGC. Furthermore, CityGML has been developed by the Special Interest Group 3D (SIG 3D) of the initiative Geodata Infrastructure North-Rhine Westphalia, Germany[44].

CityGML aims to define the basic entities, attributes and relations within 3D city domain, which mainly describes the geometry, attributes and semantics of different kinds of 3D city objects[35]. The base class of all objects is CityObject that all the objects inherit the properties from CityObject. Moreover, CityGML not only represents the graphical appearance of city models but specifically addresses the representation of the semantic and thematic properties, taxonomies and aggregations[44]. CityGML includes a geometry model that allows for the consistent

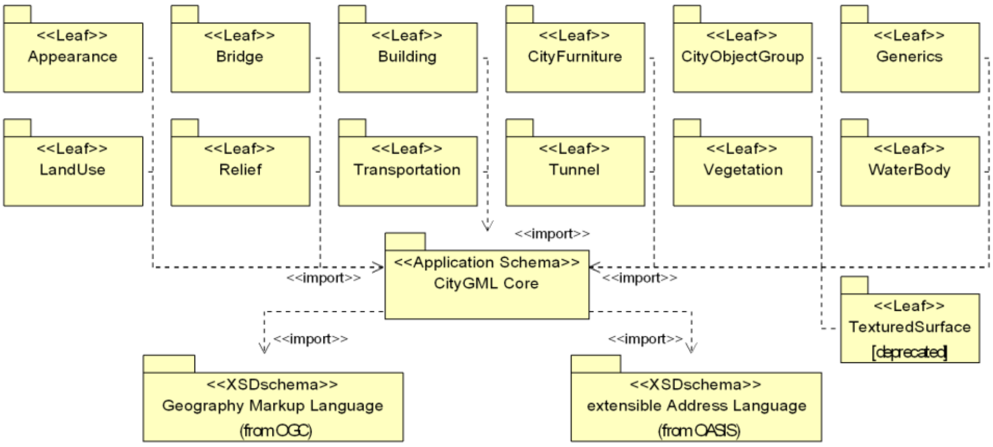


Figure 2.1: CityGML Modules (source: OGC)

and homogeneous definition of geometrical and topological properties of spatial objects within 3D city models. Nevertheless, CityGML is also designed as a common semantic information model for the representation of 3D urban objects that can be shared over different applications[36]. This makes a semantically ample CityGML model could be used in GIS applications as the data resource for relevant semantic analysis. The data in CityGML model can be interpreted by both computers and humans, and there is extensive semantics which is directly linked to the geometries in a spatially aggregated hierarchy. In fig. 2.2 the UML diagram for the relevant semantics of a CityGML building module is shown.

Furthermore, CityGML supports different Levels of Detail (LOD), which is an omnipresent concept in geographic information. Five levels of detail is defined to (as shown in fig. 2.3) represent a city object, where higher level contains more detailed features and properties of a CityObject [31]. Moreover, one object can be represented in different LODs, which enables the analysis and visualization of the same object with regard to different degrees of resolution[36] to satisfy different detail requirements.

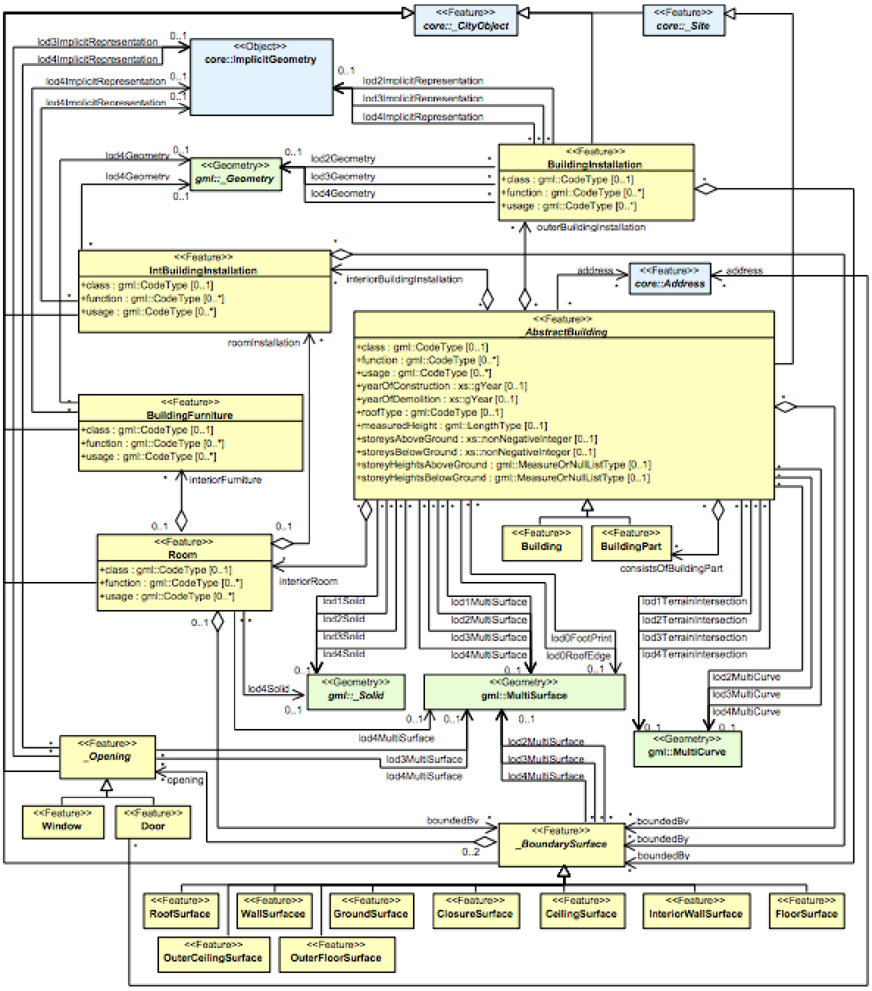


Figure 2.2: CityGML UML for Building Module(source: OGC)

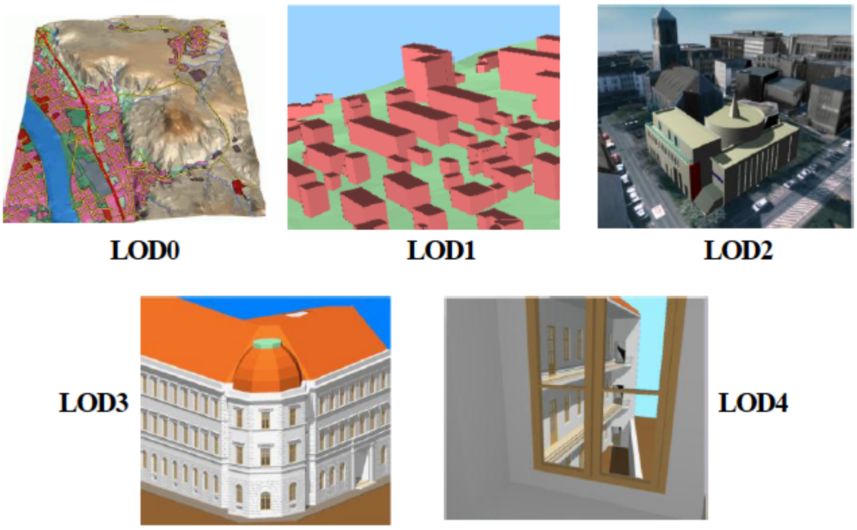


Figure 2.3: The five levels of detail (LOD) defined by CityGML (source: IGG Uni Bonn)

2.2 OKSTRA

The OKSTRA® (Objektkatalog für das Strassen- und Verkehrswesen; English version: Object catalog for road and traffic-related data) is a feature catalogue and application schema for road and traffic-related data covering the whole life-cycle of a road and with the goal of a lossless exchange of road-related data, known as a standardized, conceptual data model for various areas of roads and transport[40]. OKSTRA unifies the data description of objects from traffic engineering and also contains a data model for geometry alignment data for align with other geometry and geographical sources (shown as fig. 2.4, the OKSTRA geometry model in the green color is aligning with the landscape terrain surface).

The main design goal of OKSTRA is to ensure a consistent object representation and the simple unified graphical/geometric data exchange in the road and transport sector between different applications that implement the standard[32]. Moreover, in terms of industrial knowledge and IT-based techniques, OKSTRA® is strongly oriented to coordinate with existing regulations and standards [55], since road design is being targeted in current developments by organizations such as buildingSMART and the Open Geospatial Consortium (OGC) [4].

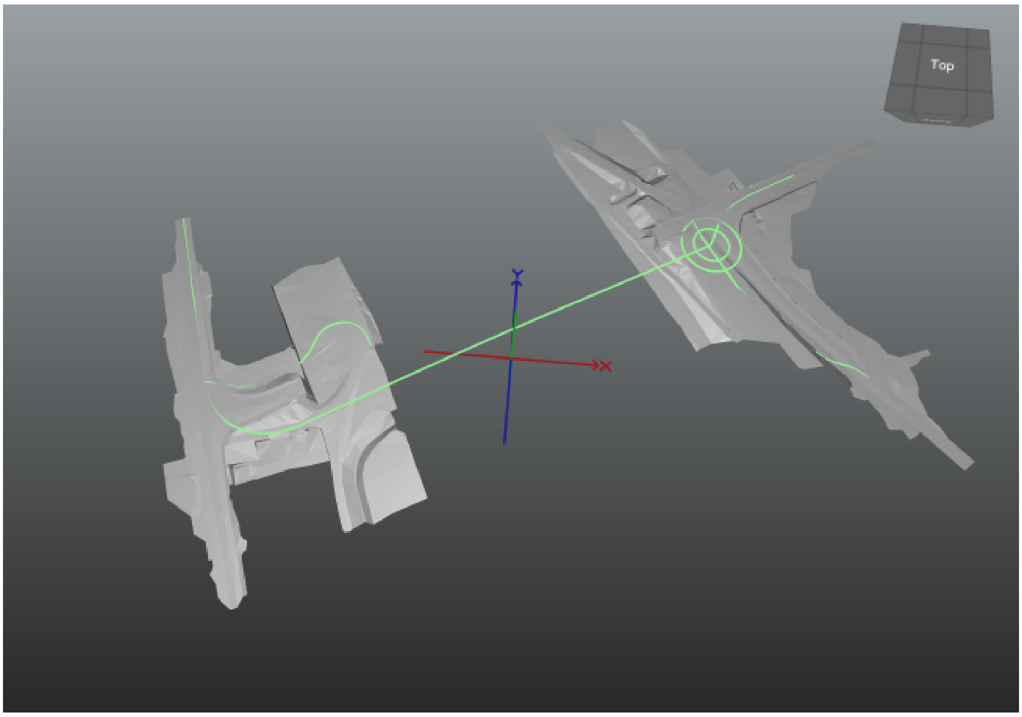


Figure 2.4: The example of OKSTRA alignment with landscape terrain data (source: OKSTRA)

Virtually all work processes in road and traffic are now supported by IT procedures. The digital loss-free transfer of the data in these processes is only possible in a limited extent and is characterized by media breaches. The idea of using uniform objects of the road and traffic in the IT process provides a remedy for the prevention of media breaches and their consequences. The summary of the objects is the object catalog in the road and traffic (OKSTRA)[53].

The current OKSTRA standard (version 2.017) was published in April 2016, providing the data model solely as an XSD schema. In the past, the OKSTRA standard was provided as an EXPRESS schema with the CTE data format. However, recently the OKSTRA developers moved from EXPRESS to XSD, from NAIM Diagrams (like EXPRESS-G Diagrams) to UML (partial example is shown as fig. 2.5) and from STEP for storing instance files to XML[4].

The OKSTRA is owned by BASt (German Federal Highway Research Institute) and distributed under an open license that permits its use for commercial projects. The OKSTRA standard was developed for the German market and initiated in the research company for road and traffic engineering and implemented within the scope of research assignments[4]. Meanwhile, OKSTRA was introduced as a nationwide standard for the area of the Federal Highway and recommended to the road construction administrations of the countries for application. Therefore, the official XSD schema specifies all identifiers in German, however, currently some the documentation is also available in English.

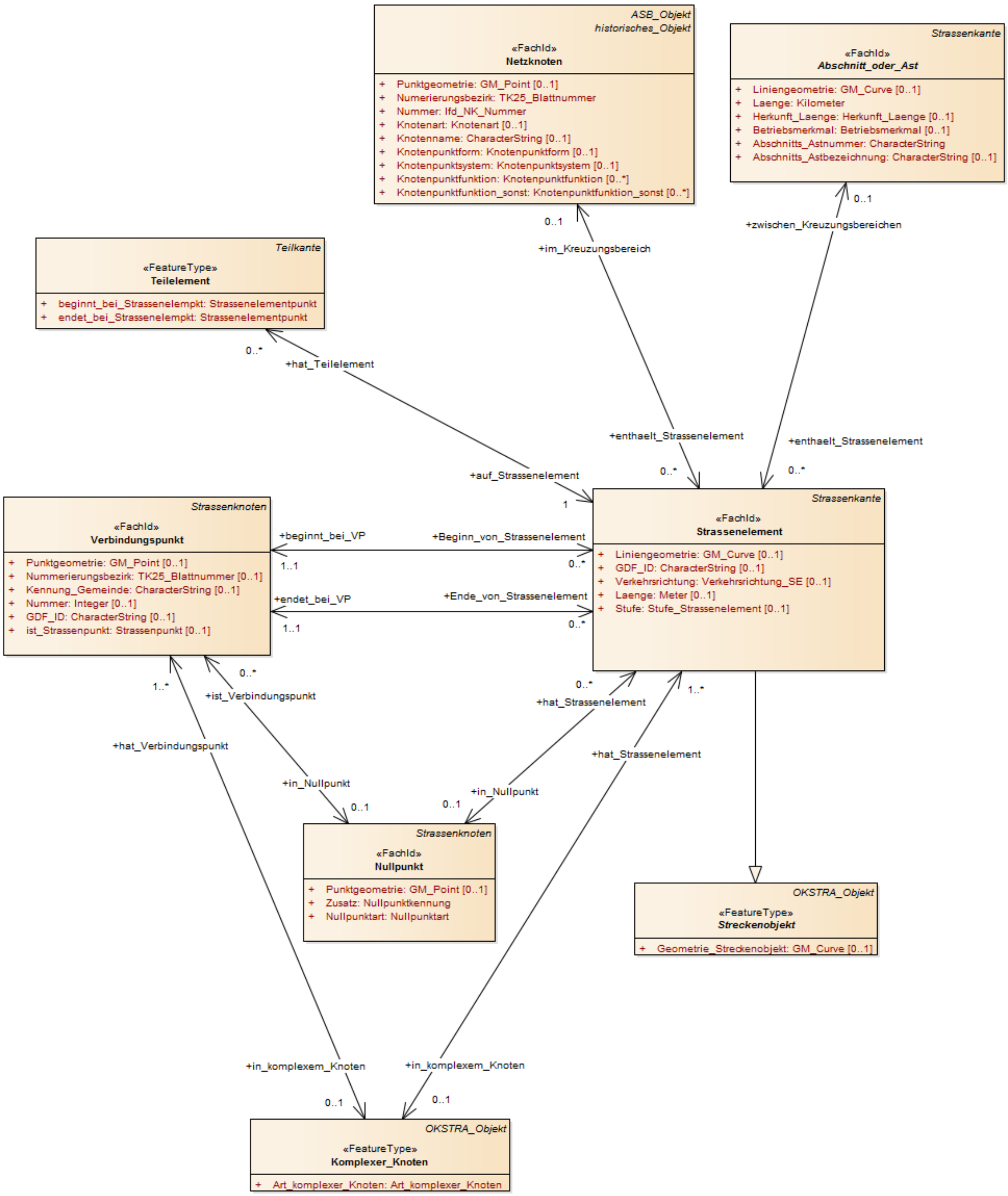

ASB_Objekt
historisches_Objekt
«FachId»
Netzknoten
+ Punktgeometrie: GM_Point [0..1]
+ Numerierungsbezirk: TK25_Blattnummer
+ Nummer: lfd_NK_Nummer
+ Knotenart: Knotenart [0..1]
+ Knotenname: CharacterString [0..1]
+ Knotenpunktform: Knotenpunktform [0..1]
+ Knotenpunktsystem: Knotenpunktsystem [0..1]
+ Knotenpunktfunktion: Knotenpunktfunktion [0..*]
+ Knotenpunktfunktion_sonst: Knotenpunktfunktion_sonst [0..*]
Strassenkante
«FachId»
Abschnitt_oder_Ast
+ Liniengeometrie: GM_Curve [0..1]
+ Laenge: Kilometer
+ Herkunft_Laenge: Herkunft_Laenge [0..1]
+ Betriebsmerkmal: Betriebsmerkmal [0..1]
+ Abschnitts_Astnummer: CharacterString
+ Abschnitts_Astbezeichnung: CharacterString [0..1]
0..1
+zwischen_Kreuzungsbereichen
0..1
+im_Kreuzungsbereich
Teilkante
«FeatureType»
Teilelement
+ beginnt_bei_Strassenelempkt: Strassenelementpunkt
+ endet_bei_Strassenelempkt: Strassenelementpunkt
0..*
+hat_Teilelement
+enthaelt_Strassenelement
+enthaelt_Strassenelement
0..*
0..*
+auf_Strassenelement
1
Strassenknoten
«FachId»
Verbindungspunkt
+ Punktgeometrie: GM_Point [0..1]
+ Nummerierungsbezirk: TK25_Blattnummer [0..1]
+ Kennung_Gemeinde: CharacterString [0..1]
+ Nummer: Integer [0..1]
+ GDF_ID: CharacterString [0..1]
+ ist_Strassenpunkt: Strassenpunkt [0..1]
+beginnt_bei_VP
+Beginn_von_Strassenelement
1..1
0..*
+endet_bei_VP
+Ende_von_Strassenelement
1..1
0..*
Strassenkante
«FachId»
Strassenelement
+ Liniengeometrie: GM_Curve [0..1]
+ GDF_ID: CharacterString [0..1]
+ Verkehrsrichtung: Verkehrsrichtung_SE [0..1]
+ Laenge: Meter [0..1]
+ Stufe: Stufe_Strassenelement [0..1]
1..*
0..*
+ist_Verbindungspunkt
+hat_Verbindungspunkt
0..*
+hat_Strassenelement
1..*
+hat_Strassenelement
+in_Nullpunkt
0..1
0..1
+in_Nullpunkt
Strassenknoten
«FachId»
Nullpunkt
+ Punktgeometrie: GM_Point [0..1]
+ Zusatz: Nullpunktkennung
+ Nullpunktart: Nullpunktart
OKSTRA_Objekt
«FeatureType»
Streckenobjekt
+ Geometrie_Streckenobjekt: GM_Curve [0..1]
+in_komplexem_Knoten
+in_komplexem_Knoten
0..1
0..1
OKSTRA_Objekt
«FeatureType»
Komplexer_Knoten
+ Art_komplexer_Knoten: Art_komplexer_Knoten

Figure 2.5: OKSTRA UML example of 'Straßenelement und Verbindungspunkt' diagram (source: http://www.okstra.de/docs/2017/html/index.htm)

2.3 Semantic Web

2.3.1 Introduction of Semantic Web

The Semantic Web is defined as a Web of Data, which is known as an extension of the Web through standards by the World Wide Web Consortium (W3C). The concept was addressed by Tim Berners-Lee in 1998 [10]. The architecture of the Semantic Web is shown in fig. 2.6. The Semantic Web will be built by adding more layers on top of existing ones and may take around ten years to complete.[33]. The main objective of the Semantic Web is to create a universal medium for the exchange of data, which can reach its full potential only if it becomes a place where data can be shared and processed by automated tools as well as by people.[27]. In fact, the Semantic Web involves publishing in languages specifically designed for data: Resource Description Framework (RDF), Web Ontology Language (OWL), and Extensible Markup Language (XML), which has a possibility to describe arbitrary things[38].

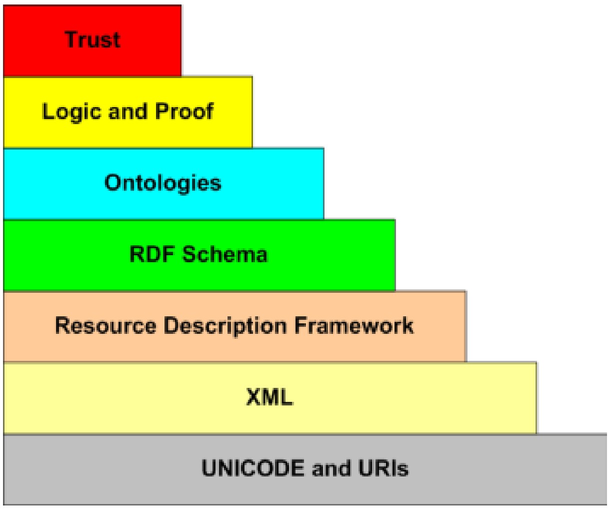


Figure 2.6: Semantic Web layered architecture (source: CCLRC Rutherford Appleton Laboratory)

The collection of Semantic Web technologies (RDF, OWL, RDFS, SPARQL, and etc.) provides an environment where application can query that data and draw inferences using vocabularies[13]. Basic Semantic Web technologies include representation languages of the Semantic Web: OWL and RDF, with RDF serving as the foundation and OWL as a formalism Web Ontology Language, as well as RDF Schema and RDF query language SPARQL.

2.3.2 OWL and RDF

OWL

The W3C Web Ontology Language (OWL) is a Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things[22], which is a computational logic-based language such that knowledge expressed in OWL format can be exploited by computer programs. OWL is built on RDF and RDF Schema and has potential to add more vocabulary for describing properties and classes[7]. Meanwhile,OWL documents, known as ontologies, can be published in the World Wide Web and may refer to or be referred from other OWL ontologies. OWL is part of the W3C's Semantic Web technology stack, which includes RDF, RDFS, SPARQL. etc.

RDF

RDF (Resource Description Framework) is a standard model for data interchange on the Semantic Web, which addresses one fundamental issue in the Semantic Web: managing distributed data[1]. The data model of RDF is deceptively simple, which is the triple of subject, predicate and object. The triple merely defines a directed binary relation between subject and object, which is called a predicate. The RDF triple can be visualized as a directed labelled graph (s, p, o model), consisting of subject, predicate, and object (as shown in fig. 2.7). Moreover, sometimes predicate and object maybe referred to as property and value, this is known as another representation of RDF triples: subject-property-value (s, p, v) model (shown as fig. 2.8). The fundamental concepts of RDF are resources, properties and statements. Resources can be considered as subjects or objects, which usually has a Uniform Resource Identifier (URI) which can be a Uniform Resource Locator (URL) to describe as a web resources. Properties are a special kind of resources that they describe relations between resources, known as predicate. While statements assert the properties of resources. A statement is an object attribute-value triple, consisting of a resource, a property, and a value[38]. Moreover, values can either be resources or literals[5]. The triple concept enables RDF to represent simple statements about resources as a graph of nodes and arcs representing the resources, and their properties and values and lets users describe resources using their own vocabulary.[38]

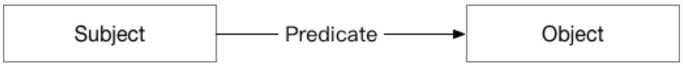


Figure 2.7: Graph of subject, predicate and object RDF triple

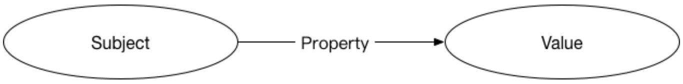


Figure 2.8: Graph of subject, property and value RDF triple

In the Semantic Web, RDF is serving as the foundation. RDF relies heavily on the infrastructure of the Web, using many of its familiar and proven features, while extending them to provide a foundation for a distributed network of data, which has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed.

As same as OWL, RDF is intended for situations in which the information needs to be processed by applications, rather than being only displayed to people. RDF provides a common framework for expressing (his information so it can he exchanged between applications without loss of meaning. Since it is a common framework, application designers can leverage the availability of common RDF parser and processing tools.

2.3.3 The Linked Data Approach

The Semantic Web was created not only publishing data on the web, but also aims to create links between data sources in order to use data-in-hand to find other related data [9]. This method called Linked Data, which lies at the heart of what Semantic Web, is about employing the Resource Description Framework (RDF) and the Hypertext Transfer Protocol (HTTP) to publish structured data on the Web and to connect data between different data sources, effectively allowing data in one data source to be linked to data in another data source.[14]. In another words, Linked data is simply about using the Web to create typed links between data from different sources (shown as fig. 2.9), which can be utilized to interoperate heterogeneous sources at the data level[13]. Linked

Data provides a publishing paradigm in which not only documents, but also data, can be a first class citizen of the Web, thereby enabling the extension of the Web of Data, known as Semantic Web[39].The four Linked Data principles are as follows [9]:

1. use URIs as names for things;

2. use HTTP URIs so those names can be looked up;

3. return useful information upon lookup of those URIs;

4. include links by using URIs which dereference to remote documents.

What should be mentioned is Linked Data refers to data published on the Web is machine-readable. Therefore, the meaning of data is explicitly defined, which can in turn be linked to from external data sets[3].

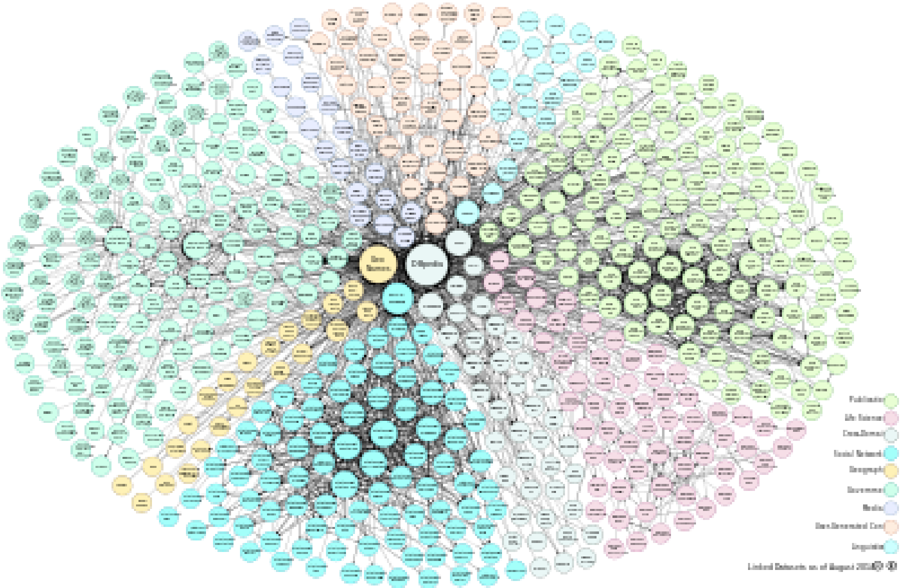


Figure 2.9: Example of The Linking Open Data cloud diagram source:lod-cloud.net

2.3.4 SPARQL

SPARQL Protocol and RDF Query Language (SPARQL) is the query language and protocol for the RDF[21], which is developed for semantic query of RDF databases and is able to retrieve and manipulate data stored in RDF data sets. A SPARQL query consists of a set of triple patterns. In these triple patterns, each element (the subject, predicate or object) could be a variable. The solution of the variables is found by matching the patterns in the query to triples in the RDF data set. SPARQL provide four types of queries[2]:

1.ASK: Ask whether there is at least one match of the query pattern in the RDF graph data;

2.SELECT: Select all or some of those matches in tabular form ;

3.CONSTRUCT: Construct a new RDF graph by substituting the variables in those matches in a set of triple templates;

4.DESCRIBE: describe the matches found by constructing a relevant RDF graph.

Moreover, SPARQL has an advantage to navigate all the relations in RDF graph data through graph pattern matching. Furthermore, multiple of simple patterns can construct a complex pattern based on logic relationships to explore more elaborate relations and result in the data.

2.4 Literature Summary and Related Work

2.4.1 Role and advantages of Semantic Web in AECO/FM industry

In order to theoretically demonstrate Semantic Web technologies can be used as the approach of CityGML and OKSTRA integration, the literature study initially started with an inspection of roles the Semantic Web technologies is playing of solving current issue in the AECO/FM domain and the advantages can Semantic Web technologies can bring based its feature.

Currently in the industry, the interoperability between heterogeneous data source is known as a long-standing challenge. Various information sources (e.g. BIM and GIS) is required to be combined and federated for enhancing the availability, efficiency and interoperability of information[50]. This demand of combinations of information and data source requires an effective data integration approach with high quality and universality. Therefore, the current circumstance stimulates the increasing interest in the use of semantic web technologies and linked data technologies, which has features that represent information in structured graphs and efficiently integrate various kinds AECO data sources. The Semantic Web technologies provide several advantages such as using one single data model RDF for representing any kind of information with its inherent semantics[11][49], using OWL to describe the knowledge and various data standard and structure among the AECO/FM industry[57], and has potential to link diverse RDF graphs of information across the different domains easily. Therefore, Semantic Web technologies could be the ideal technical means to solve the interoperability issue[50].

Based on these natures, the roles and usage of the Semantic Web technologies in the industry are mainly about 1) solving the interoperability issue among various data formats, standards, and tools or at least improve information exchange processes between them: Casey and Vankadara [17] stated that enabling technologies of Semantic Web such as XML and RDF, could provide a universal accessibility to ontologies with foundations for software services capable of reasoning and making inferences based on domain knowledge in the form of ontologies. Consoli et al.[20] addressed that the use of Semantic Web technologies such as ontologies and Linked Open Data (LOD) have high potential and can be known as a solution towards the current data management for modern smart cities is facing different kinds of challenges, especially for integrating heterogeneous urban data sources from multiple domains. Moreover, the use of linked data technologies has meaningful usage of improving interoperability between systems and data models in a generic way and enable a a higher level of data analysis[25].; 2) as a bridge to link to various domains that are relevant with AECO domains, for examples: Stefan et al.[12] stated the use of Semantic Web technologies like linked-data could enrich processing of the annotated data, especially for a variety of Smart city data sources like data from WoT and IoT to be processed. Meanwhile,Zhang et al.[63] suggested the technologies of the Semantic Web could be considered as a key element to solve the complexity of IoT data obtained for smart cities and can offer an interface to facilitate the fusion of IoT data with existing knowledge. Furthermore, Lecue et al.[45] developed a system supporting semantic traffic analytic and reasoning for city based on Semantic Web technologies to integrates heterogeneous data and historical and real-time traffic conditions.

2.4.2 Related Works

Recently, Semantic Web technologies has been increasingly adopted by in the domains of AECO/FM, however, the most of the applications are typically centre on buildings and BIM/IFC. One of the perspective is about converting BIMs in to Semantic Web data representation. For examples, Schevers and Drogemuller[54] developed a unidirectional conversion from an IFC to an OWL to convert not all the IFC data to OWL but supports the search for a more appropriate mapping. Later on, Beetz et al.[8] addressed that the EXPRESS modelling language used by IFC has limitations in resources reuse and interoperability. They developed an approach to achieve the translation from IFC schema to Semantic Web graph. Moreover, with development of data transformation approaches, researches also focused on considering implement Semantic Web as the middleware to improve the interoperability of BIM/IFC with other data sources. Pauwels et.al[49] suggested considering the semantic web as an alternative approach to enhance the interoperability of BIM and indicate how it could be used to solve the BIM interoperability issue. Moreover, Curry et.al[23][24] employed Linked Data approach to overcome interoperability challenges in order to enable data from multiple sources to be merged into a holistic scenario models for different stakeholders of buildings and further used the Linked Data as mean for developing cloud-based building data services in order to create an integrated well connected graph of relevant information for managing a building. As a conclusion, although these previous researches focusing on Semantic Web applications in BIM domain, they have contributed to prove Semantic Web technologies could extend the development possibilities of BIM and appear as an alternative approach to improve interoperability and interaction of heterogeneous data source.

Similar with BIM, the greatest challenge of CityGML integration and interoperability issue is the semantically inconsistent with other data sources[15][19]. Therefore, in the early stage, most of the researches put the efforts on data translating and format conversion between CityGML and other data resources. In 2007, Döllner and Hagedorn[28] discussed the integration of urban GIS, CAD and BIM data based on unified 3D model system, in which BIM is converted to CityGML. Isikdag and Zlatanova (2009b) have built a framework for conversion of IFC to CityGML automatically by transforming both semantic and geometric information. De Laat and Van Berlo[26] introduced their GeoBIM CityGML extension, which translate the semantics inside IFC as GIS CityGML contexts.

With the development of both CityGML and Semantic Web, more attention has turned to involve CityGML as a data source into Semantic Web world. Métral et al.[46] argued CityGML is insufficient for representing the semantics of urban information, which leads 3D city models based on CityGML are not able to be used in urban tasks that involving multiple actors and multiple tools. Therefore, for sake of overcoming this limitation, CityGML should considering involving ontology concept to improve its interoperability with other information systems. Later in 2012, Métral et al.[48] developed a direct translation of the CityGML 2.0 schema to an OWL description by using XSLT (http://cui.unige.ch/isi/onto/). This developed ontology is reused in this research.

In 2013, Van den Brink and Janssen[59], develop an approach based on XSLT to transfer the GML data to RDF representation. Their research has successfully translated the GML instance data into RDF graphs and provided a general XSLT template.

Furthermore, several researches have implemented the Semantic Web technologies approach as the means of integration CityGML data with heterogeneous data sources, which have threw light on this research. El-Mekawy and Östman[30] has develop a formal mapping between IFC and CityGML ontologies, which proposes a more expressive reference ontology between IFC, CityGML semantic models and an intermediate unified building modelled (UBM), based on reference ontology, a bidirectional formal mapping between IFC and CityGML ontologies that allows bidirectional conversion.

Vilgertshofer et al.[60] addressed CityGML and IFC models cannot be comprehensively mapped onto one another without data loss. Thus, they employed Linked Data approach to provide a semantically rich connection between the domains of BIM and GIS. In their research, EXPRESS schema and CityGML XML schema are translated to OWL representation, then manually inspect both schemas and creating an OWL mapping that contains the similarities on the schema level of IFC Tunnel and CityGML, at last an instance RDF file is created to connect the corresponding entities of the IFC and GML instance files.

In 2015, Métral et al.[47] addressed a RDF-based specification of 3D city model visualization technique. In their research, all the heterogeneous data sets are translate to isolated RDF graphs and then store in RDF in the triple store. While in the visualization phase, each RDF graphs apply the desired visualization transformation to produce a graph of abstract visual entities. However, they considering CityGML files as a semi-RDF/XML file and the gml:id attribute can serve as a resource identifier for the RDF rather than URI. Therefore, it only requires a slight modification of add RDF elements in CityGML file to translate CityGML data to RDF/XML representation.

Moreover, Hor et.al in 2016 [41] proposed a novel approach for integrating BIM and GIS using Semantic Web technologies and RDF graphs, which uses Semantic web ontologies is as nutshell between BIM and GIS to integrate BIM and GIS technologies into one unified model. The approach including several step of integrating IFC and CityGML:1)constructing IFC ontology 2)constructing GIS ontology 3)Ontology mapping by linking similar concepts and relationships between two ontologies 4)Querying integrated ontology based on application domain.5)Load the instance data. This research shows a paradigm of using Semantic Web technologies and Linked Data method to integrate IFC and CityGML model, which has contribution to inspiring the establishment of this research's methodology.

As introduced above, the CityGML is able to contain semantic information of city entities, but it is insufficient for representing the semantics of urban information and requires enhance it interoperability with other data source. The Semantic Web could be regard as integration approach of various information resources as well as generating and sharing new knowledge.

2.5 Summary

In this chapter, firstly the basic knowledge of the relevant concepts and technologies are introduced in order to illustrate the nature of these concepts and technologies. Both CityGML and OKSTRA data models are represented in XML format, contains both geometric data and semantic information. Meanwhile, their data schema is in XSD format. Moreover, the RDF is the standard model for data interchange on the Semantic Web and OWL is a Semantic Web language to represent data structure. Therefore, in order to employ Semantic Web technologies to integrate CityGML and OKSTRA data models, the first process is data transformation, which aims to convert the CityGML XML model and OKSTRA XML model into Semantic Web data representation, which will be the process of XML to RDF and XSD to OWL. After, the Linked Data approach could be involved to realize the data integration.

The following part of this chapter is a brief summary of the literature and related work. As a conclusion, the Semantic Web technologies in the AECO/FM domain are mainly implemented to solve 1) the interoperability issue among various data formats, standards, and tools and 2) as a bridge to link to various domains that are relevant with AECO/FM domain. Meanwhile, several researches has been done to employ Semantic Web technologies with CityGML domain, the main purposes of these researches aim to enhance the interoperability of GIS and CityGML with other data sources, especially for BIM. All these efforts proved that Semantic Web could be a solution to for CityGML integration issue and have reference value for establish the integration methodology in this research.

.

Chapter 3

Methodology

In this chapter, the method to integrate CityGML and OKSTRA by implementing the Semantic Web technologies is firstly described in Section 4.1. Following with the framework design and detailed process description of the experimental implementation are based on this method with CityGML and OKSTRA models, including data conversion, data linking and query development.

3.1 Integration Method

The integration method of CityGML and OKSTRA proposed in this thesis is based on three concepts or technologies:

1. CityGML model and standard
2. OKSTRA model and standard that have been already mentioned in the preliminary chapter
3. Semantic Web technologies of Linked Data and RDF graphs.

The general method is to implement Semantic Web technologies as the approach to integrate CityGML and OKSTRA models. Therefore, based this general method, the integration methodology consists of several steps as follow:

1. Constructing Ontology of CityGML ($O_{citygml}$) and OKSTRA (O_{okstra}) under the application domain:

Initially, the ontology models under the application domain of both CityGML and OKSTRA should be developed in order to illustrate the predicates between each subject and object. Aforementioned in the previous chapter, both CityGML and OKSTRA data are written in XML format and their schema is XSD. Therefore, a series of process that transforms CityGML and OKSTRA data schema from a traditional format to RDF/OWL graphs are required.

2. Transform both CityGML and OKSTRA Data Models into RDF graphs based on $O_{citygml}$ and O_{okstra}:

After $O_{citygml}$ and O_{okstra} are generated, a data translation from XML-format CityGML model and OKSTRA model into RDF graph should be conducted corresponded with these two ontologies. Aforementioned, RDF is a graph where the nodes are URI references, Blank Nodes or Literals. URIRefs and Blank Nodes can both be thought of as resources. The core idea behind converting every data-XML including CityGML and OKSTRA into RDF graphs is that every complex XML element maps to a resource and every attribute maps to a property of this resource node [61]. Thus, the translate principle of both CityGML and OKSTRA XML-format models into RDF graphs is every class in CityGML or OKSTRA XML model map to a resource and are followed by its properties. After the transformation, two RDF graphs should be uploaded to a triple store that able to store and create new RDF data for further data linking.

3. Create links between the constructed ontologies of $O_{citygml}$ and O_{okstra} based on Linked Data method to generate the integrated ontology $O_{citygml-okstra}$:

In this part, based on the Linked Data method, two ontologies $O_{citygml}$ and O_{okstra} will be integrated as one ontology $O_{citygml-okstra}$ based on recognizing the semantic correspondences between two ontologies. This ontology integrating process has two steps: 1)ontology mapping and 2) involving intermediate ontologies. In terms of the process of ontology mapping, it is the premiere and mandatory step that it inspect both of ontologies to determine the objects in distinct ontologies that can be directly linked and then creates corresponded links. The direct relationships can be classified as two different types: 1) As same, i.e., $Object_{citygml}$ (an object in CityGML ontology) is equivalent to $Object_{okstra}$ (an object OKSTRA ontology), which means they have similar concepts. Therefore, during the mapping process, these two objects can be matched and joined as one object in the newly integrated ontology. 2) required the establishment of direct relationships: there are direct logic relationships between a $Object_{citygml}$ and a $Object_{okstra}$ existed but no links between them. Therefore, this circumstance requires creating new relationships between objects in order to directly links two objects. The process of mapping of ontology is illustrated in fig. 3.1. As seen, the enriched ontology $O_{citygml-okstra}$ contains all the objects from both $O_{citygml}$ and O_{okstra}, and objects in two ontologies are directly linked or overlapped based on ontology mapping. Mathematically, this ontology mapping process can also be given as:

$$O_{citygml-okstra} = O_{citygml} \cup O_{okstra}$$

.

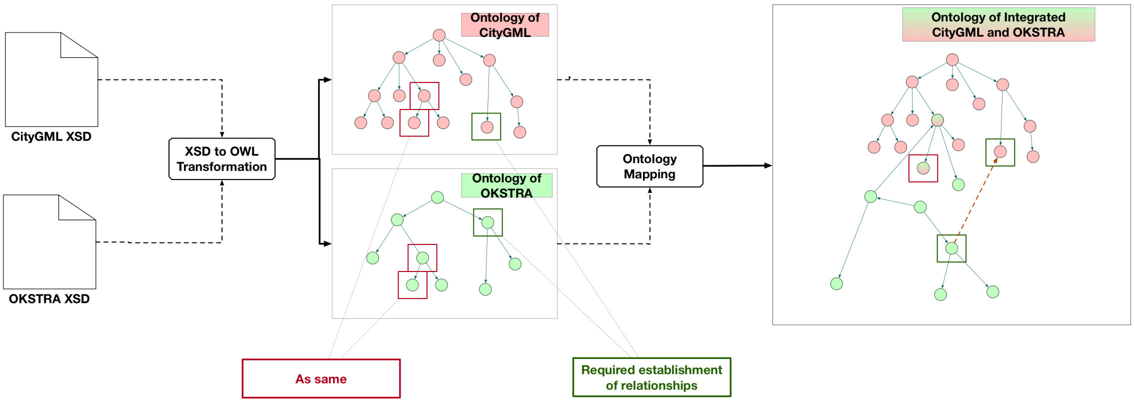

CityGML XSD
OKSTRA XSD
XSD to OWL Transformation
Ontology of CityGML
Ontology of OKSTRA
Ontology Mapping
Ontology of Integrated CityGML and OKSTRA
As same
Required establishment of relationships

Figure 3.1: The Process of Ontology Mapping

However, the second step of involving intermediate ontologies is optional. Only if the relationships between respective ontologies can not be found and created as direct links in the ontology mapping step, involving intermediate ontology is necessary. Due to the relationship between in two ontologies can not be found as direct link, thus, it requires an additional ontology to be an intermediary that both of the CityGML ontology and OKSTRA ontology can be directly link with this it. Thus, the integrated ontology is constituted by $O_{citygml-okstra}$,O_{okstra} and intermediate ontology with links between them. This process is shown in fig. 3.2

4. Link the RDF graphs of CityGML and OKSTRA models:

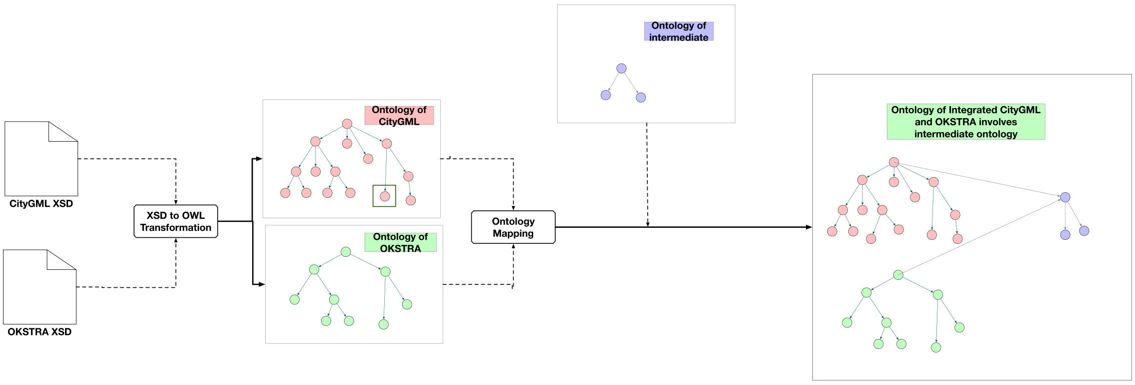


Figure 3.2: The Process of Ontology Mapping with involving intermediate ontology

Since the relationships between objects in two ontologies have been defined in the ontology linking process, the corresponding linkages can be created between two RDF graphs by creating RDF statements to describe the relationships. These statement triples should be merged with existed RDF graphs of both CityGML and OKSTRA model as the final integrated RDF set. By using the same URI system, the integration is automatically realized during the RDF graph merging process by computer. The integrated RDF set will be uploaded in the triple store for next step validation and application.

5. Query the integrated RDF set for practical application:

After the integration process has been accomplished, the integrated RDF result should be validated and utilized for practical applications afterwards. The validation and application are realized by using SPARQL approach to query the RDF set to verify if the integration is successful and this RDF set could be further implemented as a semantic data base for city information query.

Corresponded with the integration method discussed previously, an architecture of the integration system and data flow are designed and presented as shown in fig. 3.3 . The integration system consists of four parts: 1) CityGML and OKSTRA models input, 2) RDF transformation 3) Data linking 4) Application. The input models contain CityGML and OKSTRA data represented as XML formats. RDF transformation refers to the process for translation the models to RDF graphs based on the developed ontologies. Data linking part aims to associate two RDF graphs corresponded with the integrated ontology to achieve the data linking. Application part aims to utilize the integration product, for example, extracting useful semantic information for specific purpose.

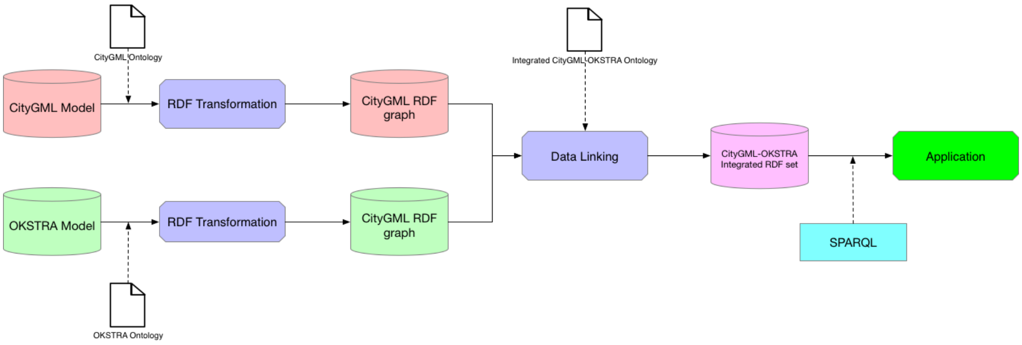


Figure 3.3: The Integration system and data flow

3.2 Process Map of Experimental Implementation

In order to realize and validate the created integration method, it is necessary to conduct an experimental implementation corresponded with the integration method aforementioned. In this section, the process map of the experimental implementation will be designed based on the integration method to guide the practical implementation process. The general process map for this experimental implementation is shown in fig. 3.4 , which includes four main parts: data preparation, data transformation, data linking, and query development. More detailed descriptions of the process will be displayed as follow.

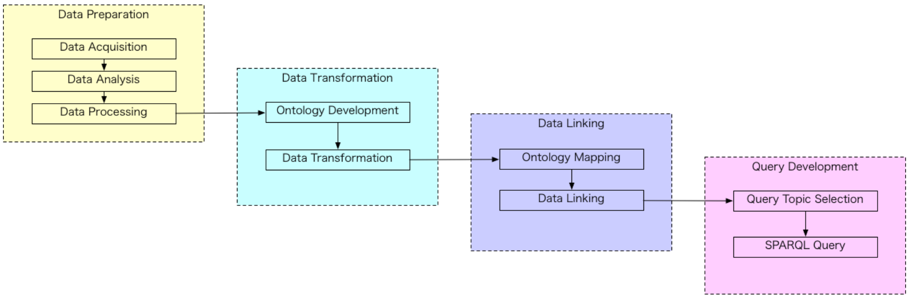


Figure 3.4: The process map of experimental implementation

3.2.1 Data Preparation

Data Acquisition

The initial step of the experimental implementation is to prepare data for the whole integration process, which first requires obtaining complete available data source. Therefore, the process starts with data acquisition, the step that could affect the whole process results. The aim of the data acquisition is to obtain suitable raw data resource that with the legal license. The key criteria for data selection include the availability of the data and suitability of this research. Apart from the key criterion, the accuracy, richness, and stability of the data should be considered as important criteria.

Data Analysis

After obtaining the raw data sources, the next step is to analyze. This data analysis process includes analyzing the data schema and data sets themselves. Analyzing the data schema aims to gain insight of the data structure, concepts and the relationships among different concepts, which is helpful for further ontology development and XML to RDF transformation. To look up data sources themselves could gain a clear understanding of the instance data and semantics within the data sets.

Data Process

The data process aims to produce a meaningful appropriate data set by modifying or adjusting the raw data set for further integration, since the raw data sets could be exceed the research scope or need to coordinate with the further application. The data process in the implementation mainly focus on modifying the raw CityGML and OKSTRA XML files to ensure the data is not exceed the research scope and suitable for RDF translation without data loss.

3.2.2 Data Transformation

The data transformation is mainly about to translate the XML format CityGML and OKSTRA instance models into RDF graphs. With the purpose of unifying data format and preparing for data linking, the data transformation is structured into two steps: 1)Ontology development and 2) Transforming data-XML to RDF.

Ontology development

Before the transforming of XML instance models to RDF graphs, ontologies of both CityGML and OKSTRA to describe resources and relationships inside the instance model and within practical application domain should be established. In this research, there are two approaches for the development of the ontologies, first is to search for existing ontologies if can be applied. Second is to develop the ontologies based on the data schema.

Transforming data-XML to RDF

Transforming data-XML to RDF includes two parts: 1) extract the data/semantics in XML files and 2) recombine them into RDF graphs. Currently, there are no comprehensive tools can be directly utilized to convert data-XML to RDF graphs, all the tools are adapted to correspond specific XML file, because of the diversity of XML that every type of XML file for different uses that contains different elements and restricted by its XML schema. Therefore, suitable transformation tools should have functions to achieve the functions of parse and extract the XML elements and write RDF triples. Meanwhile, an appropriate serialization format should also be considered. Furthermore, for evaluating of the data transforming and merging result, several SPARQL queries can be executed. If the SPARQL queries go smoothly and get correct results, it proves that the data transforming and merging part works well.

3.2.3 Data Linking

After the RDF graphs from different data sources are generated, the graphs are still isolated and should be merged in one RDF set and associated via the data linking process.

Ontology Mapping

To select which dependent classes from various sources to perform linking is based on the logic relationship between all the classes in the individual ontologies. Therefore, identifying of the linking objects is achieved by ontology mapping, and the method has been described in the last section .

Data Linking

The newly created links should perform as a bridge to navigate between different semantics island and could be used to extract useful semantic information. The linkage can be a reference of existing ontology vocabularies or self-created. Moreover, abundant Semantic Web applications can be used for data linking, such as Apache Jena, Silk, RDFlib and etc.

3.2.4 Query Development

There are two purposes of designing the query part in the implementation: 1) verify the integration method and process: SPARQL queries can be utilized as an evaluation approach of data conversion and data linking, which are able to evaluate the RDF data transformation or data linking are successful. 2) reflects practical usage of the integration product: SPARQL queries can be used to query useful semantic information based specific topics which cannot be achieved in isolate data set.

Query topics selection

In this research, CityGML and OKSTRA models are the data sources. Two data resources have their data range that CityGML model contains city and building relevant semantic information and geometric data, while OKSTRA model contains road information and semantics. Therefore, all the queries should be designed based on available data range and meaningful usage, which requires an analysis to define topics that not only can be realized based on data-in-hand but also have meaningful usage.

SPARQL query development

According to the selected query topics, the SPARQL query could be conducted. The SPARQL query aims to use the SPARQL language to extract query result among direct or indirect resources relationships.

Chapter 4

Transformation of the Data

In the previous chapter, a complete integrating methodology has been introduced. In the next two chapters an experimental implementation based on this methodology is described, which has a purpose of validating the developed data linking method and simultaneously testing the interoperability of two data platforms CityGML and OKSTRA by using Semantic Web technology. The implementation itself includes two parts: data transformation and data linking. In this chapter the data transformation process of both CityGML and OKSTRA data model into RDF graphs is described, including the explanation of data preparation phase, ontology development of each data and data transformation approaches description. This is followed by a description of data linking and query process in the next chapter.

4.1 Data Preparation

4.1.1 Instance Data Resource Acquisition and Introduction

In order to realize the experimental implementation of integrating CityGML and OKSTRA data, the instance data resource acquisition is the first step. Initially, a target area should be determined according to several scenarios of searching an optimal target area for the experiment. The selecting scenarios were set up including:1) considering the data access availability 2)searching within the Germany territory (due to OKSTRA is a German infrastructure information standard that only widely applied in Germany currently) 3)the area contains with several main roads and a number of buildings to ensure the rich and sufficient building and road information that can be used and linked in the data linking process. Therefore, in this experimental implementation, the city of Aachen is the selected. Aachen is a German city in the North-Rhine Westphalia state, known as the westernmost city in Germany that near the borders with Belgium and the Netherlands. Within the territory of Aachen, several roads such as A54, B1, B1A,B264, B57, B258, L136 and etc have linked with the entire city(shown as fig. 4.1).

Nevertheless, the whole city is a huge scale that exceed the scope and operability of the experiment. Therefore, the selected target area was further shrink as a square region in the northeast part of Aachen. In this part, road B1, B1A and L136 cross by and a number of buildings for various functions are located. The coordinate of four square vertices in two dimensions is (295000.0, 5629000.0), (295000.0, 5630000.0), (296000.0, 562000.0) and (296000.0, 563000.0) in ETRS89-UTM32 Geo-coordinate system (shown as fig. 4.2).

The CityGML model of the target area in Aachen is acquired from CityGML organization homepage, which is published as an open Geo-data set of 3D city models for the entire North Rhine-Westphalia state. Moreover, all the CityGML models in this model set are separated in different cities and named based on the lower corner's coordinates, which is easily navigated. The

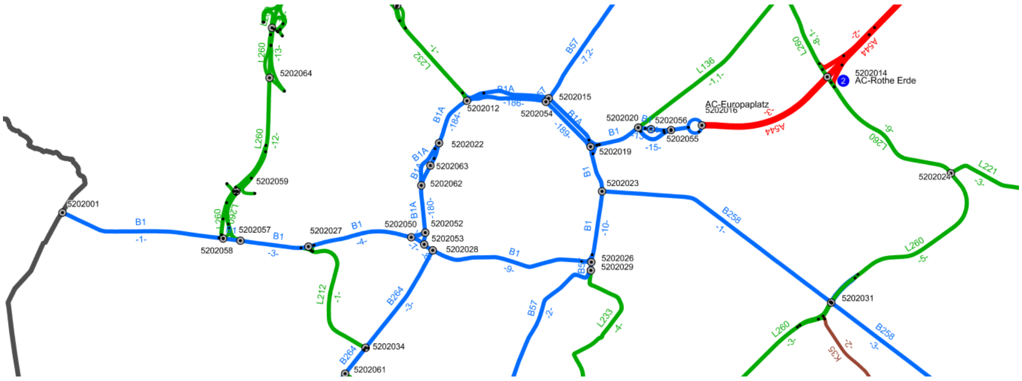


Figure 4.1: The Road Network in Aachen, source: the North Rhine-Westphalia Road Information Bank

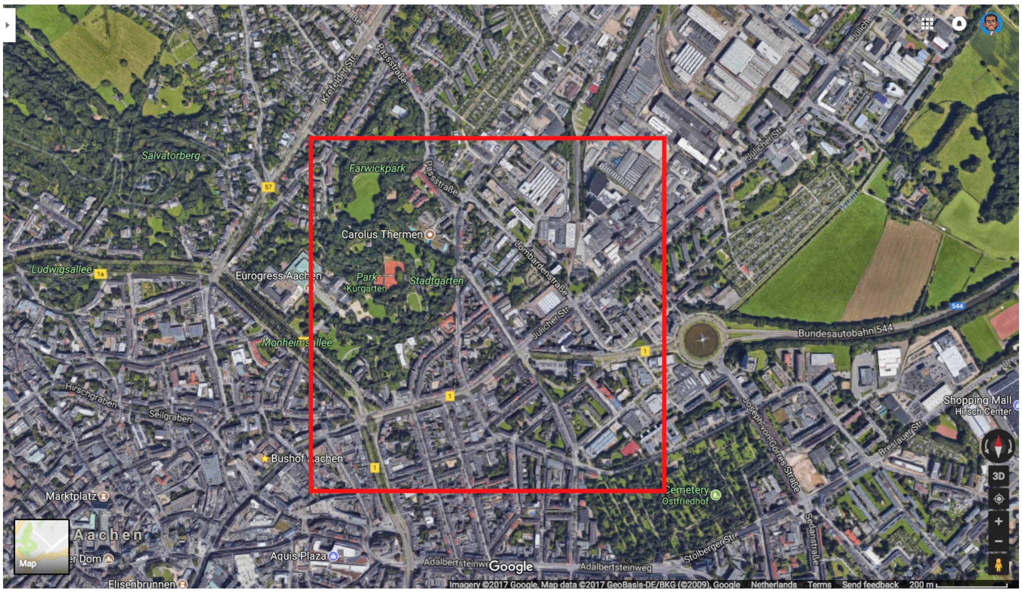


Figure 4.2: The Target Area (in the red square), source: Google Map

model set contains both LoD1 and LoD2 models of the buildings are present and partial of the relevant semantics is present as well. The chosen model is LoD2 model due to obtain more semantics, however, in the LoD2 model there are still some LoD1 buildings without rooftop. The visualization of the target area in CityGML visualization tool 'Azul' is shown in fig. 4.3.

Meanwhile, the OKSTRA data was obtained from Mr.Stefan Wick in the North Rhine-Westphalia Road Information Bank. The provided data set contains basic OKSTRA infrastructure/road information of the whole North-Rhine Westphalia state, which is huge and complex, therefore the OKSTRA data in-hand need a further data processing to extract the OKSTRA data of the target area, and the process of data processing will be described in next section.

4.1.2 OKSTRA Data Processing

Aforementioned, the OKSTRA data-in-hand covers the entire North Rhine-Westphalia state, which is exceed the scope of the experiment. Thus, in order to find the road network data model in the target area, a data processing is conducted in order to extract the OKSTRA data of the

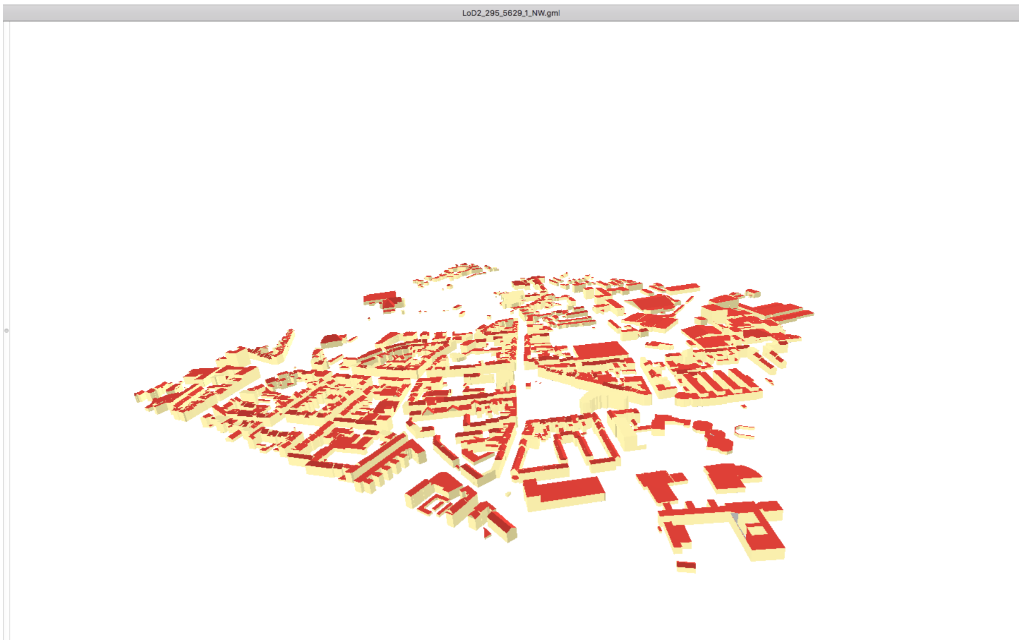


Figure 4.3: The Target Area CityGML Model Viewing in Azul

target area.

Data Analysis
The road model in OKSTRA is shown in the fig. 4.4 , in which a real-world road is represented by

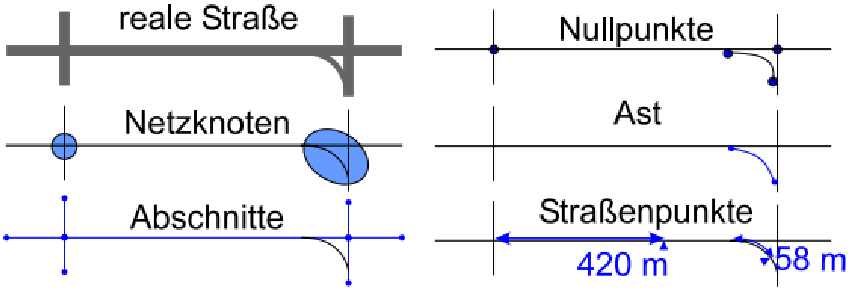


Figure 4.4: The Constitution of OKSTRA Road Model, Resource: okstra.de

several model components: Nullpunkte (English: Zero point of the Road), Netzknoten(English: Road Network Node), Ast(English: Road Branch) and Abschnitte (English:Road Section), all these components are elements in the OKSTRA XML file as "gml:featureMember".Furthermore, all the roads constructed by these components constitute the road network model (shown as fig. 4.5). Therefore, to extract the road information in the target area is equivalent to extract all the road represent elements within the target area from the entire North Rhine-Westphalia OKSTRA XML file.

Obtaining the Target Area Data
In the OKSTRA XML data file, a Netzknoten is a OKSTRA object type for representing a network node, which is a height-equal node that results from the traffic connection of two or more roads of the relevant road network. A Netzknoten has sole position 3D coordinates, therefore the

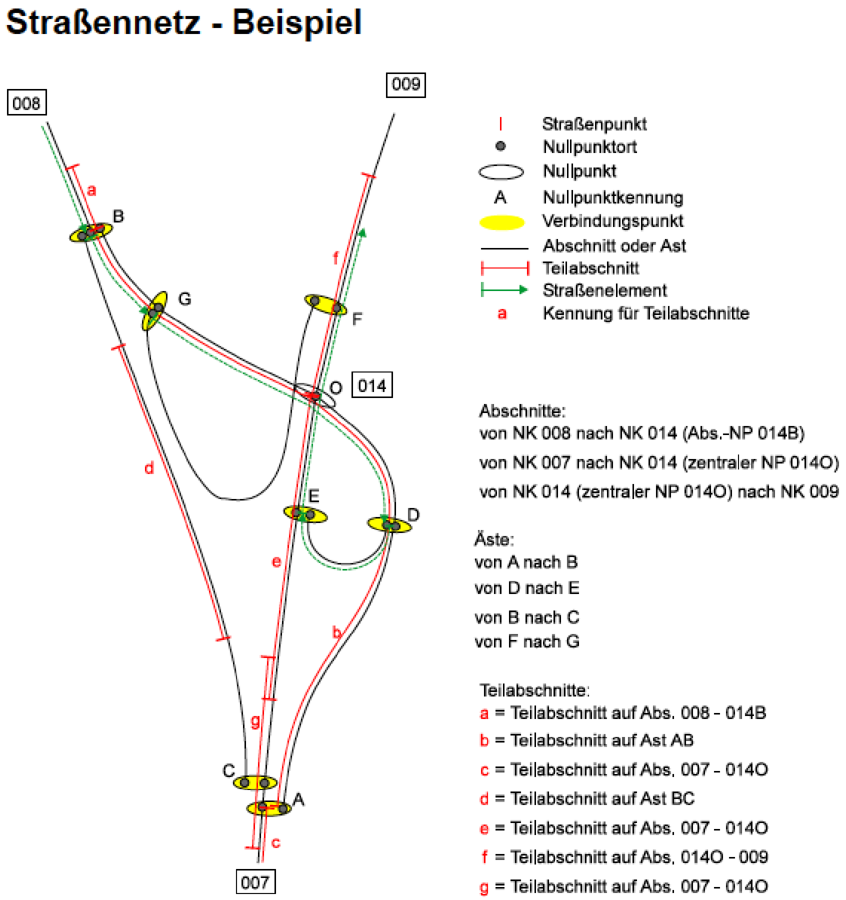

Straßennetz - Beispiel
008
009
014
007
A
B
C
D
E
F
G
O
a
b
c
d
e
f
g
Straßenpunkt
Nullpunktort
Nullpunkt
Nullpunktkennung
Verbindungspunkt
Abschnitt oder Ast
Teilabschnitt
Straßenelement
Kennung für Teilabschnitte
Abschnitte:
von NK 008 nach NK 014 (Abs.-NP 014B)
von NK 007 nach NK 014 (zentraler NP 014O)
von NK 014 (zentraler NP 014O) nach NK 009
Äste:
von A nach B
von D nach E
von B nach C
von F nach G
Teilabschnitte:
a = Teilabschnitt auf Abs. 008 - 014B
b = Teilabschnitt auf Ast AB
c = Teilabschnitt auf Abs. 007 - 014O
d = Teilabschnitt auf Ast BC
e = Teilabschnitt auf Abs. 007 - 014O
f = Teilabschnitt auf Abs. 014O - 009
g = Teilabschnitt auf Abs. 007 - 014O

Figure 4.5: OKSTRA Road Network Model, source:okstra.de

coordinates of Netzknoten could be utilized to navigate all the Netzknoten within the target area, sequentially according the elements linkage to find all the other road elements. Thus, the procedure to extract the OKSTRA road data of the target area is initially to find all the Netzknoten in the file, which should parse the entire OKSTRA XML and search all the Netzknoten elements to find their coordinates . This step is realized by developing a Python program using several Python packages, parsing the file and using the Xpath to find the coordinates for 'Netzknoten'. The Python script for seeking the Netzknoten within the target area is shown in the the Appendix B, the output of the program is a CSV file contains 3D coordinates of the 'Netzknoten'. Next, those 'Netzknoten' whose coordinates that are inside the boundary of the target area need to be determined. To realize this, a Python program (shown as Appendix C) is implemented in order to find satisfied result based on the generated 'Netzknoten' coordinates CSV file.

Eventually, four 'Netzknoten' were found inside the target area (shown as table 4.1), and these Netzknoten have linkage with other road elements such as Nullpunkt. Based on the relationships of between each elements, all of the essential road elements in XML file could be navigated and extracted to constitute a new OKSTRA-XML file for the target area (shown as fig. 4.6).

Table 4.1: List of Netzknoten Coordinates in Target Area

Netzknoten Coordinates List			
Netzknoten	X	Y	Z
Oklabi.15927.090001lu7ad	295157.167	5629195.85	0
Oklabi.15927.090001luamn	295601.203	5629372.571	0
Oklabi.15927.090001lvr8n	295719.931	5629358.3	0
Oklabi.15927.090001m323	295905.711	5629349.295	0

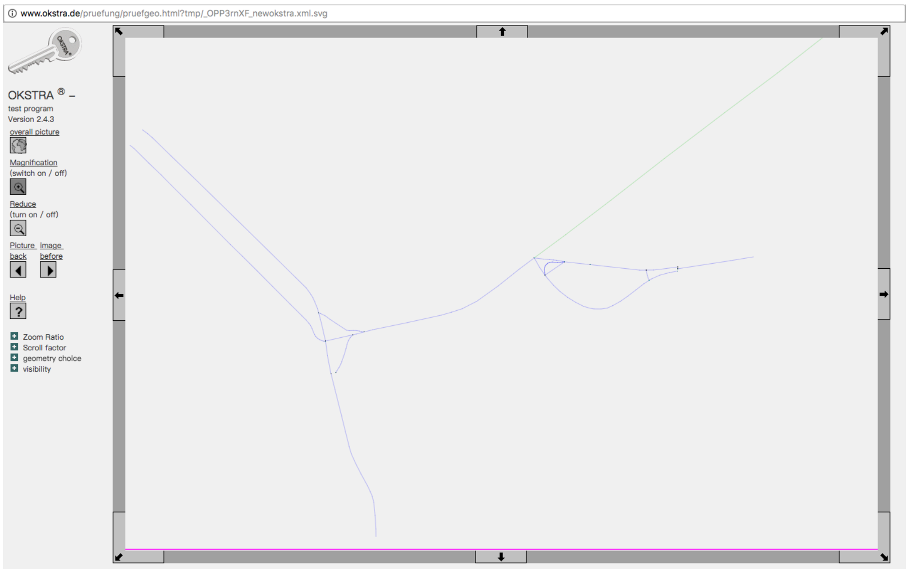


Figure 4.6: The Visualization of the OKSTRA Model in the Target Area

4.2 Data Transformation

After data preparation phase is completed, the data transformation could be conducted. The general idea of data-XML to RDF transformation consists two phases. The first one is to distinguish and derive the classes and properties from XML. This phase could be achieved by collecting of classes and properties from the XML schema which describes structure of data-XML. The second phase is producing RDF graphs by extracting the data in the generated XML file from first step and compiling to RDF triples. Therefore, based on the general transformation idea, the procedure of the transformation itself includes two parts: ontology development and the data-XML to RDF conversion. In terms of the process of ontology development, it coincides with the first phase distinguish and derive the classes and properties from CityGML and OKSTRA XML Schema file. Meanwhile, the data-XML to RDF conversion aims to produce the RDF graphs for both platforms.

4.2.1 Constructing Ontologies for the XML Files of the Implementation

An ontology is a formal specification of a shared conceptualization [37], which aims to illustrate the interrelationships between various entities from a particular domain including their name, types, properties, and data type. Thus, the ontology could be used as a representation to derive the classes and properties from XML. In this experimental implementation, both of CityGML and OKSTRA ontologies should be developed within the research scope to illustrate the data structure, especially different classes and their properties. Therefore, the created ontologies could be applied to guide the data conversion and for further data linking process.

The ontology of entire CityGML standard has been developed by University of Geneva in OWL format, which is constructed based on the XML schema of CityGML standard. This existed CityGML ontology is determined to be directly utilized for this research, due to the fact that utilizing the existed ontology could enhance the veracity of further ontology mapping process. Nevertheless, from the data perspective, the CityGML instance model for this experimentation only contains the data of buildings in the target area. Therefore, the existed ontology should be adapted in order to create coordinated CityGML ontology corresponded with the instance model. The representation of the ontology in OWL can be seen in the Appendix E. Notice that in fact, few of current open CityGML model sources containing the data of the road or infrastructure, most of they only contain the partial semantics of buildings compare with the full scale data schema.

Moreover, in terms of OKSTRA ontology, since there is no direct artifacts for this domain, it should be self-constructed in this research. According to the OKSTRA data XML schema, the ontology of OKSTRA model can be produced to describe relationships of classes and properties that derives from OKSTRA XML schema which well described the OKSTRA data structure. The principle of distinguishing the element in XSD is a class or a property in OWL is that each complexType in OKSTRA XML schema is consider as a class and each sub-element belong to this class is regarded as a property or sub-classes of this class.

Since both OKSTRA schema and OWL are written in XML format, the main approach to create the OKSTRA ontology of well-structured XML is to convert their XML schema into OWL by using the Extensible Style sheet Language Transformations (XSLT), shown as fig. 4.7. XSLT is a widely-used language for transforming XML documents into other XML documents by using a XSLT stylesheet with converting principles. In order to convert OKSTRA XML schema, an XSL (Extensible Stylesheet Language) template is scripted to generate the OWL of OKSTRA road network module XML schema in RDF/XML format (As shown in Appendix D). The conversion principle is corresponded with the distinguishing principle that convert each class in OKSTRA XSD to an OWL class and the properties belong to this class translate to its property restrictions.

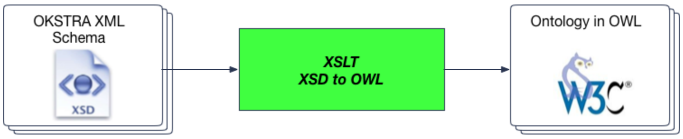
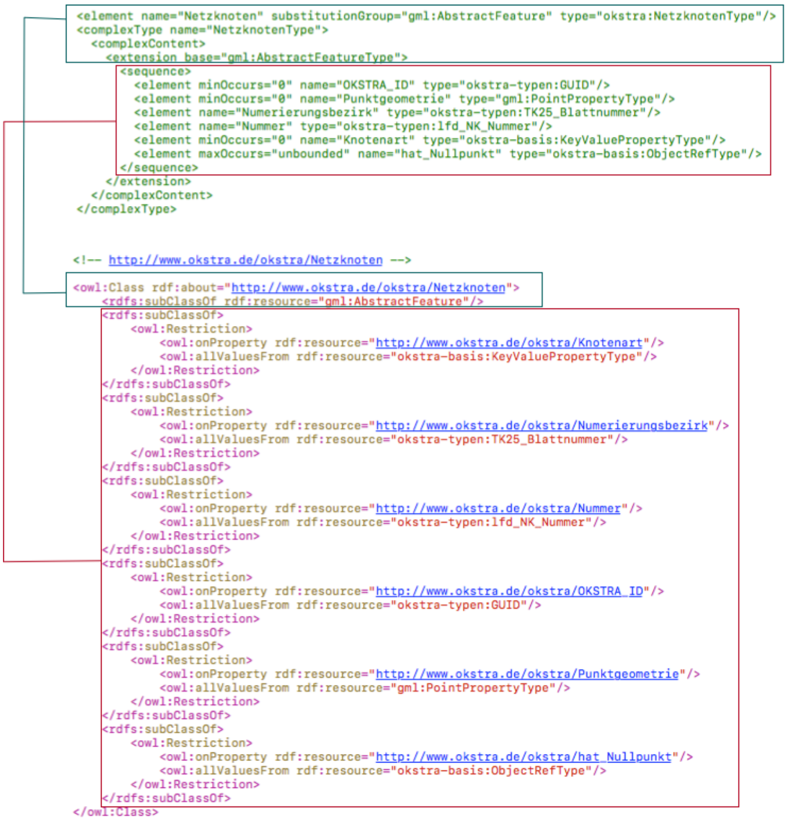


Figure 4.7: The XSLT

Due to the generated OKSTRA instance data model for the target area only contains the data from OKSTRA road network module. Thus, in this case the OKSTRA ontology development process only focuses on this module. After some manual adaption, the ontology of OKSTRA road network module is constructed. The partial ontology in OWL of ’Netzknoten’ class is shown in fig. 4.8, compared with the original XML schema. The full ontology represents in OWL can be seen in the Appendix F.

4.2.2 Data Conversion: from data-XML to RDF

After constructed the data ontology of both OKSTRA and CityGML, the process of converting data-XML to RDF graphs could be started. In this section, conversion principles and transform-



```
<element name="Netzknoten" substitutionGroup="gml:AbstractFeature" type="okstra:NetzknotenType"/>
<complexType name="NetzknotenType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <element minOccurs="0" name="OKSTRA_ID" type="okstra-typen:GUID"/>
        <element minOccurs="0" name="Punktgeometrie" type="gml:PointPropertyType"/>
        <element name="Numerierungsbezirk" type="okstra-typen:TK25_Blattnummer"/>
        <element name="Nummer" type="okstra-typen:lfd_NK_Nummer"/>
        <element minOccurs="0" name="Knotenart" type="okstra-basis:KeyValuePropertyType"/>
        <element maxOccurs="unbounded" name="hat_Nullpunkt" type="okstra-basis:ObjectRefType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>


<!-- http://www.okstra.de/okstra/Netzknoten -->

<owl:Class rdf:about="http://www.okstra.de/okstra/Netzknoten">
    <rdfs:subClassOf rdf:resource="gml:AbstractFeature"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/Knotenart"/>
            <owl:allValuesFrom rdf:resource="okstra-basis:KeyValuePropertyType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/Numerierungsbezirk"/>
            <owl:allValuesFrom rdf:resource="okstra-typen:TK25_Blattnummer"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/Nummer"/>
            <owl:allValuesFrom rdf:resource="okstra-typen:lfd_NK_Nummer"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/OKSTRA_ID"/>
            <owl:allValuesFrom rdf:resource="okstra-typen:GUID"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/Punktgeometrie"/>
            <owl:allValuesFrom rdf:resource="gml:PointPropertyType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/hat_Nullpunkt"/>
            <owl:allValuesFrom rdf:resource="okstra-basis:ObjectRefType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Figure 4.8: The Original XSD and OWL of OKSTRA Class 'Netzknoten'

atio and corresponding result will be introduced.

Conversion Principles

As discussed in the methodology chapter, the conversion principle of both CityGML and OKSTRA XML instance data into RDF graphs should follow this general idea that every class in XML map to a resource and is followed with its properties. For every tag in the XML document, it has been verified whether it is a name of a class or a property in the developed ontologies. Therefore, according to the developed ontologies, the class should be the URIRefs or Blank Nodes in the RDF graphs. To distinguish a node has URI or not depends on if it has attributes of gml:id. Moreover, in this experiment, the creation of URIs are fake URLs. The naming path of each OKSTRA data URIRefs begins with "mycitygml.tue.nl/", pluses the type of the elements and its gml:id. The naming path for the OKSTRA elements begins with "myokstra.tue.nl/" and continues with the description and its gml:id. Meanwhile, in the XML file there is 'xlink' be used to refer relevant classes, thus, during the transformation process the 'xlinks' are converting to corresponded URIRef.

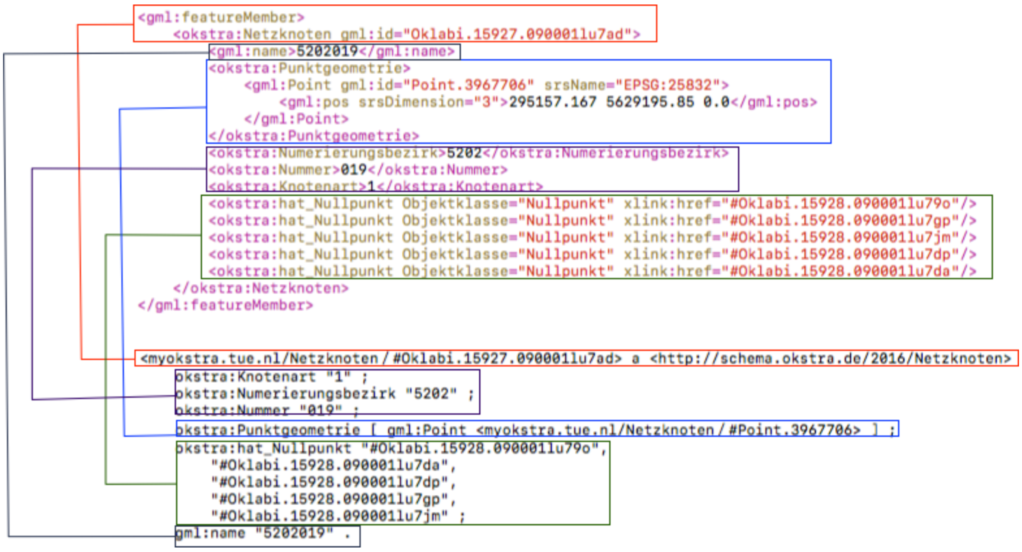
One thing should be mentioned as well is that the data-XML to RDF transformation should use the RDF property name rather than the XML name for the property [61], which means to use more RDF statement to describe the XML elements. However, in this case, the original XML data statement are used with the original namespace and tag name without converting to RDF statements, since both of CityGML and OKSTRA are well-structured XML standard and all their XML elements are well-defined. Therefore, the reuse of XML statement as subjects, predicates and objects in the RDF is sufficient and correlated rather than using RDF statement to describe the elements.

Transformation Approach and Result

For the sake of translating data-XML into RDF graphs, two procedures is required that first is extracting the XML data out and the second is creating the triples in RDF graphs. However, with currently available tools and languages, translating between XML and RDF is not a simple task, since the XML provides a popular format for data exchange accompany with a diversity of the XML data sets. On the other hand, the XML file is restricted by its data schema (usually DTD or XSD), which should be considered as well in the data transformation. All these conditions lead to the complexity of XML to RDF transformation process that there is no general converting tool and each specific type of XML data set should have a corresponding and generic converting tool to be developed.

Currently, there are several approaches and tools claim can convert data-XML into RDF graphs. In this research, the chosen approach to convert CityGML and OKSTRA data into RDF is implementing several existed Python packages, including lxml and beautiful soup for parsing the XML file and extracting XML data out and RDFlib for creating new triples and inputting the XML data into triples. The programs of transformation in Python code is shown in the Appendix G and H.

Furthermore, the serialization of RDF graphs has several format, including RDF/XML, Turtle, N3-triples and RDF/JSON. In this case, in order to have a clear view of readable triples in the RDF graphs and further convenient data linking process, the turtle format is selected.An example of the transformation is shown in fig. 4.9.

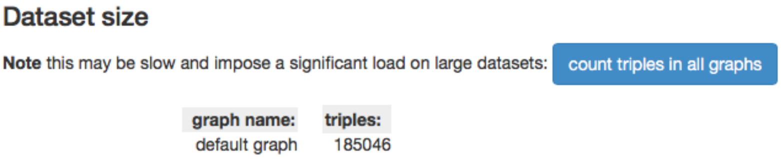


```
<gml:featureMember>
    <okstra:Netzknoten gml:id="Oklabi.15927.090001lu7ad">
        <gml:name>5202019</gml:name>
        <okstra:Punktgeometrie>
            <gml:Point gml:id="Point.3967706" srsName="EPSG:25832">
                <gml:pos srsDimension="3">295157.167 5629195.85 0.0</gml:pos>
            </gml:Point>
        </okstra:Punktgeometrie>
        <okstra:Numerierungsbezirk>5202</okstra:Numerierungsbezirk>
        <okstra:Nummer>019</okstra:Nummer>
        <okstra:Knotenart>1</okstra:Knotenart>
        <okstra:hat_Nullpunkt Objektklasse="Nullpunkt" xlink:href="#Oklabi.15928.090001lu79o"/>
        <okstra:hat_Nullpunkt Objektklasse="Nullpunkt" xlink:href="#Oklabi.15928.090001lu7gp"/>
        <okstra:hat_Nullpunkt Objektklasse="Nullpunkt" xlink:href="#Oklabi.15928.090001lu7jm"/>
        <okstra:hat_Nullpunkt Objektklasse="Nullpunkt" xlink:href="#Oklabi.15928.090001lu7dp"/>
        <okstra:hat_Nullpunkt Objektklasse="Nullpunkt" xlink:href="#Oklabi.15928.090001lu7da"/>
    </okstra:Netzknoten>
</gml:featureMember>
```

```
<myokstra.tue.nl/Netzknoten/#Oklabi.15927.090001lu7ad> a <http://schema.okstra.de/2016/Netzknoten>
    okstra:Knotenart "1" ;
    okstra:Numerierungsbezirk "5202" ;
    okstra:Nummer "019" ;
    okstra:Punktgeometrie [ gml:Point <myokstra.tue.nl/Netzknoten/#Point.3967706> ] ;
    okstra:hat_Nullpunkt "#Oklabi.15928.090001lu79o",
        "#Oklabi.15928.090001lu7da",
        "#Oklabi.15928.090001lu7dp",
        "#Oklabi.15928.090001lu7gp",
        "#Oklabi.15928.090001lu7jm" ;
    gml:name "5202019" .
```

Figure 4.9: An Example of Transformation for OKSTRA XML to RDF

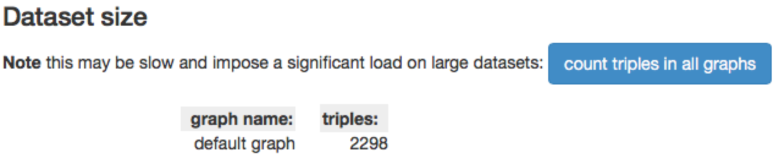
So far, the data transformation process is done with a result of two turtle-format RDF graphs. For CityGML RDF graphs contain over 180,000 triples (shown as fig. 4.10). Meanwhile the RDF graphs of OKSTRA contain over 2000 triples (shown as fig. 4.11). Two RDF graphs will be used in further data linking process.


Dataset size

Note this may be slow and impose a significant load on large datasets:

graph name:	triples:
default graph	185046

Figure 4.10: The count of triples in CityGML RDF graphs


Dataset size

Note this may be slow and impose a significant load on large datasets:

graph name:	triples:
default graph	2298

Figure 4.11: The count of triples in OKSTRA RDF graphs

4.3 Summary

In this chapter, the data transformation process is described, the detailed BPMN graph to illustrate the data flow is shown as fig. 4.12. Started with an introduction of the experimental implementation and the raw data process is addressed and followed with the description of the data transformation process which is intended to illustrate the detailed data transformation principle and approach based on the developed methodology that aim to achieve the translation of CityGML and OKSTRA XML data to RDF representation. As the second chapter introduced, both CityGML and OKSTRA data are represented as XML, which providing a tree-based data model. In order to translate them to RDF graph-based models, the transformation process should identify and overly structured data. These syntactic artifacts must be translated into a proper semantic model where objects and properties are typed and semantically related to common vocabularies. Thus, initially, ontologies of both OKSTRA and CityGML in the certain data domain is created. Based on the ontologies, the translation of data-XML of both data platform to RDF graphs is conducted. The result of data transformation will be the preparation for the next step: Data Linking.

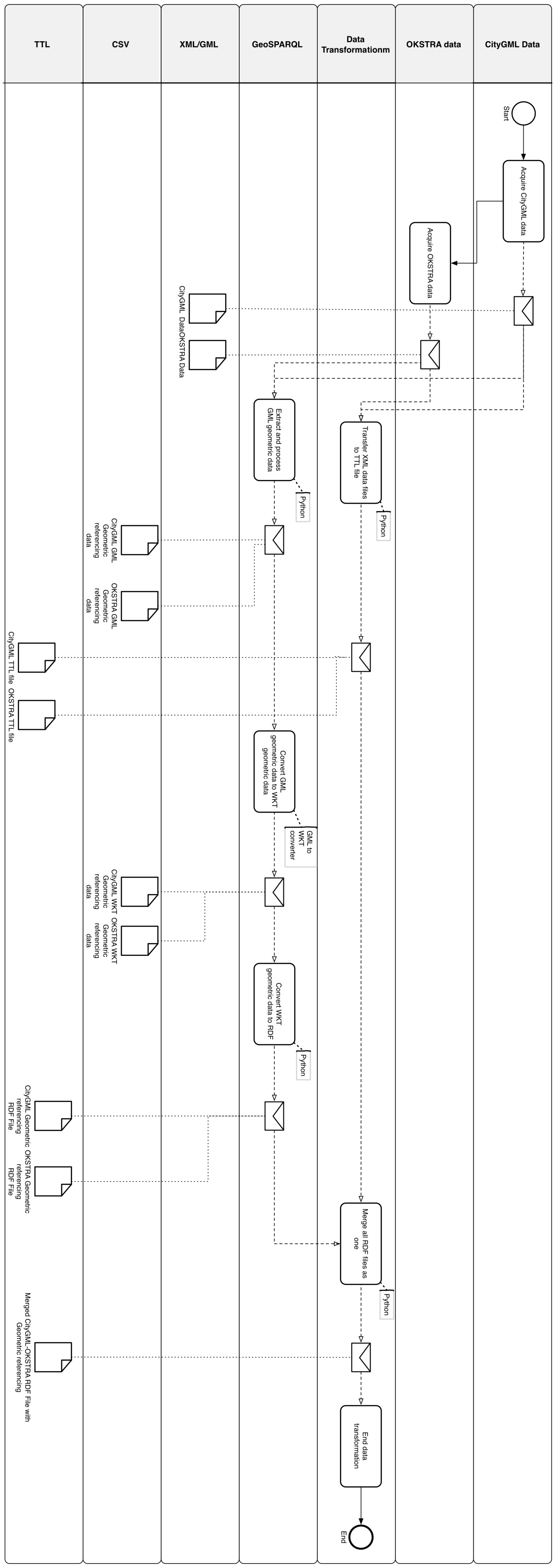

TTL
CSV
XML/GML
GeoSPARQL
Data Transformationm
OKSTRA data
CityGML Data
Start
Acquire CityGML data
Acquire OKSTRA data
CityGML Data
OKSTRA Data
Extract and process GML geometric data
Python
Transfer XML data files to TTL file
Python
CityGML GML Geometric referencing data
OKSTRA GML Geometric referencing data
CityGML TTL file
OKSTRA TTL file
Convert GML geometric data to WKT geometric data
GML to WKT converter
CityGML WKT Geometric referencing data
OKSTRA WKT Geometric referencing data
Convert WKT geometric data to RDF
Python
CityGML Geometric referencing RDF File
OKSTRA Geometric referencing RDF File
Merge all RDF files as one
Python
Merged CityGML-OKSTRA RDF File with Geometric referencing
End data transformation
End

Figure 4.12: The BPMN graph of transformation data flow

Chapter 5

Data Linking

In the previous chapter, the data transformation process is introduced. As a result, both of the CityGML data and OKSTRA data models are transformed as RDF graphs that contain with all the statement triples in the Turtle format. However, at this stage, the heterogeneous data from both platforms in RDF graphs is still isolated, which requires a further data linking process to associate the two RDF graphs.

In this chapter, the data linking process of the experimental implementation will be introduced. This data linking process aims to create linkages between specific object classes in both RDF graphs based on linked data method and GeoSPARQL concept in order to realize the integration of CityGML and OKSTRA. Furthermore, the following with an introduction of the application of querying the integrated RDF graphs based on specific topics, which can be also known as a validation of the linking process.

5.1 Link the RDF Graphs of OKSTRA and CityGML

5.1.1 Ontology Mapping and Linking Definition

As discussed in the methodology chapter, the data linking process should be started with ontology linking in order to create a unified integrated ontology (shown as fig. 5.1). The first step is the ontology mapping process that inspecting the classes in two different ontologies that have might have direct relationships and then create corresponded links. Otherwise, if the ontology mapping process cannot find direct relationships between two ontologies, the ontology linking

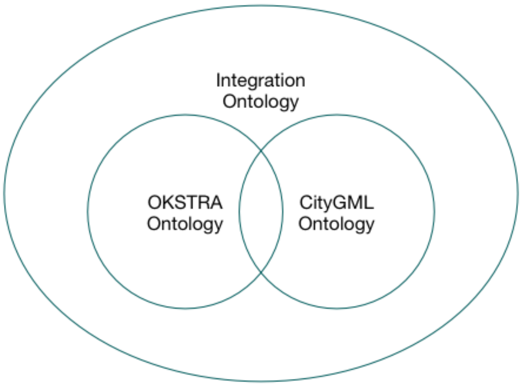


Figure 5.1: The ideal composition of integrated ontology

should be achieved by involving a intermediate ontology to link both ontologies. In terms of this experimental implementation, in the adapted OKSTRA ontology, there are classes and properties describing the road elements. Hence, in the corresponding instance RDF graph, there are instance data of these classes and properties, including semantics and geometry data. Meanwhile, the CityGML ontology consists of the classes of basic elements and properties about of the buildings. Therefore, from the perspective of ontology mapping, there are no equivalents or defined direct relationship between objects in two ontologies. Thus, in this research, the ontology linking process is designed to create a link between the classes in OKSTRA and CityGML ontologies by applying additional ontology as the intermediary.

According to the real-world and topological circumstance, the buildings have their closest roads sections (shown as fig. 5.2). Therefore, this spatial relationship could be used as the self-defined property for these buildings, known as "has the closest road section". From ontology perspective, the OKSTRA ontology class of 'okstra:Abschnitt' expressing the road section can be linked within CityGML ontology class 'bldg:Building' by the relationship of 'has closest road section' as a literal property of 'bldg:building' class. This connection links two separated information part as one resource-interrelated model. The corresponding data linking process of two RDF graphs is realized in the merged RDF set by adding triples that state the relationship of "building" - "has closest road section" - "road section" with the same URI of both building node in CityGML and road section node in OKSTRA.

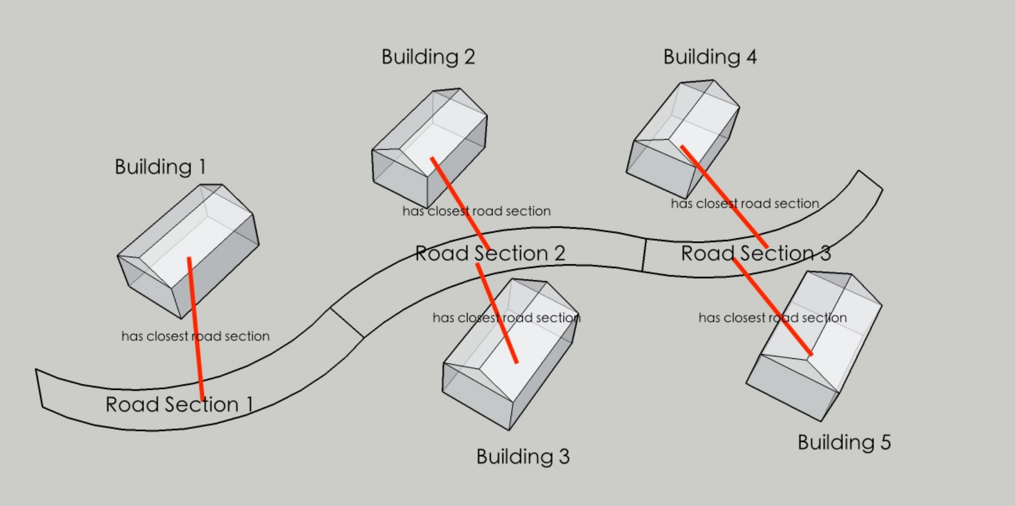


Figure 5.2: The schematic diagram of spatial relationship between buildings and road sections

5.1.2 Data Linking

In order to realize data linking by automatically determining the spatial relationship between buildings and road sections, here the GeoSPARQL technology is selected to be employed. The GeoSPARQL standard is developed by OGC, which is designed to support representing and querying geospatial data on the Semantic Web[51]. GeoSPARQL can be regarded as an extension of SPARQL query language. It is designed for processing geospatial data on Semantic Web and performs efficiently geospatial reasoning. GeoSPARQL defines a vocabulary dictionary for representing geospatial data in RDF and has its own ontology. Meanwhile, GeoSPARQL has a series of query transformation rules that expand a feature-only query into a geometry-based query from both topological query and non-topological spatial query [43] about spatial and geometric relations between different features, once they have the same specification in both qualitative systems and quantitative systems.[6]. Thus, considering these features, GeoSPARQL is one of appropriate approaches for this case to determine the spatial relationship between buildings and road sections.

Aforementioned, in order to employ GeoSPARQL to determine the spatial relationship between buildings and road sections, firstly both buildings and road sections node in RDF should have the same specification. This means to involve the geometry classes in GeoSPARQL ontology as the middleware to link the ontologies of both CityGML and OKSTRA. For building and road section class, a property of 'hasGeometry' is created with geometry class in GeoSPARQL ontology (shown as fig. 5.3). So far, three ontologies have been linked. Corresponding with the linked ontology, each building node and road section node in RDF graphs requires property of 'hasGeometry' with instance data of referencing geometry class. In order to have same qualitative systems and quantitative systems, the referencing geometries are required to be described by using specific GeoSPARQL RDF vocabulary with same format and same coordinate referencing system. Thus, in this case, a data processing is required to create GeoSPARQL RDF geometric references for all the buildings and road sections.

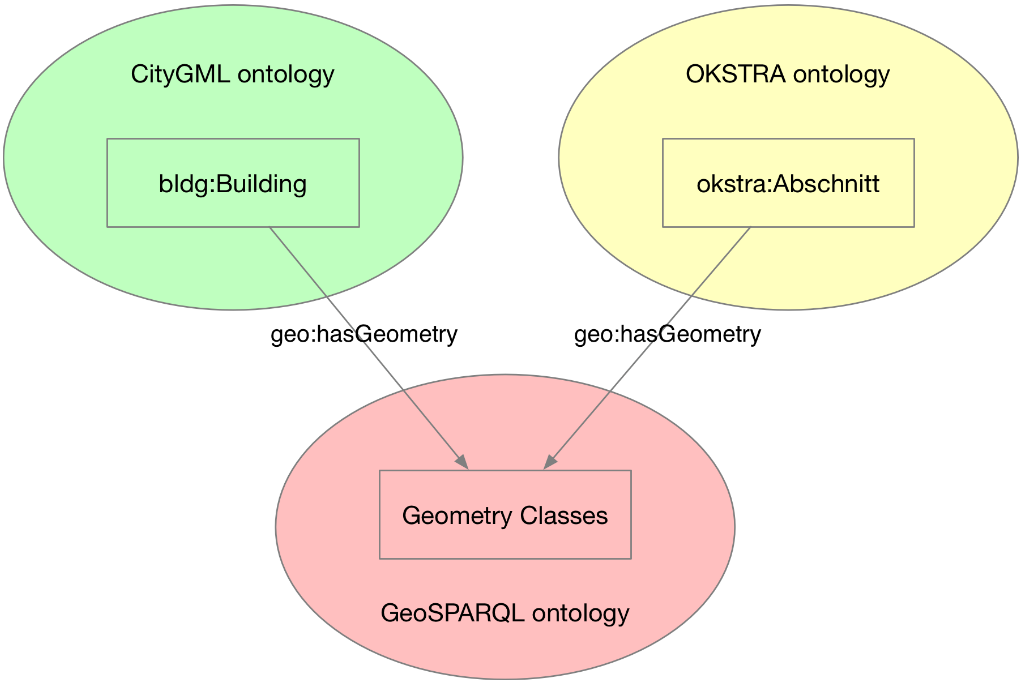


Figure 5.3: The illustration of conceptual ontology linking

In terms of the LoD2 buildings in the CityGML model, they have at least one ground surface, whose geometric type is known as 3D polygon. Therefore, geometric coordinate list (gml:poslist) of each ground surface polygon is used here to extract geometric data for constructing GeoSPARQL geometric references for each building (shown shown as fig. 5.4). Moreover, since the geometry representation of each road section is 2D linestring, in order to ensure the distance calculation during GeoSPARQL query process, each geometric reference of building is simplified from the 3D ground surface polygon to a 2D referencing point by calculating the mean value of X and Y coordinates of polygons. This data process is achieved by a Python code to parse and calculate the CityGML file, shown in Appendix I. All these coordinates of referencing points are then transformed to WKT format under WGS84 system along with corresponded building id via open online WKT translator (https://mygeodata.cloud/) and stored in a CSV file. Moreover, aforementioned OKSTRA the geometries of OKSTRA road section are representing as 2D linestrings, so they can be directly extracted from OKSTRA XML file and converted to WKT format in CSV file via the

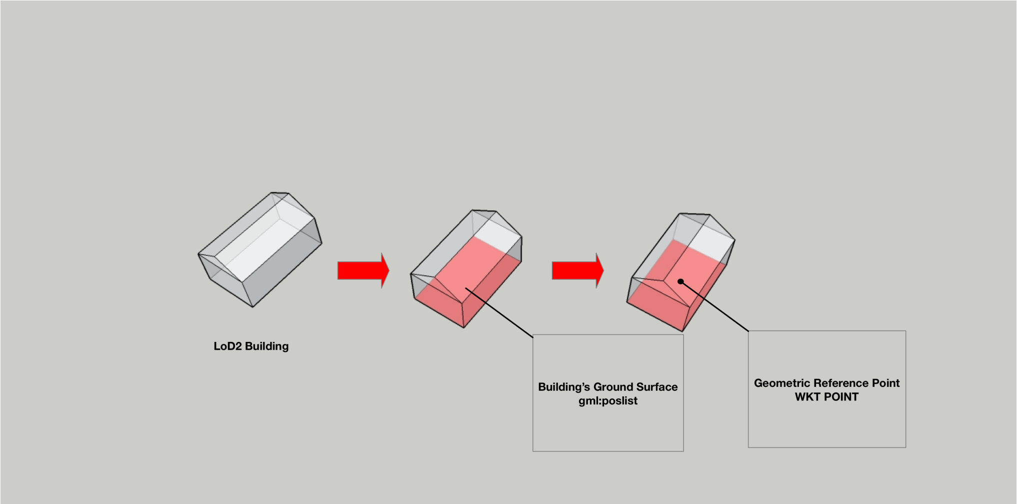


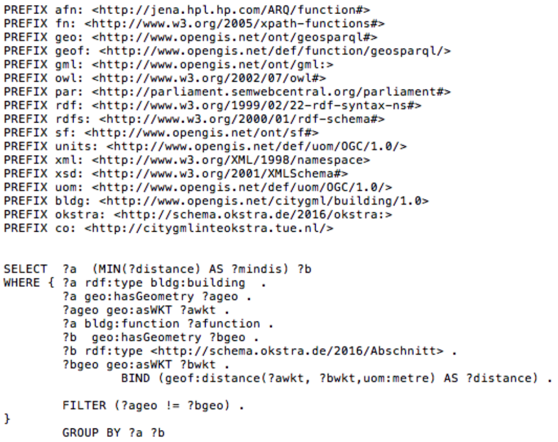
Figure 5.4: The geometry reference of building

open online WKT translator.

By using Python RDFlib package to translate these two CSV files (the script is shown in Appendix J), one RDF graph is created to contain all the geometries in GeoSPARQL RDF vocabulary. It contains buildings with their point referencing geometries in WKT and road sections with their WKT linestring geometries. Afterwards the translation of geometry to RDF has been done, the referencing geometry RDF is merged with CityGML and OKSTRA RDF graphs as one RDF set that contains all triples of CityGML building and OKSTRA road section together with their geometric references. Since, the URIs of buildings and road section in three RDF graphs are consistentthe computer will automatically append the referencing geometry triples to each building and road section. The merging process is realized by applying Java Apache Jena package (script is shown in Appendix K).

As so far, the preparation of applying GeoSPARQL has been done. However, the linking is not accomplished yet. To determine the spatial relationship, the GeoSPARQL distance function is utilized here, which is a filter function to return the shortest distance in units between any two geometric objects as calculated in the spatial reference system of the first object. Here, using GeoSPARQL distance function and SPARQL basic bind function to query the merged RDF set of buildings and road sections with their geometry, the distances between one building to every road section can be calculated (shown as fig. 5.5). This query result is exported as CSV file and a small Python program (shown in Appendix L) is scripted to find the shortest distance value to indicate the closest road section for every building. Till now, the relationship of building and its closest road section is determined, which is translated as ”building URI” - ”hasclosestroadsection” - ”road section URI” triples and added to the RDF set (the script is shown in Appendix M).

After the spatial relationship between each building and road section is defined, all the required RDF graphs have been generated and should be gathering and storing in one RDF set in order to realize the data linking. The Python code of this process is shown in Appendix N. Since the relationship RDF contains the consistent URI of both buildings and road sections with CityGML and OKSTRA RDF graph, the merged RDF set will automatically link the CityGML and OKSTRA RDF graph based on URI. In this research, the GraphDB is used for the storage and the retrieval of all the RDF triples as one merged RDF graph set. The merged model consists all the RDF triples from both RDF graphs of CityGML model and OKSTRA model and their relationship. Once the merged RDF set consists of 198599 RDF triples, shown as fig. 5.6.

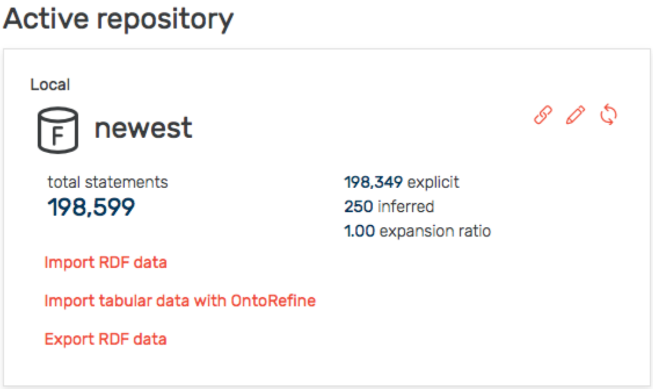


```
PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#>
PREFIX fn: <http://www.w3.org/2005/xpath-functions#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX gml: <http://www.opengis.net/ont/gml:>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX par: <http://parliament.semwebcentral.org/parliament#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sf: <http://www.opengis.net/ont/sf#>
PREFIX units: <http://www.opengis.net/def/uom/OGC/1.0/>
PREFIX xml: <http://www.w3.org/XML/1998/namespace>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX uom: <http://www.opengis.net/def/uom/OGC/1.0/>
PREFIX bldg: <http://www.opengis.net/citygml/building/1.0>
PREFIX okstra: <http://schema.okstra.de/2016/okstra:>
PREFIX co: <http://citygmlinteokstra.tue.nl/>


SELECT  ?a  (MIN(?distance) AS ?mindis) ?b
WHERE { ?a rdf:type bldg:building  .
        ?a geo:hasGeometry ?ageo .
        ?ageo geo:asWKT ?awkt .
        ?a bldg:function ?afunction .
        ?b  geo:hasGeometry ?bgeo .
        ?b rdf:type <http://schema.okstra.de/2016/Abschnitt> .
        ?bgeo geo:asWKT ?bwkt .
                BIND (geof:distance(?awkt, ?bwkt,uom:metre) AS ?distance) .

        FILTER (?ageo != ?bgeo) .
}
        GROUP BY ?a ?b
```

Figure 5.5: The GeoSPARQL query for calculate distance between one building and all road sections



Active repository

Local

newest

total statements
198,599

198,349 explicit
250 inferred
1.00 expansion ratio

Figure 5.6: The information of final linked RDF set

5.2 Query the Integration Result

As long as the data linking phase is accomplished, the CityGML data and OKSTRA data have been theoretically integrated. To verify whether if the integration is successful and able to conduct spatial query for useful semantics, several thematic queries based on selected topics are developed by using SPARQL.

5.2.1 Query Topic Selection

The selection of the topics has two principles: 1) within the available range based on the instance RDF set and 2) has practical usage. In the CityGML model, available semantics are basic information about the building including building name, building function, roof type and measured height. While in the OKSTRA model, the most interested semantics are the road name, road section name, number of lanes of road section, and road section is separated or not. Compared with the data schema of both CityGML building module and OKSTRA road network module, the instance data sources contain less available semantic. However, it is still able to query the integration product for interested topics that can not achieved in sole data models. Therefore, considering the scenarios, several topics are determined:

1.Find the kindergartens and secondary schools that within 100m distance to road and the information of their closest road sections

Road safety is an crucial issue to neighborhoods and communities. A short distance between buildings and highways can bring potential safety hazard to vulnerable groups such as children and students. Thus, especially for buildings as kindergarten or secondary school that contain a number of children, an inspection of road ambience is required. Here, the integrated RDF set can be applied for searching the secondary schools and kindergartens that close to roads, and the information of their closest road sections. The result could help the local authorities to inspect the safe condition about the schools in the neighborhood and improve the safety for children. If some of the kindergartens or secondary schools are found within 50 metres distance to road, they should be marked and reminded to pay more attention on prevent traffic accident with actions like employing more safety facilities and enhancing the education of travel safety to students and parents.

2.Find gas stations and car wash service buildings in the target area and their closest road section information

Both of gas stations and car wash shops are important facilities for vehicles on the road. These functional buildings provide specific car services. Therefore, the information such as the traffic volume and number of traffic lanes about the closest road section of these buildings could be supportive to the operators' decision making process regarding to the management of business and facilities.

3.Find the buildings over 20m height that within 50m distance to road

In the architecture and urban design domain, the skyline of the buildings close to road is always considered as an essential design index for architects, urban designers and engineers. The design height of the project which close to the road usually is corresponded with other buildings in order to create erratic skyline. Moreover, there is a relationship between residential floor level and concentration of traffic-related airborne pollutants. Therefore, if the integrated RDF set can be used to query for buildings close to street and with specific height, architects can use the building height information to support design meanwhile for urban traffic pollutant research can easily find target building to research.

5.2.2 Query and Result

So far, three selected topics have been determined, the thematic queries are able to be conducted by applying SPARQL technology to query the integrated RDF set. Notice that the query result output of SPARQL are tables with the varieties and their values. Moreover, in order to illustrate the result clearly, the result about the specific buildings are also visualized in 3D representation in the 3D city model. However, currently there are no existed tools that directly visualize the selected 3D objects inside RDF file. Therefore, the visualization of result building is realized still based on CityGML visualization through FME workbench. FME workbench provides functions of input CityGML and GML models, and various translating functions to translate the models into visual results with different feature. In this case, the 'testfilter' translator is implemented to use buildings' property 'gml:id' as the index to select buildings in the query result and 'geometry appearance setter' is used to change their color of appearance in output 3D-PDF file (shown as fig. 5.7). Therefore in the 3D-PDF file, the buildings as the query result are highlighted in the entire city model, so the enquirer can easily navigate the target building and get an intuitive impression from a city scale. In the following parts, the query process of three thematic queries are briefly discussed and the query results are illustrated.

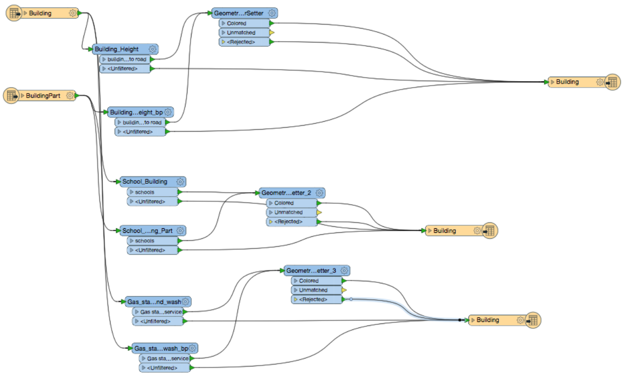
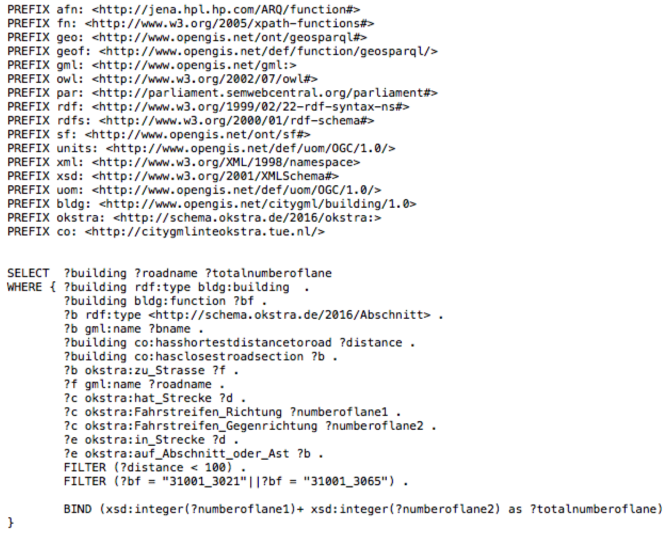


Figure 5.7: The work space of visualizing query result buildings in FME

1.Find the kindergartens and secondary schools that within 100m distance to road and the information of their closest road sections

As shown in the fig. 5.8, the query is conducted in order to find the buildings with function of kindergarten (function code:"31001-3065") and secondary school(function code:"31001-3021") that within 100 metres to road and the information of the name of road that the buildings are close to and number of lanes the road section has. There are two restrictions in the query, one is the building function only can be kindergarten or secondary school and the other is must within 50 metres distance to road. These two restrictions are used as filter to constraint the result in SPARQL query. The fig. 5.9 shows result of the target kindergartens and secondary school inside the CityGML model. Meanwhile, the table 5.1 is shown as to illustrate the result of road information.



```
PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#>
PREFIX fn: <http://www.w3.org/2005/xpath-functions#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX gml: <http://www.opengis.net/gml:>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX par: <http://parliament.semwebcentral.org/parliament#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sf: <http://www.opengis.net/ont/sf#>
PREFIX units: <http://www.opengis.net/def/uom/OGC/1.0/>
PREFIX xml: <http://www.w3.org/XML/1998/namespace>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX uom: <http://www.opengis.net/def/uom/OGC/1.0/>
PREFIX bldg: <http://www.opengis.net/citygml/building/1.0>
PREFIX okstra: <http://schema.okstra.de/2016/okstra:>
PREFIX co: <http://citygmlinteokstra.tue.nl/>


SELECT  ?building ?roadname ?totalnumberoflane
WHERE { ?building rdf:type bldg:building  .
        ?building bldg:function ?bf .
        ?b rdf:type <http://schema.okstra.de/2016/Abschnitt> .
        ?b gml:name ?bname .
        ?building co:hasshortestdistancetoroad ?distance .
        ?building co:hasclosestroadsection ?b .
        ?b okstra:zu_Strasse ?f .
        ?f gml:name ?roadname .
        ?c okstra:hat_Strecke ?d .
        ?c okstra:Fahrstreifen_Richtung ?numberoflane1 .
        ?c okstra:Fahrstreifen_Gegenrichtung ?numberoflane2 .
        ?e okstra:in_Strecke ?d .
        ?e okstra:auf_Abschnitt_oder_Ast ?b .
        FILTER (?distance < 100) .
        FILTER (?bf = "31001_3021"||?bf = "31001_3065") .

        BIND (xsd:integer(?numberoflane1)+ xsd:integer(?numberoflane2) as ?totalnumberoflane)
}
```

Figure 5.8: The SPARQL query for finding the kindergartens and secondary schools that within 100m distance to road and the information of their closest road sections

Table 5.1: The SPARQL result of topic 1

building	roadname	total number of lane
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MiE8	B1	2
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MiuR	B1	2
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MjMS	B1	2
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000Mjbu	B1	2
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39ALGN00001F	B1	4

2.Find gas stations and car wash service buildings in the target area and their closest road section information

The SPARQL query script for this topic is shown in fig. 5.10, in which the restriction is to find the selected building only have the specific function of gas stations or car wash service.fig. 5.10 According to CityGML building function code-book, the codes of two building functions are ”31001-2131” and ”31001-2130”. Shown as fig. 5.11, the highlight buildings in blue color are the result of the target buildings inside the CityGML model. And the table 5.2 is shown as to illustrate the result of road information.

3.Find the buildings over 20m height that within 50m distance to road

In this query (shown as fig. 5.12), the target buildings are selected based their shortest distance to the closest road section and measured height. The restrictions of the distance is less than 50 metres and the height is over 20 metres. The query aims to find the buildings satisfy with this restriction and their semantic information. The result is shown in the fig. 5.13 and table 5.3.

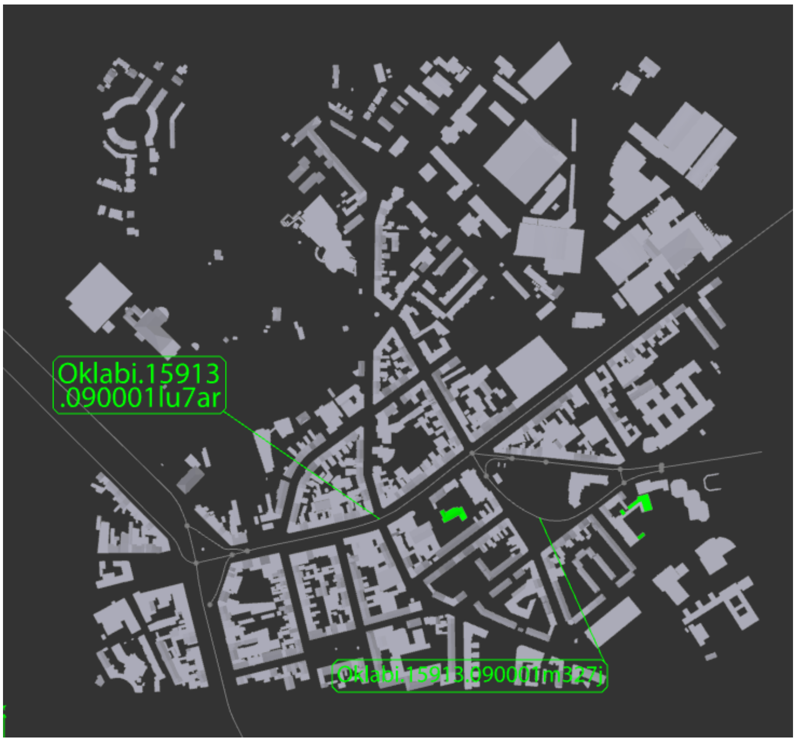
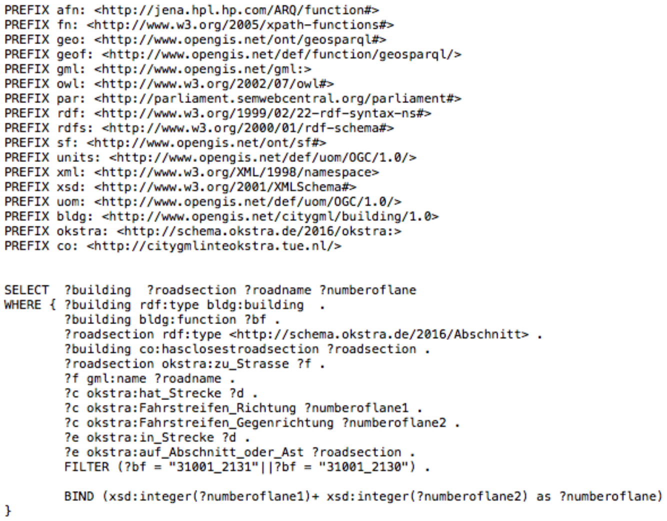


Figure 5.9: The visualization for the target kindergartens and secondary schools (in green)

Table 5.2: The SPARQL result of topic 2

building	roadname	number of lane
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000Mk6I	B1	2
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000Mice	B1	2
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MjcM	L136	4



```
PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#>
PREFIX fn: <http://www.w3.org/2005/xpath-functions#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX gml: <http://www.opengis.net/gml:>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX par: <http://parliament.semwebcentral.org/parliament#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sf: <http://www.opengis.net/ont/sf#>
PREFIX units: <http://www.opengis.net/def/uom/OGC/1.0/>
PREFIX xml: <http://www.w3.org/XML/1998/namespace>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX uom: <http://www.opengis.net/def/uom/OGC/1.0/>
PREFIX bldg: <http://www.opengis.net/citygml/building/1.0>
PREFIX okstra: <http://schema.okstra.de/2016/okstra:>
PREFIX co: <http://citygmlinteokstra.tue.nl/>


SELECT  ?building  ?roadsection ?roadname ?numberoflane
WHERE { ?building rdf:type bldg:building  .
        ?building bldg:function ?bf .
        ?roadsection rdf:type <http://schema.okstra.de/2016/Abschnitt> .
        ?building co:hasclosestroadsection ?roadsection .
        ?roadsection okstra:zu_Strasse ?f .
        ?f gml:name ?roadname .
        ?c okstra:hat_Strecke ?d .
        ?c okstra:Fahrstreifen_Richtung ?numberoflane1 .
        ?c okstra:Fahrstreifen_Gegenrichtung ?numberoflane2 .
        ?e okstra:in_Strecke ?d .
        ?e okstra:auf_Abschnitt_oder_Ast ?roadsection .
        FILTER (?bf = "31001_2131"||?bf = "31001_2130") .

        BIND (xsd:integer(?numberoflane1)+ xsd:integer(?numberoflane2) as ?numberoflane)
}
```

Figure 5.10: The SPARQL query for finding the gas stations and car wash service buildings in the target area and their closest road section information

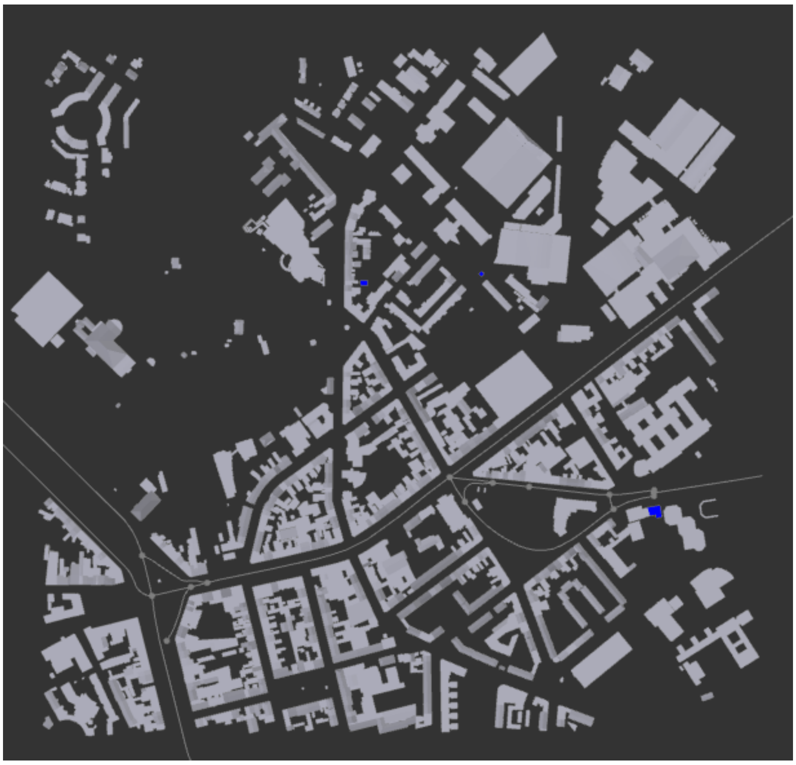
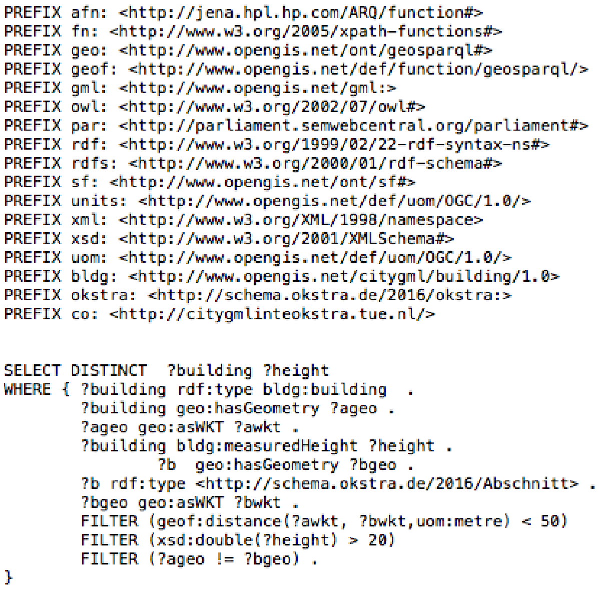


Figure 5.11: The visualization for the target kindergartens and secondary schools (in blue)



```
PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#>
PREFIX fn: <http://www.w3.org/2005/xpath-functions#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX gml: <http://www.opengis.net/gml:>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX par: <http://parliament.semwebcentral.org/parliament#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sf: <http://www.opengis.net/ont/sf#>
PREFIX units: <http://www.opengis.net/def/uom/OGC/1.0/>
PREFIX xml: <http://www.w3.org/XML/1998/namespace>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX uom: <http://www.opengis.net/def/uom/OGC/1.0/>
PREFIX bldg: <http://www.opengis.net/citygml/building/1.0>
PREFIX okstra: <http://schema.okstra.de/2016/okstra:>
PREFIX co: <http://citygmlinteokstra.tue.nl/>


SELECT DISTINCT  ?building ?height
WHERE { ?building rdf:type bldg:building  .
        ?building geo:hasGeometry ?ageo .
        ?ageo geo:asWKT ?awkt .
        ?building bldg:measuredHeight ?height .
                ?b  geo:hasGeometry ?bgeo .
        ?b rdf:type <http://schema.okstra.de/2016/Abschnitt> .
        ?bgeo geo:asWKT ?bwkt .
        FILTER (geof:distance(?awkt, ?bwkt,uom:metre) < 50)
        FILTER (xsd:double(?height) > 20)
        FILTER (?ageo != ?bgeo) .
}
```

Figure 5.12: The SPARQL query for finding the buildings over 20m height that within 50m distance to road

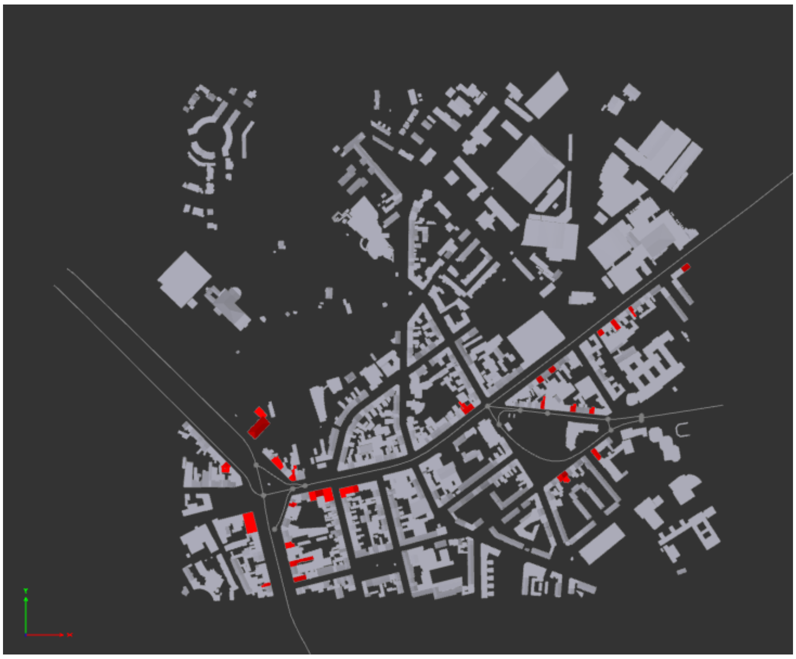


Figure 5.13: The visualization for the target kindergartens and secondary schools (in red)

Table 5.3: The SPARQL result of topic 2

building	height
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL10005W9z	23.308
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL10005WMp	20.082
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000Mimi	20.205
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000Mivp	24.021
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000Mj3R	24.38
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL10005VtZ	22.238
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MiFR	21.082
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MiQl	25.723
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MiDn	20.376
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000Mij5	20.83
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MjB5	20.311
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MjDt	33.997
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MjKb	21.58
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MjL6	20.234
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MjLL	20.665
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MjZn	20.981
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MiFW	20.135
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MiP8	20.063
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MiTt	20.571
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MiYW	21.066
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MitM	20.347
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000Mj6I	22.293
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MjIg	22.135
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MieQ	23.778
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000Mi91	20.185
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MiJA	21.914
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000Mihq	20.111
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000Mixi	20.348
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000Mj8n	21.149
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000MjqG	20.402
file:///Users/pc/Desktop/geolinkrdf/mycitygml.tue.nl/city/building/DENW39AL1000Mjya	20.521

Chapter 6

Conclusions, Recommendation and Future Work

In this chapter, the answers all research questions based on the accomplishment of the research goals will constitute the conclusions of this research. Following are the discussion of the limitation and process of the research, and recommendations based on research result. At last, the future work of this study will be described.

6.1 Conclusions

Aforementioned, the research purpose is divided into two goals, each aims to provide insight in a specific part of the research of the CityGML and OKSTRA integration and corresponded with different research questions. In this concluding part, the sub-questions will be answered successively at first according to different research goals, and then the main question will be answered.

The first goal aims to answer the first part of how the semantic web technology can be utilized to integrate CityGML and OKSTRA models, as described in chapters 3 and 4. The answer to this question is based upon the findings in the previous studies and literature that related with the Semantic Web implementation in AECO/FM domain and used to generate an output as the own integration method of integrating CityGML and OKSTRA models.

1. How Semantic Web technology can enhance the semantic information sharing during the integration process?

From preliminary and literature study, it can be concluded that the Semantic Web technologies including RDF, OWL and Linked Data have obvious advantages to merge various information sources. Since the Semantic Web technology have capabilities to combine heterogeneous data sources by implying RDF data model across islands of information. Therefore, interoperability and efficiency during the information exchange process can be improved. Moreover, the implementation of Semantic Web technologies like linked-data could optimize the processing of the annotated data and perform as middleware to link heterogeneous data source without information loss during the conversion between different sources and is able to integrate data various sources regardless of their format or domain boundaries.

2. How to use semantic web technology to integrate CityGML and OKSTRA from theoretical aspect?

From theoretical perspective, the integration of CityGML and OKSTRA models could be realized via implementing Linked Data approach, which allows data in one data source to be linked to data

in another data source. However, in order to employ the Linked Data approach, it is required to create the ontologies of both CityGML and OKSTRA models and use RDF graphs to represent these models, and then integrate the ontologies to guide create RDF statements to describe the relationship between two graphs and merge all the RDF graphs as one linked RDF set. Therefore, according to these requirements, an integration methodology is designed and consists several parts:
1. Constructing Ontology of CityGML ($O_{citygml}$) and OKSTRA (O_{okstra}) under the application domain
2. Transform both CityGML and OKSTRA Data Models into RDF graphs based on $O_{citygml}$ and O_{okstra}
3. Create links between the constructed ontologies of $O_{citygml}$ and O_{okstra} based on Linked Data method to generate the integrated ontology $O_{citygml-okstra}$
4. Link the RDF graphs of CityGML and OKSTRA models
5. Query the integrated RDF set for practical application

Moreover, the second research goal aims to answer the question about how the integration can be realized based on a specific implementation by describing the detail integration process and in chapter 5 and 6.

3. What are detailed process to transform the CityGML and OKSTRA model into RDF graphs?

As we known, both CityGML and OKSTRA models are serialized as XML format. Therefore, the data transformation can be regarded as data-XML to RDF issue. From the literature study, several approaches have been applied to translate data-XML to RDF. In this research, the transformation method is to extract the data from XML files and reconstruct it as RDF triple under the instruction of the developed ontologies of both CityGML and OKSTRA within the research scope. The ontologies illustrate classes and properties in the XML file. In terms of the classes in ontologies, they are subjects or objects in the triples, refers to URI nodes or blank nodes. For properties, they are predicates in the triples. Moreover, the values in XML also need to be extracted to construct the entire RDF graph. Furthermore, the transformation tools are several Python packages that can realize parsing XML file and write RDF triples for the sake of translation.

4. What kind of linking between CityGML data and OKSTRA data can be created?

Theoretically from the Linked Data method, the resource links can be created when there are semantics have similarity, equivalents or logic relationships existed between the data resources. However, in experimental implementation for the research, the CityGML model in hand only contains geometric data and limited semantics of buildings in the target area. Similarly, the OKSTRA model only contains geometric data and limited semantics of the roads. Therefore, between CityGML and OKSTRA models, there are no direct relationships between classes in both ontologies. Considering this circumstance, a spatial relationship should be self-defined to link buildings and roads. In this case, since there is no semantics to be used, specific geographical reflection of both buildings and road sections are append to them and serialized as RDF by extracting and processing their geometric data for employing GeoSPARQL, which can be used to calculate the distance between each buildings and road section in order to establish the linkage between building and road section, known as: building 'hastheclosestroadsection'. This relationship describes the geographical relation between one building and its closest road section. The linkage is generated through creating corresponding RDF triples in RDFlib that represent the links and adding them to the merged RDF model.

5. How the linked CityGML-OKSTRA model could be utilized by querying integrated product?

In order to utilize the integrated product, known as the linked CityGML-OKSTRA model, several queries based on a specific topic can be conducted through the SPARQL query and the query results processing, which aims to query the semantic questions that could not be achieved when the models are isolated. To be mentioned, the result of query process can be considered as the validation of integration method as well.

During the experimental implementation of the research, although there is a limitation of the semantics richness in both CityGML and OKSTRA data source, the query topics are created to analyze the available useful building environment semantics, such as the average building height and variety of building functions of a specific road section. In fact, the query process can be realized in several existed Semantic Web tools. In this research, a python program is created to perform all the topic queries and data processing.

Main question: How to implement the integration of CityGML and OKSTRA semantics based on Semantic Web technologies in order to enhance the city semantic information interoperation and cooperation?

As a conclusion, the main research question can be answered. The implementation of the integration of CityGML and OKSTRA based on Semantic Web technologies can be achieved through a framework including several steps: 1) Collect the necessary semantics data resource from CityGML and OKSTRA platforms 2) Analyze the data schema of both platform and develop their ontologies 3)Transform the data into RDF graphs under the developed ontologies, 4) Map the developed ontologies to determine the links between RDF resources 5) Merge all the triples and link the determined related resources 6) develop SPARQL query or use other RDF processing methods to apply the integrated product.

Overall, from this research, it can be proved that Semantic Web technologies are suitable approaches to integrate CityGML and OKSTRA data source, which performs to integrate them by determining the rational relationships among different data sources. Therefore, the all the data in different domains is all kept without data loss. Moreover, the implementation of Semantic Web technologies can be used for further applications such as thematic semantics queries across two different data domains. In other words, it overcomes the data isolation.

6.2 Limitation and Discussion

In this research, it is clearly evident that the main limitation is the fact that both of the CityGML models and OKSTRA only contains limited semantics. In terms of the CityGML model with only contains LoD1 and LoD2 buildings and there is limited semantics for each building. Meanwhile, the OKSTRA model only contains road elements with their basic information, so the available semantics are limited as well. This leads to a consequence that for ontologies adapted based on the data range, during linking process there is no direct relationship can be found between two ontologies, which requires an additional GeoSPARQL ontology get involved for determining the spatial relationship between building and road section. Moreover, it extremely confined further application, only a few query topic could be conducted, which undermined the practicability of implementation.

6.3 Recommendations

As a reflection, several research recommendations can be made based on the entire research. Considering the different aspects, the recommendations can be classified into two kinds: scientific recommendation and practical recommendation.

6.3.1 Practical Recommendations

The practical recommendations consider the current development of the CityGML and OKSTRA data resource, and Semantic Web technology practical utilization. The following recommendations are made with regard to the practical development of use case utilizing 3D city models in real estate applications:

1. Increase the richness and quality of semantics in the CityGML model and OKSTRA model

As discussed in the limitation part, the current drawback of the CityGML and OKSTRA data source is the models are scarce of semantics in the model. Moreover, the limitation of the semantics in both models has restricted the linking creation and further practical application domain and possible use cases. Therefore, in order to solve this data lack issue, several recommendations could be made from different perspective: 1) improve the modification tools to enhance the ability of appending necessary semantics to 3D objects that might be generated from LiDAR or 3D point cloud; 2) enrich the semantics of building objects in CityGML model by integrate from the corresponded building's BIM model 3) enlarge the city model to contain more variety of city objects according to the CityGML thematic modules.

2. Develop and publish the official full-scale ontology of both CityGML and OKSTRA standard

In order to enhance the interoperability of both CityGML and OKSTRA for further linking with other possible data source, the ontology of both data standard should be established at the first stage. In this research, the used ontology of CityGML is adapted from the existed ontology that created by the University of Geneva, while the used ontology of OKSTRA is generated through the translation from OKSTRA data schema to OWL via the XSLT. However, from the perspective of standardization and normalization, the ontologies should be developed and published by the creator in order to guide the future integration.

6.3.2 Scientific Recommendations and Future Work

The scientific recommendations consider to discuss the shortcomings of this research and recommendations for further research. In this research, three obvious shortcoming have limited the research, and should be overcome in the future studies:

1. Visualization of CityGML and OKSTRA semantics in RDF format

In this research, the visualization of the query result is realized by return the result to the FME workbench in order to highlight buildings in the entire CityGML model by changing the appearance of a geometric item of the building in the output 3D-PDF file. However, this approach can not present of semantic annotation of the query result. Therefore, new means of visualization are demanded that can achieve both 3D visualization of city object and can show the annotation semantics of query result direct from the RDF file. This approach requires first parsing the both semantic and geometric data store in the RDF file. Afterwards, recompiling the geometric data to 3D object and translate the semantics to human-readable annotation information.

2. Integrate with more data sources

This research focuses on the integration of CityGML and OKSTRA, however, there are possibilities to integrate more heterogeneous city data source in order to rich the semantics for achieving more possible applications in order to realize more applications and use cases. Currently, various data sources about the city information are becoming as open data sources. Meanwhile, considering the trend of smart city, abundant ICT devices like sensors are employed in the cities throughout the world. In this research, the RDF data set can be translated from CSV file and data-XML file, however, most of the various data format can be translate to RDF as well. Therefore, there is a possibility to integrate these sorts of sensor data in different format with the CityGML model as a solution to build up a Semantic Web data base of smart city.

3. Research on feasibility of the combination of Linked Data approach and Big Data analysis for city semantic knowledge and spatial data analysis

From this research, it can be proved that Linked Data approach has the ability to integrate heterogeneous data source. Therefore, the Linked Data approach might be an effective approach to enlarge the data size and add more variables in the data set in a certain range. Considering this nature, the Linked Data approach could be supportive regarding the city big data analysis from the data variety perspective. In fact, there has been a clear connection and mutual interest from both Semantic Web and Big Data. However, current Semantic Web technologies has limitation to cooperate with Big Data concept. One the most acute issues is that Big Data analysis requires complex data processing and calculation. However, currently within the Semantic Web domain it cannot be realized. If the data process issue can be overcome, it will be more convenient if the possible data mining processes can be directly conducted within the RDF graphs.

Bibliography

[1] Resource description framework (rdf). `https://www.w3.org/RDF/`. 17

[2] What is sparql - semantic search query language. `https://ontotext.com/knowledgehub/fundamentals/what-is-sparql/`. 18

[3] Dean Allemang and James Hendler. *Semantic web for the working ontologist: effective modeling in RDFS and OWL*. Elsevier, 2011. 18

[4] Julian Amann and André Borrmann. Okstra internationalization. `http://www.okstra.de/forschung/okstra_internationalization_EN.html`. 13, 14

[5] Grigoris Antoniou and Frank Van Harmelen. *A semantic web primer*. MIT press, 2004. 17

[6] Robert Battle and Dave Kolas. Geosparql: enabling a geospatial semantic web. *Semantic Web Journal*, 3(4):355–370, 2011. 41

[7] Sean Bechhofer. Owl: Web ontology language. In *Encyclopedia of Database Systems*, pages 2008–2009. Springer, 2009. 16

[8] Jakob Beetz, Jos Van Leeuwen, and Bauke De Vries. Ifcowl: A case of transforming express schemas into ontologies. *Ai Edam*, 23(1):89–101, 2009. 20

[9] Tim Berners-Lee. Linked data. `https://www.w3.org/DesignIssues/LinkedData.html`, Jun 2009. 17, 18

[10] Tim Berners-Lee et al. Semantic web roadmap, 1998, 1998. 16

[11] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001. 3, 19

[12] Stefan Bischof, Athanasios Karapantelakis, Cosmin-Septimiu Nechifor, Amit P Sheth, Alessandra Mileo, and Payam Barnaghi. Semantic modelling of smart city data. 2014. 19

[13] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227, 2009. 16, 17

[14] Christian Bizer, Tom Heath, Kingsley Idehen, and Tim Berners-Lee. Linked data on the web (ldow2008). In *Proceedings of the 17th international conference on World Wide Web*, pages 1265–1266. ACM, 2008. 17

[15] André Borrmann, Thomas H Kolbe, Andreas Donaubauer, Horst Steuer, Javier Ramos Jubierre, and Matthias Flurl. Multi-scale geometric-semantic modeling of shield tunnels for gis and bim applications. *Computer-Aided Civil and Infrastructure Engineering*, 30(4):263–281, 2015. 20

[16] C Brenner, N Haala, and D Fritsch. Towards fully automated 3d city model generation. *Automatic Extraction of Man-Made Objects from Aerial and Space Images (III)*, pages 47–57, 2001. 1

[17] Michael J Casey and Sriharsha Vankadara. Semantics in cad/gis integration. *CAD and GIS Integration*, 143, 2010. 19

[18] S Chan, G Sohn, L Wang, and W Lee. Dynamic wifi-based indoor positioning in 3d virtual world. *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 1(4):1–6, 2013. 1

[19] Jack Cheng, Yichuan Deng, and Qianru Du. Mapping between bim models and 3d gis city models of different levels of detail. In *13th international conference on construction applications of virtual reality, London*, pages 30–31, 2013. 20

[20] Sergio Consoli, Misael Mongiovic, Andrea G Nuzzolese, Silvio Peroni, Valentina Presutti, Diego Reforgiato Recupero, and Daria Spampinato. A smart city data model based on semantics best practice and principles. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1395–1400. ACM, 2015. 19

[21] World Wide Web Consortium. Sparql protocol for rdf. `https://www.w3.org/TR/rdf-sparql-protocol/`. 18

[22] World Wide Web Consortium. Web ontology language (owl). `https://www.w3.org/2001/sw/wiki/OWL`. 16

[23] Edward Curry, James O'Donnell, and Edward Corry. Building optimisation using scenario modeling and linked data. In *First International Workshop on Linked Data in Architecture and Construction (LDAC 2012)*, pages 6–8, 2012. 20

[24] Edward Curry, James O'Donnell, Edward Corry, Souleiman Hasan, Marcus Keane, and Seán O'Riain. Linking building data in the cloud: Integrating cross-domain building data using linked data. *Advanced Engineering Informatics*, 27(2):206–219, 2013. 20

[25] Mathieu d'Aquin, John Davies, and Enrico Motta. Smart cities' data: Challenges and opportunities for semantic technologies. 19:66–70, 11 2015. 19

[26] Ruben de Laat and Leon Van Berlo. Integration of bim and gis: The development of the citygml geobim extension. In *Advances in 3D geo-information sciences*, pages 211–225. Springer, 2011. 20

[27] Jiri Dokulil, Jaroslav Tykal, Jakub Yaghob, and Filip Zavoral. Semantic web infrastructure. In *Semantic Computing, 2007. ICSC 2007. International Conference on*, pages 209–215. IEEE, 2007. 16

[28] Jürgen Döllner and Benjamin Hagedorn. Integrating urban gis, cad, and bim data by service-based virtual 3d city models. *Urban and regional data management-annual*, pages 157–160, 2007. 20

[29] Jürgen Döllner, Thomas H Kolbe, Falko Liecke, Takis Sgouros, Karin Teichmann, et al. The virtual 3d city model of berlin-managing, integrating, and communicating complex urban information. In *Proceedings of the 25th Urban Data Management Symposium UDMS*, volume 2006, pages 15–17, 2006. 1

[30] Mohamed El-Mekawy and Anders Östman. Semantic mapping: an ontology engineering method for integrating building models in ifc and citygml. In *3rd ISDE DIGITAL EARTH SUMMIT, 12-14 June,*, 2010. 20

[31] Mohamed El-Mekawy, Anders Östman, and Ihab Hijazi. A unified building model for 3d urban gis. *ISPRS International Journal of Geo-Information*, 1(2):120–145, 2012. 10

[32] Reinhard Erstling and Clemens Portele. Standardization of graphic data in road and transport part 1: Study. *research and development, ss number = 724, year = 1996.* 13

[33] Vladimir Geroimenko. *Dictionary of XML technologies and the semantic web*, volume 1. Springer Science & Business Media, 2004. 16

[34] G Gröger, TH Kolbe, C Nagel, and KH Häfele. Ogc city geography markup language (citygml) encoding standard, version 2.0, ogc doc no. 12-019. *Open Geospatial Consortium*, 2012. 9

[35] Gerhard Gröger, Thomas H Kolbe, Angela Czerwinski, and Claus Nagel. Opengis city geography markup language (citygml) encoding standard. *Open Geospatial Consortium Inc*, pages 1–234, 2008. 9

[36] Gerhard Gröger, Thomas H Kolbe, Claus Nagel, and Karl-Heinz Häfele. Open geospatial consortium ogc city geography markup language (citygml) encoding standard, 2012. 10

[37] Thomas R Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993. 33

[38] Xingui He, Ertian Hua, Yun Lin, and Xiaozhu Liu. *Computer, Informatics, Cybernetics and Applications: Proceedings of the CICA 2011*, volume 107. Springer Science & Business Media, 2011. 16, 17

[39] Tom Heath and Christian Bizer. Linked data: Evolving the web into a global data space. *Synthesis lectures on the semantic web: theory and technology*, 1(1):1–136, 2011. 18

[40] Jochen HETTWER. The object catalog for traffic and transport - okstra ®. 13

[41] A-H Hor, A Jadidi, and G Sohn. Bim-gis integrated geospatial information model using semantic web and rdf graphs. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 3(4), 2016. 21

[42] Ebrahim P Karan and Javier Irizarry. Extending bim interoperability to preconstruction operations using geospatial analyses and semantic web services. *Automation in Construction*, 53:1–12, 2015. 1

[43] Dave Kolas, Matt Perry, and John Herring. Getting started with geosparql. *OGC presentation*, 2013. 40

[44] Thomas H Kolbe. Representing and exchanging 3d city models with citygml. *3D geo-information sciences*, pages 15–31, 2009. 9

[45] Freddy Lécué, Simone Tallevi-Diotallevi, Jer Hayes, Robert Tucker, Veli Bicer, Marco Sbodio, and Pierpaolo Tommasi. Smart traffic analytics in the semantic web with star-city: Scenarios, system and lessons learned in dublin city. *Web Semantics: Science, Services and Agents on the World Wide Web*, 27:26–33, 2014. 19

[46] Claudine Métral, Roland Billen, Anne-Francoise Cutting-Decelle, and Muriel Van Ruymbeke. Ontology-based approaches for improving the interoperability between 3d urban models. *Journal of Information Technology in Construction*, 15:169–184, 2010. 20

[47] Claudine Métral and Gilles Falquet. Prototyping information visualization in 3d city models: a model-based approach. *arXiv preprint arXiv:1505.07267*, 2015. 21

[48] Claudine Metral, Nizar Ghoula, and Gilles Falquet. An ontology of 3d visualization techniques for enriched 3d city models. In *Usage, Usability, and Utility of 3D City Models–European COST Action TU0801*, page 02005. EDP Sciences, 2012. 20

[49] Pieter Pauwels, Ronald De Meyer, and Jan Van Campenhout. Interoperability for the design and construction industry through semantic web technology. In *International Conference on Semantic and Digital Media Technologies*, pages 143–158. Springer, 2010. 19, 20

[50] Pieter Pauwels, Sijie Zhang, and Yong-Cheol Lee. Semantic web technologies in aec industry: A literature overview. *Automation in Construction*, 73:145–165, 2017. 9, 19

[51] Matthew Perry and John Herring. Ogc geosparql-a geographic query language for rdf data. *OGC Implementation Standard. Sept*, 2012. 40

[52] F Prandi, M Soave, F Devigili, M Andreolli, and R De Amicis. Services oriented smart city platform based on 3d city model visualization. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2(4):59, 2014. 1

[53] W Rueffer. Okstra-the key to road and traffic data. *Road ss e And Motorway*, 52(2), 2001. 13

[54] Hans Schevers and Robin Drogemuller. Converting the industry foundation classes to the web ontology language. In *Semantics, Knowledge and Grid, 2005. SKG'05. First International Conference on*, pages 73–73. IEEE, 2005. 20

[55] Christian Schultze and Erich Buhmann. Developing the okstra® standard for the needs of landscape planning in context of implementation for mitigation and landscape envelope planning of road projects. In *International conference on information technologies in landscape architecture*, pages 310–320, 2008. 13

[56] Alexandra Stadler and Thomas H Kolbe. Spatio-semantic coherence in the integration of 3d city models. In *Proceedings of the 5th International Symposium on Spatial Data Quality, Enschede*, 2007. 1

[57] Igor Svetel and Milica Pejanović. The role of the semantic web for knowledge management in the construction industry. *Informatica*, 34(3), 2010. 19

[58] Pavel Tobiáš. An investigation into the possibilities of bim and gis cooperation and utilization of gis in the bim process. *Geoinformatics FCE CTU*, 14(1):65–78, 2015. 2

[59] Linda Van den Brink, Paul Janssen, and Wilko Quak. From geo-data to linked data: automated transformation from gml to rdf. *Linked Open Data-Pilot Linked Open Data Nederland. Deel 2-De Verdieping, Geonovum, 2013, pp. 249-261*, 2013. 20

[60] S Vilgertshofer, J Amann, B Willenborg, A Borrmann, and TH Kolbe. Linking bim and gis models in infrastructure by example of ifc and citygml. In *Computing in Civil Engineering 2017*, pages 133–140. 21

[61] Jan Wielemaker. A structured approach to covert xml into rdf. `http://semanticweb.cs.vu.nl/xmlrdf/`. 23, 35

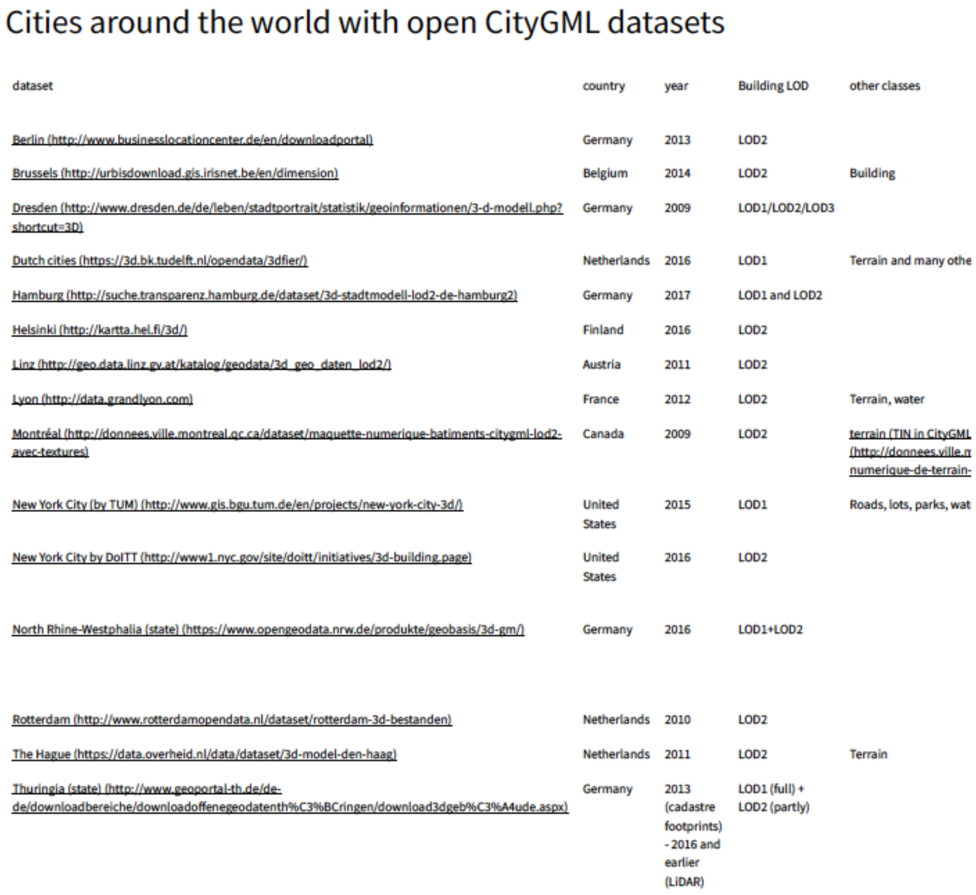
[62] Xun Xu, Lieyun Ding, Hanbin Luo, and Ling Ma. From building information modeling to city information modeling. *Journal of Information Technology in Construction (ITcon)*, 19(17):292–307, 2014. 1

[63] Ningyu Zhang, Huajun Chen, Xi Chen, and Jiaoyan Chen. Semantic framework of internet of things for smart cities: case studies. *Sensors*, 16(9):1501, 2016. 19

[64] Qing Zhu, Junqiao Zhao, Zhiqiang Du, Yeting Zhang, Weiping Xu, Xiao Xie, Yulin Ding, Fei Wang, and Tingsong Wang. Towards semantic 3d city modeling and visual explorations. In *Advances in 3D Geo-Information Sciences*, pages 275–294. Springer, 2011. 1

Appendix A

Cities around the world with open CityGML data sets



Cities around the world with open CityGML datasets

dataset	country	year	Building LOD	other classes
Berlin (http://www.businesslocationcenter.de/en/downloadportal)	Germany	2013	LOD2	
Brussels (http://urbisdownload.gis.irisnet.be/en/dimension)	Belgium	2014	LOD2	Building
Dresden (http://www.dresden.de/de/leben/stadtportrait/statistik/geoinformationen/3-d-modell.php?shortcut=3D)	Germany	2009	LOD1/LOD2/LOD3	
Dutch cities (https://3d.bk.tudelft.nl/opendata/3dfier/)	Netherlands	2016	LOD1	Terrain and many othe
Hamburg (http://suche.transparenz.hamburg.de/dataset/3d-stadtmodell-lod2-de-hamburg2)	Germany	2017	LOD1 and LOD2	
Helsinki (http://kartta.hel.fi/3d/)	Finland	2016	LOD2	
Linz (http://geo.data.linz.gv.at/katalog/geodata/3d_geo_daten_lod2/)	Austria	2011	LOD2	
Lyon (http://data.grandlyon.com)	France	2012	LOD2	Terrain, water
Montréal (http://donnees.ville.montreal.qc.ca/dataset/maquette-numerique-batiments-citygml-lod2-avec-textures)	Canada	2009	LOD2	terrain (TIN in CityGML (http://donnees.ville. numerique-de-terrain-
New York City (by TUM) (http://www.gis.bgu.tum.de/en/projects/new-york-city-3d/)	United States	2015	LOD1	Roads, lots, parks, wat
New York City by DoITT (http://www1.nyc.gov/site/doitt/initiatives/3d-building.page)	United States	2016	LOD2	
North Rhine-Westphalia (state) (https://www.opengeodata.nrw.de/produkte/geobasis/3d-gm/)	Germany	2016	LOD1+LOD2	
Rotterdam (http://www.rotterdamopendata.nl/dataset/rotterdam-3d-bestanden)	Netherlands	2010	LOD2	
The Hague (https://data.overheid.nl/data/dataset/3d-model-den-haag)	Netherlands	2011	LOD2	Terrain
Thuringia (state) (http://www.geoportal-th.de/de-de/downloadbereiche/downloadoffenegeodatenth%C3%BCringen/download3dgeb%C3%A4ude.aspx)	Germany	2013 (cadastre footprints) - 2016 and earlier (LiDAR)	LOD1 (full) + LOD2 (partly)	

Figure A.1: Cities around the world with open CityGML datasets (source: OGC)

Appendix B

Python code for parsing OKSTRA data

```
'''

@author: JacobBeetz&YuanZheng
'''
from lxml import etree
import sys
import csv

# variable for the data sets
source = r'/Users/pc/Desktop/Models/INS_NRW/INS_NRW.xml'
# source = r'd:\data\okstra\NRW_2016hj2\OKSTRA_Daten_QStr_20160705.xml'
# source = r'd:\data\okstra\NRW_2016hj2\
    OKSTRA_Daten_Netzdaten_ohne_Dnst_VerwB_20160705.xml'

def fast_iter(context, func, *args, **kwargs):
    """
        http://lxml.de/parsing.html#modifying-the-tree
        Based on Liza Daly's fast_iter
        http://www.ibm.com/developerworks/xml/library/x-hiperfparse/
        See also http://effbot.org/zone/element-iterparse.htm
        """
    for event, elem in context:
        func(elem)
        # It's safe to call clear() here because no descendants will be
        # accessed
        elem.clear()
    # Also eliminate now-empty references from the root node to elem
    # for ancestor in elem.xpath('ancestor-or-self::*'):
    #     while ancestor.getprevious() is not None:
    #         del ancestor.getparent()[0]
    del context



#global variable to count e.g. the number of okstraObjekt instances
count = 0
lineofpos=[]

def getPosition(elem):
    """
        :param elem: gml:featureMember XML node to retrieve a position from
        :return:
        """
    # get the XPath of a gml position in the graph
    positions = elem.xpath('okstra:Netzknoten/okstra:Punktgeometrie/gml:Point/gml:
        pos',namespaces={ 'okstra': 'http://schema.okstra.de/2016/okstra', 'gml':'
        http://www.opengis.net/gml/3.2'})
    # positions = elem.xpath('okstra:Abschnitt',namespaces={ 'okstra': 'http://
        schema.okstra.de/2016/okstra', 'gml':'http://www.opengis.net/gml/3.2'})



    for pos in positions:
        global count
        count = count +1
        if count % 1000 ==0:
            print(count)
        # print the coordinates of the gml:pos element
        print ("netzknoten",pos.text)
        lineofpos.append(pos.text + '\n')



#search for all gml:featureMember nodes in the file
context = etree.iterparse(source, tag='{http://www.opengis.net/gml/3.2}
    featureMember', events = ('end', ))
# the callback function to execute, when a featuerNode member is found
fast_iter(context, getPosition)


# print the total number of positions found
print (count)
print(len(lineofpos))
file= open ('output.csv','w')
file.writelines(lineofpos)
file.close()
```

Appendix C

Python code for finding Netzknoten in the target area

```
'''

@author: YuanZheng
'''
import csv
import pandas as pd
import numpy as np

data = pd.read_csv('/Users/pc/Desktop/data/okstra data extract/output/output1.csv')
xcoord = np.array(data.X)
ycoord = np.array(data.Y)
zcoord = np.array(data.Z)

#search for the coordinates within target area in the file
indices=[]
for i in range(0,np.size(xcoord)):
    if xcoord[i] <=296000 and  xcoord[i] >= 295000 and ycoord[i] <= 5630000 and
        ycoord[i] >= 5629000:
        indices.append(i)
output = {'X':xcoord[indices],
    'Y':ycoord[indices],
        'Z':zcoord[indices]}
df = pd.DataFrame(output)
df.to_csv('rect_area_new.csv',index= False)
```

Appendix D

XML Schema List for converting OKSTRA Schema to OWL

```
<?xml version='1.0' encoding='utf-8'?>

<!DOCTYPE xsl:stylesheet [
 <!ENTITY owl "http://www.w3.org/2002/07/owl#">
<!ENTITY gml "http://www.w3.org/2002/07/owl#">
<!ENTITY core "http://www.opengis.net/citygml/1.0#">
]>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:gml="http://www.opengis.net/gml/3.2#"
    xmlns:okstra="http://www.okstra.de/okstra/2.017#"
    xmlns:okstra-basis="http://www.okstra.de/okstra/2.017#"
    xmlns:okstra-types="http://www.okstra.de/okstra/2.017#"
  >


<xsl:template match='xs:schema'>

<xsl:text>

</xsl:text>

<rdf:RDF xmlns="http://www.w3.org/2001/XMLSchema#"
     xml:base="http://www.okstra.de/okstra/2.017#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:gml="http://www.opengis.net/gml/3.2#"
     xmlns:okstra="http://www.okstra.de/okstra/2.017#"
     xmlns:okstra-basis="http://www.okstra.de/okstra/2.017#"
     xmlns:okstra-types="http://www.okstra.de/okstra/2.017#">

<owl:Ontology rdf:about="myokstra.tue.nl/RoadnetworkOntology"/>

<xsl:apply-templates select="xs:complexType"/>
<xsl:apply-templates select="xs:simpleType"/>
<xsl:apply-templates select="xs:element[@substitutionGroup]"/>

</rdf:RDF>
```

```
</xsl:template>

<xsl:template match="xs:complexType">
  <xsl:text>

  </xsl:text>
<owl:Class>
    <!-- Content: template -->
    <xsl:attribute name="rdf:about"> <!-- was about" -->
      <xsl:value-of select="@name" />
     </xsl:attribute>
<xsl:text>

</xsl:text> <!-- LINE FEED -->
  <xsl:variable name="txt">
  <xsl:value-of select="@name" />
  </xsl:variable>


  <xsl:if test="contains($txt, 'Property')">
  <xsl:comment>association</xsl:comment>
  </xsl:if>
  <xsl:apply-templates select="./xs:annotation/xs:documentation"></xsl:apply-
      templates>
  <xsl:apply-templates select="./xs:complexContent/xs:extension"/>
  <xsl:apply-templates select="./xs:complexContent/xs:extension/xs:sequence/
      xs:element[@name]"/>
  <xsl:apply-templates select="./xs:complexContent/*/xs:sequence/xs:element[@ref]"
      />

</owl:Class>

  <!-- generate datatype properties -->
  <xsl:for-each select="./xs:complexContent/xs:extension/xs:sequence/xs:element[
      starts-with(@type,'xs:')]">
    <owl:DatatypeProperty>
      <xsl:attribute name="rdf:about"><xsl:value-of select="@name"/></xsl:attribute
          > <!-- was rdf:about -->
    </owl:DatatypeProperty>
  </xsl:for-each>
</xsl:template>

<xsl:template match="xs:documentation">
   <rdfs:comment><xsl:value-of select="."/>
   </rdfs:comment>

</xsl:template>
<xsl:template match="xs:simpleType">
   <xsl:text>
   </xsl:text> <!-- LINE FEED -->
  <owl:Class>
    <xsl:attribute name="rdf:about"> <!-- about -->
      <xsl:value-of select="@name" />
    </xsl:attribute>
    <xsl:comment>simple class</xsl:comment>

   </owl:Class>
</xsl:template>


<xsl:template match="xs:extension">
   <rdfs:subClassOf>
     <xsl:attribute name="rdf:resource"><xsl:value-of select="@base"/></
         xsl:attribute>
   </rdfs:subClassOf>
</xsl:template>
```

```
<xsl:template match="/xs:schema/xs:element">
      <owl:ObjectProperty>
         <xsl:attribute name="rdf:about">  <!-- about -->
            <xsl:value-of select="@name"/>
         </xsl:attribute>

         <xsl:if test="@substitutionGroup != ''">
            <rdfs:subPropertyOf>
              <xsl:attribute name="rdf:resource"> <xsl:value-of select="
                  @substitutionGroup"/>
              </xsl:attribute>
            </rdfs:subPropertyOf>
         </xsl:if>

         <rdfs:range>
            <xsl:attribute name="rdf:resource"> <xsl:value-of select="@type"/>
            </xsl:attribute>
         </rdfs:range>


      </owl:ObjectProperty>

</xsl:template>

<xsl:template match="xs:complexType//xs:element[@name]">
  <xsl:text>

  </xsl:text>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <xsl:attribute name="rdf:resource"><xsl:value-of select="@name"/></
            xsl:attribute>
      </owl:onProperty>
      <owl:allValuesFrom>
        <xsl:attribute name="rdf:resource"><xsl:value-of select="@type"/></
            xsl:attribute>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>

  <xsl:if test="not(@minOccurs='0')"> <!--- mandatory field -->
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <xsl:attribute name="rdf:resource"><xsl:value-of select="@name"/></
              xsl:attribute>
        </owl:onProperty>
        <owl:someValuesFrom>
          <xsl:attribute name="rdf:resource"><xsl:value-of select="@type"/></
              xsl:attribute>
        </owl:someValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
  </xsl:if>
</xsl:template>

<xsl:template match="xs:complexType//xs:element[@ref]">
    <xsl:text>

    </xsl:text>
    <xsl:variable name="refElt">
       <xsl:value-of select="@ref"/>
    </xsl:variable>
    <xsl:variable name="refEltType">
       <xsl:value-of select="//xs:element[@name=$refElt]/@type"/>
    </xsl:variable>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <xsl:attribute name=”rdf:resource”>
             <xsl:value-of select=”@ref”/>
          </xsl:attribute>
        </owl:onProperty>
        <owl:allValuesFrom>
          <xsl:attribute name=”rdf:resource”><xsl:value-of select=”$refEltType”/></
              xsl:attribute>
        </owl:allValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>

    <xsl:if test=”not(@minOccurs=’0’)”> <!--- mandatory field -->
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty>
            <xsl:attribute name=”rdf:resource”><xsl:value-of select=”@ref”/></
                xsl:attribute>
          </owl:onProperty>
          <owl:someValuesFrom>
            <xsl:attribute name=”rdf:resource”><xsl:value-of select=”$refEltType”
                /></xsl:attribute>
          </owl:someValuesFrom>
        </owl:Restriction>
      </rdfs:subClassOf>
    </xsl:if>
</xsl:template>



</xsl:stylesheet>
```

Appendix E

Adapted CityGML Ontology for Experiment

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://www.opengis.net/citygml/1.0#"
     xml:base="http://www.opengis.net/citygml/1.0"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Ontology rdf:about="http://www.opengis.net/citygml/1.0"/>



    <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Object Properties
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->




    <!-- http://www.opengis.net/citygml/
        _GenericApplicationPropertyOfAbstractBuilding -->

    <owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/
        _GenericApplicationPropertyOfAbstractBuilding"/>



    <!-- http://www.opengis.net/citygml/
        _GenericApplicationPropertyOfBoundarySurface -->

    <owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/
        _GenericApplicationPropertyOfBoundarySurface"/>



    <!-- http://www.opengis.net/citygml/_GenericApplicationPropertyOfBuilding -->

    <owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/
        _GenericApplicationPropertyOfBuilding"/>
```

```
<!-- http://www.opengis.net/citygml/_GenericApplicationPropertyOfBuildingPart
    -->

<owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/
    _GenericApplicationPropertyOfBuildingPart"/>



<!-- http://www.opengis.net/citygml/_GenericApplicationPropertyOfGroundSurface
    -->

<owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/
    _GenericApplicationPropertyOfGroundSurface"/>



<!-- http://www.opengis.net/citygml/_GenericApplicationPropertyOfRoofSurface --
    >

<owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/
    _GenericApplicationPropertyOfRoofSurface"/>



<!-- http://www.opengis.net/citygml/_GenericApplicationPropertyOfWallSurface --
    >

<owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/
    _GenericApplicationPropertyOfWallSurface"/>



<!-- http://www.opengis.net/citygml/boundedBy -->

<owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/boundedBy"/>



<!-- http://www.opengis.net/citygml/consistsOfBuildingPart -->

<owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/
    consistsOfBuildingPart"/>



<!-- http://www.opengis.net/citygml/function -->

<owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/function"/>



<!-- http://www.opengis.net/citygml/lod1MultiSurface -->

<owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/lod1MultiSurface"
    />



<!-- http://www.opengis.net/citygml/lod1Solid -->

<owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/lod1Solid"/>
```

```
<!-- http://www.opengis.net/citygml/lod1TerrainIntersection -->

<owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/
    lod1TerrainIntersection"/>



<!-- http://www.opengis.net/citygml/lod2MultiCurve -->

<owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/lod2MultiCurve"/>



<!-- http://www.opengis.net/citygml/lod2MultiSurface -->

<owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/lod2MultiSurface"
    />



<!-- http://www.opengis.net/citygml/lod2Solid -->

<owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/lod2Solid"/>



<!-- http://www.opengis.net/citygml/lod2TerrainIntersection -->

<owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/
    lod2TerrainIntersection"/>



<!-- http://www.opengis.net/citygml/measuredHeight -->

<owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/measuredHeight"/>



<!-- http://www.opengis.net/citygml/roofType -->

<owl:ObjectProperty rdf:about="http://www.opengis.net/citygml/roofType"/>



<!--
///////////////////////////////////////////////////////////////////////////////////////

//
// Classes
//
///////////////////////////////////////////////////////////////////////////////////////

 -->




<!-- core:AbstractCityObject -->

<owl:Class rdf:about="core:AbstractCityObject"/>



<!-- core:AbstractSite -->

<owl:Class rdf:about="core:AbstractSite"/>
```

```
<!-- gml:CodeType -->

<owl:Class rdf:about="gml:CodeType"/>


<!-- gml:LengthType -->

<owl:Class rdf:about="gml:LengthType"/>


<!-- gml:MultiCurvePropertyType -->

<owl:Class rdf:about="gml:MultiCurvePropertyType"/>


<!-- gml:MultiSurfacePropertyType -->

<owl:Class rdf:about="gml:MultiSurfacePropertyType"/>


<!-- gml:SolidPropertyType -->

<owl:Class rdf:about="gml:SolidPropertyType"/>


<!-- http://www.opengis.net/citygml/AbstractBoundarySurface -->

<owl:Class rdf:about="http://www.opengis.net/citygml/AbstractBoundarySurface">
    <rdfs:subClassOf rdf:resource="core:AbstractCityObject"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.opengis.net/citygml/
                _GenericApplicationPropertyOfBoundarySurface"/>
            <owl:allValuesFrom rdf:resource="xs:anyType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.opengis.net/citygml/
                lod2MultiSurface"/>
            <owl:allValuesFrom rdf:resource="gml:MultiSurfacePropertyType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:comment>A BoundarySurface is a thematic object which classifies
        surfaces bounding an _AbstractBuilding, Room,
    BuildingInstallation, and IntBuildingInstallation. The geometry of a
        BoundarySurface is given by MultiSurfaces. As it is a
    subclass of _CityObject, it inherits all atributes and relations, in
        particular the external references, and the
    generalization relations. </rdfs:comment>
</owl:Class>


<!-- http://www.opengis.net/citygml/AbstractBuilding -->

<owl:Class rdf:about="http://www.opengis.net/citygml/AbstractBuilding">
    <rdfs:subClassOf rdf:resource="core:AbstractSite"/>
    <rdfs:subClassOf>
```

```
            <owl:Restriction>
                <owl:onProperty rdf:resource=”http://www.opengis.net/citygml/
                    _GenericApplicationPropertyOfAbstractBuilding”/>
                <owl:allValuesFrom rdf:resource=”xs:anyType”/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource=”http://www.opengis.net/citygml/
                    boundedBy”/>
                <owl:allValuesFrom rdf:resource=”http://www.opengis.net/citygml/
                    BoundarySurfacePropertyType”/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource=”http://www.opengis.net/citygml/
                    consistsOfBuildingPart”/>
                <owl:allValuesFrom rdf:resource=”http://www.opengis.net/citygml/
                    BuildingPartPropertyType”/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource=”http://www.opengis.net/citygml/
                    function”/>
                <owl:allValuesFrom rdf:resource=”gml:CodeType”/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource=”http://www.opengis.net/citygml/
                    lod1MultiSurface”/>
                <owl:allValuesFrom rdf:resource=”gml:MultiSurfacePropertyType”/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource=”http://www.opengis.net/citygml/
                    lod1Solid”/>
                <owl:allValuesFrom rdf:resource=”gml:SolidPropertyType”/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource=”http://www.opengis.net/citygml/
                    lod1TerrainIntersection”/>
                <owl:allValuesFrom rdf:resource=”gml:MultiCurvePropertyType”/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource=”http://www.opengis.net/citygml/
                    lod2MultiCurve”/>
                <owl:allValuesFrom rdf:resource=”gml:MultiCurvePropertyType”/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource=”http://www.opengis.net/citygml/
                    lod2MultiSurface”/>
                <owl:allValuesFrom rdf:resource=”gml:MultiSurfacePropertyType”/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
```

```
                <owl:onProperty rdf:resource="http://www.opengis.net/citygml/
                    lod2Solid"/>
                <owl:allValuesFrom rdf:resource="gml:SolidPropertyType"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="http://www.opengis.net/citygml/
                    lod2TerrainIntersection"/>
                <owl:allValuesFrom rdf:resource="gml:MultiCurvePropertyType"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="http://www.opengis.net/citygml/
                    measuredHeight"/>
                <owl:allValuesFrom rdf:resource="gml:LengthType"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="http://www.opengis.net/citygml/
                    roofType"/>
                <owl:allValuesFrom rdf:resource="gml:CodeType"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:comment>Type describing the thematic and geometric attributes and the
             associations of buildings. It is an abstract
        type, only its subclasses Building and BuildingPart can be instantiated. An
            _AbstractBuilding may consist of
        BuildingParts, which are again _AbstractBuildings by inheritance. Thus an
            aggregation hierarchy between _AbstractBuildings
        of arbitrary depth may be specified. In such an hierarchy, top elements are
             Buildings, while all other elements are
        BuildingParts. Each element of such a hierarchy may have all attributes and
             geometries of _AbstractBuildings. It must,
        however, be assured than no inconsistencies occur (for example, if the
            geometry of a Building does not correspond to the
        geometries of its parts, or if the roof type of a Building is saddle roof,
            while its parts have an hip roof). As subclass
        of _CityObject, an _AbstractBuilding inherits all attributes and relations,
             in particular an id, names, external
        references, and generalization relations. </rdfs:comment>
    </owl:Class>



    <!-- http://www.opengis.net/citygml/BoundarySurfacePropertyType -->

    <owl:Class rdf:about="http://www.opengis.net/citygml/
        BoundarySurfacePropertyType"/>



    <!-- http://www.opengis.net/citygml/Building -->

    <owl:Class rdf:about="http://www.opengis.net/citygml/Building">
        <rdfs:subClassOf rdf:resource="http://www.opengis.net/citygml/
            AbstractBuilding"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="http://www.opengis.net/citygml/
                    _GenericApplicationPropertyOfBuilding"/>
                <owl:allValuesFrom rdf:resource="xs:anyType"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>
```

```
<!-- http://www.opengis.net/citygml/BuildingPart -->

<owl:Class rdf:about="http://www.opengis.net/citygml/BuildingPart">
    <rdfs:subClassOf rdf:resource="http://www.opengis.net/citygml/
        AbstractBuilding"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.opengis.net/citygml/
                _GenericApplicationPropertyOfBuildingPart"/>
            <owl:allValuesFrom rdf:resource="xs:anyType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>



<!-- http://www.opengis.net/citygml/BuildingPartPropertyType -->

<owl:Class rdf:about="http://www.opengis.net/citygml/BuildingPartPropertyType"/
    >



<!-- http://www.opengis.net/citygml/GroundSurface -->

<owl:Class rdf:about="http://www.opengis.net/citygml/GroundSurface">
    <rdfs:subClassOf rdf:resource="http://www.opengis.net/citygml/
        AbstractBoundarySurface"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.opengis.net/citygml/
                _GenericApplicationPropertyOfGroundSurface"/>
            <owl:allValuesFrom rdf:resource="xs:anyType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>



<!-- http://www.opengis.net/citygml/RoofSurface -->

<owl:Class rdf:about="http://www.opengis.net/citygml/RoofSurface">
    <rdfs:subClassOf rdf:resource="http://www.opengis.net/citygml/
        AbstractBoundarySurface"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.opengis.net/citygml/
                _GenericApplicationPropertyOfRoofSurface"/>
            <owl:allValuesFrom rdf:resource="xs:anyType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>



<!-- http://www.opengis.net/citygml/WallSurface -->

<owl:Class rdf:about="http://www.opengis.net/citygml/WallSurface">
    <rdfs:subClassOf rdf:resource="http://www.opengis.net/citygml/
        AbstractBoundarySurface"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.opengis.net/citygml/
                _GenericApplicationPropertyOfWallSurface"/>
                <owl:allValuesFrom rdf:resource="xs:anyType"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- xs:anyType -->

    <owl:Class rdf:about="xs:anyType"/>
</rdf:RDF>



<!-- Generated by the OWL API (version 4.3.1) https://github.com/owlcs/owlapi -->
```

Appendix F

Adapted OKSTRA Ontology for Experiment

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://www.okstra.de/okstra/myokstra.tue.nl/RoadnetworkOntology#"
     xml:base="http://www.okstra.de/okstra/myokstra.tue.nl/RoadnetworkOntology"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Ontology rdf:about="http://www.okstra.de/okstra/myokstra.tue.nl/
        RoadnetworkOntology"/>



    <!--
    ///////////////////////////////////////////////////////////////////////////////////////

    //
    // Object Properties
    //
    ///////////////////////////////////////////////////////////////////////////////////////

     -->




    <!-- http://www.okstra.de/okstra/Abschnitts_Astbezeichnung -->

    <owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/
        Abschnitts_Astbezeichnung"/>



    <!-- http://www.okstra.de/okstra/Abschnitts_Astnummer -->

    <owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/Abschnitts_Astnummer
        "/>



    <!-- http://www.okstra.de/okstra/Abschnittsfolgenummer -->

    <owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/
        Abschnittsfolgenummer"/>

```

```
<!-- http://www.okstra.de/okstra/Beginn_von_Abschnitt_oder_Ast -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/
    Beginn_von_Abschnitt_oder_Ast"/>



<!-- http://www.okstra.de/okstra/Betriebsmerkmal -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/Betriebsmerkmal"/>



<!-- http://www.okstra.de/okstra/Ende_von_Abschnitt_oder_Ast -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/
    Ende_von_Abschnitt_oder_Ast"/>



<!-- http://www.okstra.de/okstra/Knotenart -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/Knotenart"/>



<!-- http://www.okstra.de/okstra/Laenge -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/Laenge"/>



<!-- http://www.okstra.de/okstra/Liniengeometrie -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/Liniengeometrie"/>



<!-- http://www.okstra.de/okstra/Nullpunktart -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/Nullpunktart"/>



<!-- http://www.okstra.de/okstra/Numerierungsbezirk -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/Numerierungsbezirk"/
    >



<!-- http://www.okstra.de/okstra/Nummer -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/Nummer"/>



<!-- http://www.okstra.de/okstra/Nummer_gehoert_zu_Strasse -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/
    Nummer_gehoert_zu_Strasse"/>



<!-- http://www.okstra.de/okstra/OKSTRA_ID -->
```

```
<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/OKSTRA_ID"/>


<!-- http://www.okstra.de/okstra/Punktgeometrie -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/Punktgeometrie"/>


<!-- http://www.okstra.de/okstra/Seitenarm -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/Seitenarm"/>


<!-- http://www.okstra.de/okstra/Zusatz -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/Zusatz"/>


<!-- http://www.okstra.de/okstra/beginnt_bei_NP -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/beginnt_bei_NP"/>


<!-- http://www.okstra.de/okstra/endet_bei_NP -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/endet_bei_NP"/>


<!-- http://www.okstra.de/okstra/getrennt_verlaufende_Fahrbahn -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/
    getrennt_verlaufende_Fahrbahn"/>


<!-- http://www.okstra.de/okstra/hat_AoA_zugeordnet -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/hat_AoA_zugeordnet"/
    >


<!-- http://www.okstra.de/okstra/hat_Nullpunkt -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/hat_Nullpunkt"/>


<!-- http://www.okstra.de/okstra/hat_Nullpunktort -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/hat_Nullpunktort"/>


<!-- http://www.okstra.de/okstra/hat_Strassenbezeichnung -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/
    hat_Strassenbezeichnung"/>
```

```
<!-- http://www.okstra.de/okstra/hat_Strassenbezugsobjekt -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/
    hat_Strassenbezugsobjekt"/>



<!-- http://www.okstra.de/okstra/in_Netzknoten -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/in_Netzknoten"/>



<!-- http://www.okstra.de/okstra/zu_Strasse -->

<owl:ObjectProperty rdf:about="http://www.okstra.de/okstra/zu_Strasse"/>



<!--
///////////////////////////////////////////////////////////////////////////////////////

//
// Classes
//
///////////////////////////////////////////////////////////////////////////////////////

 -->




<!-- gml:AbstractFeature -->

<owl:Class rdf:about="gml:AbstractFeature"/>



<!-- gml:CurvePropertyType -->

<owl:Class rdf:about="gml:CurvePropertyType"/>



<!-- gml:PointPropertyType -->

<owl:Class rdf:about="gml:PointPropertyType"/>



<!-- http://www.okstra.de/okstra/Abschnitt -->

<owl:Class rdf:about="http://www.okstra.de/okstra/Abschnitt">
    <rdfs:subClassOf rdf:resource="gml:AbstractFeature"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Abschnitts_Astnummer"/>
            <owl:someValuesFrom rdf:resource="http://www.w3.org/2001/
                XMLSchemastring"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Abschnitts_Astbezeichnung"/>
            <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/
                XMLSchemastring"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Abschnitts_Astnummer"/>
            <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/
                XMLSchemastring"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Abschnittsfolgenummer"/>
            <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/
                XMLSchemainteger"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Betriebsmerkmal"/>
            <owl:allValuesFrom rdf:resource="okstra-basis:KeyValuePropertyType"
                />
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/Laenge"/>
            <owl:allValuesFrom rdf:resource="okstra-typen:Kilometer"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Liniengeometrie"/>
            <owl:allValuesFrom rdf:resource="gml:CurvePropertyType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Nummer_gehoert_zu_Strasse"/>
            <owl:allValuesFrom rdf:resource="okstra-basis:ObjectRefType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/OKSTRA_ID
                "/>
            <owl:allValuesFrom rdf:resource="okstra-typen:GUID"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/Seitenarm
                "/>
            <owl:allValuesFrom rdf:resource="okstra-basis:KeyValuePropertyType"
                />
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                beginnt_bei_NP"/>
            <owl:allValuesFrom rdf:resource="okstra-basis:ObjectRefType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                endet_bei_NP"/>
            <owl:allValuesFrom rdf:resource="okstra-basis:ObjectRefType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                getrennt_verlaufende_Fahrbahn"/>
            <owl:allValuesFrom rdf:resource="okstra-basis:KeyValuePropertyType"
                />
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                zu_Strasse"/>
            <owl:allValuesFrom rdf:resource="okstra-basis:ObjectRefType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>



<!-- http://www.okstra.de/okstra/Ast -->

<owl:Class rdf:about="http://www.okstra.de/okstra/Ast">
    <rdfs:subClassOf rdf:resource="gml:AbstractFeature"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Abschnitts_Astbezeichnung"/>
            <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/
                XMLSchemastring"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Abschnitts_Astnummer"/>
            <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/
                XMLSchemastring"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Betriebsmerkmal"/>
            <owl:allValuesFrom rdf:resource="okstra-basis:KeyValuePropertyType"
                />
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/Laenge"/>
            <owl:allValuesFrom rdf:resource="okstra-typen:Kilometer"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Liniengeometrie"/>
            <owl:allValuesFrom rdf:resource="gml:CurvePropertyType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Nummer_gehoert_zu_Strasse"/>
            <owl:allValuesFrom rdf:resource="okstra-basis:ObjectRefType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/OKSTRA_ID
                "/>
            <owl:allValuesFrom rdf:resource="okstra-typen:GUID"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                beginnt_bei_NP"/>
            <owl:allValuesFrom rdf:resource="okstra-basis:ObjectRefType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                endet_bei_NP"/>
            <owl:allValuesFrom rdf:resource="okstra-basis:ObjectRefType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                zu_Strasse"/>
            <owl:allValuesFrom rdf:resource="okstra-basis:ObjectRefType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>



<!-- http://www.okstra.de/okstra/Netzknoten -->

<owl:Class rdf:about="http://www.okstra.de/okstra/Netzknoten">
    <rdfs:subClassOf rdf:resource="gml:AbstractFeature"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/Knotenart
                "/>
            <owl:allValuesFrom rdf:resource="okstra-basis:KeyValuePropertyType"
                />
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Numerierungsbezirk"/>
            <owl:allValuesFrom rdf:resource="okstra-typen:TK25_Blattnummer"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/Nummer"/>
            <owl:allValuesFrom rdf:resource="okstra-typen:lfd_NK_Nummer"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/OKSTRA_ID
                "/>
            <owl:allValuesFrom rdf:resource="okstra-typen:GUID"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Punktgeometrie"/>
            <owl:allValuesFrom rdf:resource="gml:PointPropertyType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                hat_Nullpunkt"/>
            <owl:allValuesFrom rdf:resource="okstra-basis:ObjectRefType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>



<!-- http://www.okstra.de/okstra/Nullpunkt -->

<owl:Class rdf:about="http://www.okstra.de/okstra/Nullpunkt">
    <rdfs:subClassOf rdf:resource="gml:AbstractFeature"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Beginn_von_Abschnitt_oder_Ast"/>
            <owl:allValuesFrom rdf:resource="okstra-basis:ObjectRefType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Ende_von_Abschnitt_oder_Ast"/>
            <owl:allValuesFrom rdf:resource="okstra-basis:ObjectRefType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Nullpunktart"/>
            <owl:allValuesFrom rdf:resource="okstra-basis:KeyValuePropertyType"
                />
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/OKSTRA_ID
                "/>
            <owl:allValuesFrom rdf:resource="okstra-typen:GUID"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                Punktgeometrie"/>
            <owl:allValuesFrom rdf:resource="gml:PointPropertyType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
```

```
                <owl:onProperty rdf:resource="http://www.okstra.de/okstra/Zusatz"/>
                <owl:allValuesFrom rdf:resource="okstra-typen:Nullpunktkennung"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                    hat_Nullpunktort"/>
                <owl:allValuesFrom rdf:resource="okstra-basis:ObjectRefType"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                    in_Netzknoten"/>
                <owl:allValuesFrom rdf:resource="okstra-basis:ObjectRefType"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://www.okstra.de/okstra/Strasse -->

    <owl:Class rdf:about="http://www.okstra.de/okstra/Strasse">
        <rdfs:subClassOf rdf:resource="gml:AbstractFeature"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="http://www.okstra.de/okstra/OKSTRA_ID
                    "/>
                <owl:allValuesFrom rdf:resource="okstra-typen:GUID"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                    hat_AoA_zugeordnet"/>
                <owl:allValuesFrom rdf:resource="okstra-basis:ObjectRefType"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                    hat_Strassenbezeichnung"/>
                <owl:allValuesFrom rdf:resource="
                    okstra:StrassenbezeichnungPropertyType"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="http://www.okstra.de/okstra/
                    hat_Strassenbezugsobjekt"/>
                <owl:allValuesFrom rdf:resource="okstra-basis:ObjectRefType"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://www.w3.org/2001/XMLSchemainteger -->

    <owl:Class rdf:about="http://www.w3.org/2001/XMLSchemainteger"/>



    <!-- http://www.w3.org/2001/XMLSchemastring -->
    <owl:Class rdf:about="http://www.w3.org/2001/XMLSchemastring"/>



    <!-- okstra-basis:KeyValuePropertyType -->

    <owl:Class rdf:about="okstra-basis:KeyValuePropertyType"/>



    <!-- okstra-basis:ObjectRefType -->

    <owl:Class rdf:about="okstra-basis:ObjectRefType"/>



    <!-- okstra-typen:GUID -->

    <owl:Class rdf:about="okstra-typen:GUID"/>



    <!-- okstra-typen:Kilometer -->

    <owl:Class rdf:about="okstra-typen:Kilometer"/>



    <!-- okstra-typen:Nullpunktkennung -->

    <owl:Class rdf:about="okstra-typen:Nullpunktkennung"/>



    <!-- okstra-typen:TK25_Blattnummer -->

    <owl:Class rdf:about="okstra-typen:TK25_Blattnummer"/>



    <!-- okstra-typen:lfd_NK_Nummer -->

    <owl:Class rdf:about="okstra-typen:lfd_NK_Nummer"/>



    <!-- okstra:StrassenbezeichnungPropertyType -->

    <owl:Class rdf:about="okstra:StrassenbezeichnungPropertyType"/>
</rdf:RDF>



<!-- Generated by the OWL API (version 4.3.1) https://github.com/owlcs/owlapi -->
```

Appendix G

Python code for converting CityGML Data to RDF

```
'''

@author: JerryZheng
'''
from bs4 import BeautifulSoup

import rdflib
from pip.cmdoptions import allow_all_external
from rdflib.term import BNode
from numpy import poly
from builtins import str
from rdflib.plugins.sparql.operators import string

graph = rdflib.Graph(store='IOMemory', identifier='cityGraph')

gml = rdflib.Namespace("http://www.opengis.net/gml:")
core = rdflib.Namespace("http://www.opengis.net/citygml/1.0")
bldg = rdflib.Namespace("http://www.opengis.net/citygml/building/1.0")
gen = rdflib.Namespace("http://www.opengis.net/citygml/generics/1.0")
graph.bind('gml', gml)
graph.bind('core', core)
graph.bind('bldg', bldg)
graph.bind('gen', gen)

soup = BeautifulSoup(open("/Users/pc/Desktop/LoD2_295_5629_1_NW.gml"), 'lxml-xml')

city = soup.CityModel

cityObjectMembers = city.find_all('cityObjectMember')

for cityObjectMember in cityObjectMembers:
    building = cityObjectMember.Building
    building_id = building['gml:id']

    print(building_id)

    building_node = rdflib.URIRef('mycitygml.tue.nl/city/building/' + building_id)

    graph.add((building_node, rdflib.RDF.type,bldg.building))

    date = building.creationDate.text

    graph.add((building_node, core.creationDate, rdflib.Literal(date)))

    externalRef = BNode()
```

```
graph.add((building_node,core.externalReference,externalRef))

informationsystem = building.externalReference.informationSystem.text

graph.add((externalRef,core.informationSystem,rdflib.Literal(informationsystem)
    ))

externalObj = BNode()
graph.add((externalRef,core.externalObject,externalObj))

graph.add((externalObj,core.name,rdflib.Literal(building_id)))

stringattributes = building.find_all('stringAttribute')
for stringattribute in stringattributes:
    attr_name = stringattribute['name']
    attr_value = stringattribute.value.text

    string_attri = BNode()
    graph.add((building_node,gen.stringAttribute,string_attri))
    graph.add((string_attri,gen.name,rdflib.Literal(attr_name)))
    graph.add((string_attri,gen.value,rdflib.Literal(attr_value)))



building_function = building.function.text
graph.add((building_node,bldg.function,rdflib.Literal(building_function)))

measuredHeight = building.measuredHeight.text
graph.add((building_node,bldg.measuredHeight,rdflib.Literal(measuredHeight)))

if building.find('lod2Solid'):
#lod2
    if building.find('consistsOfBuildingPart'):
        # lod2withbuidlingparts
        buildingparts = building.find_all('BuildingPart')

        for buildingpart in buildingparts:
            buildingpart_id = buildingpart['gml:id']
            buildingpart_node = rdflib.URIRef('mycitygml.tue.nl/city/building/
                BuildingPart/' + buildingpart_id)
            graph.add((building_node,bldg.consistsOfBuildingPart,
                buildingpart_node))

            date = buildingpart.creationDate.text

            graph.add((buildingpart_node, core.creationDate, rdflib.Literal(
                date)))


            stringattributes = buildingpart.find_all('stringAttribute')
            for stringattribute in stringattributes:
                attr_name = stringattribute['name']
                attr_value = stringattribute.value.text

            if buildingpart.find('roofType'):
                buildingpart_rooftype = buildingpart.roofType.text
                graph.add((buildingpart_node,bldg.roofType,rdflib.Literal(
                    buildingpart_rooftype)))

            measuredHeight = buildingpart.measuredHeight.text
            graph.add((buildingpart_node,bldg.measuredHeight,rdflib.Literal(
                measuredHeight)))

            if buildingpart.find('lod2Solid'):
                solid = buildingpart.lod2Solid.Solid
                solid_id = solid['gml:id']
```

```
                    solid_node = rdflib.URIRef('mycitygml.tue.nl/city/building/
                        solid/' + solid_id)

                    graph.add((buildingpart_node,bldg.lod2Solid,solid_node))



                    compositesurface = solid.exterior.CompositeSurface
                    compositesurface_id = solid.exterior.CompositeSurface['gml:id']
                    compositesurface_node = rdflib.URIRef('mycitygml.tue.nl/city/
                        building/buildingpart/solid/compositesurface/' +
                        compositesurface_id)

                    graph.add((solid_node,gml.exterior,compositesurface_node))


                    surfacemembers = compositesurface.find_all('surfaceMember')
                    for surfacemember in surfacemembers:
                        surfacemember_xlink = surfacemember['xlink:href']
                        surfacemember_identifier = rdflib.URIRef('mycitygml.tue.nl/
                            city/building/buildingpart/solid/compositesurface/
                            surfaceMember' + surfacemember_xlink)
                        graph.add ((compositesurface_node,gml.surfaceMember,
                            surfacemember_identifier))

                multicurve =BNode()
                if buildingpart.find('lod2TerrainIntersection'):
                    multiCurve = buildingpart.lod2TerrainIntersection.MultiCurve

                    graph.add((buildingpart_node, bldg.lod2TerrainIntersection,
                        multicurve))

                    curvemembers = multiCurve.find_all('curveMember')
                    for curvemember in curvemembers:
                        linestring = BNode()
                        graph.add((multicurve,gml.curveMember,linestring))

                        poslist = curvemember.LineString.posList.text
                        graph.add((linestring,gml.posList,rdflib.Literal(poslist)))

                boundbys = buildingpart.find_all('boundedBy')
                for boundby in boundbys:
                    surface = list(boundby.children)[1]

                    surfaceType = surface.name

                    surface_id = surface['gml:id']
                    surface_node = rdflib.URIRef('mycitygml.tue.nl/city/building/
                        buildingpart/boundedBy/'+surfaceType+'' + surface_id)
                    graph.add((buildingpart_node,bldg.boundedBy,surface_node))

                    creationdate = surface.creationDate.text
                    graph.add((surface_node,core.creationdate,rdflib.Literal(
                        creationdate)))

                    multisurface_id = surface.lod2MultiSurface.MultiSurface['gml:id
                        ']
                    multisurface_node =  rdflib.URIRef('mycitygml.tue.nl/city/
                        building/buildingpart/boundedBy/'+surfaceType+'/
                        MultiSurface' + multisurface_id)
                    graph.add((surface_node,bldg.lod2MultiSurface,multisurface_node
                        ))


                    surfaceMember = surface.lod2MultiSurface.MultiSurface.
                        surfaceMember
                    polygon = surfaceMember.Polygon
                    polygon_id = polygon['gml:id']
                    polygon_node = rdflib.URIRef('mycitygml.tue.nl/city/building/
                        boundedBy/'+surfaceType+'/MultiSurface/Polygon'+polygon_id)
                    graph.add((multisurface_node,gml.surfaceMember,polygon_node))

                    linearring = surfaceMember.Polygon.exterior.LinearRing
                    linearring_id = linearring['gml:id']
                    linearring_node = rdflib.URIRef('mycitygml.tue.nl/city/building
                        //buildingpart/boundedBy/'+surfaceType+'/MultiSurface/
                        Polygon/LinearRing'+linearring_id)
                    graph.add((polygon_node,gml.exterior,linearring_node))

                    poslist = linearring.posList.text
                    graph.add((linearring_node,gml.posList,rdflib.Literal(poslist))
                        )


        else:
        #lod2
            building_rooftype = building.roofType.text
            graph.add((building_node,bldg.roofType,rdflib.Literal(building_rooftype
                )))

            measuredHeight = building.measuredHeight.text
            graph.add((building_node,bldg.measuredHeight,rdflib.Literal(
                measuredHeight)))

            solid = building.lod2Solid.Solid
            solid_id = solid['gml:id']
            solid_node = rdflib.URIRef('mycitygml.tue.nl/city/building/solid/' +
                solid_id)

            graph.add((building_node,bldg.lod2Solid,solid_node))



            compositesurface = solid.exterior.CompositeSurface
            compositesurface_id = solid.exterior.CompositeSurface['gml:id']
            compositesurface_node = rdflib.URIRef('mycitygml.tue.nl/city/building/
                solid/compositesurface/' + compositesurface_id)

            graph.add((solid_node,gml.exterior,compositesurface_node))


            surfacemembers = compositesurface.find_all('surfaceMember')
            for surfacemember in surfacemembers:
                surfacemember_xlink = surfacemember['xlink:href']
                surfacemember_identifier = rdflib.URIRef('mycitygml.tue.nl/city/
                    building/solid/compositesurface/surfaceMember' +
                    surfacemember_xlink)
                graph.add ((compositesurface_node,gml.surfaceMember,
                    surfacemember_identifier))

            multicurve =BNode()
            multiCurve = building.lod2TerrainIntersection.MultiCurve

            graph.add((building_node, bldg.lod2TerrainIntersection, multicurve))

            curvemembers = multiCurve.find_all('curveMember')
            for curvemember in curvemembers:
                linestring = BNode()
                graph.add((multicurve,gml.curveMember,linestring))

                poslist = curvemember.LineString.posList.text
                graph.add((linestring,gml.posList,rdflib.Literal(poslist)))

            boundbys = building.find_all('boundedBy')
            for boundby in boundbys:

                surface = list(boundby.children)[1]

                surfaceType = surface.name

                surface_id = surface['gml:id']
                surface_node = rdflib.URIRef('mycitygml.tue.nl/city/building/boundedBy/'+surfaceType+'' + surface_id)
                graph.add((building_node,bldg.boundedBy,surface_node))

                creationdate = surface.creationDate.text
                graph.add((surface_node,core.creationdate,rdflib.Literal(creationdate)))

                multisurface_id = surface.lod2MultiSurface.MultiSurface['gml:id']
                multisurface_node =  rdflib.URIRef('mycitygml.tue.nl/city/building/boundedBy/'+surfaceType+'/MultiSurface' + multisurface_id)
                graph.add((surface_node,bldg.lod2MultiSurface,multisurface_node))


                surfaceMember = surface.lod2MultiSurface.MultiSurface.surfaceMember
                polygon = surfaceMember.Polygon
                polygon_id = polygon['gml:id']
                polygon_node = rdflib.URIRef('mycitygml.tue.nl/city/building/boundedBy/'+surfaceType+'/MultiSurface/Polygon'+polygon_id)
                graph.add((multisurface_node,gml.surfaceMember,polygon_node))

                linearring = surfaceMember.Polygon.exterior.LinearRing
                linearring_id = linearring['gml:id']
                linearring_node = rdflib.URIRef('mycitygml.tue.nl/city/building/boundedBy/'+surfaceType+'/MultiSurface/Polygon/LinearRing'+linearring_id)
                graph.add((polygon_node,gml.exterior ,linearring_node))

                poslist = linearring.posList.text
                graph.add((linearring_node,gml.posList,rdflib.Literal(poslist)))





graph.serialize(destination='/Users/pc/Desktop/citygml2rdfnew.ttl', format='turtle')









pass
```

Appendix H

Python code for converting OKSTRA Data to RDF

```
’’’

@author: JerryZheng
’’’
from bs4 import BeautifulSoup

import rdflib
from pip.cmdoptions import allow_all_external
from rdflib.term import BNode
from numpy import poly
from builtins import str
from rdflib.plugins.sparql.operators import string

graph = rdflib.Graph(store=’IOMemory’, identifier=’okstraGraph’)

gml = rdflib.Namespace(”http://www.opengis.net/gml:”)
okstra = rdflib.Namespace(”http://schema.okstra.de/2016/okstra:”)

graph.bind(’gml’, gml)
graph.bind(’okstra’,okstra)

bsoup = BeautifulSoup(open(”/Users/pc/Desktop/finalokstra.xml”), ’lxml−xml’)

roadobject = bsoup.FeatureCollection

FeatureMembers = roadobject.find_all(’featureMember’)

#Convert Abschnitt
for FeatureMember in FeatureMembers:
    if FeatureMember.find(’Abschnitt’):
        Abschnitt = FeatureMember.Abschnitt
        Abschnitt_id = Abschnitt[’gml:id’]
        Abschnitt_node = rdflib.URIRef(’myokstra.tue.nl/Abschnitt/#’ + Abschnitt_id
            )
        graph.add((Abschnitt_node, rdflib.RDF.type, rdflib.URIRef(’http://schema.
            okstra.de/2016/Abschnitt’)))
        Name = list(Abschnitt.find_all(’gml:name’))[0].text
        graph.add((Abschnitt_node, gml.name, rdflib.Literal(Name)))
        zu_strasse_xlink = Abschnitt.zu_Strasse[’xlink:href’]
        zu_strasse_identifier = rdflib.URIRef(’myokstra.tue.nl/Strasse/’ +
            zu_strasse_xlink)


        graph.add((Abschnitt_node, okstra.zu_Strasse, zu_strasse_identifier))
```

```
Liniengeometrie = BNode()
graph.add((Abschnitt_node, okstra.Liniengeometrie, Liniengeometrie))

curve = Abschnitt.Liniengeometrie.Curve
curve_id = curve['gml:id']
curve_node = rdflib.URIRef('myokstra.tue.nl/Abschnitt/#' +curve_id)
graph.add((Liniengeometrie, gml.Curve, curve_node))

segment =BNode()
graph.add((curve_node, gml.segments, segment))

linestringsegment = BNode()
graph.add((segment, gml.LineStringSegment, linestringsegment))

coordinate = curve.segments.LineStringSegment.coordinates.text
graph.add((linestringsegment, gml.coordinates, rdflib.Literal(coordinate)))

laenge = Abschnitt.Laenge.text
graph.add((Abschnitt_node, okstra.Laenge,  rdflib.Literal(laenge)))

betriebsmerkmal =  Abschnitt.Betriebsmerkmal.text
graph.add((Abschnitt_node, okstra.Betriebsmerkmal, rdflib.Literal(
    betriebsmerkmal)))

abschnitts_Astnummer = Abschnitt.Abschnitts_Astnummer.text
graph.add((Abschnitt_node, okstra.Abschnitts_Astnummer, rdflib.Literal(
    abschnitts_Astnummer)))

abschnitts_Astbezeichnung = Abschnitt.Abschnitts_Astbezeichnung.text
graph.add((Abschnitt_node, okstra.Abschnitts_Astbezeichnung, rdflib.Literal(
    abschnitts_Astbezeichnung)))

nummer_gehoert_zu_Strasse = Abschnitt.Nummer_gehoert_zu_Strasse
nummer_gehoert_zu_Strasse_class = nummer_gehoert_zu_Strasse['Objektklasse']
nummer_gehoert_zu_Strasse_xlink = nummer_gehoert_zu_Strasse['xlink:href']
nummer_gehoert_zu_Strasse_identifier = rdflib.URIRef('myokstra.tue.nl/' +
    nummer_gehoert_zu_Strasse_class +'/'+ nummer_gehoert_zu_Strasse_xlink)
graph.add((Abschnitt_node, okstra.Nummer_gehoert_zu_Strasse,
    nummer_gehoert_zu_Strasse_identifier))

Endet_null = Abschnitt.endet_bei_NP
Endet_null_class = Endet_null['Objektklasse']
Endet_null_xlink = Endet_null['xlink:href']
Endet_null_identifier = rdflib.URIRef('myokstra.tue.nl/' + Endet_null_class
     + '/' + Endet_null_xlink)
graph.add((Abschnitt_node, okstra.endet_bei_NP, Endet_null_identifier))

Beginnt_null = Abschnitt.beginnt_bei_NP
Beginnt_null_class = Beginnt_null['Objektklasse']
Beginnt_null_xlink = Beginnt_null['xlink:href']
Beginnt_null_identifier = rdflib.URIRef('myokstra.tue.nl/' +
    Beginnt_null_class + '/' + Beginnt_null_xlink)
graph.add((Abschnitt_node, okstra.endet_bei_NP, Beginnt_null_identifier))


seitenarm = Abschnitt.Seitenarm.text
graph.add((Abschnitt_node, okstra.Seitenarm, rdflib.Literal(seitenarm)))

getrennt_Fahrbahn = Abschnitt.getrennt_verlaufende_Fahrbahn.text
graph.add((Abschnitt_node, okstra.getrennt_verlaufende_Fahrbahn, rdflib.
    Literal(getrennt_Fahrbahn)))

abschnittsfolgenummer = Abschnitt.Abschnittsfolgenummer.text
graph.add((Abschnitt_node, okstra.Abschnittsfolgenummer, rdflib.Literal(
    abschnittsfolgenummer)))
```

```
#Convert Ast
if FeatureMember.find('Ast'):
    Ast = FeatureMember.Ast
    Ast_id = Ast['gml:id']
    Ast_node = rdflib.URIRef('myokstra.tue.nl/Ast/#' + Ast_id)
    graph.add((Ast_node, rdflib.RDF.type, rdflib.URIRef('http://schema.okstra.
        de/2016/Ast')))
    Name = list(Ast.find_all('gml:name'))[0].text
    graph.add((Ast_node, gml.name, rdflib.Literal(Name)))
    zu_Strasse = Ast.zu_Strasse['xlink:href']
    zu_Strasse_identifier = rdflib.URIRef('myokstra.tue.nl/Strasse/' +
        zu_Strasse)

    graph.add((Ast_node,okstra.zu_Strasse,  zu_Strasse_identifier))

    Liniengeometrie = BNode()
    graph.add((Ast_node,okstra.Liniengeometrie,Liniengeometrie))

    curve = Ast.Liniengeometrie.Curve
    curve_id = curve['gml:id']
    curve_node = rdflib.URIRef('myokstra.tue.nl/Ast/#' +curve_id)
    graph.add((Liniengeometrie,gml.Curve,curve_node))

    segment =BNode()
    graph.add((curve_node,gml.segments,segment))

    linestringsegment = BNode()
    graph.add((segment,gml.LineStringSegment,linestringsegment))

    coordinate = curve.segments.LineStringSegment.coordinates.text
    graph.add((linestringsegment,gml.coordinates,rdflib.Literal(coordinate)))

    laenge = Ast.Laenge.text
    graph.add((Ast_node,okstra.Laenge, rdflib.Literal(laenge)))

    betriebsmerkmal =  Ast.Betriebsmerkmal.text
    graph.add((Ast_node,okstra.Betriebsmerkmal,rdflib.Literal(betriebsmerkmal))
        )

    abschnitts_Astnummer = Ast.Abschnitts_Astnummer.text
    graph.add((Ast_node,okstra.Abschnitts_Astnummer,rdflib.Literal(
        abschnitts_Astnummer)))

    abschnitts_Astbezeichnung = Ast.Abschnitts_Astbezeichnung.text
    graph.add((Ast_node,okstra.Abschnitts_Astbezeichnung,rdflib.Literal(
        abschnitts_Astbezeichnung)))

    nummer_gehoert_zu_Strasse = Ast.Nummer_gehoert_zu_Strasse
    nummer_gehoert_zu_Strasse_class = nummer_gehoert_zu_Strasse['Objektklasse']
    nummer_gehoert_zu_Strasse_xlink = nummer_gehoert_zu_Strasse['xlink:href']
    nummer_gehoert_zu_Strasse_identifier = rdflib.URIRef('myokstra.tue.nl/' +
        nummer_gehoert_zu_Strasse_class + '/' + nummer_gehoert_zu_Strasse_xlink
        )
    graph.add((Ast_node,okstra.nummer_gehoert_zu_Strasse,
        nummer_gehoert_zu_Strasse_identifier))

    endet_null = Ast.endet_bei_NP
    endet_null_class = endet_null['Objektklasse']
    endet_null_xlink = endet_null['xlink:href']
    endet_null_identifier = rdflib.URIRef('myokstra.tue.nl/' + endet_null_class
         + '/' + endet_null_xlink)
    graph.add((Ast_node,okstra.endet_bei_NP,endet_null_identifier))

    beginnt_null = Ast.beginnt_bei_NP
    beginnt_null_class = beginnt_null['Objektklasse']
    beginnt_null_xlink = beginnt_null['xlink:href']
```

```
        beginnt_null_identifier = rdflib.URIRef('myokstra.tue.nl/' +
            beginnt_null_class + '/' + beginnt_null_xlink)
        graph.add((Ast_node,okstra.beginnt_bei_NP,rdflib.Literal(beginnt_null_class
            )))
        graph.add((Ast_node,okstra.beginnt_bei_NP,beginnt_null_identifier))




    #Convert Netzknoten
    if FeatureMember.find('Netzknoten'):
        Netzknoten = FeatureMember.Netzknoten
        Netzknoten_id = Netzknoten['gml:id']
        Netzknoten_node = rdflib.URIRef('myokstra.tue.nl/Netzknoten/#' +
            Netzknoten_id)

        graph.add((Netzknoten_node, rdflib.RDF.type, rdflib.URIRef('http://schema.
            okstra.de/2016/Netzknoten')))

        Name = list(Netzknoten.find_all('gml:name'))[0].text
        graph.add((Netzknoten_node,gml.name, rdflib.Literal(Name)))

        punktgeometrie = BNode()
        graph.add((Netzknoten_node,okstra.Punktgeometrie,punktgeometrie))

        point = Netzknoten.Punktgeometrie.Point
        point_id = point['gml:id']
        point_node = rdflib.URIRef('myokstra.tue.nl/Netzknoten/#' + point_id)
        graph.add((punktgeometrie,gml.Point,point_node))

        Pos = point.pos.text
        graph.add((point_node,gml.pos,rdflib.Literal(Pos)))

        numerierungsbezirk = Netzknoten.Numerierungsbezirk.text
        graph.add((Netzknoten_node,okstra.Numerierungsbezirk,rdflib.Literal(
            numerierungsbezirk)))

        nummer = Netzknoten.Nummer.text
        graph.add((Netzknoten_node,okstra.Nummer,rdflib.Literal(nummer)))

        knotenart = Netzknoten.Knotenart.text
        graph.add((Netzknoten_node,okstra.Knotenart,rdflib.Literal(knotenart)))

        Hat_Nullpunkts = Netzknoten.find_all('hat_Nullpunkt')
        for Hat_Nullpunkt in Hat_Nullpunkts:
            Hat_Nullpunkt_xlink = Hat_Nullpunkt['xlink:href']
            Hat_Nullpunkt_identifier = rdflib.URIRef('myokstra.tue.nl/Nullpunkt/' +
                 Hat_Nullpunkt_xlink)
            graph.add((Netzknoten_node,okstra.hat_Nullpunkt,
                Hat_Nullpunkt_identifier))

    #Convert Nullpunkt
    if FeatureMember.find('Nullpunkt'):
        Nullpunkt = FeatureMember.Nullpunkt
        Nullpunkt_id = Nullpunkt['gml:id']
        Nullpunkt_node = rdflib.URIRef('myokstra.tue.nl/Nullpunkt/#' + Nullpunkt_id
            )

        graph.add((Nullpunkt_node, rdflib.RDF.type, rdflib.URIRef('http://schema.
            okstra.de/2016/Nullpunkt')))

        Name = list(Nullpunkt.find_all('gml:name'))[0].text
        graph.add((Nullpunkt_node, gml.name, rdflib.Literal(Name)))

        punktgeometrie = BNode()
        graph.add((Nullpunkt_node,okstra.Punktgeometrie,punktgeometrie))
        point = Nullpunkt.Punktgeometrie.Point
        point_id = point['gml:id']
        point_node = rdflib.URIRef('myokstra.tue.nl/Nullpunkt/#' + point_id)
        graph.add((punktgeometrie,gml.Point,point_node))

        Pos = point.pos.text
        graph.add((point_node,gml.pos,rdflib.Literal(Pos)))

        Zusatz = Nullpunkt.Zusatz.text
        graph.add((Nullpunkt_node,okstra.Zusatz,rdflib.Literal(Zusatz)))

        Nullpunktart = Nullpunkt.Nullpunktart.text
        graph.add((Nullpunkt_node,okstra.Nullpunktart,rdflib.Literal(Nullpunktart))
            )

        Beginn_von_Abschnitt_oder_Asts = Nullpunkt.find_all('
            Beginn_von_Abschnitt_oder_Ast')
        for Beginn_von_Abschnitt_oder_Ast in Beginn_von_Abschnitt_oder_Asts:
            Beginn_von_Abschnitt_oder_Ast_class = Beginn_von_Abschnitt_oder_Ast['
                Objektklasse']
            Beginn_von_Abschnitt_oder_Ast_xlink = Beginn_von_Abschnitt_oder_Ast['
                xlink:href']
            Beginn_von_Abschnitt_oder_Ast_identifier = rdflib.URIRef('myokstra.tue.
                nl/'+ Beginn_von_Abschnitt_oder_Ast_class +'/'+
                Beginn_von_Abschnitt_oder_Ast_xlink)
            graph.add((Nullpunkt_node,okstra.Beginn_von_Abschnitt_oder_Ast,
                Beginn_von_Abschnitt_oder_Ast_identifier))

        Ende_von_Abschnitt_oder_Asts = Nullpunkt.find_all('
            Ende_von_Abschnitt_oder_Ast')
        for Ende_von_Abschnitt_oder_Ast in Ende_von_Abschnitt_oder_Asts:
            Ende_von_Abschnitt_oder_Ast_class = Ende_von_Abschnitt_oder_Ast['
                Objektklasse']
            Ende_von_Abschnitt_oder_Ast_xlink = Ende_von_Abschnitt_oder_Ast['xlink:
                href']
            Ende_von_Abschnitt_oder_Ast_identifier = rdflib.URIRef('myokstra.tue.nl
                /'+ Ende_von_Abschnitt_oder_Ast_class +'/'+
                Ende_von_Abschnitt_oder_Ast_xlink)
            graph.add((Nullpunkt_node,okstra.Ende_von_Abschnitt_oder_Ast,
                Beginn_von_Abschnitt_oder_Ast_identifier))

        in_Netzknoten = Nullpunkt.in_Netzknoten
        in_Netzknoten_xlink = in_Netzknoten['xlink:href']
        in_Netzknoten_identier = rdflib.URIRef('myokstra.tue.nl/Netzknoten/' +
            in_Netzknoten_xlink)
        graph.add((Nullpunkt_node,okstra.in_Netzknoten,in_Netzknoten_identier))

        hat_Nullpunktorts = Nullpunkt.find_all('hat_Nullpunktort')
        for hat_Nullpunktort in hat_Nullpunktorts:
            hat_Nullpunktort_xlink = hat_Nullpunktort['xlink:href']
            hat_Nullpunktort_identifier = rdflib.URIRef('myokstra.tue.nl/
                Nullpunktort/' + hat_Nullpunktort_xlink)
            graph.add((Nullpunkt_node,okstra.hat_Nullpunktort,
                hat_Nullpunktort_identifier))

    #Convert Nullpunktort
    if FeatureMember.find('Nullpunktort'):
        Nullpunktort = FeatureMember.Nullpunktort
        Nullpunktort_id = Nullpunktort['gml:id']
        Nullpunktort_node = rdflib.URIRef('myokstra.tue.nl/Nullpunktort/#' +
            Nullpunktort_id)

        graph.add((Nullpunktort_node, rdflib.RDF.type, rdflib.URIRef('http://schema
            .okstra.de/2016/Nullpunktort')))

        bei_Strassenpunkt = BNode()
        graph.add((Nullpunktort_node, okstra.bei_Strassenpunkt, bei_Strassenpunkt))
        Strassenpunkt = BNode()
        graph.add((bei_Strassenpunkt,okstra.Strassenpunkt,Strassenpunkt))

        Station = Nullpunktort.bei_Strassenpunkt.Strassenpunkt.Station.text
        Auf_Abschnitt_oder_Ast =  Nullpunktort.bei_Strassenpunkt.Strassenpunkt.
            auf_Abschnitt_oder_Ast
        Auf_Abschnitt_oder_Ast_class = Auf_Abschnitt_oder_Ast['Objektklasse']
        Auf_Abschnitt_oder_Ast_xlink = Auf_Abschnitt_oder_Ast['xlink:href']
        Auf_Abschnitt_oder_Ast_identifier = rdflib.URIRef('myokstra.tue.nl/'+
            Auf_Abschnitt_oder_Ast_class +'/'+Auf_Abschnitt_oder_Ast_xlink)
        graph.add((Strassenpunkt,okstra.Station,rdflib.Literal(Station)))
        graph.add((Strassenpunkt,okstra.auf_Abschnitt_oder_Ast_class,
            Auf_Abschnitt_oder_Ast_identifier))

        Bei_Nullpunkt = Nullpunktort.bei_Nullpunkt
        Bei_Nullpunkt_xlink = Bei_Nullpunkt['xlink:href']
        Bei_Nullpunkt_identifier = rdflib.URIRef('myokstra.tue.nl/Nullpunkt/' +
            Bei_Nullpunkt_xlink)
        graph.add((Nullpunktort_node,okstra.bei_Nullpunkt,Bei_Nullpunkt_identifier)
            )

    #Convert Strasse
    if FeatureMember.find('Strasse'):
        Strasse = FeatureMember.Strasse
        Strasse_id = Strasse['gml:id']

        Strasse_node = rdflib.URIRef('myokstra.tue.nl/Strasse/#' + Strasse_id)

        graph.add((Strasse_node,rdflib.RDF.type,rdflib.URIRef('http://schema.okstra
            .de/2016/Strasse')))

        Name = list(Strasse.find_all('gml:name'))[0].text

        graph.add((Strasse_node, gml.name, rdflib.Literal(Name)))

        strassenbezeichnung = Strasse.hat_Strassenbezeichnung.Strassenbezeichnung
        Strassenbezeichnung = BNode()
        graph.add((Strasse_node,okstra.hat_Strassenbezeichnung,Strassenbezeichnung)
            )

        strassenklasse = strassenbezeichnung.Strassenklasse.text
        strassennummer = strassenbezeichnung.Strassennummer.text

        graph.add((Strassenbezeichnung,okstra.Strassenklasse,rdflib.Literal(
            strassenklasse)))
        graph.add((Strassenbezeichnung,okstra.Strassennummer,rdflib.Literal(
            strassennummer)))

        if strassenbezeichnung.find('Zusatzbuchstabe'):
            zusatzbuchstabe = strassenbezeichnung.Zusatzbuchstabe.text
            graph.add((Strassenbezeichnung,okstra.Zusatzbuchstabe,rdflib.Literal(
                zusatzbuchstabe)))

        hat_AoA_zugeordnets = Strasse.find_all('hat_AoA_zugeordnet')
        for hat_AoA_zugeordnet in hat_AoA_zugeordnets:
            hat_AoA_zugeordnet_class = hat_AoA_zugeordnet['Objektklasse']
            hat_AoA_zugeordnet_xlink = hat_AoA_zugeordnet['xlink:href']
            hat_AoA_zugeordnet_identifier = rdflib.URIRef('myokstra.tue.nl/'+
                hat_AoA_zugeordnet_class +'/'+hat_AoA_zugeordnet_xlink)
            graph.add((Strasse_node,okstra.hat_AoA_zugeordnet,
                hat_AoA_zugeordnet_identifier))

        hat_Strassenbezugsobjekts = Strasse.find_all('hat_Strassenbezugsobjekt')
        for hat_Strassenbezugsobjekt in hat_Strassenbezugsobjekts:
            hat_Strassenbezugsobjekt_class = hat_Strassenbezugsobjekt['Objektklasse
                ']
            hat_Strassenbezugsobjekt_xlink = hat_Strassenbezugsobjekt['xlink:href']
            hat_Strassenbezugsobjekt_identifier = rdflib.URIRef('myokstra.tue.nl/'+
                 hat_Strassenbezugsobjekt_class +'/'+hat_Strassenbezugsobjekt_xlink
                )
            graph.add((Strasse_node,okstra.hat_Strassenbezugsobjekt,
                hat_Strassenbezugsobjekt_identifier))

    #Convert Anzahl Fahrstrifen:
    if FeatureMember.find('Anzahl_Fahrstreifen'):
        Anzahl_Fahrstreifen = FeatureMember.Anzahl_Fahrstreifen
        Anzahl_Fahrstreifen_id = Anzahl_Fahrstreifen['gml:id']

        Anzahl_Fahrstreifen_node = rdflib.URIRef('myokstra.tue.nl/
            Anzahl_Fahrstreifen/#' + Anzahl_Fahrstreifen_id)

        graph.add((Anzahl_Fahrstreifen_node,rdflib.RDF.type,rdflib.URIRef('http://
            schema.okstra.de/2016/Anzahl_Fahrstreifen')))

        hat_Strecke = Anzahl_Fahrstreifen.hat_Strecke
        hat_Strecke_class = hat_Strecke['Objektklasse']
        hat_Strecke_xlink = hat_Strecke['xlink:href']
        hat_Strecke_identifier = rdflib.URIRef('myokstra.tue.nl/'+
            hat_Strecke_class +'/'+hat_Strecke_xlink)
        graph.add((Anzahl_Fahrstreifen_node,okstra.hat_Strecke,
            hat_Strecke_identifier))

        Fahrstreifen_Gegenrichtung = Anzahl_Fahrstreifen.Fahrstreifen_Gegenrichtung
            .text
        graph.add((Anzahl_Fahrstreifen_node,okstra.Fahrstreifen_Gegenrichtung,
            rdflib.Literal(Fahrstreifen_Gegenrichtung)))

        Fahrstreifen_Richtung = Anzahl_Fahrstreifen.Fahrstreifen_Richtung.text
        graph.add((Anzahl_Fahrstreifen_node,okstra.Fahrstreifen_Richtung,rdflib.
            Literal(Fahrstreifen_Richtung)))

        Fahrstreifen_beide_Richtungen = Anzahl_Fahrstreifen.
            Fahrstreifen_beide_Richtungen.text
        graph.add((Anzahl_Fahrstreifen_node,okstra.Fahrstreifen_beide_Richtungen,
            rdflib.Literal(Fahrstreifen_beide_Richtungen)))


    #Convert Bahnigkeit:
    if FeatureMember.find('Bahnigkeit'):
        Bahnigkeit = FeatureMember.Bahnigkeit
        Bahnigkeit_id = Bahnigkeit['gml:id']

        Bahnigkeit_node = rdflib.URIRef('myokstra.tue.nl/Bahnigkeit/#' +
            Bahnigkeit_id)

        graph.add((Bahnigkeit_node,rdflib.RDF.type,rdflib.URIRef('http://schema.
            okstra.de/2016/Bahnigkeit')))

        hat_strecke = Bahnigkeit.hat_Strecke
        hat_strecke_class = hat_strecke['Objektklasse']
        hat_strecke_xlink = hat_strecke['xlink:href']
        hat_strecke_identifier = rdflib.URIRef('myokstra.tue.nl/'+
            hat_strecke_class +'/'+hat_strecke_xlink)
        graph.add((Bahnigkeit_node,okstra.hat_Strecke,hat_strecke_identifier))

        Kennzeichen_Bahnigkeit = Bahnigkeit.Kennzeichen_Bahnigkeit.text
        graph.add((Bahnigkeit_node,okstra.Kennzeichen_Bahnigkeit,rdflib.Literal(
            Kennzeichen_Bahnigkeit)))

    #Convert Strecke:
    if FeatureMember.find('Strecke'):
        Strecke = FeatureMember.Strecke
        Strecke_id = Strecke['gml:id']
        Strecke_node = rdflib.URIRef('myokstra.tue.nl/Strecke/#' + Strecke_id)

        graph.add((Strecke_node,rdflib.RDF.type,rdflib.URIRef('http://schema.okstra
            .de/2016/Strecke')))

        zu_Streckenobjekt = Strecke.zu_Streckenobjekt
        zu_Streckenobjekt_class = zu_Streckenobjekt['Objektklasse']
        zu_Streckenobjekt_xlink = zu_Streckenobjekt['xlink:href']
        zu_Streckenobjekt_identifier = rdflib.URIRef('myokstra.tue.nl/'+
            zu_Streckenobjekt_class +'/'+zu_Streckenobjekt_xlink)
        graph.add((Strecke_node,okstra.zu_Streckenobjekt,
            zu_Streckenobjekt_identifier))

        Textfeld = Strecke.Textfeld.text
        graph.add((Strecke_node,okstra.Textfeld,rdflib.Literal(Textfeld)))

        entlang_Streckenkomponente = Strecke.entlang_Streckenkomponente
        entlang_Streckenkomponente_class = entlang_Streckenkomponente['Objektklasse
            ']
        entlang_Streckenkomponente_xlink = entlang_Streckenkomponente['xlink:href']
        entlang_Streckenkomponente_identifier = rdflib.URIRef('myokstra.tue.nl/'+
            entlang_Streckenkomponente_class +'/'+entlang_Streckenkomponente_xlink)
        graph.add((Strecke_node,okstra.entlang_Streckenkomponente,
            entlang_Streckenkomponente_identifier))

    #Convert Querschnittstreifen:
    if FeatureMember.find('Querschnittstreifen'):
        Querschnittstreifen = FeatureMember.Querschnittstreifen
        Querschnittstreifen_id = Querschnittstreifen['gml:id']
        Querschnittstreifen_node = rdflib.URIRef('myokstra.tue.nl/
            Querschnittstreifen/#' + Querschnittstreifen_id)
        graph.add((Querschnittstreifen_node,rdflib.RDF.type,rdflib.URIRef('http://
            schema.okstra.de/2016/Querschnittstreifen')))

        Hat_strecke = Querschnittstreifen.hat_Strecke
        Hat_strecke_class = Hat_strecke['Objektklasse']
        Hat_strecke_xlink = Hat_strecke['xlink:href']
        Hat_strecke_identifier = rdflib.URIRef('myokstra.tue.nl/'+
            Hat_strecke_class +'/'+Hat_strecke_xlink)
        graph.add((Querschnittstreifen_node,okstra.hat_Strecke,
            Hat_strecke_identifier))

        x_Wert_von_Station_links = Querschnittstreifen.x_Wert_von_Station_links
        x_Wert_von_Station_links_uom = x_Wert_von_Station_links['uom']
        x_Wert_von_Station_links_number = x_Wert_von_Station_links.text
        x_Wert_von_Station_links_value = x_Wert_von_Station_links_number +
            x_Wert_von_Station_links_uom
        graph.add((Querschnittstreifen_node,okstra.x_Wert_von_Station_links,rdflib.
            Literal(x_Wert_von_Station_links_value)))

        x_Wert_von_Station_rechts = Querschnittstreifen.x_Wert_von_Station_rechts
        x_Wert_von_Station_rechts_uom = x_Wert_von_Station_rechts['uom']
        x_Wert_von_Station_rechts_number = x_Wert_von_Station_rechts.text
        x_Wert_von_Station_rechts_value = x_Wert_von_Station_rechts_number +
            x_Wert_von_Station_rechts_uom
        graph.add((Querschnittstreifen_node,okstra.x_Wert_von_Station_rechts,rdflib
            .Literal(x_Wert_von_Station_rechts_value)))

        x_Wert_bis_Station_links = Querschnittstreifen.x_Wert_bis_Station_links
        x_Wert_bis_Station_links_uom = x_Wert_bis_Station_links['uom']
        x_Wert_bis_Station_links_number = x_Wert_bis_Station_links.text
        x_Wert_bis_Station_links_value = x_Wert_bis_Station_links_number +
            x_Wert_bis_Station_links_uom
        graph.add((Querschnittstreifen_node,okstra.x_Wert_bis_Station_links,rdflib.
            Literal(x_Wert_bis_Station_links_value)))

        x_Wert_bis_Station_rechts = Querschnittstreifen.x_Wert_bis_Station_rechts
        x_Wert_bis_Station_rechts_uom = x_Wert_bis_Station_rechts['uom']
        x_Wert_bis_Station_rechts_number = x_Wert_bis_Station_rechts.text
        x_Wert_bis_Station_rechts_value = x_Wert_bis_Station_rechts_number +
            x_Wert_bis_Station_rechts_uom
        graph.add((Querschnittstreifen_node,okstra.x_Wert_bis_Station_rechts,rdflib
            .Literal(x_Wert_bis_Station_rechts_value)))

        mittlere_Breite = Querschnittstreifen.mittlere_Breite
        mittlere_Breite_uom = mittlere_Breite['uom']
        mittlere_Breite_number = mittlere_Breite.text
        mittlere_Breite_value = mittlere_Breite_number + mittlere_Breite_uom
        graph.add((Querschnittstreifen_node,okstra.mittlere_Breite,rdflib.Literal(
            mittlere_Breite_value)))

        Streifenart = Querschnittstreifen.Streifenart.text
        graph.add((Querschnittstreifen_node,okstra.Streifenart,rdflib.Literal(
            Streifenart)))

        unscharfe_Breite = Querschnittstreifen.unscharfe_Breite.text
        graph.add((Querschnittstreifen_node,okstra.unscharfe_Breite,rdflib.Literal(
            unscharfe_Breite)))

        tatsaechliche_Laenge = Querschnittstreifen.tatsaechliche_Laenge
        tatsaechliche_Laenge_uom = tatsaechliche_Laenge['uom']
        tatsaechliche_Laenge_number = tatsaechliche_Laenge.text
        tatsaechliche_Laenge_value = tatsaechliche_Laenge_number +
            tatsaechliche_Laenge_uom
        graph.add((Querschnittstreifen_node,okstra.tatsaechliche_Laenge,rdflib.
            Literal(tatsaechliche_Laenge_value)))


        tatsaechliche_Flaeche = Querschnittstreifen.tatsaechliche_Flaeche
        tatsaechliche_Flaeche_uom = tatsaechliche_Flaeche['uom']
        tatsaechliche_Flaeche_number = tatsaechliche_Flaeche.text
        tatsaechliche_Flaeche_value = tatsaechliche_Flaeche_number +
            tatsaechliche_Flaeche_uom
        graph.add((Querschnittstreifen_node,okstra.tatsaechliche_Flaeche,rdflib.
            Literal(tatsaechliche_Flaeche_value)))

    #Convert Teilabschnitt:
    if FeatureMember.find('Teilabschnitt'):
        Teilabschnitt = FeatureMember.Teilabschnitt
        Teilabschnitt_id = Teilabschnitt['gml:id']
        Teilabschnitt_node = rdflib.URIRef('myokstra.tue.nl/Teilabschnitt/#' +
            Teilabschnitt_id)
        graph.add((Teilabschnitt_node,rdflib.RDF.type,rdflib.URIRef('http://schema.
            okstra.de/2016/Teilabschnitt')))

        if Teilabschnitt.find('in_Strecke'):
            in_Strecke = Teilabschnitt.in_Strecke
            in_Strecke_class = in_Strecke['Objektklasse']
            in_Strecke_xlink = in_Strecke['xlink:href']
            in_Strecke_identifier = rdflib.URIRef('myokstra.tue.nl/Strecke/'+
                in_Strecke_xlink)
            graph.add((Teilabschnitt_node,okstra.in_Strecke,in_Strecke_identifier))

            beginnt_bei_SP = BNode()
            graph.add((Teilabschnitt_node, okstra.beginnt_bei_SP, beginnt_bei_SP))

            strassenpunkt = BNode()
            graph.add((beginnt_bei_SP,okstra.Strassenpunkt,strassenpunkt))

            station = Teilabschnitt.beginnt_bei_SP.Strassenpunkt.Station.text
            Auf_abschnitt_oder_Ast =  Teilabschnitt.beginnt_bei_SP.Strassenpunkt.
                auf_Abschnitt_oder_Ast
            Auf_abschnitt_oder_Ast_class = Auf_abschnitt_oder_Ast['Objektklasse']
            Auf_abschnitt_oder_Ast_xlink = Auf_abschnitt_oder_Ast['xlink:href']
            Auf_abschnitt_oder_Ast_identifier = rdflib.URIRef('myokstra.tue.nl/'+
                Auf_abschnitt_oder_Ast_class +'/'+Auf_abschnitt_oder_Ast_xlink)
            graph.add((strassenpunkt,okstra.Station,rdflib.Literal(station)))
            graph.add((strassenpunkt,okstra.auf_Abschnitt_oder_Ast,
                Auf_abschnitt_oder_Ast_identifier))

            endet_bei_SP = BNode()
            graph.add((Teilabschnitt_node, okstra.endet_bei_SP, endet_bei_SP))

            StrassenPunkt = BNode()
            graph.add((endet_bei_SP,okstra.StrassenPunkt,StrassenPunkt))

            station = Teilabschnitt.endet_bei_SP.Strassenpunkt.Station.text
            Auf_abschnitt_oder_ast =  Teilabschnitt.endet_bei_SP.Strassenpunkt.
                auf_Abschnitt_oder_Ast
            Auf_abschnitt_oder_ast_class = Auf_abschnitt_oder_ast['Objektklasse']
            Auf_abschnitt_oder_ast_xlink = Auf_abschnitt_oder_ast['xlink:href']
            Auf_abschnitt_oder_ast_identifier = rdflib.URIRef('myokstra.tue.nl/'+
                Auf_abschnitt_oder_ast_class +'/'+Auf_abschnitt_oder_ast_xlink)
            graph.add((StrassenPunkt,okstra.Station,rdflib.Literal(station)))
            graph.add((StrassenPunkt,okstra.auf_Abschnitt_oder_Ast,
                Auf_abschnitt_oder_ast_identifier))

            Auf_Abschnitt_Oder_Ast = Teilabschnitt.auf_Abschnitt_oder_Ast
            Auf_Abschnitt_Oder_Ast_class = Auf_Abschnitt_Oder_Ast['Objektklasse']
            Auf_Abschnitt_Oder_Ast_xlink = Auf_Abschnitt_Oder_Ast['xlink:href']
            Auf_Abschnitt_Oder_Ast_identifier = rdflib.URIRef('myokstra.tue.nl/'+
                Auf_Abschnitt_Oder_Ast_class +'/'+Auf_Abschnitt_Oder_Ast_xlink)

            graph.add((Teilabschnitt_node,okstra.auf_Abschnitt_oder_Ast,
                Auf_Abschnitt_Oder_Ast_identifier))


        if Teilabschnitt.find('in_Netzbereich'):
            in_Netzbereich = Teilabschnitt.in_Netzbereich
            in_Netzbereich_class = in_Netzbereich['Objektklasse']
            in_Netzbereich_xlink = in_Netzbereich['xlink:href']
            in_Netzbereich_identifier = rdflib.URIRef('myokstra.tue.nl/'+
                in_Netzbereich_class+'/'+in_Netzbereich_xlink)
            graph.add((Teilabschnitt_node,okstra.in_Netzbereich,
                in_Netzbereich_identifier))

            beginnt_bei_SP = BNode()
            graph.add((Teilabschnitt_node, okstra.beginnt_bei_SP, beginnt_bei_SP))

            Strassenpunkt = BNode()
            graph.add((beginnt_bei_SP,okstra.Strassenpunkt,Strassenpunkt))

            station = Teilabschnitt.beginnt_bei_SP.Strassenpunkt.Station.text
            Auf_abschnitt_oder_Ast =  Teilabschnitt.beginnt_bei_SP.Strassenpunkt.
                auf_Abschnitt_oder_Ast
            Auf_abschnitt_oder_Ast_class = Auf_abschnitt_oder_Ast['Objektklasse']
            Auf_abschnitt_oder_Ast_xlink = Auf_abschnitt_oder_Ast['xlink:href']
            Auf_abschnitt_oder_Ast_identifier = rdflib.URIRef('myokstra.tue.nl/'+
                Auf_abschnitt_oder_Ast_class +'/'+Auf_abschnitt_oder_Ast_xlink)
            graph.add((Strassenpunkt,okstra.Station,rdflib.Literal(station)))
            graph.add((Strassenpunkt,okstra.auf_Abschnitt_oder_Ast,
                Auf_abschnitt_oder_Ast_identifier))

            endet_bei_SP = BNode()
            graph.add((Teilabschnitt_node, okstra.endet_bei_SP, endet_bei_SP))

            StrassenPunkt = BNode()
            graph.add((endet_bei_SP,okstra.Strassenpunkt,StrassenPunkt))

            station = Teilabschnitt.endet_bei_SP.Strassenpunkt.Station.text
            Auf_abschnitt_oder_ast =  Teilabschnitt.endet_bei_SP.Strassenpunkt.
                auf_Abschnitt_oder_Ast
            Auf_abschnitt_oder_ast_class = Auf_abschnitt_oder_ast['Objektklasse']
            Auf_abschnitt_oder_ast_xlink = Auf_abschnitt_oder_ast['xlink:href']
            Auf_abschnitt_oder_ast_identifier = rdflib.URIRef('myokstra.tue.nl/'+
                Auf_abschnitt_oder_ast_class +'/'+Auf_abschnitt_oder_ast_xlink)
            graph.add((StrassenPunkt,okstra.Station,rdflib.Literal(station)))
            graph.add((StrassenPunkt,okstra.auf_Abschnitt_oder_Ast,
                Auf_abschnitt_oder_ast_identifier))

            Auf_Abschnitt_Oder_Ast = Teilabschnitt.auf_Abschnitt_oder_Ast
            Auf_Abschnitt_Oder_Ast_class = Auf_Abschnitt_Oder_Ast['Objektklasse']
            Auf_Abschnitt_Oder_Ast_xlink = Auf_Abschnitt_Oder_Ast['xlink:href']
            Auf_Abschnitt_Oder_Ast_identifier = rdflib.URIRef('myokstra.tue.nl/'+
                Auf_Abschnitt_Oder_Ast_class +'/'+Auf_Abschnitt_Oder_Ast_xlink)

            graph.add((Teilabschnitt_node,okstra.auf_Abschnitt_oder_Ast,
                Auf_Abschnitt_Oder_Ast_identifier))

        if Teilabschnitt.find('zu_Streckenobjekt'):
            zu_Streckenobjekt = Teilabschnitt.zu_Streckenobjekt
            zu_Streckenobjekt_class = zu_Streckenobjekt['Objektklasse']
            zu_Streckenobjekt_xlink = zu_Streckenobjekt['xlink:href']
            zu_Streckenobjekt_identifier = rdflib.URIRef('myokstra.tue.nl/'+
                zu_Streckenobjekt_class+'/'+zu_Streckenobjekt_xlink)
            graph.add((Teilabschnitt_node,okstra.zu_Streckenobjekt,
                zu_Streckenobjekt_identifier))

            beginnt_bei_SP = BNode()
            graph.add((Teilabschnitt_node, okstra.beginnt_bei_SP, beginnt_bei_SP))

            Strassenpunkt = BNode()
            graph.add((beginnt_bei_SP,okstra.Strassenpunkt,Strassenpunkt))

            station = Teilabschnitt.beginnt_bei_SP.Strassenpunkt.Station.text
            Auf_abschnitt_oder_Ast =  Teilabschnitt.beginnt_bei_SP.Strassenpunkt.
                auf_Abschnitt_oder_Ast
            Auf_abschnitt_oder_Ast_class = Auf_abschnitt_oder_Ast['Objektklasse']
            Auf_abschnitt_oder_Ast_xlink = Auf_abschnitt_oder_Ast['xlink:href']
            Auf_abschnitt_oder_Ast_identifier = rdflib.URIRef('myokstra.tue.nl/'+
                Auf_abschnitt_oder_Ast_class +'/'+Auf_abschnitt_oder_Ast_xlink)
            graph.add((Strassenpunkt,okstra.Station,rdflib.Literal(station)))
            graph.add((Strassenpunkt,okstra.auf_Abschnitt_oder_Ast,
                Auf_abschnitt_oder_Ast_identifier))

            endet_bei_SP = BNode()
            graph.add((Teilabschnitt_node, okstra.endet_bei_SP, endet_bei_SP))

            StrassenPunkt = BNode()
            graph.add((endet_bei_SP,okstra.Strassenpunkt,StrassenPunkt))

            station = Teilabschnitt.endet_bei_SP.Strassenpunkt.Station.text
            Auf_abschnitt_oder_ast =  Teilabschnitt.endet_bei_SP.Strassenpunkt.
                auf_Abschnitt_oder_Ast
            Auf_abschnitt_oder_ast_class = Auf_abschnitt_oder_ast['Objektklasse']
            Auf_abschnitt_oder_ast_xlink = Auf_abschnitt_oder_ast['xlink:href']
            Auf_abschnitt_oder_ast_identifier = rdflib.URIRef('myokstra.tue.nl/'+
                Auf_abschnitt_oder_ast_class +'/'+Auf_abschnitt_oder_ast_xlink)
            graph.add((StrassenPunkt,okstra.Station,rdflib.Literal(station)))
            graph.add((StrassenPunkt,okstra.auf_Abschnitt_oder_Ast,
                Auf_abschnitt_oder_ast_identifier))

            Auf_Abschnitt_Oder_Ast = Teilabschnitt.auf_Abschnitt_oder_Ast
            Auf_Abschnitt_Oder_Ast_class = Auf_Abschnitt_Oder_Ast['Objektklasse']
            Auf_Abschnitt_Oder_Ast_xlink = Auf_Abschnitt_Oder_Ast['xlink:href']
            Auf_Abschnitt_Oder_Ast_identifier = rdflib.URIRef('myokstra.tue.nl/'+
                Auf_Abschnitt_Oder_Ast_class +'/'+Auf_Abschnitt_Oder_Ast_xlink)
                graph.add((Teilabschnitt_node,okstra.auf_Abschnitt_oder_Ast,
                     Auf_Abschnitt_Oder_Ast_identifier))


graph.serialize(destination='/Users/pc/Desktop/okstra2rdfnew.ttl', format='turtle')
```

Appendix I

Python code for parsing and CityGML file and constructing geometric reference

```
'''

@author: JerryZheng
'''
from bs4 import BeautifulSoup
import numpy as np

from pip.cmdoptions import allow_all_external
from numpy import poly
from builtins import str
from itertools import count

soup = BeautifulSoup(open("/Users/pc/Desktop/Datalinking files/data model/
    LoD2_295_5629_1_NW.gml"), 'lxml-xml')
city = soup.CityModel

avg_geo_coords = dict()
with open('csv.csv', mode='w') as f:
    cityObjectMembers = city.find_all('cityObjectMember')
    for cityObjectMember in cityObjectMembers:
        building = cityObjectMember.Building
        building_id = building['gml:id']
        #building_node = 'mycitygml.tue.nl/city/building/' + building_id
        if building.find('roofType'):

            groundsurfaces = building.find_all('GroundSurface')
            for groundsurface in groundsurfaces:
                surfaceMember = groundsurface.lod2MultiSurface.MultiSurface.
                    surfaceMember
                polygon = surfaceMember.Polygon
                linearring = polygon.exterior.LinearRing
                poslist = linearring.posList.text
                float_poslist = [float(e) for e in poslist.split(" ")]
                x = np.mean(float_poslist[0::3])
                y = np.mean(float_poslist[1::3])
                z = np.mean(float_poslist[2::3])
                #poslist_csv = poslist.replace(' ', ',')
                f.write(building_id + ',' + str(x) + ',' + str(y) + ',' + str(z) +
                    '\n')
```

Appendix J

Python code for parsing CityGML and OKSTRA geometric reference CSV file and translating to RDF statements

```python
import csv as csv
import numpy as np
import pandas as pd
import math
import rdflib
from rdflib.plugins.sparql.operators import string
from test.test_functools import decimal


graph = rdflib.Graph(store='IOMemory', identifier='Graph')
gml = rdflib.Namespace("http://www.opengis.net/gml:")
sf = rdflib.Namespace("http://www.opengis.net/ont/sf#")
geo = rdflib.Namespace("http://www.opengis.net/ont/geosparql#")
bldg = rdflib.Namespace("http://www.opengis.net/citygml/building/1.0")
okstra = rdflib.Namespace("http://schema.okstra.de/2016/okstra:")

road = pd.read_csv(r"/Users/pc/Desktop/Abschnitt.csv")
roadbranch = pd.read_csv(r"/Users/pc/Desktop/ast.csv")
building = pd.read_csv(r"/Users/pc/Desktop/geolink/buildingref2wkt.csv")
rposref = road.loc[:,["WKT"]]
roaddata = road.values
roadbranchref = roadbranch.loc[:,["WKT"]]
roadbranchdata = roadbranch.values
bposref = building.loc[:,["WKT"]]
buildingdata = building.values
graph.bind('geo',geo)
graph.bind('sf',sf)
graph.bind('bldg', bldg)
graph.bind('okstra',okstra)

#with open('closestroad.csv', mode='w') as f:
# # print(roaddata)
for i in range(len(buildingdata)):
    row1 = buildingdata[i]
    building_WKT = row1[0]
    building_id = row1[3]
#     asbuilding_WKT = str(building_WKT)
#     Xi = float(row1[1])
#     Yi = float(row1[2])
#       xi = str(Xi)
#       yi = str(Yi)
    building_node = rdflib.URIRef('mycitygml.tue.nl/city/building/' + building_id)
    buildingrefpoint_node = rdflib.URIRef('mycitygml.tue.nl/city/building/RefPoint'
         + building_id)
    graph.add((building_node,geo.hasGeometry,buildingrefpoint_node))
    graph.add((buildingrefpoint_node,rdflib.RDF.type, sf.Point))
    graph.add((buildingrefpoint_node,geo.asWKT,rdflib.Literal(building_WKT,
        datatype=geo.wktLiteral)))
for j in range(len(roaddata)):
    row2 = roaddata[j]
    road_WKT = row2[0]
    road_id = row2[1]
    road_node = rdflib.URIRef('myokstra.tue.nl/Abschnitt/#' + road_id)
    roadrefpoint_node = rdflib.URIRef('myokstra.tue.nl/Abschnitt/RefPoint#' +
        road_id)
    graph.add((road_node,geo.hasGeometry,roadrefpoint_node))
    graph.add((roadrefpoint_node,rdflib.RDF.type, sf.LineString))
    graph.add((roadrefpoint_node,geo.asWKT,rdflib.Literal(road_WKT, datatype=geo.
        wktLiteral)))

for k in range(len(roadbranchdata)):
    row3 = roadbranchdata[k]
    roadbranch_WKT = row3[0]
    roadbranch_id = row3[1]
    roadbranch_node = rdflib.URIRef('myokstra.tue.nl/Ast/#' + roadbranch_id)
    roadbranchrefpoint_node = rdflib.URIRef('myokstra.tue.nl/Ast/RefPoint#' +
        roadbranch_id)
    graph.add((roadbranch_node,geo.hasGeometry,roadbranchrefpoint_node))
    graph.add((roadbranchrefpoint_node,rdflib.RDF.type, sf.LineString))
    graph.add((roadbranchrefpoint_node,geo.asWKT,rdflib.Literal(roadbranch_WKT,
        datatype=geo.wktLiteral)))


graph.serialize(destination='/Users/pc/Desktop/geometry.ttl', format='turtle')
```

Appendix K

Java code for merging CityGML RDF graph and OKSTRA graph with geometric reference RDF graph

```
import org.apache.jena.rdf.model.*;
import org.apache.jena.util.FileManager;

import java.io.*;
import org.apache.log4j.BasicConfigurator;


public class geolink {

  static final String inputFilePath1 = "/Users/pc/Desktop/okstra2rdfnew.ttl";
  static final String inputFilePath2 = "/Users/pc/Desktop/testgeobuilidngnew.ttl";
  static final String inputFilePath3 = "/Users/pc/Desktop/citygml2rdfnew.ttl";


  public static void main(String args[]) throws FileNotFoundException{

    //create 2 models from OKSTRA rdf file and CityGML rdf file
    Model model1 = ModelFactory.createDefaultModel().read(inputFilePath1);
    Model model2 = ModelFactory.createDefaultModel().read(inputFilePath2);
    Model model3 = ModelFactory.createDefaultModel().read(inputFilePath3);

    //put all the triples inside a created new model
    Model integration = ModelFactory.createDefaultModel();
    Model[] models = {model1,model2,model3};

    for (Model part: models) {
      integration.add(part );
    }


    //model out put
    OutputStream out = new FileOutputStream(new File("/Users/pc/Desktop/
        geolinkednew.ttl"));
    integration.write(out,"TURTLE");
    }

}
```

Appendix L

Python code for finding the CityGML building and their closest OKSTRA road section

```
import csv
import numpy as np
import pandas as pd

result = pd.read_csv('/Users/pc/Downloads/query-result-5.csv')

resultlist = list(zip(result['a'],result['b'],result['distance']))

resultdict={}

for item in resultlist:
    a = item[0]
    b = item[1]
    dist = item[2]
    if not resultdict.get(a):
            resultdict[a]=[]
    res = (b,dist)
    resultdict[a].append(res)
mindist={}
for a, tuple in resultdict.items():
    distset = np.array([item[1] for item in tuple])
    array = [item[0] for item in tuple]
    index_min = np.argmin(distset)
    b_value = array[index_min]
    mindist[a]=(b_value,min(distset))
print(mindist)

df = pd.DataFrame(mindist)
df.to_csv('mindis.csv',index= False)
```

Appendix M

Python code for link the CityGML building and OKSTRA road section

```python
import csv as csv
import rdflib
from rdflib.plugins.sparql.operators import string
import pandas as pd

graph = rdflib.Graph(store='IOMemory', identifier='Graph')
co = rdflib.Namespace("http://citygmlinteokstra.tue.nl/")
graph.bind('co',co)

dataset = pd.read_csv(r"/Users/pc/Desktop/mindis.csv")
data = dataset.values
dataref = dataset.loc[:,["building"]]
for i in range(len(data)):
    row = data[i]
    building = row[0]
    section = row[1]
    building_node = rdflib.URIRef(building)
    section_node = rdflib.URIRef(section)
    graph.add((building_node,co.hasclosestroadsection,section_node))

graph.serialize(destination='/Users/pc/Desktop/closestroadsection.ttl', format='
    turtle')
```

Appendix N

Python code for merging CityGML and OKSTRA RDF graphs and geometry RDF graph

```
from rdflib import Graph
g1 = Graph()
g1.parse('/Users/pc/Desktop/geolinked.ttl', format="turtle")

g2 = Graph()
g2.parse('/Users/pc/Desktop/closestroadsection.ttl', format="turtle")

g3 = Graph()
g3.parse('/Users/pc/Desktop/closestdistance.ttl', format="turtle")



graph =  g1 + g2 + g3

graph.serialize(destination='/Users/pc/Desktop/linked.ttl', format='turtle')
```