



**Technische Universität München**  
Ingenieur fakultät Bau Geo Umwelt  
Lehrstuhl für Geoinformatik

# Integration and Management of Time-dependent Properties with Semantic 3D City Models

Kanishk Chaturvedi

Vollständiger Abdruck der von der Ingenieur fakultät Bau Geo Umwelt der Technischen Universität München zur Erlangung des akademischen Grades eines

**Doktor-Ingenieurs (Dr.-Ing.)**

genehmigten Dissertation.

Vorsitzender: Prof. Dr.-Ing. habil. Thomas Wunderlich  
Prüfer der Dissertation: 1. Prof. Dr. rer. nat. Thomas H. Kolbe  
2. Prof. Dr. Gilles Gesquière  
Universität Lumière Lyon 2, France  
3. Prof. Dr.-Ing. André Borrmann

Die Dissertation wurde am 15.04.2020 bei der Technischen Universität München eingereicht und durch die Ingenieur fakultät Bau Geo Umwelt am 28.06.2021 angenommen.



## ABSTRACT

Semantic 3D city models describe spatial, graphical and thematic aspects of the cityscapes by decomposing and classifying the occupied physical space according to a semantic data model. The relevant real-world entities are represented by the ontological structure, including thematic classes, attributes, and their interrelationships. The main advantage of such semantic data models is that they make it possible for applications and simulation tools to distinguish urban objects (like buildings and streets) and use their rich thematic and geometric information for queries, statistical computations, simulations, and visualisations. There are international standards such as CityGML and IFC, which provide not only well-defined data models for describing spatial, graphical and semantic information of physical objects but also an exchange format for exchanging entire city models among different software systems and applications. For this reason, semantic 3D city models are used worldwide for different application domains ranging from Smart Cities, Simulations, Planning to History and Archaeology.

However, most of the applications involve scenarios where city objects and their properties are not static and change with time, and current generation semantic 3D city models do not support such changes explicitly. These changes can be associated with different properties of city objects such as geometry, semantics, topology, or appearance. Furthermore, these changes can also be slower (e.g. evolution of a city over ten years) or highly dynamic (e.g. varying air quality in a room, the energy consumption of a building, and traffic density in a road segment). Hence, such semantic data models must be capable of representing changes taking place in cities and their properties over time.

The objective of this thesis is to extend semantic 3D city models to support different types of time-dependent properties. Based on the comprehensive review of numerous application domains, the thesis provides a systematic analysis and identifies key requirements for extensions of 3D city models to support time-dependent properties. Considering the gathered requirements, a complete “ecosystem” is provided. It not only allows extending 3D city models using the conceptual data models, but also illustrates ways to implement, manage, and visualise them using applications. From the modelling perspective, the research work introduces two novel concepts for incorporating the listed requirements: (i) the "Versioning" concept for managing slower changes in the form of historic and parallel versions of 3D city models and (ii) the "Dynamizer" concept, that allows the representation of highly dynamic data and provides a method for injecting dynamic variations of city object properties into the static representation. Both concepts are realised as extensions of the international OGC standard CityGML.

Furthermore, the thesis also proposes ways to manage static and dynamic properties of the extended 3D city models allowing them to be queried by applications in an integrated fashion. The time-dependent properties can be stored in database management systems along with the precise description and metadata of time-series. In other scenarios, where the properties are highly dynamic (e.g. real-time

observations from a sensor or an IoT device), the storage of time-varying values is cumbersome. Hence, the concept of Spatial Data Infrastructure is utilised. The SDI allows city objects and time-varying properties to manage in distributed systems and to integrate them when they are used. For this purpose, the research work introduces a new concept called "InterSensor Service" allowing to establish interoperability over different types of time-dependent properties. The service allows connecting to multiple IoT platforms, simulation specific data, databases, and simple files and representing the observations using open and international standards. In this way, the heterogeneous observations can be analysed and visualised in a unified way.

The concepts are implemented and demonstrated within this thesis in the context of real-world Smart City projects. Although the concepts are developed for the CityGML standard, they can also be applied to other GML-based application schemas including the European INSPIRE data themes and national standards for topography and cadasters, like the British Ordnance Survey Mastermap or the German cadaster standard ALKIS.

## ZUSAMMENFASSUNG

Semantische 3D-Stadtmodelle beschreiben räumliche, grafische und thematische Aspekte von Stadtlandschaften, indem sie den belegten physischen Raum gemäß der Struktur eines semantischen Datenmodells zerlegen und klassifizieren. Die relevanten Objekte der realen Welt werden durch die ontologische Struktur mit thematischen Klassen, Attributen und deren Beziehungen repräsentiert. Der Hauptvorteil solcher semantischen Datenmodelle besteht darin, dass sie es Anwendungen und Simulationswerkzeugen ermöglichen, Stadtobjekte (wie Gebäude und Straßen) zu unterscheiden und deren umfassende thematische und geometrische Informationen für Abfragen, statistische Berechnungen, Simulationen und Visualisierungen zu nutzen. Es gibt internationale Standards wie CityGML und IFC, die nicht nur wohldefinierte Datenmodelle zur Beschreibung räumlicher, grafischer und semantischer Informationen von physischen Objekten bereitstellen, sondern auch ein Austauschformat für den Austausch ganzer Stadtmodelle zwischen verschiedenen Softwaresystemen und Anwendungen. Aus diesem Grund werden semantische 3D-Stadtmodelle weltweit für verschiedene Anwendungsbereiche eingesetzt, die von Smart Cities, Simulationen, Planung bis hin zu Geschichte und Archäologie reichen.

Die meisten Anwendungen beinhalten jedoch Szenarien, in denen Stadtobjekte und ihre Eigenschaften nicht statischer Natur sind, sondern sich über die Zeit ändern. Die aktuelle Generation semantischer 3D-Stadtmodelle unterstützt solche Änderungen nicht explizit. Diese Änderungen können mit verschiedenen Eigenschaften von Stadtobjekten wie Geometrie, Semantik, Topologie oder Erscheinung verbunden sein. Darüber hinaus können diese Änderungen auch langsamer (z.B. Entwicklung einer Stadt über 10 Jahre) oder hochdynamisch (z.B. schwankende Luftqualität in einem Raum, Energieverbrauch eines Gebäudes und Verkehrsdichte in einem Straßenabschnitt) stattfinden. Daher ist es wichtig, dass die Datenmodelle für semantische 3D-Stadtmodelle in der Lage sind, Veränderungen in Städten und deren Eigenschaften im Laufe der Zeit zu repräsentieren.

Das Ziel dieser Arbeit ist die Erweiterung semantischer 3D-Stadtmodelle zur Unterstützung verschiedener Arten von zeitabhängigen Eigenschaften. Basierend auf der umfassenden Betrachtung zahlreicher Anwendungsbereiche bietet die Arbeit eine systematische Analyse und identifiziert die wichtigsten Anforderungen für Erweiterungen von 3D-Stadtmodellen zur Unterstützung zeitabhängiger Eigenschaften. Unter Berücksichtigung der gesammelten Anforderungen wird ein vollständiges "Ökosystem" bereitgestellt, das nicht nur die Erweiterung von 3D-Stadtmodellen mit Hilfe der konzeptuellen Datenmodelle ermöglicht, sondern auch Möglichkeiten zur Implementierung, Verwaltung und Visualisierung mit Hilfe von Anwendungen aufzeigt. Aus der Perspektive der Modellierung stellt die Forschungsarbeit zwei neuartige Konzepte zur Berücksichtigung der gesammelten Anforderungen vor: (i) das "Versionierungskonzept" zur Verwaltung langsamerer Änderungen in Form von historischen und parallelen Versionen von 3D-Stadtmodellen und (ii) das "Dynamizer"-Konzept,

das es ermöglicht, hochdynamische Daten zu repräsentieren und ein Verfahren bereitstellt, mit dem dynamische Änderungen von Stadtobjekteigenschaften in die statische Repräsentation eingebracht werden können. Beide Konzepte sind als Erweiterungen des internationalen OGC-Standards CityGML realisiert.

Darüber hinaus schlägt die Arbeit auch vor, wie man statische und dynamische Eigenschaften der erweiterten 3D-Stadtmodelle verwalten kann, so dass sie von Anwendungen in einer integrierten Weise abgefragt werden können. Die zeitabhängigen Eigenschaften können in Datenbankmanagementsystemen zusammen mit einer genauen Beschreibung und Metadaten der Zeitreihen gespeichert werden. In anderen Szenarien, in denen die Eigenschaften der Objekte hochdynamisch sind (z.B. Echtzeitbeobachtungen von einem Sensor oder einem IoT-Gerät), ist die integrierte Speicherung von zeitvariablen Werten der Eigenschaften von Stadtobjekten in Datenbanksystemen umständlich. Für diese Fälle wird daher das Konzept der Geodateninfrastruktur genutzt, das es vorsieht, Stadtobjekte und zeitvariable Beobachtungen in einem verteilten System zu speichern und erst bei der Nutzung zusammenzuführen. Zu diesem Zweck wird in der Forschungsarbeit ein neues Konzept namens "InterSensor Service" vorgestellt, das es ermöglicht, die Interoperabilität über verschiedene Arten von zeitabhängigen Eigenschaften hinweg herzustellen. Der Service ermöglicht die Anbindung an mehrere IoT-Plattformen, simulationsspezifische Daten, Datenbanken und einfache Dateien sowie die Repräsentation der Beobachtungen mit Hilfe offener internationaler Standards. Auf diese Weise können die heterogenen Beobachtungen einheitlich analysiert und visualisiert werden.

Die Konzepte wurden im Rahmen der Arbeit in Smart City-Projekten aus der Praxis umgesetzt und demonstriert. Obwohl die Konzepte für den CityGML-Standard entwickelt wurden, sind sie auch auf andere GML-basierte Anwendungsschemata übertragbar, darunter die Datenthemen der EU-Richtlinie INSPIRE und nationale Standards für Topographie und Kataster wie die British Ordnance Survey Mastermap oder den deutschen Katasterstandard ALKIS.

## ACKNOWLEDGEMENTS

This thesis represents the outcome of several years of research conducted at the Chair of Geoinformatics of the Technische Universität München. In doing this research and writing this thesis, I enjoyed support and encouragement from many people without whom this thesis would not have been completed and to whom I want to express my sincere gratitude.

First and foremost, I extend my deepest gratitude to my supervisor Prof. Dr. Thomas H. Kolbe. It was indeed an honour to work under his valuable guidance and supervision. I thank him for his continuous support and encouragement, fruitful discussions, constructive criticism, and valuable scientific remarks. For the rest of my professional life, he will be my true inspiration.

I would also like to extend my heartiest thanks to Prof. Dr. Gilles Gesquière and Prof. Dr.-Ing. André Borrmann for co-supervising this research. Their invaluable suggestions and encouragement helped me a lot in completing my thesis.

My special thanks go to my colleagues Dr. Andreas Donaubaue and Dr. Tatjana Kutzner, for all the fruitful discussions related to my research and for having been a reliable mentor during all these years. I also thank them for providing valuable feedback in completing my thesis. I enjoyed working at the Chair of Geoinformatics. Thanks a million to my peers Dr. Zhihang Yao, Maximilian Sindram, Mandana Moshrefzadeh, Mostafa ElFouly, Son H. Nguyen, Benedikt Schwab, Bruno Willenborg, Christof Beil, Caroline Marx, Dr. Aftab Khan, and Dr. Ihab Hijazi. I also express my special gratitude to Dr. Gabriele Aumann, Tanja Nyc, and Roland Dietrich.

The research would not have been possible without the sponsors of the research projects in which I have been involved. A significant part of this thesis was carried out within the project Smart District Data Infrastructure (SDDI) Demonstrator funded by the Climate-KIC of the European Institute of Innovation and Technology (EIT). I want to thank Climate-KIC and the EIT for providing continuous support. I also thank project partners Mr. Jim Wood and Mr. Ben Edmonds (London Legacy Development Corporation), Prof. Nilay Shah and Dr. Koen H. Van Dam (Imperial College London), Mr. Paul Oesten-Creasey (VU.City), and Dr. Andreas Matheus (Secure Dimensions GmbH). Some of the concepts in this thesis were also developed for the project OGC Future City Pilot Phase 1. I would like to thank the pilot sponsors and participants for making this project a big success. I acknowledge Mr. Bart De Lathouwer (Open Geospatial Consortium), Mr. Simon Navin (Ordnance Survey), Mr. Emmanuel Devys (IGN France), Dr. Claus Nagel and Dr. Lutz Ross (virtualcitySYSTEMS GmbH), Dr. Mohsen Kalantari (University of Melbourne), and Mr. Guy Schumann (Remote Sensing Solutions Inc., U.S.A.). I would also like to thank the project Digital Twin Munich for allowing me to apply my research skills to the project. I acknowledge Mr. Jan Liebscher, Mr. Markus Mohl, and Mr. Maximilian Müller (Landeshauptstadt München), and Dr. Ralf Kerschner (CGI Deutschland).

This thesis would not have been possible without the lasting patience, support, and love of my family. All my love goes to my wife Kimi for her belief in me and for sharing this journey with me. My heartfelt gratitude goes to my parents Harsh and Sadhana, aunts Rani and Vibha, parents-in-law Vineet and Pallavi, brother Shantanu, sister-in-law Kanika, my adorable nephews Raghav and Kriday, and our cat Pablo. This thesis is also their accomplishment; hence I dedicate it to them.



# Contents

<b>Chapter 1: Introduction</b>	<b>1</b>
1.1 Semantic 3D City Models . . . . .	1
1.2 Motivation and Problem Statement . . . . .	2
1.3 Research Hypotheses and Questions . . . . .	3
1.4 Outline of the thesis . . . . .	5
1.4.1 Part I: Integration of time-dependent properties . . . . .	6
1.4.2 Part II: Management of time-dependent properties . . . . .	6
1.4.3 Part III: Proof of Concept . . . . .	7
1.5 Projects . . . . .	7
1.6 Open Source Solutions . . . . .	8
<b>Chapter 2: Background</b>	<b>11</b>
2.1 Time-dependent properties in the applications of 3D city models . . . . .	12
2.1.1 Smart Cities and Digital Twins . . . . .	12
2.1.2 Urban Simulations . . . . .	18
2.1.3 Mobility . . . . .	23
2.1.4 Urban Development . . . . .	25
2.1.5 Requirements Summary . . . . .	28
2.2 Review of existing standards for the listed requirements . . . . .	28
2.2.1 Sensor and IoT data access and management . . . . .	28
2.2.2 Representation and management of time-series . . . . .	31
2.2.3 Managing alerts and events . . . . .	32
2.2.4 Representation of moving objects . . . . .	33
2.2.5 History and Version Management . . . . .	34
2.3 Evaluation of city modelling standards for the listed requirements . . . . .	35
2.3.1 CityGML 2.0 . . . . .	35
2.3.2 IFC v4 . . . . .	38
2.3.3 EU INSPIRE . . . . .	39
<b>Chapter 3: Methodology</b>	<b>41</b>
3.1 Time-dependent properties in the context of 3D city models . . . . .	42
3.1.1 Identification of city object properties affected by time . . . . .	42
3.1.2 Classification of changes in cities . . . . .	43
3.2 Overview of the CityGML standard . . . . .	45
3.2.1 Data Modelling with CityGML . . . . .	45
3.2.2 Management of CityGML-based city models . . . . .	47

3.3	Realisation of the concepts with the CityGML standard . . . . .	49
3.3.1	Data Models . . . . .	49
3.3.2	Data Management . . . . .	50
3.3.3	Proof of concept . . . . .	51
<b>PART I: INTEGRATION OF TIME-DEPENDENT PROPERTIES</b>		<b>53</b>
<b>Chapter 4: Modelling Slower Changes</b>		<b>55</b>
4.1	Versioning in semantic 3D city models . . . . .	56
4.1.1	Requirements for modelling the new Versioning concept . . . . .	58
4.2	Modelling the Versioning concept within the CityGML standard . . . . .	61
4.2.1	Versionable Features . . . . .	61
4.2.2	Version - a new Feature Type . . . . .	63
4.2.3	Version Transitions . . . . .	64
4.2.4	Complete UML Model of the Versioning concept . . . . .	65
4.3	Illustration of the Concept . . . . .	66
4.3.1	Using new CityGML identifiers . . . . .	66
4.3.2	Using Version and Version Transitions . . . . .	68
4.4	Discussions . . . . .	70
<b>Chapter 5: Modelling Highly Dynamic Changes</b>		<b>73</b>
5.1	Making 3D City Models Dynamic . . . . .	74
5.2	Modelling the Dynamizer concept within the CityGML standard . . . . .	75
5.2.1	Dynamizer - a new Feature Type . . . . .	76
5.2.2	SensorConnection . . . . .	78
5.2.3	Atomic Timeseries . . . . .	81
5.2.4	Composite Timeseries . . . . .	87
5.2.5	Complete UML Model of Dynamizer . . . . .	90
5.3	Illustration of the Concept . . . . .	90
5.3.1	Integrating city object properties with real-time sensors . . . . .	91
5.3.2	Representing timeseries values in-line within city objects . . . . .	99
5.3.3	Representing complex periodic patterns using Dynamizers . . . . .	103
5.4	Discussions . . . . .	109
<b>PART II: MANAGEMENT OF TIME-DEPENDENT PROPERTIES</b>		<b>111</b>
<b>Chapter 6: Management on the level of databases</b>		<b>113</b>
6.1	Managing CityGML ADEs within databases . . . . .	114
6.1.1	Managing the Versioning ADE within the 3DCityDB . . . . .	115
6.1.2	Managing the Dynamizer ADE within the 3DCityDB . . . . .	117
6.2	New Relational Data Model for the Dynamizer ADE . . . . .	119
6.2.1	Dynamizer Core Module . . . . .	120
6.2.2	Timeseries Metadata Module . . . . .	120
6.2.3	Timeseries Module . . . . .	122

---

6.3	Import and Export of Dynamizer ADE data to/from the 3DCityDB . . . . .	124
6.4	Discussions . . . . .	125
<b>Chapter 7: Management of Dynamic City Models on the level of SDIs</b>		<b>127</b>
7.1	Spatial Data Infrastructures (SDI) . . . . .	128
7.2	Establishing cross-platform interoperability for sensor and time-series data . . . . .	129
7.2.1	OGC Sensor Observation Service . . . . .	130
7.2.2	OGC SensorThings API . . . . .	133
7.2.3	Further recommendations on working with the OGC SWE standards . . . . .	136
7.3	Introduction to the InterSensor Service . . . . .	136
7.3.1	Architecture . . . . .	138
7.3.2	Data Model . . . . .	139
7.4	Illustration of the concept . . . . .	141
7.4.1	Adding a data source . . . . .	141
7.4.2	Automated generation of the standardised interfaces . . . . .	143
7.5	Discussions . . . . .	146
<b>PART III: PROOF OF CONCEPT</b>		<b>149</b>
<b>Chapter 8: Using Dynamic 3D City Models in Smart Cities</b>		<b>151</b>
8.1	OGC Future City Pilot Phase 1 . . . . .	152
8.1.1	Integrating city object properties with real-time sensor data . . . . .	152
8.1.2	Enriching city object properties with solar potential simulation time-series . . . . .	157
8.1.3	Integrated management and visualisation of static and dynamic properties . . . . .	161
8.2	Smart District Data Infrastructure (SDDI) . . . . .	162
8.2.1	Deployment options for the InterSensor Service . . . . .	164
8.2.2	Joint visualisation and analysis of heterogeneous sensor data . . . . .	166
8.2.3	Integrated management and visualisation of static and dynamic properties . . . . .	167
8.2.4	Easy deployment of interoperable solutions . . . . .	169
<b>Chapter 9: Securing Data Infrastructures for Smart Cities</b>		<b>173</b>
9.1	Securing the Smart District Data Infrastructure (SDDI) . . . . .	174
9.2	Gathering requirements for securing the infrastructure . . . . .	175
9.2.1	Smart Cities . . . . .	176
9.2.2	Spatial Data Infrastructures (SDI) . . . . .	177
9.2.3	Security . . . . .	177
9.3	Demonstration scenario for securing the SDDI framework . . . . .	178
9.4	Implementations . . . . .	180
9.4.1	Implementing Single-Sign-On . . . . .	180
9.4.2	Linked Protected Data . . . . .	181
9.4.3	Setting up the core security services . . . . .	183
9.5	Illustration of the Concept . . . . .	184
<b>Chapter 10: Conclusions and future work</b>		<b>187</b>
10.1	Thesis Summary . . . . .	187

---

10.2 Discussion of the results . . . . .	188
10.3 Scientific Contributions . . . . .	193
10.4 Outlook and future prospects . . . . .	194
<b>Bibliography</b>	<b>197</b>
<b>Original publications</b>	<b>209</b>

## List of Figures

1.1	Organisation of the thesis. . . . .	5
2.1	Illustration of a Smart City concept. . . . .	13
2.2	Possible ways for linking sensors with city objects. . . . .	17
2.3	Examples of urban simulations with semantic 3D city models. . . . .	19
2.4	Textures of the global irradiation values for the months . . . . .	20
2.5	Tabular representation of monthly irradiation values of a building. . . . .	21
2.6	Visualisation of estimated heat demand values of a building in Berlin. . . . .	22
2.7	Illustration of mobility applications. . . . .	24
2.8	Illustration of the evolution of Singapore City. . . . .	26
2.9	Management of historic and parallel versions within semantic 3D city models. . . . .	27
3.1	Topographic object properties within semantic 3D city models . . . . .	43
3.2	UML diagram of CityGML 2.0 Core module. . . . .	46
3.3	Two new modules proposed for CityGML . . . . .	49
3.4	Data Management of new CityGML modules within 3DCityDB . . . . .	50
4.1	An illustration of historical succession. . . . .	56
4.2	Reconstruction of events to handle alternative models. . . . .	57
4.3	Managing parallel or alternative versions. . . . .	58
4.4	Representation of Version Transitions. . . . .	59
4.5	Issues with versioning of aggregated features. . . . .	60
4.6	Versionable Features of CityGML. . . . .	62
4.7	Introduction of the new Version feature. . . . .	63
4.8	Introduction of the new VersionTransition feature. . . . .	64
4.9	Complete UML model of the CityGML Versioning concept. . . . .	66
4.10	An instance example of versions representing modifications of a building. . . . .	67
5.1	Conceptual illustration of CityGML Dynamizers. . . . .	76
5.2	Dynamizer modelled as a new FeatureType. . . . .	77
5.3	Representation of the data type SensorConnection. . . . .	79
5.4	Representation of the feature type AtomicTimeseries . . . . .	82
5.5	Representation of the StandardFileTimeseries class. . . . .	83
5.6	Representation of the TabulatedFileTimeseries class . . . . .	84
5.7	Representation of the GenericTimeseries class. . . . .	86
5.8	Representation of Composite Timeseries. . . . .	88
5.9	Complete UML model of the Dynamizer concept. . . . .	89
5.10	Dynamizer <i>SensorConnection</i> linking to different sensor platforms. . . . .	90

5.11	Representation of time-series in-line using Atomic Timeseries. . . . .	100
5.12	Example of composing AtomicTimeseries to a pattern. . . . .	104
5.13	Example of complex CompositeTimeseries. . . . .	108
6.1	Transformation Workflow of the 3DCityDB ADE Plugin Manager. . . . .	114
6.2	Management of CityGML ADEs within the 3DCityDB. . . . .	115
6.3	Version Management in the 3DCityDB using the Versioning ADE. . . . .	116
6.4	Issues with the access of Dynamizer AtomicTimeseries by CityGML Viewers. . . . .	118
6.5	Issues with the access of Dynamizer SensorConnection by CityGML Viewers. . . . .	118
6.6	High Level Overview of the implementation of the Dynamizer ADE within the 3DCityDB. . . . .	119
6.7	Relational Logical Model of the Dynamizer ADE. . . . .	121
6.8	Geometry hierarchy managed within the table SURFACE_GEOMETRY for a LoD1 solid geometry. . . . .	123
7.1	Illustration of heterogeneous data sources for sensor and time-series data. . . . .	129
7.2	Interoperability of sensor and time-series data using the OGC Sensor Web Enablement standard suite. . . . .	130
7.3	Interoperability of sensor and time-series data using the OGC SOS standard. . . . .	132
7.4	Official UML Data Model of the OGC SensorThings API standard. . . . .	133
7.5	Interoperability of sensor and time-series data using the OGC SensorThings API standard. . . . .	135
7.6	Motivation of developing the InterSensor Service. . . . .	137
7.7	The three-layer architecture of the InterSensor Service. . . . .	138
7.8	Key resources of the InterSensor Service. . . . .	139
7.9	Representation of types of data sources which can be used by the InterSensor Service. . . . .	140
7.10	Resolving the issue of accessing Dynamizer AtomicTimeseries by the InterSensor Service. . . . .	146
7.11	Resolving the issue of accessing observations from heterogeneous sensor platforms by the InterSensor Service. . . . .	147
8.1	Thematic attributes of a buiding including description and links for sensor based services. . . . .	155
8.2	Timeseries graph visualisation of real-time sensor observations. . . . .	156
8.3	Management of CityGML Dynamizer ADEs for Future City Pilot Phase 1. . . . .	161
8.4	Integrated management and visualisation of static and dynamic properties of CityGML Dynamizers. . . . .	162
8.5	Joint usage of standardised web services in the Smart District Data Infrastructure. . . . .	163
8.6	Implementation scenario of the InterSensor Service. . . . .	164
8.7	Deployment of the InterSensor Service. . . . .	165
8.8	Joint visualisaion of observations being retrieved directly from heterogeneous data sources. . . . .	166
8.9	Real-time energy notification system for buildings within Queen Elizabeth Olympic Park . . . . .	167
8.10	Joint visualisation of geo-tagged tweets retrieved by the InterSensor Service along with CityGML based 3D building objects. . . . .	168

---

8.11	Joint visualisation of available rental car information being retrieved by the InterSensor Service along with CityGML based 3D building objects. . . . .	169
8.12	Illustration of the SDDI functionalities with highly detailed models provided by external visualisation solution providers. . . . .	170
8.13	Illustration of the SDDI functionalities with an external visualisation platform. . . . .	171
9.1	Illustration of secure and controlled access to the distributed applications and services within the SDDI framework. . . . .	175
9.2	Venn Diagram illustrating the key focus of the research contribution. . . . .	176
9.3	Representation of chaining of distributed resources in the SDDI framework. . . . .	179
9.4	Illustration of the security demonstration scenario showing that users identified by different identity providers can access the distributed components. . . . .	180
9.5	An overview of the security demonstrator architecture. . . . .	181
9.6	Selection of the appropriate Identity Provider to access the resources. . . . .	184
9.7	SOS2 can only be accessed by the user with a valid eduGAIN login. . . . .	185





## List of Tables

2.1	List of Smart City projects aiming on developing Digital Twins of cities. The table shows the Project name, Project location, and Use cases. The last two columns show that for their use cases, all the projects use semantic 3D city models and require integration of city models with real-time sensor and IoT information. . . . .	14
2.2	List of key requirements considered for extending semantic 3D city models for supporting time-dependent properties. . . . .	29
2.3	Evaluation of 3D city modelling standards for the listed requirements. . . . .	36
3.1	Classification of slower and highly dynamic changes. The first two columns refer to the requirements listed in Chapter 2. . . . .	44



## LISTINGS

4.1	Representation of multiple object versions within one single CityGML dataset . . . .	67
4.2	Representation of version transitions within one single CityGML dataset. This listing extends Listing 4.1. . . . .	69
5.1	OCLExpression for defining that either column number or column name must be provided for time and value columns . . . . .	85
5.2	OCLExpression for defining that only one type of time-series value can be represented within a single GenericTimeseries and only one type of encoding is represented by TimeValuePair . . . . .	87
5.3	Illustration of a CityGML Building object having a generic attribute 'temperature' . .	91
5.4	Dynamizer defined within the CityGML 3.0 document having direct links to sensor observations available at ThingSpeak platform . . . . .	92
5.5	CityGML Dynamizer having direct links to sensor observations available at the FROST Server . . . . .	94
5.6	Illustration of a CityGML RoofSurface object having a generic attribute named 'outside_temperature' . . . . .	95
5.7	CityGML Dynamizer having direct links to sensor observations available at the Weather Underground platform . . . . .	96
5.8	CityGML Dynamizer subscribing to a sensor data stream using the MQTT protocol .	98
5.9	Illustration of a CityGML Building WallSurface having a generic attribute to record monthly solar irradiation value . . . . .	99
5.10	CityGML Dynamizer TabulatedFileTimeseries referring to time-series stored in an external CSV file . . . . .	100
5.11	Alternative representation of CityGML Dynamizer TabulatedFileTimeseries . . . . .	102
5.12	CityGML Dynamizer GenericTimeseries representing time-series in-line . . . . .	103
5.13	CityGML Dynamizer CompositeTimeseries representing periodic patterns of energy consumption values . . . . .	104
7.1	Example of configuring the data source connection to a ThingSpeak channel . . . . .	142
7.2	Example of configuring the data source connection to a Twitter channel . . . . .	142
7.3	Example of configuring the data source connection to a Dynamizer stored in 3DCityDB	143
7.4	Illustration of the InterSensor Service resource endpoints generated for each data source connection . . . . .	144

---

7.5	Illustration of the OGC SensorThings API endpoints automatically generated for each data source connection . . . . .	144
7.6	Illustration of the OGC Sensor Observation Service endpoints automatically generated for each data source connection . . . . .	145
7.7	Illustration of the 52°North Timeseries API endpoints automatically generated for each data source connection . . . . .	145
8.1	Illustration of Dynamizer SensorConnection to link to a weather station sensor running over the OGC SOS . . . . .	154
8.2	Representation of monthly solar irradiation values as individual generic attributes in the current version of CityGML 2.0 . . . . .	157
8.3	Illustration of monthly solar irradiation values represented according to the OGC TimeseriesML 1.0 standard . . . . .	159
8.4	Illustration of a Dynamizer AtomicTimeseries representing dynamic solar irradiation values for a specific Building Wall Surface . . . . .	160

# Chapter 1

---

## Introduction

### 1.1 Semantic 3D City Models

Virtual 3D city models are digital models that represent urban objects such as terrain surfaces, buildings, vegetation, water bodies, infrastructure, and landscape elements in a 3-Dimensional (3D) space. They have been used for many years for the visualisation and graphical exploration of cityscapes. The most notable examples of virtual 3D city models are Google Earth, Apple Maps, and Bing Maps. While the mentioned examples provide a highly realistic 3D representation of cities, they lack semantic aspects of city objects. The interpretation of the rendered 3D model happens entirely by the (human) viewer relying on his or her ability to recognise individual urban objects. However, the complexities involved in many disciplines, such as planning and decision making, require a virtual representation of cityscapes allowing much more than mere visualisation. For example, computing the total roof surface area of a city quarter, estimating the energy demand of a district within a city, calculating the monthly solar irradiation on a building roof surfaces, and so on. These requirements lead to the development of "Semantic 3D city models" (Kolbe 2009), which not only describe spatial and graphical aspects of the city objects, but also provide the ontological structure including thematic classes, attributes, and their interrelationships. Such information models make it possible for applications and simulation tools to distinguish urban objects (like buildings and streets) and use their rich thematic and geometric information for queries, statistical computations, simulations, and visualisations. Hence, many cities worldwide such as Berlin, Singapore, New York, London, and Helsinki, have already developed semantic 3D city models for different use cases and applications.

Several organisations and standard working groups provide semantic data models for cities and their objects. CityGML (Gröger et al. 2012), issued by the Open Geospatial Consortium (OGC), is one of the widely accepted standards for modelling and exchanging semantic 3D city models. This standard facilitates the integration of heterogeneous data from multiple sources. It allows for the representation of geometrical and semantic attributes of the city level objects (such as buildings, streets, water bodies, vegetation, etc.) along with their interrelationship to the other objects. INSPIRE (Infrastructure for Spatial Information in the European Community) (INSPIRE 2013) is an initiative of the European Commission for developing a European Spatial Data Infrastructure. The INSPIRE Directive addresses 34 spatial data themes (such as Administrative units, Buildings, Transport networks, Land use, Geology, etc.), which are discoverable and interoperable through the implementation of a common set of standards, data models and web services. Industry Foundation Classes (IFC 2016) is also a very popular standard for modelling and exchanging Building Information Models (BIM) (Borrmann et al. 2015). This standard has been developed by buildingSMART International and provides open data

models for sharing building and construction industry data among different software applications. IndoorGML (Lee et al. 2014) is another OGC standard that specifies an open data model and exchange format for indoor spatial information. This standard intentionally focuses on modelling indoor spaces for navigation purposes. Another notable example is the SEDRIS standard (Synthetic Environment Data Representation and Interchange Specification) (Kang et al. 2015), which is one of the oldest semantic information models for representing and sharing the environment data. The environment data may be concrete (such as trees and mountains) or abstract (such as the behaviour of light). The standard SEDRIS offers a data representation model, augmented with its environmental data coding specification and spatial reference model, to capture and communicate meanings and semantics.

## 1.2 Motivation and Problem Statement

Semantic 3D city models are an important source of information for different types of applications and simulations. The well-defined city objects give the spatial context to all information that is related to the physical entities in cities and provide a means for interactive and spatio-semantic queries and aggregations. There are many application and simulation scenarios, which highly benefit from the use of semantic 3D city models such as energy demand estimations (Strzalka et al. 2011; Agugiaro 2016; Kaden and Kolbe 2013), solar potential simulations (Zahn 2015; Biljecki et al. 2015a; Salimzadeh et al. 2018), disaster management (Morel and Gesquière 2014), training simulators and autonomous driving (Randt et al. 2007; Schwab and Kolbe 2019), and indoor navigation (Mäs et al. 2006). An extensive review of different applications of 3D city models is also provided by (Biljecki et al. 2015b). They distinguished applications into two categories: the first category is non-visualisation use cases, which do not require the visualisation of the 3D city model as well as the results of the operations that the use case comprises. The second category is visualisation-based use cases, where visualisation of the city model and the results play an important role. For example, solar potential analysis is an instance of a non-visualisation use case, in which the simulation results can be visualised, but this is not essential to achieve the purpose of the use case. The results can be stored in a database, which can be queried without the need of being visualised. On the other hand, the applications related to navigation, gaming, and also urban planning fall into the category of visualisation-based use cases. In these cases, the visualisation of the objects is essential, and the use cases would not make much sense without it. In total, the authors identified more than 29 use cases including more than 100 applications, which are arranged into these categories.

However, most of these applications involve scenarios where city object properties are not static and change with time (c.f. chapter 2), and current generation semantic 3D city models do not support such changes explicitly. In general, these changes can be associated with different properties of city objects. For example, a construction event leads to the change in geometry of a building (i.e. addition of a new building floor or demolition of an existing door). The geometry of an object can be further classified according to its shape, location, and extent, which can also be changed with time. A moving car object involves varying only the position and orientation of the car; however, a flood incident involves variations in location and shape of water. There are other properties which result in changes in the semantics of city objects over time, e.g., hourly variations in energy or gas consumption of a building or changing the building's type from residential to commercial. Some properties involve changes in appearances over time, such as building textures changing over years or traffic cameras recording videos of moving traffic over definite intervals. Semantic 3D city models comprise relevant real-world entities and also represent interrelationships between objects. Such interrelationships may also change over time. Besides, the city objects may be decomposed into parts based on deeply nested

structures that can be observed in the real world. For example, a building may be decomposed into different (main) building parts like walls, stairs, etc. and these may again consist of parts like windows or doors. The changes may also be related to such semantic decompositions.

The main objective of the thesis is to extend current generation semantic 3D city models by providing explicit support of time-dependent properties. The thesis reviews multiple application domains and identifies key requirements for temporal and dynamic extensions of city object properties. Further, it defines approaches to extend semantic 3D city models based on the specified requirements. The thesis provides a complete “ecosystem”, which not only allows extensions using the conceptual data models, but also offers ways to (i) implement, (ii) manage, and (iii) visualise them in the applications. The concepts presented in the thesis have been developed for the CityGML standard. However, they can also be applied to other GML-based application schemas including the European INSPIRE data themes and national standards for topography and cadasters like the British Ordnance Survey Mastermap or the German cadaster standard ALKIS.

### 1.3 Research Hypotheses and Questions

The main research question that this thesis seeks to answer is

How can semantic 3D city models be extended to support time-dependent properties?

The research question is further subdivided into the following hypotheses and questions:

From the 'modelling' perspective,

**Question 1.1:** What are time-dependent properties in the context of semantic 3D city models?

This is a fundamental question determining the time-dependent properties associated with the static properties of city objects.

**Question 1.2:** What kinds of time-dependent properties are required in various applications of semantic 3D city models?

This question attempts to review several application domains and investigates over different types of time-dependent properties used in those applications. The question tries to identify essential requirements that arise from these applications enabling to extend city objects and their properties to support time-dependent properties in a systematic way.

**Question 1.3:** How can existing data models be extended to support the identified time-dependent properties?

This question aims to find out how can we extend data models for semantic 3D city models to support the identified time-dependent properties. The possible sub-questions are "*Are the identified time-dependent properties similar or fundamentally different from each other?*" and thus, "*Is one common data model sufficient or do we require individual data models for representing all the identified time-dependent properties?*".

**Hypothesis 1.4:** Existing modelling standards for representing various time-dependent properties can be utilised in extending semantic 3D city models.

The modern research and application fields such as Sensor and IoT, Big Data Management, Urban Mobility already provide sophisticated standards and platforms for managing various kinds of time-dependent properties. Such standards and platforms are stable, well-defined, and used worldwide. This hypothesis states that many of such standards can be utilised in developing the extensions for semantic 3D city models.

From the 'management' perspective,

**Question 1.5:** How can we manage time-dependent properties along with static properties of 3D city models?

There are already several sophisticated database management systems, which allow efficient management of semantic 3D city models. However, the available solutions only store the static properties of city objects. This question finds ways on how time-dependent properties can be managed along with the static properties of city objects.

**Hypothesis 1.6:** It is not always required to store time-dependent properties along with the static properties of city objects in database management systems.

In many scenarios, time-dependent properties may be highly frequent. For example, sensors and IoT devices are capable of measuring values up to every millisecond, which produces a massive amount of data in a short period. The concept of Spatial Data Infrastructures (SDI) can be beneficial in such cases allowing managing and accessing highly dynamic data using open and international standards in an interoperable way.

**Question 1.7:** How can we achieve cross-platform interoperability for heterogeneous data sources of time-dependent properties?

This question aims to determine the right approach for accessing heterogeneous data belonging to different stakeholders and running over different platforms in a unified way.

From the 'applications' perspective,

**Question 1.8:** How can city modelling applications achieve integrated access to static as well as dynamic properties of city models?

This question tries to find ways on how the 3D city modelling applications can interpret static as well as dynamic information in an integrated fashion.

**Question 1.9:** How can we have a unified visualisation for multiple heterogeneous time-dependent properties using the same framework?



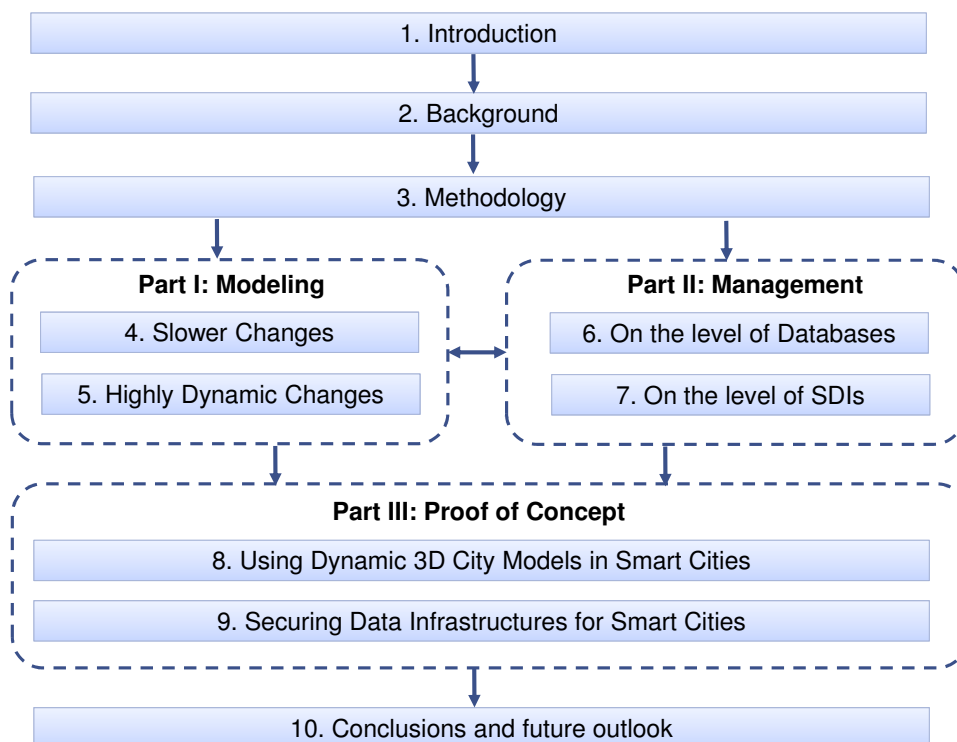
This question aims at reviewing multiple data sources and achieving an interoperable framework allowing accessing and visualising the data in a unified way.

**Question 1.10:** What are secure ways allowing users and applications to access the static and dynamic properties of semantic 3D city models?

The concept of SDIs allows integrating different data sources such as sensors, IoT devices, simulation tools, and 3D city models within a common operational framework. However, such distributed systems, if not secured, may cause a significant threat by disclosing sensitive information to untrusted or unauthorised entities. This question attempts to investigate how can such distributed access and management be secured from the users and applications perspectives.

## 1.4 Outline of the thesis

The research work carried out within this thesis has made a significant contribution to 15 Scientific Publications including Journal Articles, Book Chapters, Conference Papers, and Engineering Reports (listed in the "Original Publications" section). The results of this thesis have been applied and implemented successfully in several research projects (c.f. section 1.5) contributing directly to multiple Open Source solutions (c.f. section 1.6).



**Figure 1.1:** Organisation of the thesis. The report is divided into 3 main parts and 10 chapters.

The thesis is organised into 3 parts and 10 chapters (Figure 1.1), as follows:

Chapter 2 is empirically oriented. It discusses the fundamentals that are crucial to an understanding before delving deeper into the topic: *what are time-dependent properties in the context of semantic 3D city models?*, *what are the requirements that arise from different applications for temporal extensions of 3D city models?*, *what are the available standards and technologies to represent time-dependent properties?*, and *to which degree the existing city modelling standards support the listed requirements?*. Based on the comprehensive review, key requirements are gathered for extending semantic 3D city models.

Chapter 3 defines a methodology by classifying the key requirements according to two broad categories: slower changes and highly dynamic changes. Accordingly, data modelling and data management approaches are defined for the research. Since the realisation of the new concepts is based on the OGC CityGML standard, this chapter also provides a brief overview of the data modelling aspects and available tools for the CityGML standard.

The subsequent chapters are organised into 3 parts:

### **1.4.1 Part I: Integration of time-dependent properties**

Chapter 4 focuses on changes in cities that are slower such as history and evolution of cities. For managing such gradual changes, the chapter introduces a new Versioning concept for the CityGML standard. It extends the existing CityGML data model and allows exchanging different versions and version transitions within one dataset. In this way, a complete history or evolution of the city model is supported by version transitions having bi-temporal attributes. The concept also allows managing parallel alternative versions of the objects at the same time. The Versioning concept includes a new identifier approach allowing users to refer to a specific historic or parallel version in the dataset in an efficient way.

Chapter 5 focuses on changes that represent high frequent or dynamic variations of the object properties. For example, such variations are related to (i) thematic attributes such as changes of physical quantities (energy demands, temperature, solar irradiation levels), (ii) spatial properties such as changing feature's position (moving objects), and (iii) appearances such as changing building's textures or colours. In these cases, only some of the properties of otherwise static objects need to represent such time-varying values. Such changes are supported within 3D city models using the newly proposed Dynamizer concept. The approach allows integrating different kinds of dynamic data such as time-series obtained from sensor and IoT devices, simulation databases and external files. It also provides a method to inject them into the static attributes of city models.

### **1.4.2 Part II: Management of time-dependent properties**

The second part of the thesis focuses on managing time-dependent properties with semantic 3D city models. Chapter 6 discusses the approaches for managing newly introduced Versioning and Dynamizer concepts in the database management systems for performing queries and analysis. Further, this chapter provides a new relational database model for managing time-series and its metadata values associated with Dynamizers. This functionality allows performing queries based on the time-series related to a specific city object.

Chapter 7 discusses how time-dependent properties of city objects along with the static properties can be retrieved using web services. For this purpose, the chapter presents the concept of the Spatial Data Infrastructure (SDI) and describes ways to access time-series data using open and international

standards. Further, the chapter introduces a new concept InterSensor Service allowing to retrieve time-series data from CityGML Dynamizers and translates them "on-the-fly" according to the international standards. Besides, the service can also retrieve time-series data from the arbitrary sensor and IoT platforms, databases, and external tabulated files and perform such translations. In this way, time-series data from heterogeneous data sources can be retrieved and accessed in a unified way.

### 1.4.3 Part III: Proof of Concept

The third part of the thesis provides proofs of concepts that have been developed in the previous two parts. Chapter 8 provides implementations for the developed concepts in the context of two real-world Smart City projects. Various demonstrations show the applicability of CityGML Dynamizers in linking with real-time sensor data as well as representing solar-potential simulation results in-line with city objects. The demonstrations also include the cross-platform interoperability of many heterogeneous data sources of time-series data using the InterSensor Service. Such unified representations improve decision-making in Smart City scenarios.

Furthermore, Chapter 9 highlights the importance of security in realising distributed infrastructures (as mentioned in Chapter 8). The chapter presents a novel solution for securing the overall access and management of distributed applications and services. The proposed concept facilitates privacy, security and controlled access to all stakeholders and the respective components by establishing proper authorisation and authentication mechanisms. The approach offers Single-Sign-On (SSO) authentication by a novel combination in the use of the state-of-the-art security concepts such as OAuth2 access tokens, OpenID Connect user claims and Security Assertion Markup Language (SAML).

Chapter 10 concludes the thesis with the key takeaways, answers to the research questions, main contributions of the research, and proposes a roadmap for future work.

## 1.5 Projects

The research described in this thesis have been applied successfully to the following projects:

1. **OGC CityGML 3.0<sup>1</sup>**: To increase the usability of CityGML for more user groups and areas of application, the OGC CityGML Standards Working Group (SWG) and the Special Interest Group 3D (SIG 3D) of the initiative Geodata Infrastructure Germany (GDI-DE) have been working since 2014 on the further development of the CityGML standards. This development intends to the next major version CityGML 3.0. The requirement of supporting time-dependent properties by city objects was considered as one of the core packages within the development of CityGML 3.0. The data models developed within this thesis (Chapters 4 and 5) have been developed considering the requirements and motivation of the CityGML 3.0 SWG. The new Versioning and Dynamizer concepts are planned to become part of CityGML 3.0 (Kutzner et al. 2020).
2. **OGC Future City Pilot Phase 1<sup>2</sup>**: The Future City Pilot, Phase 1 (FCP1), an initiative from the Open Geospatial Consortium (OGC), successfully demonstrated that the use of geospatial technologies including international standards such as CityGML and Industry Foundation Classes

<sup>1</sup><https://github.com/opengeospatial/CityGML-3.0CM>

<sup>2</sup><https://www.opengeospatial.org/projects/initiatives/fcp1>

(IFC) can provide stakeholders with information, knowledge and insight which enhances financial, environmental, and social outcomes for citizens living in cities. During the pilot, multiple scenarios were set up based on real-world requirements and were put forward by the pilot sponsors: Sant Cugat del Vallès (Barcelona, Spain), Ordnance Survey Great Britain (UK), virtualcitySYSTEMS GmbH (Germany), and Institut National de l'Information Géographique et Forestière - IGN (France). The scenarios focused on the following areas: Urban Planning, Urban Flood Mapping, Adult Social Care, and Dynamic Resource Modelling. The solutions for the respective scenarios were developed by the pilot participants: University of Melbourne (Australia), Remote Sensing Solutions, Inc. (U.S.A), and Technical University of Munich (Germany). The solutions were developed for (i) achieving interoperability between the IFC and CityGML standards, (ii) illustrating a new level of interoperability between flood models and semantic 3D city models based on the CityGML standard, and (iii) integrating dynamic data such as real-time sensor streams and solar potential analysis results with 3D city models. The Dynamizer concept (c.f. Chapter 5) developed within this thesis was implemented as an Application Domain Extension (ADE) for covering the use cases of this project. The results of Dynamizers for this project are shown in Chapter 8.

3. **Smart District Data Infrastructure (SDDI) Demonstrator**<sup>3</sup>: This project runs within the Smart Sustainable Districts Program of the Climate-KIC of the European Institute for Innovation and Technology (EIT). The SDDI framework allows integrating diverse components such as multiple stakeholders, sensors, IoT devices, simulation tools with a virtual district model representing the physical reality of the district. Within the project, the owners of the district, London Legacy Development Corporation (LLDC), have identified different use cases related to the reduction of resource and energy usage, reduction of waste, reduction of emissions, improvements of well-being, mobility, and in general concerning efficiency. The concepts in this thesis (Chapters 7, 8, and 9) have been developed considering the requirements of the use cases of this project.
4. **Digital Twin Munich**<sup>4</sup>: Digital Twin Munich (Digitaler Zwilling in München) is an ongoing project initiated by the City of Munich (Landeshauptstadt München). It is funded by the Federal Ministry of Transport and Digital Infrastructure, Germany. The Digital Twin intends to create a complete digital image of Munich. In addition to a three-dimensional presentation, this "digital copy" will contain extensive information (including real-time data from sensors and IoT devices). The aim is to improve the basis for urban, traffic and environmental planning, for example, by modelling what-if scenarios. The concepts developed within this thesis (Chapter 7) enable ways to integrate real-time information from various sensors and IoT devices with the 3D City Model of Munich in a standardised and interoperable manner.

## 1.6 Open Source Solutions

The concepts developed in this thesis have been implemented and contributed to the following Open Source solutions:

1. **InterSensor Service** (<https://github.com/tum-gis/InterSensorService>): InterSensor Service is a lightweight web service that allows users to connect to multiple IoT platforms, databases and

<sup>3</sup><https://www.lrg.tum.de/gis/projekte/smart-district-data-infrastructure/>

<sup>4</sup><https://muenchen.digital/twin/>

external tabulated files and retrieving their observations without worrying about data storage and the multitude of different APIs. It is a Java application based on the Spring framework. The application is free and Open Source. This application has been developed and implemented within this thesis (Chapter 7).

2. **IoT-FROST-Ecosystem** (<https://github.com/tum-gis/iot-frost-ecosystem>): IoT-FROST-Ecosystem is a guideline that has been developed within this thesis for covering use cases of projects. It provides a step-by-step process to work with various IoT platforms and allows their observations to be managed and visualised in standardised ways. It describes ways (i) to interact with several sensors and IoT platforms, (ii) to manage their observations according to open and international standards, and (iii) to visualise heterogeneous observations using a common dashboard application. The guideline utilises Open Source applications which are used to set up IoT ecosystems based on open and international standards. By following the steps in this repository, a user can very easily install and set up the required applications on his/her personal desktop/laptop or a remote (virtual) machine.
3. **3DCityDB** (<https://github.com/3dcitydb>): 3DCityDB (3D City Database) is an Open Source geodatabase that stores, represents, and manages the large CityGML datasets on top of a standard spatial relational database management systems such as Oracle Spatial and PostgreSQL. It provides a Java front-end application named '3DCityDB Importer/Exporter', which allows for high performance importing and exporting the CityGML datasets with arbitrary file sizes. It also allows exporting the contents in the form of different visualisation formats such as KML, COLLADA, and glTF, allowing the 3D objects to be viewed and interactively explored in the web applications. Furthermore, 3DCityDB software suite also includes a visualisation client named '3DCityDB-Web-Map-Client', which provides rich 3D visualisation and interactive exploration of arbitrarily large semantic 3D city models based on the CityGML standard. The concepts developed within this thesis (Chapter 6 and 7) contribute towards the extension of the 3DCityDB and the 3DCityDB-Web-Map-Client.
4. **CityGML 3.0 Conceptual Models** (<https://github.com/opengeospatial/CityGML-3.0CM>): The conceptual models of Versioning and Dynamizer concepts developed within this thesis (Chapter 4 and 5) are planned to become a part of the CityGML 3.0 release. The data models have been provided to the official OGC CityGML 3.0 repository and are openly available at the mentioned GitHub repository.



## Chapter 2

---

### Background

This chapter discusses the fundamentals that are crucial to an understanding before delving deeper into the topic. The chapter is oriented towards the basic questions such as "*what are time-dependent properties in the context of semantic 3D city models?*", "*what are the requirements that arise from different applications for temporal extensions of 3D city models?*", and "*to which degree the existing 3D city modelling standards support the listed requirements?*". The discussion provides a systematic analysis on identifying essential requirements which should be considered for extending semantic 3D city models for supporting time-dependent properties. Major applications and use cases of semantic 3D city models are studied to gather scenarios for dealing with different types of time-dependent properties. The chapter also provides a comprehensive review of relevant standards that could be considered and re-used for extending data models for supporting the listed requirements. Finally, the chapter evaluates three widely used standards OGC CityGML, buildingSMART IFC, and the European Union INSPIRE for the listed requirements for supporting time-dependent properties. The results help determine the current state of the standards as well as defining the methodology for extending semantic 3D city models.

Some of the discussions in this chapter have been presented in the published paper

**Chaturvedi, K.** and Kolbe, T. H. (2019). 'A Requirement Analysis on extending Semantic 3D City Models for supporting Time-dependent properties.' In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-4/W9, pp. 19–26. URL: <https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-4-W9/19/2019/>

## 2.1 Time-dependent properties in the applications of 3D city models

Semantic 3D city models provide spatial context to all information that is related to the physical entities in cities. At the same time, they provide a means for interactive and spatio-semantic queries and aggregations for numerous applications and use cases. Hence, they act as a central information hub and connect different applications by aligning information exchange with the city model entities. As briefly mentioned in section 1.1, semantic 3D city models are used worldwide in numerous applications. However, most of the applications require handling with different time-dependent properties, which are not supported in current generation 3D city modelling standards. This subsection reviews four key application domains where semantic 3D city models are widely used. The section further investigates types of time-dependent properties that several applications require. Based on the review, requirements are gathered for extensions of semantic 3D city models for supporting different types of time-dependent properties.

### 2.1.1 Smart Cities and Digital Twins

According to the Department of Economic and Social Affairs of the United Nations Secretariat, 53% of the world's population resided in urban areas in 2014, and the world continues to urbanize rapidly<sup>5</sup>. With an increasing urban population, it is highly essential to use the best technology and tools available to manage the development and operation of cities efficiently. Smart Cities is an emerging concept that *"relies on advanced data processing with the goals of making governance more efficient, citizens happier, businesses more prosperous and the environment more sustainable"* (Yin et al. 2015). This concept allows the efficient management of city resources like energy, water, and mobility with the help of advanced information and communication technologies including Sensors and the Internet of Things (IoT) (Hancke et al. 2013), Big Data (Hashem et al. 2016), Cloud Computing (Suciu et al. 2013), and also geospatial technologies (Roche 2014). Many cities all over the world are already developing their smart infrastructures. Commercial implementations include IBM Smarter Planet<sup>6</sup>, CityNext<sup>7</sup> from Microsoft, and The Internet of Everything for Cities by CISCO<sup>8</sup>. Some of the projects are also run as a collaboration among universities, companies and city councils such as Smart Sustainable Districts<sup>9</sup>, under Climate-KIC of the European Institute of Innovation & Technology (EIT) and the project EU ICT 30-2015<sup>10</sup> (Internet of Things and Platforms for Connected Smart Objects) funded by the European Union Horizon 2020 Programme. Sensors and IoT devices play an essential role in such smart infrastructures. They provide detailed information by sensing the environment in real-time. Many application domains, including intelligent energy management and smart grids, traffic management, home and industrial automation, and others benefit from the use of real-time sensor observations. These sensors can be stationary such as Smart Meters (Patel et al. 2016) and weather stations (Quarati et al. 2017). Some of the sensors can also be mobile such as a sensor continuously moving for measuring air quality in different parts of a city (Hagemann et al. 2014). There is also another category of virtual sensors which are not necessarily located physically. However, their sensing observations can be studied to get better information about our surroundings

<sup>5</sup><https://population.un.org/wup/>

<sup>6</sup><https://www.ibm.com/smarterplanet/us/en/>

<sup>7</sup><https://partner.microsoft.com/en-us/solutions/citynext>

<sup>8</sup>[https://www.cisco.com/c/dam/en\\_us/solutions/industries/docs/gov/everything-for-cities.pdf](https://www.cisco.com/c/dam/en_us/solutions/industries/docs/gov/everything-for-cities.pdf)

<sup>9</sup><https://www.climate-kic.org/areas-of-focus/urban-transitions/our-initiatives/smart-sustainable-districts/>

<sup>10</sup><https://ec.europa.eu/info/funding-tenders/opportunities/portal/screen/opportunities/topic-details/ict-30-2015>



and environment. For example, real-time social media analytic like Twitter feeds can be used for behavioural and sentiment analysis and to make better decisions (Batty et al. 2012).



**Figure 2.1:** Illustration of a Smart City concept. Image source: [www.houseofbots.com](http://www.houseofbots.com)

With the technological advancement and growing popularity in the field of sensors and IoT, another term, that is gaining attention, is "Digital Twins". A Digital Twin is a digital equivalent of a physical asset, which collects information via sensors and IoT devices and applies advanced analytics and artificial intelligence to gain real-time insights about the physical asset's performance, operation or profitability (Grieves and Vickers 2017). Digital Twins are also vital for Cyber-Physical Systems. The Cyber-Physical Systems comprise of smart machines and storage systems and exchange information, trigger actions under certain events and hence, control each other independently in an autonomous way (Frontoni et al. 2018). It allows bridging the virtual and physical worlds together and therefore, enable users improving their decision-making and reducing risks by predicting issues before occurrence (Mohammadi and Taylor 2017). The Digital Twin technology is spreading from its industrial origins to the Smart City environment. This approach allows a more holistic approach in terms of cross-vertical optimisation of the design, management, and operation of urban infrastructure (Batty 2018; Tomko and Winter 2019). Benefits include operational cost savings, energy efficiencies, increased resilience, improved sustainability, and a positive impact on the economic growth. Digital twin solutions include spatial modelling of the built environment, mathematical models of electric and mechanical systems, and real-time sensor data (crowd) sourced from IoT platforms. Typical use cases include flood risk modelling, multi-building energy management, renewable energy optimisation, traffic flow optimisation, occupancy tracking and evacuation simulations, and the generative design of city extensions.

There are several Smart City projects focused on developing Digital Twins of cities by integrating various city objects with real-time sensor data streams (Table 2.1). Semantic 3D city models play a central role in such Smart City projects allowing precise representation of the physical reality. For example, the Future City Pilot Phase 1<sup>11</sup> is an Interoperability Program initiated by the Open Geospatial Consortium (OGC) in collaboration with the buildingSMART International (bSI). The

<sup>11</sup><https://www.opengeospatial.org/projects/initiatives/fcp1>

Project name	Location	Use Cases	3D City Models	Integration of sensor data
OGC Future City Pilot Phase 1	London, U.K. Rennes, France	Adult care with live weather information, solar energy potential	✓	✓
Smart District Data Infrastructure	London, U.K.	Energy efficiency, Smart Park, Sentiment analyses	✓	✓
Digital Twin Munich	Munich, Germany	Urban, traffic, and environmental planning	✓	✓
Kalatatama Digital Twins Project	Helsinki, Finland	Solar energy potential, wind simulation	✓	✓
Digital Twin Rotterdam	Rotterdam, the Netherlands	Urban Planning and development	✓	✓
Virtual Singapore	Singapore	Urban Planning, collaboration and decision-making, improved accessibility	✓	✓
Fishermans Bend urban renewal project	Melbourne, Australia	Traffic flows, energy demand estimation	✓	✓

**Table 2.1:** List of Smart City projects aiming on developing Digital Twins of cities. The table shows the Project name, Project location, and Use cases. The last two columns show that for their use cases, all the projects use semantic 3D city models and require integration of city models with real-time sensor and IoT information.

pilot aims at demonstrating the use of international standards such as CityGML and IFC together can provide stakeholders with information, knowledge and insight enhancing financial, environmental, and social outcomes for citizens living in cities. One of the objectives of the pilot is to demonstrate “*how dynamic city models can provide better services to the citizens as well as can help to perform better analysis?*”. This use case requires developing the Digital Twin of the Greenwich district in London allowing static information such as buildings or houses with elderly citizens having special needs to be integrated with real-time information such as measurements of temperature and air humidity being retrieved by a weather station. Such potential integration within council-owned assets shall lead to better decision-making in the cases of extreme weather and other emergency scenarios matching human needs to the right housing or resources. The CityGML standard is used for representing the buildings of the district, and the OGC Sensor Observation Service (SOS) (Bröring et al. 2012) (c.f. section 2.2.1) is used for retrieving real-time measurements from the weather station. Similarly, the project Smart District Data Infrastructure<sup>12</sup> (SDDI) focuses on developing Smart City infrastructures for specific districts within selected European cities. This project runs within the Smart Sustainable

<sup>12</sup><https://www.lrg.tum.de/en/gis/projects/smart-district-data-infrastructure/>

Districts Program of the Climate-KIC of the European Institute for Innovation and Technology (EIT). The SDDI framework allows integrating diverse components such as multiple stakeholders, sensors, IoT devices, simulation tools with a virtual district model representing the physical reality of the district. One of its first implementations is based in the district Queen Elizabeth Olympic Park in London. In this project, the owner of the district, London Legacy Development Corporation (LLDC), have identified different use cases related to the reduction of resource and energy usage, reduction of waste, reduction of emissions, improvements of well-being, mobility, and in general concerning efficiency. To achieve its goals, the Digital Twin of the Olympic Park comprises a semantic 3D model (including buildings, streets, vegetation, and water bodies, etc.) based on the OGC CityGML standard. Within the Digital Twin, the 3D city objects are linked with real-time sensor data streams. For example, Smart Meters installed in important buildings measuring their electricity and gas consumption, weather stations located in the park measuring weather properties (such as temperature, humidity, and wind speed), and so on (more details about this project are given in Chapter 8). There is also another ambitious project in Munich called Digital Twin Munich<sup>13</sup> (Digitaler Zwilling in München). This project is initiated by the City of Munich (Landeshauptstadt München) and is funded by the Federal Ministry of Transport and Digital Infrastructure, Germany. The Digital Twin intends to create a complete digital image of Munich in a 3D environment. In addition to a 3D presentation, this "digital copy" will contain extensive information (including real-time data from sensors and IoT devices). The aim is to improve the basis for urban, traffic and environmental planning, for example, by modelling what-if scenarios. This project also aims to integrate real-time information from various sensors and IoT devices with the 3D City Model of Munich in a standardised and interoperable manner.

The Kalasatama Digital Twins Project<sup>14</sup>, a part of the Helsinki 3D+ project initiated by the Government of Finland, aims at producing high-quality digital twin city models of the Kalasatama area for smart urban development. Within the project, the digital twins have already been created using (i) the CityGML standard-based semantic city information model and (ii) a photorealistic mesh model. The 3D city model is already used for different use cases such as calculating the city's solar energy potential and assessing the effect of wind on the high-rise buildings of Kalasatama. In the future, a more significant number of the buildings will have an API to provide real-time data, replacing simulations with real measurements as quoted by (Ruohomäki et al. 2018). The Digital Twin Rotterdam project<sup>15</sup>, initiated by the City of Rotterdam, also aims at developing the Digital Twin of the Rotterdam city. Within the project, a 3D model has already been developed, comprising not only the buildings, but also trees, lampposts, and underground cables and pipes. The Digital Twin of Rotterdam allows integrating real-time data from various IoT devices with the city objects. For example, the status of municipal waste containers equipped with sensors, real-time availability of parking spaces within the parking lots, and real-time display of traffic flow, including lifting bridges where water and road traffic intersect. Virtual Singapore<sup>16</sup>, created by the National Research Foundation of Singapore, is another project in the same direction. The project offers 3D semantic modelling, in which the meaning of data can be related to the real world, displaying land attributes or the characteristics of different forms of transport, or the components of buildings and infrastructures. And, apart from the normal map and land data, the platform also incorporates other real-time dynamics, as well as information about demographics, climate or traffic, making it a tool that offers enormous potential and which can be used in many different ways. Similarly, another ongoing project "Fishermans Bend

<sup>13</sup><https://muenchen.digital/twin/>

<sup>14</sup>[https://www.hel.fi/static/liitteet-2019/Kaupunginkanslia/Helsinki3D\\_Kalasatama\\_Digital\\_Twins.pdf](https://www.hel.fi/static/liitteet-2019/Kaupunginkanslia/Helsinki3D_Kalasatama_Digital_Twins.pdf)

<sup>15</sup><https://www.3droterdam.nl/>

<sup>16</sup><https://www.nrf.gov.sg/programmes/virtual-singapore>

urban renewal project"<sup>17</sup> has been proposed by Land Use Victoria and the University of Melbourne in Australia. The project aims to develop essential design and condition information for physical assets above and below ground, plus legal boundaries to better manage current and future developments during the 30-year urban renewal project. The representation of physical information is based on semantic information modelling. On the city scale, digital twins will illustrate traffic flows and demand for resources like electricity and water throughout the day.

As described, there are several Digital Twin initiatives being implemented throughout the world and such implementations will continue to grow in the future. According to the recent research from ABI Research<sup>18</sup>, "*the installed base of Digital Twin and city modelling deployments is expected to grow from just a handful of early implementations in 2019 to more than 500 by 2025*". All the above-mentioned Digital Twin initiatives consider semantic 3D city models such as CityGML and IFC as an integral component of their infrastructures. Hence, it is important that the next generation semantic 3D city models support seamless integration with sensors and IoT devices. It will allow measuring real-time observations associated with city objects, which can be used for performing advanced analytic. This brings us with two essential requirement described as follows:

#### **2.1.1.1 Requirement R1: Integration of sensor observations with city object properties**

Ubiquitous sensor and IoT devices are used to monitor public infrastructures such as bridges, roads, and buildings and enable more efficient use of resources based on real-time observations collected by them. For example, Smart Meters installed in buildings in a specific district not only provide real-time insights about energy usage to the administration but also helps to estimate energy demand for the district. However, to perform such estimations, it is essential to define relations between city objects and the respective sensor observations. Hence, the specific city objects are required to be linked explicitly with sensors and IoT platforms and APIs. However, an important aspect to consider is how such links between city objects and sensor platforms can be established. The following discussions in this thesis explore different possibilities for integrating sensor and IoT observation with city objects.

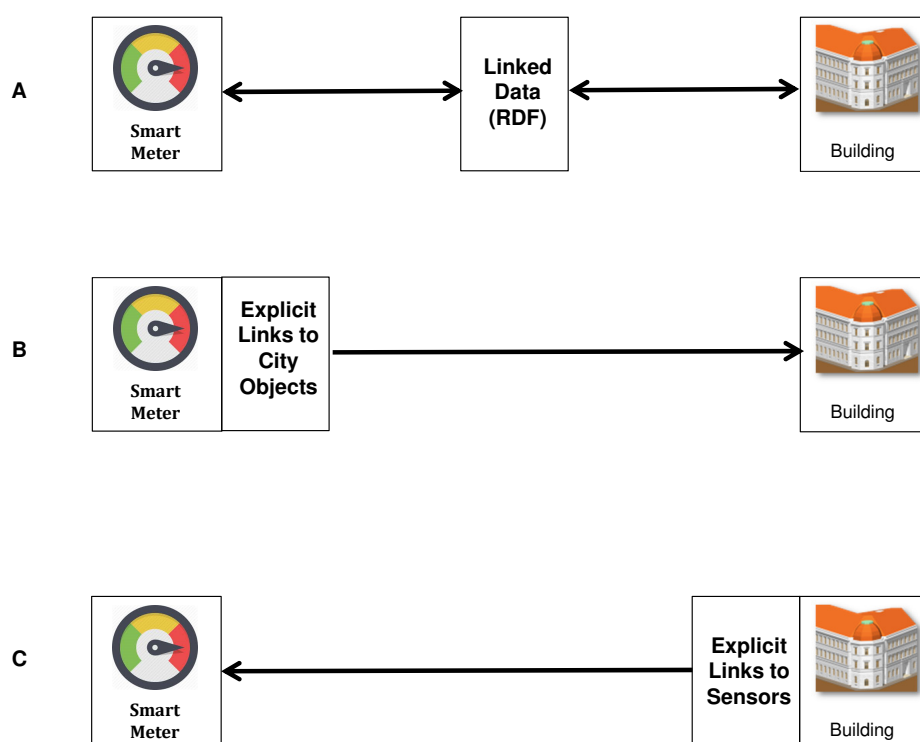
As shown in figure 2.2, one option (represented as 'A') is to link them by defining additional semantics such as Linked Data<sup>19</sup>. Linked Data provides a structured way to define semantic information for different interlinking types of data. It can be built upon standard web technologies such as Hypertext Transfer Protocol (HTTP), Resource Description Framework (RDF), and Uniform Resource Identifiers (URIs). It can also be based on Ontology representation languages such as Web Ontology Language (OWL) and Unified Modelling Language (UML). However, a probable issue with this approach is that it would add another representation to the already existing representations of sensors as well as city models. Furthermore, this would require the additional management for the semantics of Linked Data.

The second possibility (represented as 'B' in figure 2.2) is defining explicit links to city objects within the representations of sensors. There are several well-defined standards for representing sensors and their observations (c.f. section 2.2.1). This approach would allow associating time-dynamic observations directly to the city objects. For example, a sensor interface measuring real-time gas consumption of a building may comprise of direct links to the building object or building's room where the Smart Meter is installed. However, current sensor modelling standards do not allow defining the sensor's relation with the city object property. In this way, we can only name the feature of interest,

<sup>17</sup><https://www.fishermansbend.vic.gov.au/>

<sup>18</sup><https://www.abiresearch.com/market-research/product/1033835-digital-twins-smart-cities-and-urban-model/>

<sup>19</sup><http://linkeddata.org/>



**Figure 2.2:** Possible ways for linking sensors with city objects.

which typically is an object and not the property of an object. For instance, if a building object has an attribute "gas\_consumption", the issue is that the sensor operator would have to maintain the link(s) to different geospatial datasets (e.g. INSPIRE, CityGML, and IFC), that are not under his/her control.

The third option (represented as 'C') is defining explicit links to real-time observations within the city object. In this way, city object attributes can also be associated with time-dynamic sensor observations. For example, the attribute "gas\_consumption" (as mentioned above) will become dynamic by having defined links to the interface for the Smart Meter observations. In this way, 3D city modelling applications can represent dynamic variations of a specific city object property. Similarly, in an application related to Civil Engineering, Bridge Deformation Monitoring (Davila Delgado et al. 2017) is an example of predictive maintenance which involves a systematic measurement and tracking of the alteration in the shape or dimensions of the bridge as a result of stresses induced by applied loads over some time. In such cases, an explicit linking of connected sensors installed at the bridge would allow overriding the static attribute values of the Digital Twin of the Bridge object.

### 2.1.1.2 Requirement R2: Managing events and alerts

Digital Twins enable improving decision-making by predicting issues before occurrence and at the same time by notifying as soon as the issue has occurred. The former aspect of predicting issues is mostly realised by using sophisticated machine learning algorithms. However, in cases of the latter

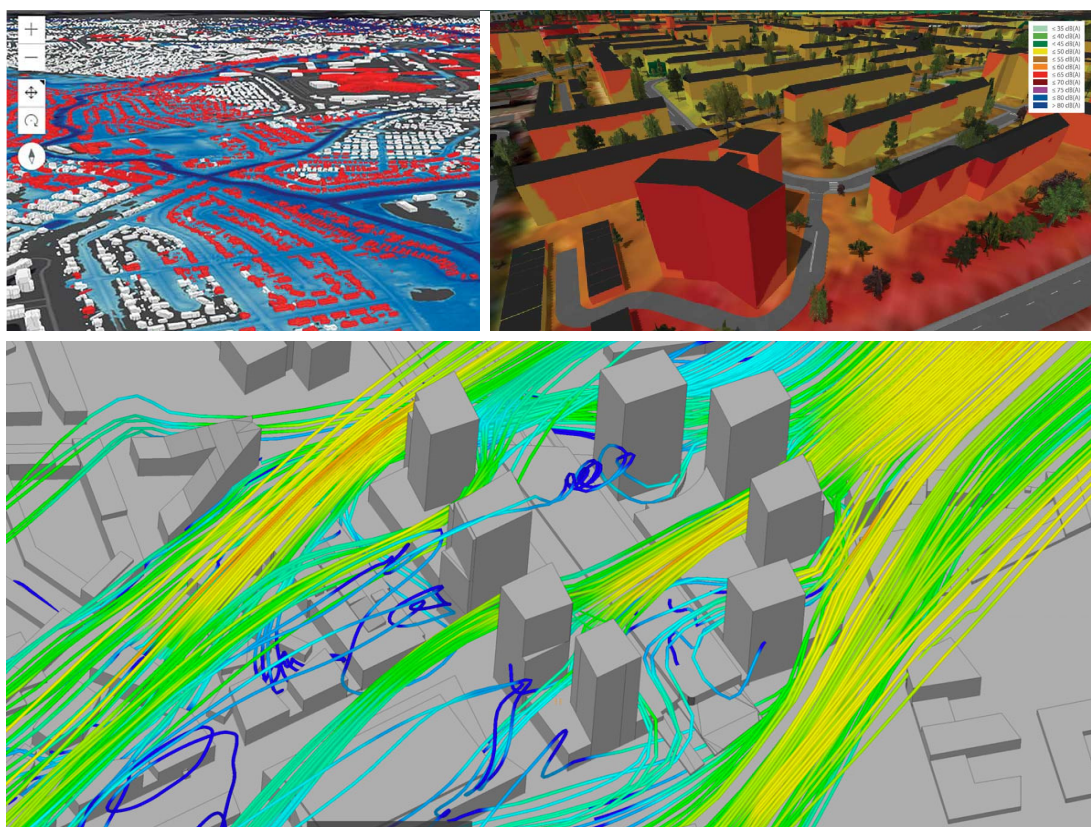
aspect, synchronous communication with sensors may be insufficient as the applications depend on asynchronous events, such as external communications, status changes, natural phenomena, or data updates. In these cases, automatic notification by using a publish/subscribe scheme are required where clients are notified with the desired information. For example, in the case of water management systems, the notifications of water gauges exceeding a threshold may prevent critical flood scenarios. Similarly, real-time processing of air quality data may alert citizens and administrators for dangerous air pollution situations. The explicit linking of city objects with sensors and IoT enable users and applications to subscribe to real-time data streams and be alerted in case of occurrence of any particular phenomenon.

### 2.1.2 Urban Simulations

*"A simulation is an approximate imitation of the reality in which a particular set of conditions is created artificially in order to study or experience something that could exist in reality"*<sup>20</sup>. In the urban context, simulators are often used by urban planners to understand how cities and city entities are likely to evolve in response to various policy decisions. UrbanSim (Borning et al. 2008) and Land Use Evolution and Impact Assessment Model (LEAM) (Deal 2006) are the most commonly used examples of large-scale urban simulation models that are used by municipal planning agencies and authorities for land use and transportation planning. Semantic 3D city models are also considered as an important source of information for different types of urban simulations. Prominent examples are noise propagation simulation and mapping (Czerwinski et al. 2007), urban and telecommunication planning (Königer and Bartel 1998; Knapp and Coors 2007), disaster management (Zlatanova and Holweg 2004; Kolbe et al. 2008; Morel and Gesquière 2014), real-time simulations for training and autonomous driving (Randt et al. 2007; Schwab and Kolbe 2019), indoor navigation (Becker et al. 2009; Mäs et al. 2006), energy demand estimations (Strzalka et al. 2011; Agugiaro 2016; Kaden and Kolbe 2013), solar potential simulation (Zahn 2015; Biljecki et al. 2015a; Salimzadeh et al. 2018), flood simulations (Amirebrahimi et al. 2016; Schulte and Coors 2009), and wind analysis (Cousins 2017). (Biljecki et al. 2015b) also provide an extensive review of different simulations based on 3D city models. The authors classified the use cases in two broad categories. The first category is the non-visualisation use cases, which do not require the visualisation of the 3D city model as well as the results of the operations that the use case comprises. The second category is the visualisation-based use cases, where visualisation of the city model and the results play an important role. For example, solar potential analysis is an instance of non-visualisation use case in which the simulation results can be visualised but this is not essential to achieve the purpose of the use case. The results can be stored in a database, which can be queried without the need of being visualized. On the other hand, the applications related to navigation, gaming, and also urban planning fall into the category of visualisation-based use cases. In these cases, the visualisation of the objects is crucial, and the use cases would not make much sense without it. In total, the authors identified more than 29 use cases including more than 100 applications, which are arranged into these categories.

Simulation specific data can be represented by particular geo-objects and properties within the city models. Further, the results of simulations can be fed back to the original 3D city models for thematic enrichment and data fusion by data from different disciplines. Most simulations generate properties which vary with time. However, current 3D city models represent such time-varying results as static values. For example, the Solar Potential Analysis Tool (Zahn 2015) developed at the Chair

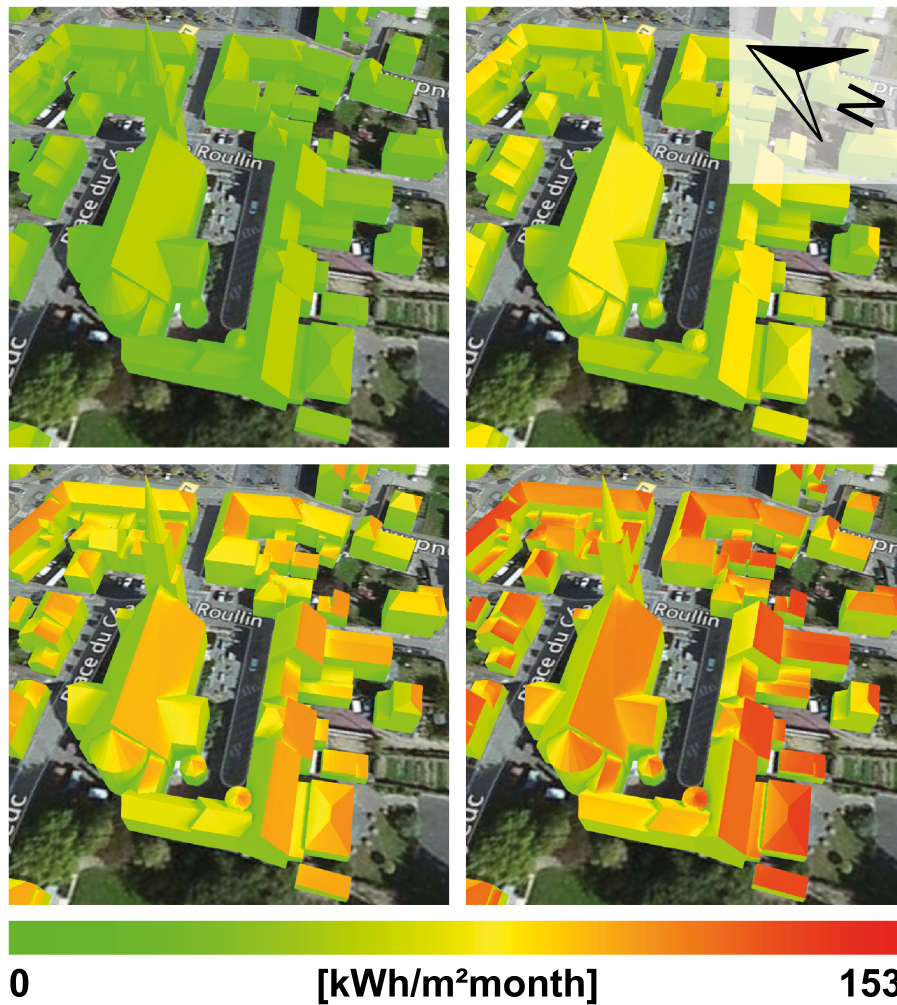
<sup>20</sup><https://www.oxfordlearnersdictionaries.com/definition/english/simulation>



**Figure 2.3:** Examples of urban simulations with semantic 3D city models. The top-left image shows the flood inundation simulation with 3D city models. The top-right image is the screenshot for noise propagation simulation (screenshot taken from ESRI), and the bottom image is the wind flow analysis with 3D buildings (screenshot taken from Helsinki Kalasatama project)

of Geoinformatics, Technical University of Munich is widely used for assessing and estimating solar energy production for the roofs and façades of the 3D building objects. The simulation tool operates on CityGML-based semantic 3D city models. By combining a transition model, sun position calculation, and an approximation of the skydome, the solar power from direct, diffuse, and global sunlight irradiation are estimated for individual months and years. The shadowing effects of the surrounding topographic features are considered by applying a ray-tracing approach. The Sky View Factor (SVF), a measure indicating the visible fraction of the sky hemisphere, is determined for each surface. As a result, each building surface is enriched by its irradiation values. These are also aggregated to the building level. Finally, a point cloud is generated from sampling points that have been produced for each building surface by the simulation. Each point is parametrised with the direct, diffuse, and global irradiation values over the different months.

Figure 2.4 shows the textures of the global irradiation values for the months: February, March, April and May. The colour gradient ranges from dark green (no irradiation) over yellow to red (maximum monthly irradiation of  $153 \text{ kWh/m}^2$ ). This type of visualisation allows for quick identification of suitable areas for photovoltaic energy production. However, to be able to perform profound analyses,



**Figure 2.4:** Textures of the global irradiation values for the months (left to right): February, March, April, and May generated for a district in Rennes, France. Screenshot taken from the 3DCityDB-Web-Map-Client.

the values are stored as attributes in the city model in addition to the purely visual representation of the solar irradiation values as textures. Based on the point grid results, the different temporal resolutions for direct, diffuse and global irradiation are computed and stored as generic attributes for each spatial aggregation level (wall, roof surfaces and building) in the city model. Currently, the temporal classification of a simulation result parameter is encoded in the attribute name as a suffix. For instance, an attribute named *globalRadMonth\_01* denotes the aggregate global irradiation estimate on a specific feature for January. Similarly, individual generic attributes can be defined for storing the irradiation values for different months. In the same way, the attributes can be defined for the aggregate global irradiation estimate on a specific feature for the entire year such as *globalRadYear*.

As depicted in figure 2.5, the monthly estimates for the three irradiation types direct, diffuse, and global for a facade surface are visualised as an attribute table. This data visualisation has some





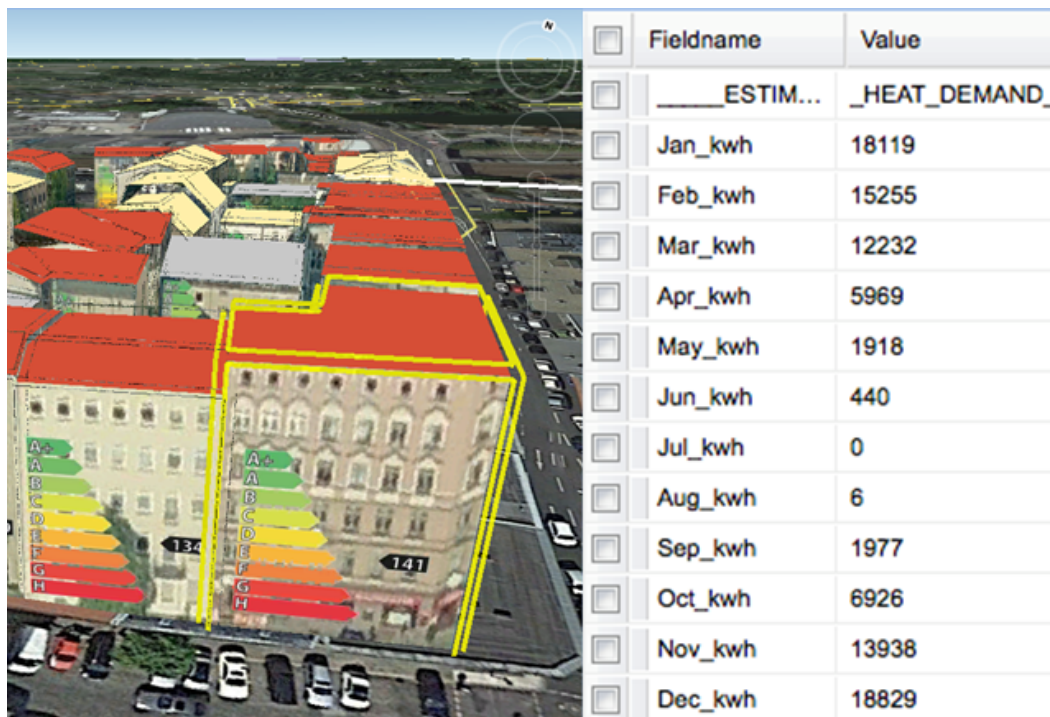
**Figure 2.5:** Tabular representation of monthly irradiation values of a building in Rennes, France. Screenshot taken from the 3DCityDB-Web-Map-Client.

limitations, as it makes it hard for the viewer to grasp relevant information from the data quickly. For instance, the table representation does not allow to observe the full range of the data directly. It would usually not be possible to view the monthly attributes for direct, diffuse and global irradiation at the same time, because a table of that size would not fit on the screen. More difficulties arise when trying to compare the three variables to each other or recognise monthly deviations. The usability of such analyses will drastically be improved, if city model standards allow time-dependent variations of such result values to a common generic attribute.

Similarly, concerning energy simulations, (Kaden and Kolbe 2013) explain the application of the CityGML standard for city-wide estimation of the energy demands of buildings. The authors discuss the project "Energy Atlas Berlin". The project aims at determining the suitability of all individual roof surfaces for each of the 550,000 buildings in Berlin for the production of photovoltaic and solar thermal energy. It also integrates methods for energy demand estimation, which includes heating energy, electrical energy and warm water. It also assesses the energetic retrofitting possibilities on the individual building level. As shown in Figure 2.6, the authorities can explore the energy demand of individual buildings for different months of a year. In this application, one (static) attribute for each month is explicitly modelled. However, in the current version of CityGML (version 2.0), it is not possible to override or replace the same attribute with respect to time.

### 2.1.2.1 Requirement R3: Supporting time-series in-line within city objects

Most of the simulations involve time-dependent attributes, for example, monthly values of solar irradiations or energy demand estimations for a building (as illustrated in the previous section). Since



**Figure 2.6:** Visualisation of estimated heat demand values of a building in Berlin, Germany. Screenshot taken from 3DCityDB-Web-Map-Client.

these simulation results are associated with city object properties, the respective city object properties must be enriched with these time-dynamic values. The results of these simulations are sometimes stored in databases or encoded in external files such as Microsoft Excel or CSV. They may also be retrieved by an external API or web interface of simulation software. In these cases, explicit links can be established from city object properties to the simulation results. However, to perform detailed realistic simulations, for example, cross-domain exchanging of simulation results with city objects for enhancing disaster management or energy assessment, it is essential to model the precise description of the time-series. It is also relevant to create a snapshot of the state(s) of a city model, including time-varying data for documentation and archiving. This approach requires modelling a data structure using which time-series data can be exchanged with appropriate metadata to allow correct machine interpretation and thus proper use for further analysis.

#### 2.1.2.2 Requirement R4: Supporting complex patterns and schedules

In many applications, it is not sufficient to provide a means for the tabulation of time-value pairs. The applications may require patterns to represent dynamic variations of properties based on statistics and general rules. For example, energy applications can be used to study patterns in the energy consumption of a building for weekdays, weekends, public holidays, or even customised period (e.g. between 6 pm and 10 pm). In these cases, time may be defined for a non-specific year (e.g. averages over many years), but still classified by the relative time of a year. For example, January monthly summaries for the energy consumption of a building might be described as "all-Januaries

2001-2010". Similarly, the energy consumption values may reflect generic patterns for individual weekdays/weekends in a week or a month. Another example scenario may also be determining patterns for specific seasons (such as spring, summer, autumn and winter) over ten years. In different simulations, such time-series can also be used as a basis for defining schedules. For example, schedules in the energy simulations may be required for specifying setpoint values for the heating and cooling systems, or for setting the operational schedules of energy systems, ventilation, lighting, and electrical appliances. Similarly, in the cases of traffic analysis, a public bus line following a schedule can also have a repeating trajectory.

### 2.1.3 Mobility

With an increasingly urban population, reducing congestion, accidents, and pollution has become a common challenge to all major cities in the world. According to the European Commission<sup>21</sup>, *"Urban mobility accounts for 40% of all CO2 emissions of road transport and up to 70% of other pollutants from transport in the European Union"*. It leads city authorities and urban planners to adopt sustainable solutions for not only reducing pollution but also allowing visitors and residents to have a more comfortable and enjoyable everyday city experience. Many cities worldwide have adopted the so-called "Smart Mobility" (Benevolo et al. 2016) solutions offering multimodal capabilities which bundle transport options such as public transport, car-sharing, bicycle-sharing, and ride-hailing (Bertolini and Le Clercq 2003; Nobis 2006; Luginger 2016). There are also many mobility simulations in major metro cities such as Helsinki<sup>22</sup> and Dublin<sup>23</sup> providing indicators for the impact of mobility solutions on accessibility, metro/rail ridership, required parking space, congestion and CO2 emissions. Such mobility simulations are also beneficial for urban planning and disaster management, especially in planning, managing, and optimising pedestrian flows in public buildings like airports, railway stations, shopping malls, and stadiums. Evacuation planning is an important aspect to determine how an evacuation plan will work in the events of a disaster such as an earthquake and fire. Evacuation simulations (Chu et al. 2019; Almeida et al. 2013; Okaya and Takahashi 2013) allow determining measures to create better emergency management for residence, offices, and in general, the community. Besides, mobility simulations also help to manage the movements of visitors during an event. For example, the Smart District Data Infrastructure (SDDI) project involves a pedestrian flow simulation (Yang et al. 2020; Yang et al. 2019) to manage pedestrians' walking behaviour, flow and peak demands during specific events organised at the Queen Elizabeth Olympic Park, London. Other notable examples of use cases for mobility simulations are navigation for robots (Kwak and Park 2012), aviation or maritime traffic monitoring (Bosson and Lauderdale 2018), wildlife tracking and conservation (Ayele et al. 2018), and autonomous driving<sup>24</sup>.

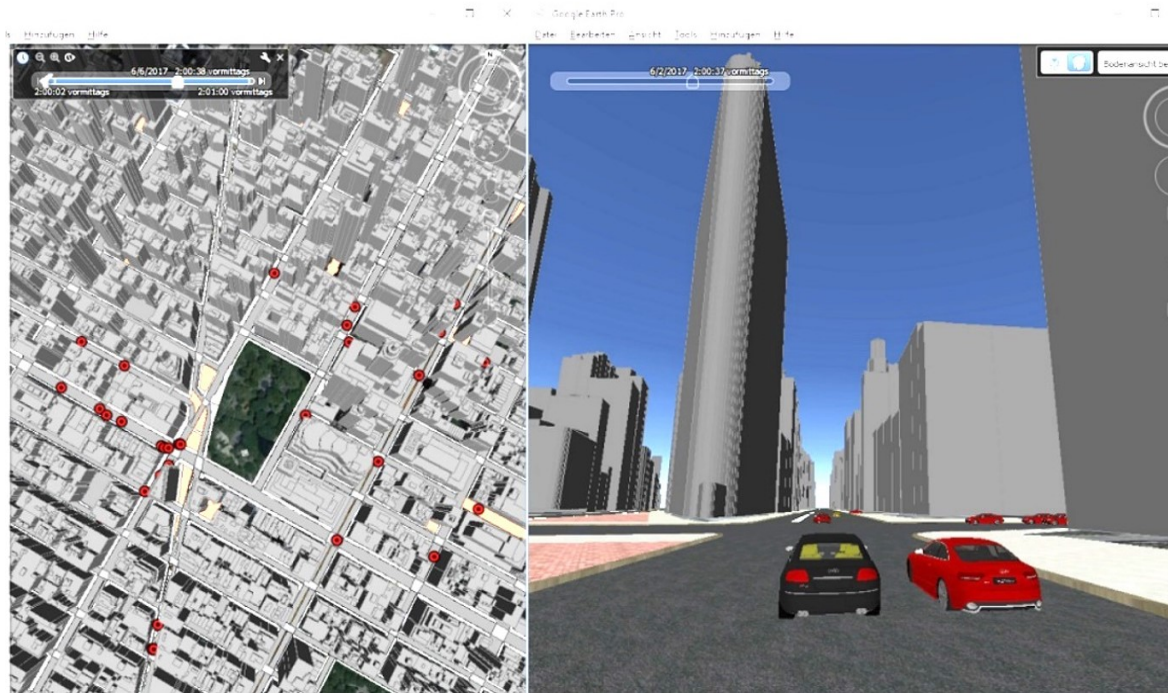
Semantic 3D city models also play a vital role in mobility applications and simulations. One of the significant advantages of 3D city models is that they allow detailed 3D geometry representation of city objects, which is more realistic than the symbolic representation provided by traditional 2D maps (Oulasvirta et al. 2009; Schilling et al. 2005). Furthermore, city objects enriched with their thematic, semantic, and cognitive properties enhance processing and visualisation for navigation applications. For example, a building of type 'restaurant' and 'brand' Starbucks offers more navigational cues than a block of grey, anonymous residential building (Nedkov 2012). Such semantically rich objects

<sup>21</sup>[https://ec.europa.eu/transport/themes/urban/urban\\_mobility\\_en](https://ec.europa.eu/transport/themes/urban/urban_mobility_en)

<sup>22</sup><https://www.itf-oecd.org/shared-mobility-simulations-helsinki>

<sup>23</sup>[https://www.oecd-ilibrary.org/transport/shared-mobility-simulations-for-dublin\\_e7b26d59-en](https://www.oecd-ilibrary.org/transport/shared-mobility-simulations-for-dublin_e7b26d59-en)

<sup>24</sup>[https://elib.dlr.de/118438/1/SUMO\\_proceedings\\_online.pdf](https://elib.dlr.de/118438/1/SUMO_proceedings_online.pdf)



**Figure 2.7:** Illustration of mobility applications. Screenshot taken from (Ruhdorfer 2017)

are also beneficial for landmark-based navigations (Krukar et al. 2017). (Ruhdorfer 2017; Beil and Kolbe 2017) also demonstrate how the detailed geometrical and semantic representation of street space can hugely benefit a variety of applications, including traffic analysis, visibility analysis, and pedestrian flow simulations. The authors showcased various demonstrations with a semantic 3D city model of New York City by generating a detailed 3D street space model for the entire city. Similarly, (Schwab and Kolbe 2019) also highlight the importance of detailed street space models and discuss the feasibility of semantic 3D city models in the context of autonomous driving. The ability of 3D city models to define navigable spaces such as steps and ramps also make them suitable for performing pedestrian flow and movement simulations (Slingsby and Raper 2008; Scholl 2019). If the 3D city model contains information on the interior of buildings, this information can also be used for wayfinding and accessibility applications (Khan 2015; Isikdag et al. 2013; Kim et al. 2014).

### 2.1.3.1 Requirement R5: Integration and overlay of dynamics of moving objects

Most of the mobility applications described previously involve objects changing their properties with time. For a navigation application, there might be a car or a tram changing its location over the day. In an evacuation simulation, there might be a group of people coming out of different exits of the building at different points in time. However, some cases of mobility applications also require objects changing their semantic properties over time. A traffic monitoring camera generates videos of traffic flow over regular intervals of a day. Similarly, a traffic detector inductive loop installed beneath the road surface at a traffic junction produces the statistics of traffic speed and density over regular intervals of a day. In other cases, applications require dealing with changing geometry and thematic

attributes with time. For example, the project 'AERO-TRAM' (Hagemann et al. 2014) involve a mobile measurement system mounted on the roof of a tram and examine the spatial distribution of pollutants (such as particle number concentrations and nitrogen oxides). Such cases require mapping locations and sensor observations with time. The other example could be to access data on moving objects that passed through a particular area at a specific time after a disaster, therefore gathering information on density and flow of people and vehicles in a more timely manner. Hence, it is crucial to extend city models allowing moving objects to be represented within them.

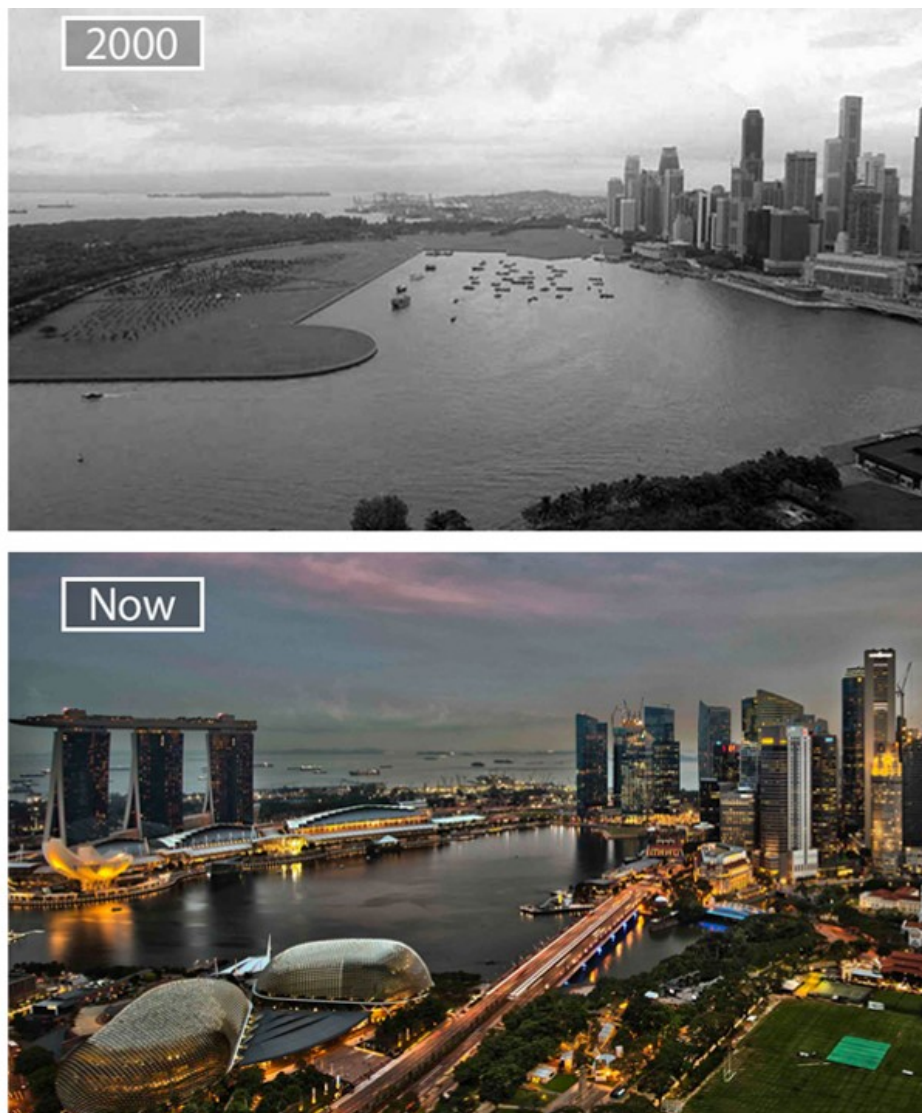
Similarly, the use cases of Smart Mobility require a well-connected infrastructure where different objects can communicate with each other. For example, the concept Vehicle-to-Everything<sup>25</sup> allows a moving vehicle to communicate with a different moving vehicle or any other static object such as a temporary roadside infrastructure that may affect the vehicle. Such concepts rely on real-time information obtained from hundreds of sensors installed onboard vehicles, transport infrastructures such as traffic light, temporary infrastructures such as a construction site, and also crowd-sourced information such as GPS feeds. In similar ways, as highlighted by (Schwab and Kolbe 2019), autonomous driving requires that every object within the moving vehicle sight becomes relevant as those objects can reflect sound and electromagnetic waves. For realising such concepts, the static, as well as moving objects within a city model, must be capable of dealing with dynamic information. Hence, it is essential to establish an overlay and integration of moving objects with the static objects of the city models.

#### 2.1.4 Urban Development

Cities change over time, and so do the city objects as shown in figure 2.8. For example, over a time sequence, a building may be constructed, modified, destructed or even replaced by other ones. However, at each stage, it is crucial to record and document the changes over time. Lessons from the past successes and failures serve as guidance for the future planning of the cities. Hence, to understand urban evolution and develop sustainable and durable cities, changes in cities are studied by historians and urban planners. Several studies in the past have been done for demonstrating the historical evolution, for example, 4D visualisation of Bastion fort (Rizvic et al., 2015), digital reconstruction of the city of Pompeii (Dell'Unto et al., 2013) and the relief maps of the past augmented with various video displays (Priestnall et al., 2012). However, some use cases also require rebuilding the past of a city, which is not an easy task. Historians and researchers make use of historical artefacts available in the present to reconstruct the scenarios in the past. For example, the Venice time machine project (Kaplan, 2015) intends to build an extensive document corpus by digitising historical archives for studying the last 1000 years of Venice. The researchers reconstruct one or more city objects and even the entire city by exploring the numerous documents available to them today. These documents include project plans, aerial views, municipal council meetings and multimedia documents like videos, photographs, old postal cards, paintings etc. Some of these documents are evidence of the actual buildings of the past, whereas others are artistic renderings. While studying the urban evolution, these traces are used to comprehend various socio-economic and political aspects of the given period.

With the advancement in technology (especially in the field of 3D city models), it becomes easier to build large scale flexible city models in a virtual environment compared to their equivalents in the real world. As a result, semantic 3D city models have been used to study historical information. For example, the Virtual Leodium project (Pfeiffer et al. 2013) allows studying the evolution of the city of

<sup>25</sup><https://www.reportsnreports.com/reports/2058475-the-v2x-vehicle-to-everything-communications-ecosystem-2019-2030-opportunities-challenges-strategies-forecasts.html>



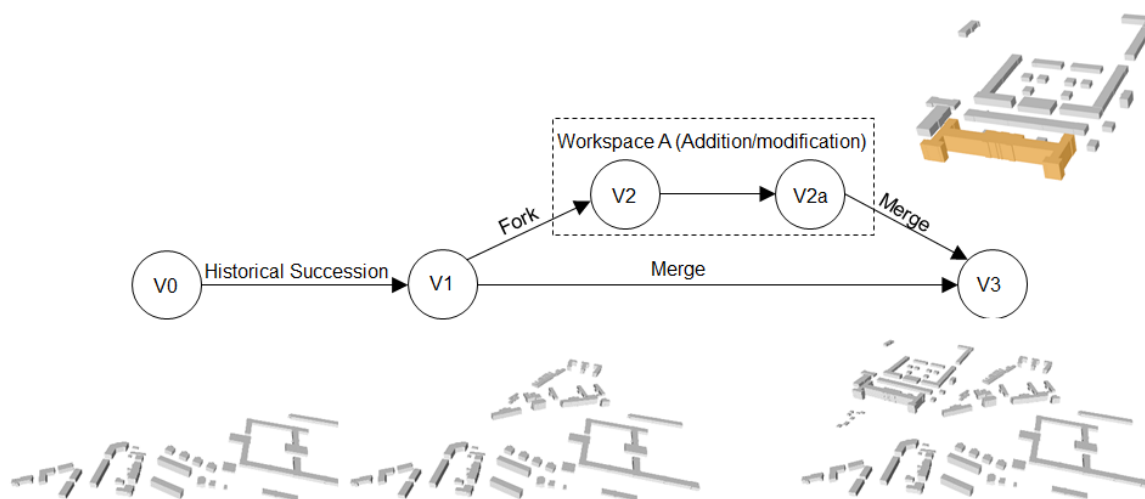
**Figure 2.8:** Illustration of the evolution of Singapore City. Screenshot taken from [www.pinterest.com](http://www.pinterest.com)

Liege in Belgium based on the CityGML standard. The study enables representing the sequence of versions of a “reality” at a specific time or in a particular time sequence.

Apart from linear sequencing, some use cases also address to work with planning alternatives or parallels of buildings or other structures, e.g. for comparison by a reviewing body. The planning alternatives are not different versions of actual structures at different times but different structures that might be substituted for one another. In urban planning scenarios, various planning authorities can also work with alternative planned versions at the same time to insert a newly generated object or delete or update any existing object (Samuel et al. 2018).

### 2.1.4.1 Requirement R6: Managing historic versions

Semantic 3D city models are an essential source of information in applications related to urban planning, architecture, business development, tourism, history, and archaeology, which often involve studying the evolution of cities representing how city objects change over a long period. For example, for a time sequence, a building may be constructed, modified, demolished and replaced by other ones. These changes are slower and involve features that begin or cease to exist over different periods. Hence, such changes are required to be managed using historic or linear versions of city models, including up-to-date information about newly added, modified, or demolished objects. Scenarios for studying city evolution also require enabling backward compatibility for semantic 3D city models to handle multiple representations of the past of a city. A given date may be a starting point to imagine the past and constructing several scenarios (c.f. section 4.1). Such backward compatibility is also beneficial for archaeology, for instance, for the urban reconstruction of ancient cities, modelling of archaeological 3D objects and their attributes, managing excavations, testing reconstruction hypotheses, and analysing the development of sites over time. Hence, it is required for semantic 3D city models to be able to manage historic versions. In order to be able to exchange and query all versions and their features, it is also important to manage all the versions within one single data file. As shown in figure 2.9, this would allow representing the evolution of the city in the form of different versions of CityGML documents. The successive versions represent the state of all features of the entire city at specific points in time. In addition, the figure shows that the authorities can work, in parallel, with different workspaces or branches to insert, delete or modify the objects. Such additions can be merged with the earlier versions of the CityGML documents to form the final versions. It leads to the next requirement of managing alternative versions.



**Figure 2.9:** Management of historic and parallel versions within semantic 3D city models.

### 2.1.4.2 Requirement R7: Managing alternative versions

Semantic 3D models also play a crucial role in the documentation and reconstruction of both historical and contemporary events. Examples include crime scene and accident reconstructions, representation

of battles and other past events, archival descriptions of ancient structures before demolition, documentation of construction and demolition of buildings. Such events involve a sequence of versions of a “reality” at a specific time or in a particular time sequence. Events are often reconstructed from conflicting and incomplete evidence, and a complete reconstruction must allow branching to handle the alternative possibilities. In this way, as shown in figure 2.9, the authorities or users can work, in parallel, with different workspaces or branches to insert, delete or modify the objects. Such additions can be merged with the earlier versions of city objects to form the final versions leading to multiple possible futures. Hence, semantic 3D city models should be able to manage such parallel and alternative versions. Again, for cross-domain applications, it is essential to exchange the city models along with their parallel versions to perform, e.g. “what-if” analysis. Hence, it requires (i) managing all the parallel versions within one dataset and (ii) interoperable data exchange format to be interpreted by different applications and software systems.

### 2.1.5 Requirements Summary

The previous sub-sections discuss important applications of semantic 3D city models which include Smart Cities, Digital Twins, different types of urban simulations like solar potential simulation and energy demand estimation, mobility, and urban development including planning and history. Based on the review of numerous use cases, key requirements are gathered for semantic 3D city models for supporting different types of time-dependent properties. The requirements are summarised in Table 2.2.

## 2.2 Review of existing standards for the listed requirements

### 2.2.1 Sensor and IoT data access and management

Due to the widespread use of heterogeneous sensors and IoT devices in numerous applications, several open and interoperable standards and protocols have been developed to access and manage their data. Sensor Web Enablement (SWE) (Bröring et al. 2011), an OGC initiative, provides a suite of standards that enable the discovery, access, tasking, as well as eventing and alerting of the sensor resources in a standardised way. The OGC SWE standards suite comprises well-defined information models. One of the standards is the SensorML (Botts 2014), which represents sensor description and metadata. This standard also describes sensor calibration records and accuracy and precision information. The other information model within OGC SWE is the Observations and Measurements (O&M) (Cox 2013). It allows describing real-time observation data. The SWE also provides comprehensive interface models and web services such as Sensor Observation Service (SOS) (Bröring et al. 2012) and SensorThings API (Liang et al. 2015) for retrieval of sensor descriptions and observations with the help of standardised requests. In comparison to SOS, SensorThings API is a relatively new standard, which is REST-ful, lightweight, and based on JSON. Owing to the well-defined and comprehensive set of open and international standards, the SWE standard suite is already being used worldwide in various domains such as Digital Twins like Digital Twin Munich<sup>26</sup>, SDDI<sup>27</sup>, and (Moshrefzadeh et al. 2020), early warning systems (Wupperverband 2017), disaster management (Jirka et al. 2009), marine science (Partescano et al. 2017; Toma et al. 2015), citizen science (Pfeil et al. 2015), environmental and air quality monitoring (IRCELINE 2018; Jirka et al. 2011) and many more. The open-source

<sup>26</sup><https://muenchen.digital/twin/>

<sup>27</sup><https://www.lrg.tum.de/en/gis/projects/smart-district-data-infrastructure/>



No.	Requirement	Example Use Cases
R1	Integrating sensor observations with city object properties	Smart Meters monitoring building energy
		Monitoring bridge deformation
		Traffic cameras recording numbers of cars
R2	Managing events and alerts	Water level exceeding a threshold
		Air Quality exceeding a dangerous limit
		Energy consumption breaching the allowed usage
R3	Integration and overlay of dynamics of moving objects	Air quality sensors mounted on a tram
		Autonomous car communicating with city objects
		Pedestrians moving into or out of a building
R4	In-line Support of Timeseries	Solar irradiation for building roof surface
		Energy demand estimation of a building
		Flood inundation simulation
R5	Complex patterns and schedules	Weekly patterns of energy consumption
		Heating schedule of energy systems
		Repeating trajectories for bus lines
R6	Managing historic versions	Documentation of changes over time
		Reconstruction of the past events
		Multiple representations of the past
R7	Managing alternative versions	Planned alternative structures for comparison by Urban Planners

**Table 2.2:** List of key requirements considered for extending semantic 3D city models for supporting time-dependent properties. The individual requirements are referred to using their unique numbers in the thesis.

implementations such as 52°North Sensor Observation Service<sup>28</sup> and the Fraunhofer Opensource SensorThings (FROST) Server<sup>29</sup> allow inserting, querying, and visualising arbitrary sensor data and observations according to the OGC Sensor Observation Service and OGC SensorThings API standards respectively.

Apart from OGC Sensor Web Enablement, there are also other architectures and frameworks which focus on interoperability of sensor and IoT devices and being applied in different projects. The FIWARE (FIWARE 2018) is a generic and open-source platform that aims to make interoperable city services, to provide access to real-time context information, and to implement smart city applications. The platform enables developers and communities to create their services based on commonly defined APIs and data models. The FIWARE is already being used in several smart city initiatives such as “City Enabler”<sup>30</sup>. It is a FIWARE-based software product allowing scattered and distributed urban data to be collected and organised in a central repository, which can be fed to different applications with the help of the standard APIs. Other than FIWARE, another project called BIG IoT (Bröring et al. 2017) focuses on cross-standard, cross-platform, and cross-domain IoT services and applications. The approach is to register an individual IoT platform to their so-called “BIG-IoT Marketplace”, which acts as a catalogue. Using the Marketplace, the BIG-IoT API allows discovering, authenticating/authorising multiple IoT resources and allows using them in a single application. Similarly, the bIoTope project (Robert et al. 2017) under the European Union’s Horizon 2020 Programme provides an ecosystem allowing registering heterogeneous IoT platforms and accessing them using standardised and open APIs. Like BIG-IoT Marketplace, the bIoTope project also includes a Marketplace called IoTBnB which can be used for discovering and authenticating the different IoT platforms. In similar ways, another EU Horizon 2020 project VICINITY (Mynzhasova et al. 2017) provides a decentralised ecosystem offering “interoperability as a service”. Its architecture involves a VICINITY Cloud acting as a Marketplace used for registering and then discovering and accessing the numerous IoT platforms using the standardised APIs. Other pertinent initiatives carried out within the EU Horizon 2020 Programme are symbIoTe (Gojmerac et al. 2016), INTER-IoT (Ganzha et al. 2016), and Thingful (Thingful 2018). Sensor Measurement Lists (SenML) (Jennings et al. 2018) is also a specification working towards interoperability of sensors. In this specification, representations share a common SenML data model. A simple sensor, such as a temperature sensor, could use this media type in protocols such as HTTP to transport the measurements of the sensor or to be configured. (Jazayeri et al. 2015) also provide a comprehensive evaluation of four open, interoperable standards for the IoT devices: OGC PUCK over Bluetooth, TinySOS, SOS over CoAP, and OGC SensorThings API.

There are several platforms which consist of a complete suite for managing sensor observations. These platforms include their own data storage, visualisation clients, and the APIs to query and retrieve the observations. ThingSpeak (Maureira et al. 2014) is an IoT platform that allows users to register different sensors attached to simple microcontrollers such as Arduino and Raspberry Pi, collect and store sensor observations in the Cloud and develop IoT applications. The ThingSpeak platform provides applications to analyse and visualise observations. The system also allows querying by location, allowing the user to have access to data from various locations in the world. Open-Sensors<sup>31</sup>, which is termed as “Twitter for Sensors”<sup>32</sup> enables users to connect various sensor devices and publish their observations for free. The data is publicly accessible, shareable and reusable by

<sup>28</sup><https://52north.org/software/software-projects/sos/>

<sup>29</sup><https://github.com/FraunhoferIOSB/FROST-Server>

<sup>30</sup><https://geographica.com/en/showcase/cedus-city-enabler/>

<sup>31</sup><https://www.opensensors.com/>

<sup>32</sup><https://www.wired.com/2014/12/the-internet-of-anything-opensensorsio/>

and for anyone. The platform provides real-time and historical access to public and private data through the API and in-browser data view. The Things Network (TTN)<sup>33</sup> is a relatively new initiative aiming at building a network for the Internet of Things by creating abundant data connectivity. The network focuses on a technology called LoRaWAN<sup>34</sup> which allows for things to talk to the Internet without 3G or WiFi, so no WiFi codes and mobile subscriptions are required. It features low battery usage, long-range and low bandwidth, which is ideal for the IoT devices. The Things Network also supports publishing observations to other platforms such as TTN-OpenSensors-Integration<sup>35</sup> and TTN-OGCSWE-Integration<sup>36</sup>. Such integration makes discovering, analysing, and visualising sensor observations even easier. The weather has a major influence on city systems ranging from energy and water, to sanitation, transportation, health care, to disaster management. Weather Underground<sup>37</sup> is a commercial weather service providing real-time weather information via the Internet. It provides weather reports for most major cities across the world on its website. It also uses observations from members with automated personal weather stations (PWS). Weather Underground currently uses observations from over 250,000 personal weather stations worldwide. Similarly, HawaDawa<sup>38</sup>, a relatively new company based in Germany, provides a well-defined API for measuring air quality parameters over a city. With this data, a city could, for example, opt to change how it routes traffic to minimise vehicle exhausts in particularly polluted streets and monitor how changes are impacting the air. HawaDawa's data currently covers over 20 cities across Germany, Switzerland, and the UK.

### 2.2.2 Representation and management of time-series

As discussed in the previous section, most sensor and IoT standards provide comprehensive structures to represent and manage time-series data obtained from sensors. However, time-series data is also generated in different simulations such as solar potential simulations and energy demand estimations. In many scenarios, such time-series data are also stored in databases. Traditional relational database management systems such as Oracle<sup>39</sup>, MySQL<sup>40</sup>, and PostgreSQL<sup>41</sup> are widely used for managing time-series data. They provide standard SQL functions to query and analyse time-series. Big Data creates a demand for efficient time-series data analysis. TimescaleDB<sup>42</sup> and InfluxDB<sup>43</sup> are good examples of Open Source time-series databases which are being used in the fields of IoT and real-time analytics. When it comes to managing more heterogeneous data generated by millions of devices and applications, each with their own data structures, databases require new levels of flexibility, agility, and scalability. In this environment, NoSQL databases such as MongoDB<sup>44</sup> are proving their value. Another new concept in this direction is the Data Stream Management System (DSMS). Such management systems continuously process arriving data without having to persist them, this speeds up the data evaluation process, achieving more timely results in comparison to traditional DBMSs.

<sup>33</sup><https://www.thethingsnetwork.org/>

<sup>34</sup><https://www.thethingsnetwork.org/docs/lorawan/>

<sup>35</sup><https://www.thethingsnetwork.org/docs/applications/opensensors/>

<sup>36</sup><https://github.com/52North/ttn-ogcswe-integration>

<sup>37</sup><https://www.wunderground.com/>

<sup>38</sup><https://hawadawa.com/>

<sup>39</sup><https://www.oracle.com/index.html>

<sup>40</sup><https://www.mysql.com/>

<sup>41</sup><https://www.postgresql.org/>

<sup>42</sup><https://www.timescale.com/>

<sup>43</sup><https://www.influxdata.com/>

<sup>44</sup><https://www.mongodb.com/>

(Anjos et al. 2014) explore the feasibility of Data Stream Management Systems (DSMSs) to support Energy Management applications, pointing out how to implement an Energy Management System capable of real-time data processing. In many scenarios, especially, when observations are not very highly frequent, time-varying data are stored in external files such as Comma Separated Values (CSV) and Excel sheets. Such files are usually generated once for a specific scenario and do not update continuously. There are also cloud-based systems such as Google Spreadsheet<sup>45</sup>, and Microsoft OneDrive<sup>46</sup> which allow users to store such time-series data in a cloud environment.

For representing time-series, the Open Geospatial Consortium (OGC) also provides various standards. The Observations and Measurements (O&M) is a generic information model for describing observations for specific timestamps. The observation is modelled as a Feature within the context of the General Feature Model (ISO 19101). An observation feature binds a result to a feature of interest, upon which the observation was made. The observed property is a property of the feature of interest. An observation uses a procedure to determine the value of the result, which may involve a sensor or an observer, analytical procedure, simulation or other numerical process. As mentioned in the previous section, O&M is one of the core standards in the OGC Sensor Web Enablement suite, providing the response model for the OGC Sensor Observation Service (SOS) and the OGC SensorThings API. WaterML 2.0 (Taylor 2014) is also an OGC standard for the representation of hydrological observations data with a specific focus on time-series structures. This standard is implemented as an application schema of the GML version 3.2.1, making use of the OGC Observations and Measurements standards. The standard allows defining time-series as discrete coverages, which means, an instance of such a coverage would be a set of ordered time instances where each time instance is associated with a single value from the attribute space. This association is often represented using time value pairs or a domain range. It also allows to define interpolation types 'per point' within time-series, establishing the relationship between time instants and the recorded values.

OGC also provides the TimeseriesML 1.0 (Tomkins and Lowe 2016) for representation and exchange of observation data as time-series. It is an extension of the work initially undertaken within the WaterML standard. However, this standard aims to provide a domain-neutral model for the representation and exchange of time-series data. The TimeseriesML schema supports two types of encodings. The first encoding is the interleaved time-value pair encoding, whereby the time and value are coupled together, and the coupling explicitly represents the mapping. The second encoding is the domain-range encoding, where the domain and range are encoded separately, with a mapping function that allows looking up the range value for a given domain value.

### 2.2.3 Managing alerts and events

ESRI ArcGIS GeoEvent Server<sup>47</sup> is a well-known commercial solution which allows enabling real-time event-based data streams to be integrated as data sources. The GeoEvent Server is capable of consuming event data from multiple real-time data streams. At the same time, users can filter, process, and subscribe to event data streams and automatically be alerted when a specific condition occurs, all in real-time. It helps users to respond faster with increased awareness whenever and wherever change occurs. The new OGC Publish/Subscribe 1.0 standard (also known as PubSub) (Braeckel et al. 2016) provides asynchronous communication across OGC service interfaces and data types, including coverages, features, and observations and enables users to subscribe to the real-time data streams. The

---

<sup>45</sup><https://www.google.com/sheets/about/>

<sup>46</sup><https://onedrive.live.com>

<sup>47</sup><https://www.esri.com/en-us/arcgis/products/arcgis-geoevent-server>

PubSub is a relatively new standard, and its first experiments have been performed in the application domains of Sensor Web and Aviation.

Message Queuing Telemetry Transport (MQTT) (Banks et al. 2019) is also a publish/subscribe, straightforward, and lightweight messaging protocol designed specifically for constrained devices with low-bandwidth. Hence, it is considered a right solution for IoT applications. The design principles are to minimise network bandwidth and device resource requirements while also attempting to ensure reliability and some degree of assurance of delivery. The MQTT extension is supported by major interfaces such as OGC SensorThings API<sup>48</sup>, The Things Network<sup>49</sup>, ThingSpeak<sup>50</sup>, Amazon IoT<sup>51</sup>, and Microsoft Azure<sup>52</sup> allowing users to publish the data and subscribe to specific events in a real-time manner. There are also applications such as Eclipse Mosquitto<sup>53</sup> enabling to set up message brokers implementing the MQTT protocol. Furthermore, to perform real-time processing for geospatial data, an extension of the MQTT protocol called GeoMQTT (Herle et al., 2016) has been developed. This extension includes new message types to support spatio-temporal tagging and filtering of events. It still uses a publish/subscribe interaction scheme although not exclusively topic-based but also timestamp and geometry-based. However, the topic mechanism is inherited from MQTT. Subscribers can specify their interests in geo events by the use of the topic, spatial and temporal filter. The broker only forwards the so-called GeoPublish message to subscribers if all three filters are satisfied.

#### 2.2.4 Representation of moving objects

As discussed in section 2.1.3, applications such as traffic simulations and navigation involve objects changing their locations with time. The data format such as GPS Exchange Format (GPX)(GPX 2004) provides an XML schema for describing waypoints, tracks, and routes. Location data, along with other information like elevation and time, is stored in tags and can be interchanged between GPS devices and software applications. Keyhole Markup Language (KML) (Burggraf 2015) is a widely known international standard maintained by the Open Geospatial Consortium (OGC). This format also provides an XML schema for representing and exchanging geographic information such as points, lines, and polygons, and image overlays over Google Maps and Google Earth. The KML also supports changing their locations over time. Another new data format Cesium Modelling Language (CZML)<sup>54</sup> developed by Analytics Graphics Inc, is gaining a lot of attention due to its lightweight structure. The CZML standard is based on JSON and allows representing time-dynamic aspects of the geographical objects such as points, lines, billboards, 3D models, and imageries. The CZML files can be exchanged and visualised over web-browser based virtual globe Cesium.

OGC Moving Features (Asahara et al. 2015) is a relatively new standard, which defines an abstract model for encoding moving feature data. Based on a conceptual model, it also includes an XML encoding in the form of an OGC Geography Markup Language (GML) application schema, and a simple CSV (comma-separated value) encoding format. The standard allows representing (i) Discrete phenomena which exist only on a set of instants, such as road accidents, (ii) Step phenomena where the changes of locations are abrupt at an instant, such as administrative boundaries, and (iii) Continuous

<sup>48</sup><https://developers.sensorup.com/tutorials/mqtt/>

<sup>49</sup><https://www.thethingsnetwork.org/docs/applications/mqtt/>

<sup>50</sup><https://blogs.mathworks.com/iot/2017/01/20/use-mqtt-to-send-iot-data-to-thingspeak/>

<sup>51</sup><https://docs.aws.amazon.com/iot/latest/developerguide/mqtt.html>

<sup>52</sup><https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-mqtt-support>

<sup>53</sup><https://mosquitto.org/>

<sup>54</sup><https://github.com/AnalyticalGraphicsInc/czml-writer/wiki/CZML-Guide>

phenomena whose locations move continuously for a period in time, such as vehicles, typhoons, or floods. The ability of the Moving Features standard to attribute time-varying properties to an object (rather than just varying its location or trajectory), has utility and value in many current application areas, including Location Based Services, Intelligent Transportation Systems, Disaster Risk Management Systems, and Smart City Applications. For example, the traffic congestion on roads and 'hotspots' of air pollution are typical moving features seen in the real world.

DATEX II<sup>55</sup> is the European standard for the exchange of traffic-related data. It is a unified XML-based format allowing data exchange between service providers, traffic control centres, and road operators. It enables representing traffic and travel information such as traffic flow and density, traffic measures, roadworks, accidents, and parking. For mobility related applications, the Mobility Data Specification<sup>56</sup> (MDS) comprises of a set of APIs focused on real-time information about e-scooters, bicycles, mopeds, and carshare. MDS provides a standardised way for municipalities or other regulatory agencies to ingest, compare and analyse data from mobility service providers, and to give municipalities the ability to express regulation in machine-readable formats. Another similar specification named the General Transit Feed Specification<sup>57</sup> (GTFS) by Google defines a standard format for public transportation schedules and associated geographic information. GTFS "feeds" let public transit agencies publish their transit data and developers write applications that consume that data in an interoperable way.

Apart from mobility data representation and exchange formats, many software solutions are being used for performing mobility simulations. Simulation of Urban Mobility (SUMO) (Behrisch et al. 2011) is an open-source solution providing continuous road traffic simulation package designed to handle large road networks. SUMO allows modelling of intermodal traffic systems including road vehicles, public transport and pedestrians, and supports various tools for route finding, visualisation, network import and emission calculation. Another popular solution is PTV VISSIM<sup>58</sup> developed by PTV Planung Transport Verkehr AG in Karlsruhe, Germany. PTV VISSIM is a microscopic multi-modal traffic flow simulation used for transportation planning and operation analysis. It helps users to realistically simulate and balancing roadway capacity as well as traffic and transport demand.

### 2.2.5 History and Version Management

Versioning is not a new concept and is a well-established term in the field of computer programming. There are several Version Control Systems (also termed as Revision Control Systems) such as Git<sup>59</sup>, Mercurial<sup>60</sup>, Concurrent Versions System (CVS) (Vesperman 2003), and Subversion<sup>61</sup> (SVN). Version Control Systems (VCS) were developed primarily to support parallel development within a single project. Although there are operational and architectural differences between these systems, they all maintain versions of collections of files comprising a project. These collections are often organised in a tree structure, similar to a directory hierarchy within a computer file system. A VCS has the representational power to manage changes, parallel updates, and merges of versions. However, versions always represent the change in the forward direction of time. The collection maintained

---

<sup>55</sup><https://www.datex2.eu/>

<sup>56</sup><https://github.com/openmobilityfoundation/mobility-data-specification>

<sup>57</sup><https://developers.google.com/transit/gtfs>

<sup>58</sup><https://www.ptvgroup.com/en/solutions/products/ptv-vissim/>

<sup>59</sup><https://mirrors.edge.kernel.org/pub/software/scm/git/docs/user-manual.html>

<sup>60</sup><https://www.mercurial-scm.org/>

<sup>61</sup><http://subversion.apache.org/>

by a VCS is rooted in an original version, that is, the versions form a rooted directed acyclic graph (DAG). The original version is the oldest, and it is not possible to create versions earlier than the root. It is important in the context of 3D city models, especially for applications related to history and archaeology, where a given date may be a starting point to imagine the past and constructing the possible scenarios.

The concept of versions has successfully been incorporated by Oracle using the Oracle Workspace Manager (Beauregard and Speckhard 2014). The Oracle Workspace Manager allows managing multiple versions of the data in the same Oracle Relational Database Management System in the form of workspaces. A workspace is a virtual copy of the data, which separates the collection of changes in the different versions from the live (production) data. The version-enabled data are stored in separate tables with additional columns representing the version metadata. Such additional columns contain the version and workspace of each data row along with the date and time of each update. The database view is created on the version-enabled table and triggers are defined to enable SQL operations such as insert, delete, and update. This approach allows preservation of the structure of the original table and shows the data of only the particular version. The Oracle Workspace Manager also provides the management of historical data by using savepoints and the means for resolving possible conflicts during the merging of the different versions. Similarly, versions can also be managed within ESRI ArcSDE Geodatabases (ESRI 2004). ESRI also supports managing history, performing “what-if” analysis and conflict detection and resolution. However, there are no interoperable exchange formats related to both Oracle and ESRI which would allow exchanging all versions of a repository as one dataset, nor which would allow the use of the same dataset by both Oracle and ESRI.

## 2.3 Evaluation of city modelling standards for the listed requirements

As mentioned in Chapter 1, there are already several organisations and standard working groups, who provide semantic data models for cities and their objects. These data models not only describe spatial and graphical aspects of the city objects, but also provide the ontological structure including thematic classes, attributes, and their interrelationships. For this reason, such models make it possible for applications and simulation tools to distinguish urban objects (like buildings and streets) and use their rich thematic and geometric information for queries, statistical computations, simulations, and visualisations. As the research in this thesis work focuses on Semantic 3D City Models, this section reviews the current state of three popular semantic information models for representing 3D city objects and evaluates up to which degree these standards cover the key requirements for time-dependent properties identified in section 2.1. The standards considered in the study are: OGC CityGML (version 2.0), buildingSMART IFC (version 4), and INSPIRE. The evaluation results are summarised in the table 2.3

### 2.3.1 CityGML 2.0

The most recent version of the CityGML standard CityGML v2.0 (Gröger et al. 2012) was adopted in 2012 and provides limited support for handling and managing time-dependent properties. For example, the CityGML Building model consists of attributes such as *yearOfConstruction* and *yearOfDemolition* for defining the lifespan of an object. Considering many use cases and applications, the requirement of supporting time-dependent properties with city objects was included as one of the core work packages for the next version of the standard CityGML 3.0 (Kutzner et al. 2020). As a result, many

No.	Requirements	IFC 4	INSPIRE	CityGML 2.0
R1	Integrating sensor observations with city object properties	o	o	o
R2	Managing events and alerts		o	
R3	Integration and overlay of dynamics of moving objects		+	o
R4	In-line support of Timeseries	+	+	o
R5	Complex Patterns and schedules			
R6	Managing historic versions	o	o	o
R7	Managing alternative versions	o	o	

**Table 2.3:** Evaluation of 3D city modelling standards for the listed requirements. The evaluation was performed by reviewing the specification official documentations. '+' sign denotes that the standard provides complete support of the requirement, and 'o' sign denotes that a limited support of the requirement is provided by the standard specification or the related extension. The details are given in section 2.3.

researchers covered different aspects for extending the CityGML standard providing explicit support of time-dependent properties.

CityGML 2.0 does not provide any explicit class to deal with sensor information. However, a number of research work have been carried out in the direction of linking sensor observations with city objects. An approach proposed by (Zhu et al. 2016) allows integrating city models based on the CityGML standard and spatio-temporal air quality data retrieved using the standard OGC Sensor Observation Service (SOS). The demonstration allows visualising dynamic variations of pollutants together with static city objects. Similarly, another architecture developed by (Santhanavanich et al. 2018) supports integration of heterogeneous sensor data with CityGML based models. The sensor data included pedelec usage from Smart Electric Bike, user fitness level from a Smart Watch, and weather data from an Open Weather Portal and integrated and visualised with city objects using the OGC SensorThings API standard. Both studies successfully demonstrate coupling real-time observations with city objects using open and interoperable standards, and visualise time-dependent properties with city objects. From the management perspective, the recently proposed CityThings concept (Santhanavanich and Coors 2019) also allows integrating dynamic sensor observations with city objects in very simple ways utilising the SensorThings API and CityGML standards. The concept enables defining the unique identifier (called *gml\_id*) of the CityGML object within the SensorThings



API and allows managing sensor and city model data in separate management systems making the solution easier to maintain. However, in this way, we can define the sensor's relation only with the city object and not with the property of the city object. As specified by the requirement [R1], it is also essential to define explicit links to real-time observations within the city object. That would allow, first of all, querying 3D city models to determine which city objects have sensors installed within them. At the same time, that would also make city object properties dynamic by overriding them according to the observations retrieved from the sensors and IoT devices. It is also very important to consider that apart from OGC, there are also many other standards and APIs related to sensors and IoT which are being used in different Smart City applications. Some of the APIs are open such as BIG-IoT and FIWARE, whereas, others are proprietary such as Microsoft Azure IoT and AWS IoT. The explicit linking to such standards and APIs within the city object give more flexibility to users and applications to define relation between the city object property and the heterogeneous API details from which the observations are retrieved.

Although CityGML does not explicitly support managing alerts and notifications, the integration with modern interfaces such as OGC SensorThings API may allow achieving Publish/Subscribe capabilities. Such interfaces already supports standard protocols like MQTT allowing users to subscribe to specific data streams. However, no implementation with the CityGML standard currently demonstrates the Publish/Subscribe functionalities so far.

The architecture proposed by (Santhanavanich et al. 2018) allows representing varying locations of the e-bike with the help of the OGC SensorThings API. The SensorThings API supports entities *Location* and *HistoricalLocation* for recording variations in location over time. The integration of SensorThings API with the visualisation application allowed the authors to visualise moving objects with the CityGML based models. The master's thesis (Ruhdorfer 2017) also proposes an architecture to integrate and visualise traffic simulation results from the simulation tool PTV VISSIM with city models based on the CityGML standard. The proposed concept also allows for an automatic derivation of road networks for the PTV VISSIM from city models in CityGML format. In the context of dam monitoring, (Baghdoust 2017) proposes a framework to visualise the varying water level information. The framework is based on the integration of CityGML *WaterBody* objects and the OGC Sensor Observation Service measuring the dynamic height observation data. All the mentioned studies successfully demonstrate representation of moving objects by coupling the sensor observations with the city models, however, the city object properties still remain static. For example, *WaterBody* object in the dam monitoring scenario should be dynamic and vary its height or change its shape of water over the time. This is not feasible with the current version of CityGML.

The CityGML standard does not allow representing in-line time-series data. However, an extension of the CityGML standard has been proposed by the Energy Application Domain Extension (ADE) Working Group to support time-series data explicitly (Agugiaro et al. 2018). The data model distinguishes between regular and irregular timeseries. In regular timeseries, the values have a defined start and end time and a constant time increment. In irregular timeseries, each value has an individual timestamp. The timeseries values itself may either be stored directly in-line within the CityGML document or in a separate file with table structure such as CSV. However, the concept focuses on varying physical values related to only energy based applications and simulations. The representation of timeseries using domain-neutral standards such as OGC TimeseriesML 1.0 may open the door to cover more applications.

Although complex patterns are not supported in the CityGML standard, the CityGML Energy ADE extends CityGML 2.0 classes by modelling four different types of schedules: (i) *ConstantValueSchedule*, specifying only one value for the complete time interval regarded, (ii) *DualValueSchedule*

defining two different values, one for operating times and one for idle times, (iii) *DailyPatternSchedule* specifying different time periods within a year where each period is related with schedules for specific days of the week (e.g. week day, weekend, or a specific day of the week), and (iv) *TimeSeriesSchedule*, where any (regular or irregular) time series may be used.

The versioning concept is not directly addressed within the CityGML standard. CityGML 2.0 already allows CityGML object properties *creationDate* and *terminationDate*. These can be used to represent multiple instances of the same real-world object for different time periods in the same file. However, that would require different *gml:id* for each instance. A modification has been proposed to the CityGML schema by adding temporal information on buildings (Pfeiffer et al. 2013). However, this method allows registering only definite states. CityGML schema modification and possible standardised exports are not discussed. Another method proposed by (Morel and Gesquière 2014) takes into account the possibility for a city object to change and the time value which fixes this change in the city life-cycle. However, this method does not support the possibility of having parallel or alternative scenarios.

### 2.3.2 IFC v4

The IFC specification also provides limited support in terms of time-dependent properties and does not deal with all the changes as identified in the previous section. The first version of IFC was adopted in 1996 and the latest version (IFC version 4 with 2nd Addendum) was released in 2016.

Although IFC does not provide any in-built module to deal with Sensor and IoT data, many researchers have made developments in this direction. A web based system called "Otaniemi3D" (Dave et al. 2018) provides information about energy usage, occupancy and user comfort by integrating BIMs and IoT devices through open messaging standards (O-MI and O-DF) and IFC models. Another conceptual web information service framework, proposed by (Wang et al. 2013), demonstrates the idea for Smart Building by combining live sensor data based on the OGC standards with IFC models. By knowing the building status at any given time and location, which is largely unseen to most users, it is possible to change occupant behaviour, improve building safety, avoid unnecessary energy consumption and facilitate better working environments. Another case study (Gunduz et al. 2017) demonstrates how a facility in the BIM environment is displayed in two dimensions on Google Maps, and real-time data from the sensors can be provided and tracked from the web browser. The approach helps facility managers, particularly in view of facility comfort analysis by real-time analysis and visualisation of the data coming from the sensors. However, no implementation currently demonstrates the Publish/Subscribe capabilities for managing alerts and events.

IFC does not provide support for moving objects and trajectories. The integration with other standards such as SensorThings API might create possibilities for supporting moving objects, however, to the best of our knowledge, no such implementations exist.

IFC provides the support of time-series using the in-built *IfcTimeseries* module. It allows a natural association of data collected over intervals of time. Within the specification, time-series can be regular or irregular. In regular timeseries, data arrive predictably at predefined intervals. In irregular timeseries, some or all time stamps do not follow a repetitive pattern and unpredictable bursts of data may arrive at unspecified points in time. The modelling of buildings and their performance involves data that are generated and recorded over a period of time. Such data cover a large spectrum, from weather data to schedules of all kinds to status measurements to reporting to everything else that has a time related aspect. Their correct placement in time is essential for their proper understanding and use, and the

*IfcTimeseries* subtypes provide the appropriate data structures to accommodate these types of data. However, *IfcTimeseries* cannot be used to make arbitrary buildings or object properties dynamic.

IFC does not provide data structures to model complex patterns and schedules based on available time-series.

The concept of temporal version is not currently implemented in IFC, but some extensions have been proposed. For example, the proposal by (Zada et al. 2014) suggests six existing entities from the IFC standard to be modified to represent as new entities within the IFC schema to support the idea of object versioning that holds the history of changes to objects of the BIM model. (Nour and Beucke 2010) also propose an approach in which both object versioning and IFC model are integrated together in an open multidisciplinary collaborative environment. Object versioning gives the possibility to have several versions of the content (attributes' values) of an object. The development of design in terms of addition of new objects, deletion of objects or modifications of attributes' values of pre-existing objects can be captured in a graph structure.

### 2.3.3 EU INSPIRE

The EU INSPIRE Directive came into force in 2007. The thematic scope of the Directive covers 34 interdependent spatial data themes, however, some data themes such as Environmental Monitoring Facilities require handling time-dependent properties which is not fully supported by the Directive.

Considering the popularity of RESTful architectures, an extension has been proposed for the OGC SensorThings API standard to be considered as a solution that meets the legal obligations stemming from the INSPIRE Directive, thus simplifying the process for extending existing spatial data infrastructures to the IoT (Kotsev et al. 2018). The proposal shares perspective on what should be done with regards to: (i) data encoding; and (ii) the use of SensorThings API as a download service for INSPIRE. The proposed integration with the OGC SensorThings API makes it possible for supporting MQTT protocols allowing users to subscribe to specific events and receive alert notifications.

The specialized observation from the INSPIRE guideline *TrajectoryObservation* allows representing a series of measurements along a trajectory, for example, along a ship's track. Each measurement is made at a separate point along the trajectory and at a separate time. The extensions from (Kotsev et al. 2018) also explore methods for grouping a set of features, as such a mechanism would provide easier handling for client applications.

The INSPIRE data models provide a 'Specialized Observation' package which define ten specializations of observations of the O&M specification. All the specialized Observation types essentially add 'constraints' to the underlying O&M model which characterise the result of the observation and the sampling regime used. One of specializations is the *PointTimeSeries Observation*, which represents a series of measurements at the same point, e.g. timeseries having regular measurements by a fixed station. Similarly, the *MultiPointObservation* is a specific type of Point-based observation. It is intended for cases in which measurements are made at a set of discrete points at the same time. For example, a sensor network reporting temperature at 10am. The points themselves are not on a grid but may be distributed in any manner, for example unevenly spaced around a coastline.

INSPIRE does not provide data structures to modern complex patterns and schedules based on available timeseries.

INSPIRE defines several requirements and recommendations for modelling life-cycle information of spatial objects which include UML stereotypes and properties allowing for bi-temporal modelling of geospatial objects. Furthermore, a separate property exists for denoting a specific version of a

geospatial object. However, currently only exchanging the last version of spatial objects is supported by INSPIRE; historic versions cannot be provided yet (and especially not within one data file).

## Chapter 3

---

### Methodology

This chapter defines the methodology for extending semantic 3D city models. Firstly, it is identified what the properties of city objects that may change with time are. Secondly, the discussions determine how these changes take place. Accordingly, two major classifications of changes: (i) slower changes, and (ii) highly dynamic changes are proposed. The city model extensions: (i) Versioning concept, and (ii) Dynamizer concept are developed for supporting slower and highly dynamic changes, respectively. The newly defined concepts are implemented for the CityGML standard. This chapter briefly discusses the overview of the CityGML 2.0 and introduces how these extensions are developed for the CityGML standard in the remaining parts of this thesis.

Some of the discussions have been presented in the published paper:

**Chaturvedi, K.** and Kolbe, T. H. (2019). 'A Requirement Analysis on extending Semantic 3D City Models for supporting Time-dependent properties.' In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences IV-4/W9*, pp. 19–26. URL: <https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-4-W9/19/2019/>

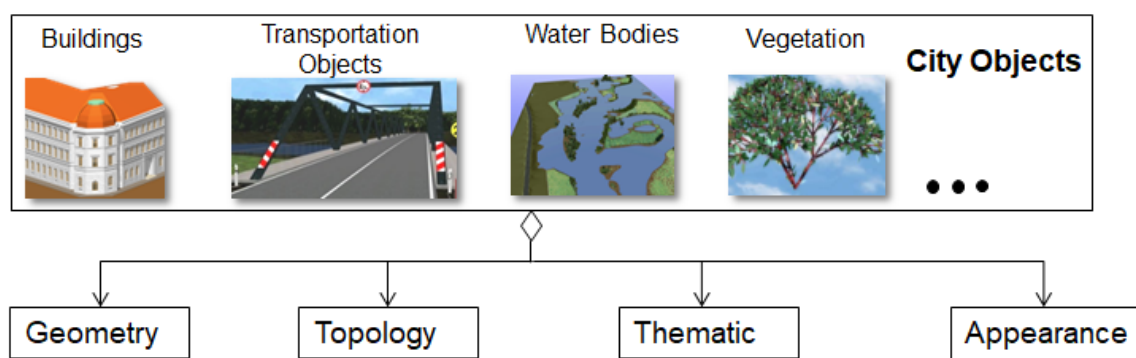
## 3.1 Time-dependent properties in the context of 3D city models

### 3.1.1 Identification of city object properties affected by time

Cities comprise several topographic objects, including built structures, elevations, vegetations, water bodies, bridges, tunnels, and more. Semantic 3D city models define classes and relations to represent each topographic object with its geometrical, topological, thematic, and appearance properties (Figure 3.1). For instance, a city object can be defined using a geometrical primitive: a zero-dimensional object is a point (e.g. location of a bus stop), a one-dimensional a curve such as a line string (e.g. representation of a street), two-dimensional a surface such as a polygon (e.g. a water body), and a three-dimensional a solid (e.g. a building). Each geometry can be defined using its coordinate reference system and with an appropriate level of detail. Similarly, for many use cases, topological correctness of the object geometries is of high importance. Topology represents the sharing of geometry objects between features or other geometries. For example, a building and an adjacent garage may be represented by two solids. The surface, describing the area where both solids touch, maybe represented only once and both solids reference it. Similarly, a geometry defining a wall of a building may be referenced twice: by the solid geometry defining the geometry of the building, and by the wall feature.

Furthermore, semantic 3D city models allow defining an arbitrary number of spatial and non-spatial attributes for geographic features. For example, in the CityGML standard, the pivotal class of the building model is *AbstractBuilding* from which the two non-abstract classes *Building* and *BuildingPart* are derived. These three classes follow the general composite design pattern (Gamma et al. 1995): a *Building* may contain *BuildingParts*, and as the latter class is also derived from *AbstractBuilding* a (recursive) aggregation hierarchy of arbitrary depth may be realised. For example, a building can be decomposed into different (main) building parts like walls, stairs, etc. and these may again consist of parts like windows or doors. Furthermore, a *Building* or *BuildingPart* is described by optional (non-spatial) attributes: function, usage, and class, year of construction and demolition, roof type, measured height, number and individual heights of storeys above and below ground. *Building* and *BuildingPart* can also be assigned multiple postal addresses and descriptions. In addition to spatial and thematic properties, semantic 3D city models allow defining appearances (i.e. observable properties) of the features' surfaces. Appearances represent visual data like materials, textures, and colours on the wall and roof surfaces. Appearances also represent arbitrary categories using themes for infra-red radiation, noise pollution, or earthquake-induced structural stress.

In general, current generation semantic 3D city models provide only static representations of the physical environment reflecting the state of an object and its properties at a specific point in time. However, in reality, these properties are not static and change with time. For instance, as proposed by (De Luca et al. 2010), a building may undergo different kinds of transformations such as : (i) creation (addition of a new building), (ii) demolition (destruction of an existing building), (iii) union, (iv) division, (v) reconstruction, and (vi) modification. These transformations may result in changes in (i) geometrical or spatial properties such as addition of a new floor in the building, (ii) topology within a building (e.g. division of a big room into two smaller rooms by the creation of a new wall), (iii) appearance of a building (such as materials, textures, and colours on the wall and roof surfaces), and (iv) semantics of a building (e.g. changing the building's type from residential to commercial). Similar changes can also be observed with other city objects. For example, in an autonomous driving scenario (Schwab and Kolbe 2019), a moving car on street results in changes in its geometrical properties such as car's location and orientation. However, the output from a traffic camera may lead to



**Figure 3.1:** Different properties of topographic objects within semantic 3D city models.

changes in appearances, e.g. recording videos of moving traffic over definite intervals. The geometry of an object can be further classified according to its shape, location, and extent, which can also be changed over time. For instance, a flooding incident in an urban area leads to overflow of water that submerges land such as streets, vegetation and buildings. From the 3D city modelling perspective, the representation of such flood inundation scenarios result in changes in the location, extent, and shape of water bodies (Amirebrahimi et al. 2016). More examples are provided in table 3.1. Hence, the extensions of semantic 3D city models must support the representation of time-dependent properties associated with geometrical, topological, thematic, and appearance properties of city objects.

### 3.1.2 Classification of changes in cities

The city object properties mentioned in the previous section can also change according to different frequencies. Some of the changes can be sudden. For instance, changing the owner of a house is a sudden event. On the other hand, changes may also be gradual and progressive. The demolition of a building is a gradual but comparatively shorter event. But the construction of a Gothic cathedral is a very long event that lasts several centuries. Similarly, historical building deteriorations may take centuries or millennia. The changes in city object properties can also be discrete or continuous. The evolution of a city can be represented as a series of time-stamped snapshots whereby each snapshot represents the state of city features at a specific point in time. Such snapshots are linear and discrete. On the other hand, rising water during a flood event is continuous.

Therefore, changes in cities can be classified according to the rate at which they change. This thesis classifies such changes according to two broad categories (Table 3.1):

1. **Slower Changes:** Some of the changes are slower, e.g. evolution of a city over a longer period. These changes may include addition, modification, or demolition of buildings and other objects over time. The urban planning scenario may also involve developing parallel or alternative versions of city models. Such slower changes include features that begin or cease to exist over different periods.
2. **Highly Dynamic Changes:** The other classification of changes represent high frequent or dynamic variations of object properties, e.g. variations of (i) thematic attributes such as changes of physical quantities (energy demands, temperature, solar irradiation levels), (ii) spatial properties such as change of a feature's location (moving objects), (iii) real-time observations from sensors and

No.	Requirement	Example Use Cases						
			Geometry	Thematic	Topology	Appearance	Slower Changes Dynamic Changes	
R1	Linking Sensors and IoT with city objects	Smart Meters monitoring Building Energy		+				✓
		Monitoring Bridge Deformation		+	+		+	✓
		Traffic Cameras recording numbers of cars		+			+	✓
R2	Managing Events and Alerts	Water level exceeding a threshold		+	+	+	+	✓
		Air Quality exceeding the dangerous limit			+		+	✓
		Energy consumption breaching the allowed usage				+		✓
R3	Moving Objects	Air Quality Sensors mounted on a car		+		+		✓
		Pedestrians moving into or out of a building		+	+	+		✓
R4	In-line Support of Timeseries	Solar Irradiation for Building Roof Surface			+		+	✓
		Energy Demand Estimation of a building			+		+	✓
		Flood Inundation Simulation		+	+	+	+	✓
R5	Complex Patterns and Schedules	Weekly patterns of energy consumption			+			✓
		Heating schedule of energy systems				+		✓
		Repeating trajectories for bus lines		+	+	+		✓
R6	Managing Historic Versions	Documentation of changes over time		+	+	+	+	✓
		Reconstruction of the past events		+	+	+	+	✓
		Multiple representations of the past		+	+	+	+	✓
R7	Managing Alternative Versions	Planned alternative structures for comparison by Urban Planners		+	+	+	+	✓

**Table 3.1:** Classification of slower and highly dynamic changes. The first two columns refer to the requirements listed in Chapter 2.



IoT devices. Such changes are often determined by discrete recordings or by using interpolation functions and can be defined as a function of time. For example, varying energy consumption values of a building can be determined for specific points of time (i) in the past by querying a database for historical data, (ii) in the present by querying a real-time sensor, and (iii) in the future by simulation software.

Both slower and highly dynamic changes are fundamentally different from each other. Slower changes involve features that begin or cease to exist over different time intervals. If a new building is added in the city model at a certain point or period in time, it is not possible to query it before that specified time as there was no existence of the feature. Similarly, the planning scenarios involve a comparison of multiple versions of the same city model by different planners. Hence, such changes require different versions of the city models having completely new or modified features. On the other hand, highly dynamic changes are mostly associated with city object properties and can be defined as a function of time. In this case, only some of the properties of otherwise static objects need to represent such time-varying values. For example, the energy consumption of a building determined by a Smart Meter installed in the same building requires only one specific property (e.g. "*energy\_consumption*") of the building to be dynamic, while other properties (e.g. "*building\_roof\_type*") remain static. More examples are provided in table 3.1. Hence, this thesis considers both slower and highly dynamic changes separately and proposes different extensions to represent and manage them.

## 3.2 Overview of the CityGML standard

The concepts developed in this thesis have been realised for the CityGML standard. However, in general, they can also be applied to other standards like IFC, EU INSPIRE, and the German cadaster standard ALKIS. This sub-section gives a brief overview of the CityGML standard, its data modelling aspects, and several applications and software systems supporting the CityGML standard.

### 3.2.1 Data Modelling with CityGML

CityGML (Gröger et al. 2012) defines a conceptual schema for describing relevant entities of urban objects such as buildings, roads, railways, tunnels, bridges, city furniture, water bodies, vegetation, and the terrain. This conceptual schema specifies how and into which parts and pieces physical objects of the real world should be decomposed and classified. All objects can be represented with their semantics, 3D geometry, 3D topology, and appearances information. The objects can further be represented using five predefined levels of details (LOD 0-4 with increasing accuracy and structural complexity). The relevant city objects are defined using the Unified Modelling Language (UML) and with an XML schema for the file exchange format.

The classes and data types in CityGML are grouped into several thematic modules. Most thematic classes are the sub-classes of *\_Feature* and *\_FeatureCollection*, the basic notions defined in the OGC Geography Markup Language (GML) (Cox et al. 2004) for the representation of spatial objects and their aggregations. Features include spatial as well as non-spatial attributes which are mapped to GML feature properties with corresponding data types. Geometric properties are represented as associations to the geometry classes for the respective thematic module. The thematic model also comprises different types of interrelationships between feature classes like aggregations, generalisations and associations.

The most important module is the *Core* module (Figure 3.2), which defines the basic CityGML components and is, hence, a mandatory package that must always be referenced by the packages

of the other modules. CityGML 2.0 comprises the thematic modules including *Building*, *Bridge*, *Transportation*, *CityObjectGroup*, *Appearance*, *Generic*, *CityFurniture*, *Relief*, *Vegetation*, *Tunnel*, *LandUse*, and *WaterBody*. In order to ensure that all the modules support the recommended time-dependent properties, it is important to first understand the basics of the CityGML *Core* module.

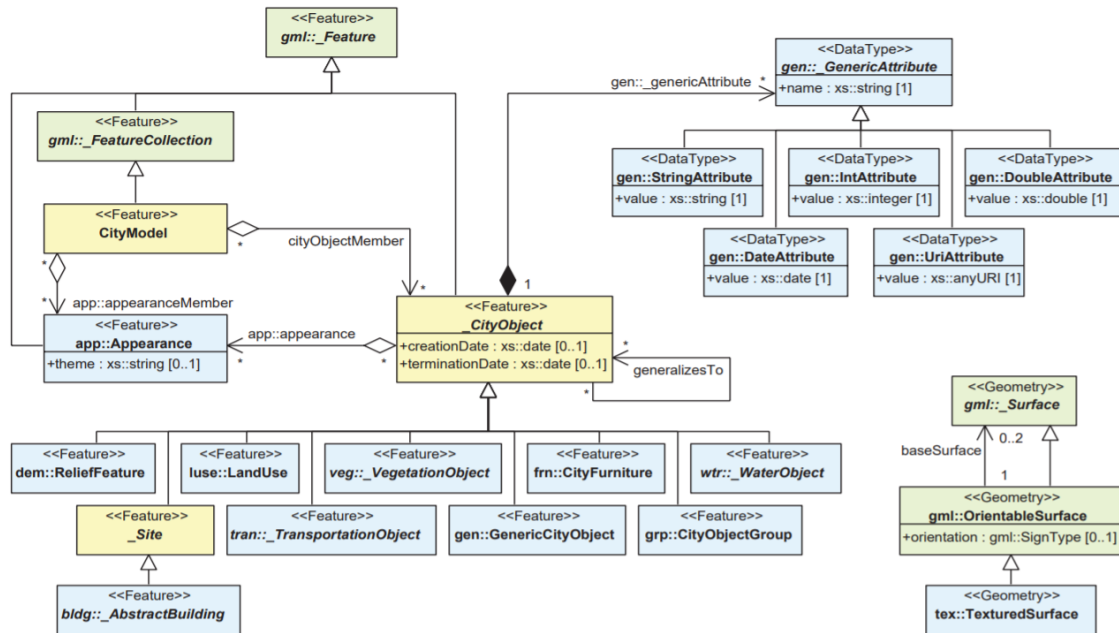


Figure 3.2: UML diagram of CityGML 2.0 Core module.

The *core* module defines the base class of all thematic classes within CityGML's data model, which is an abstract class called *\_CityObject* and all the previously mentioned thematic feature types are types of *\_CityObject*. *\_CityObject* provides a creation date and a termination date for the management of histories of features as well as the possibility to model external references to the same object in other data sets. Furthermore, two qualitative attributes *relativeToTerrain* and *relativeToWater* are provided, which enable to specify the feature's location with respect to the terrain and water surface.

*\_CityObject* is a subclass of the GML class *\_Feature*, thus it inherits the metadata property (which can be e.g. information about the lineage, quality aspects, accuracy, local CRS) and name property from the superclass *\_GML*. The previously mentioned CityGML thematic classes are subclasses of *\_CityObject* and may have further subclasses with relations, attributes and geometry. Features of the specialised subclasses of *\_CityObject* may be aggregated to a single *CityModel*, which is a feature collection with optional metadata. Generally, each feature has the attributes *class*, *function*, and *usage*, unless it is stated otherwise.

CityGML already defines many feature classes and attributes which are useful for a broad range of applications. However, in practical scenarios, it is often necessary to store and exchange extra attributes or even 3D objects which do not belong to any of the predefined classes. For these cases, CityGML generally provides two different ways of extensions. The first is the usage of generic city objects and generic attributes; both defined within the module *generics*. Any *CityObject* may have an arbitrary number of additional generic attributes. For each generic attribute of an object, the

name, type, and value have to be given within the CityGML dataset. Supported data types are string, integer, real, date, and URI (as shown in Figure 3.2). A *GenericCityObject* may be assigned arbitrary geometries (or CityGML *ImplicitGeometry*). As they are derived from *CityObject* they may also be assigned generic attributes.

The second concept for extending CityGML is the Application Domain Extension (ADE). An ADE specifies systematic extensions of the CityGML data model. These comprise the introduction of new properties, e.g. the energy-relevant attributes, to existing CityGML classes. The difference between ADEs and generic objects and attributes is that an ADE has to be defined within an additional XML schema definition file with its namespace. This file has to explicitly import the XML Schema definition of the extended CityGML modules. ADEs can be defined (and even standardised) by information communities which are interested in specific application fields. There are already many CityGML ADEs such as the Energy ADE (Agugiaro et al. 2018) and the Utility Network ADE (Kutzner et al. 2018) being used for different applications. A comprehensive list of CityGML ADEs is provided by (Biljecki et al. 2018).

### 3.2.2 Management of CityGML-based city models

Several software systems already support reading and processing CityGML files. Feature Manipulation Engine (FME<sup>62</sup>) is a commercial 'Extract Transform Load' (ETL) tool. It allows creating workbenches to read, write, and transform arbitrary CityGML datasets. Similarly, other commercial platforms such as Autodesk's InfraWorks<sup>63</sup> and ESRI 3D Cities Information Model (Reitz and Schubiger-Banz 2014) also support processing CityGML files. azul<sup>64</sup> is an open-source application developed by TU Delft for visualising CityGML models on the macOS operating system.

CityGML files can be huge as they often represent the entire city, state or even country-wide 3D geospatial data. For example, the New York City 3D Building model<sup>65</sup> based on CityGML 2.0 includes approx. 1.1 million 3D buildings. Similarly, the CityGML based 3D Building model<sup>66</sup> of the state North Rhine-Westphalia of Germany includes approx. 10.1 million 3D buildings. Besides, such large city models are also used for various complex GIS simulation and analysis tasks, which go far beyond pure 3D visualisation. Hence, such large files with geometric and semantic information must be efficiently stored and managed in database management systems allowing users to perform querying and analysis. The generic support for GML application schema is already provided by Open Source software frameworks such as deegree<sup>67</sup> and the OpenGIS Simple Features Reference Implementation (OGR) by GDAL<sup>68</sup> as well as the commercial software packages *CPA SupportGIS*<sup>69</sup> and *Snowflake GO LOADER / GO PUBLISHER*<sup>70</sup> offer generic support for GML application schemas. Since CityGML is a GML application schema, these software systems can automatically create database schemas for storing CityGML data for various database management systems like Oracle Spatial or PostgreSQL/ PostGIS, using the CityGML XML Schema definition files. Additionally, GeoRocket<sup>71</sup>

<sup>62</sup><https://www.safe.com/>

<sup>63</sup><https://www.autodesk.com/products/infraworks/overview>

<sup>64</sup><https://github.com/tudelft3d/azul>

<sup>65</sup><https://www1.nyc.gov/site/doitt/initiatives/3d-building.page>

<sup>66</sup><https://www.virtualcitysystems.de/aktuelles/458-3d-landesmodell-nrw>

<sup>67</sup><https://www.deegree.org/>

<sup>68</sup><https://gdal.org/drivers/vector/gmlas.html>

<sup>69</sup><http://www.cpa-software.de/>

<sup>70</sup><https://snowflakesoftware.com/geospatial-products/>

<sup>71</sup><https://georocket.io/>

is also a free and Open Source tool, which decomposes CityGML XML files and stores the XML fragments in a (distributed) file system like Amazon S3 or MongoDB. Furthermore, CityGML data can also be stored using the graph database Neo4j<sup>72</sup> as presented by (Nguyen et al. 2017).

The 3D City Database (3DCityDB)(Yao et al. 2018) is also an Open Source software which stores, represents, and manages the large CityGML datasets on top of a standard spatial relational database management systems such as Oracle Spatial and PostgreSQL/PostGIS. It provides a Java front-end application named '3DCityDB Importer/Exporter', which allows for high performance importing and exporting the CityGML datasets of arbitrary file sizes. It also allows exporting the contents in the form of different visualisation formats such as KML, COLLADA, glTF, and 3D Tiles allowing the 3D objects to be viewed and interactively explored in web applications. For integration into an OGC Web Service environment, the 3DCityDB contains a Web Feature Service (WFS) interface, using which the CityGML features can be requested in standardised ways. Thematic and generic attributes of city objects can be exported from 3DCityDB in tabular forms such as a CSV file or a Google Spreadsheet Document. Besides, thematic attributes can also be queried directly from the PostgreSQL REST API. Furthermore, 3DCityDB also provides functionality to validate CityGML documents.

For high-performance 3D visualisation and interactive exploration of arbitrarily large semantic 3D city models based on the CityGML standard, there are different web-based visualisation clients available. 3DCityDB-Web-Map-Client Pro (Chaturvedi et al. 2015) developed by the Chair of Geoinformatics, Technical University of Munich, is a web-based front-end client of 3DCityDB, which not only allows exploring and interacting with large semantic 3D city models, but also provides thematic querying capabilities on the 3D objects. For example, queries such as "*retrieving all the buildings objects located at a specific street*" and "*computing the total energy consumption of a specific residential area*" can be performed and results can be visualised directly on the 3DCityDB-Web-Map-Client Pro. It supports linking the 3D visualisation models (KML/glTF) with the cloud-based Google Spreadsheet documents allowing for querying the thematic data of every 3D object. virtualcityMAP<sup>73</sup> is a commercial web-based visualisation client with similar functionalities for working with large semantic 3D city models. This application has been developed by the company virtualcitySYSTEMS GmbH based in Berlin, Germany. 3DCityDB-Web-Map-Client<sup>74</sup> is a free and Open Source visualisation client developed by the Chair of Geoinformatics, the Technical University of Munich in cooperation with virtualcitySYSTEMS GmbH (Yao 2020). This client provides rich 3D visualisation and interactive exploration of arbitrarily large semantic 3D city models based on the CityGML standard. However, it does not support querying capabilities, unlike previously mentioned visualisation clients.

All of the visualisation clients use the Cesium<sup>75</sup> virtual globe as their visualisation engines. Cesium, developed by the Analytical Graphics, Inc., is an Open Source JavaScript package supporting the presentation and visualisation of 3D contents directly within web browsers. It enables users to dynamically switch between 3D globe visualisations and 2D map projections. It utilises the standards HTML5 and WebGL that allow hardware acceleration for loading 3D contents without any requirement to install additional plug-ins. Hence, it allows applications to be used as cross-platform, cross-browser, and cross-device.

---

<sup>72</sup><https://neo4j.com/product/>

<sup>73</sup><https://www.virtualcitysystems.de/en/products/virtualcitymap>

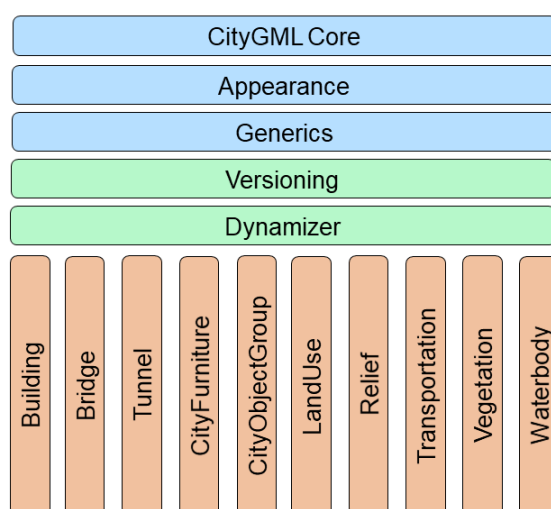
<sup>74</sup><https://github.com/3dcitydb/3dcitydb-web-map>

<sup>75</sup><https://cesium.com/>

### 3.3 Realisation of the concepts with the CityGML standard

#### 3.3.1 Data Models

As discussed in section 3.1, due to the fundamental difference between slower and highly dynamic changes, this thesis recommends extending semantic 3D city models for dealing with both types of changes in two different ways. Part I of the thesis introduces conceptual data models for the CityGML standard and describes how these data models support the requirements [R1-R7], which were identified in Chapter 2. The Versioning concept (c.f. Chapter 4) deals with slower changes and allows representing historic and parallel versions of 3D city models. The Dynamizer concept (c.f. Chapter 5) deals with highly dynamic changes and allows representing as well as linking city object properties with numerous sources of highly dynamic time-dependent properties. The Dynamizer concept also provides a method for injecting dynamic variations of city object properties into the static representation making city objects truly dynamic.

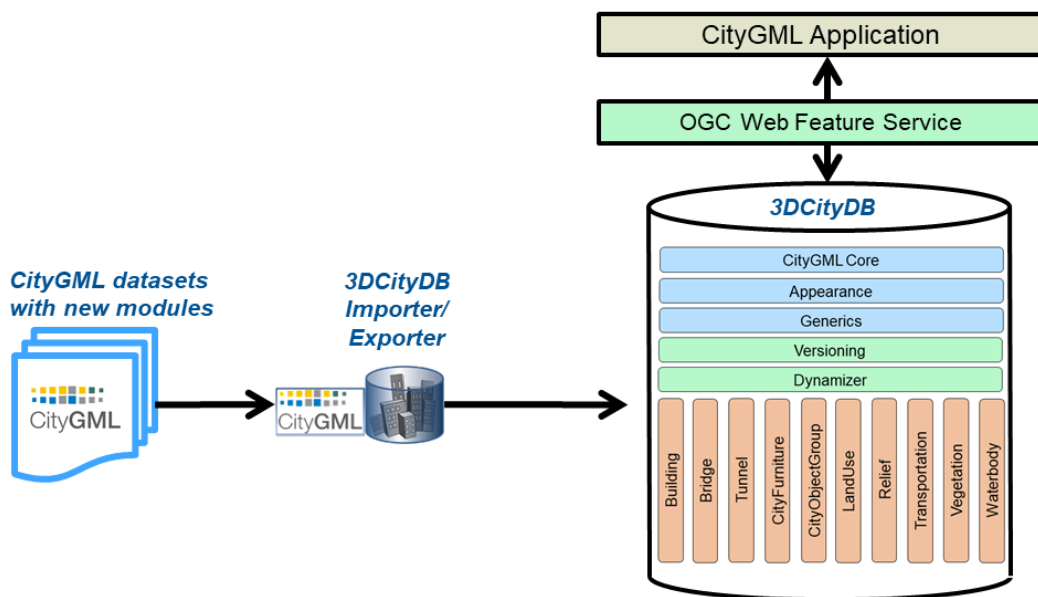


**Figure 3.3:** Addition of two new modules Versioning and Dynamizer (shown as green) for CityGML. The vertical boxes show the different thematic module. Horizontal modules specify concepts that are applicable to all thematic modules.

Both concepts have been proposed in a way that they apply to all the thematic modules (Figure 3.3). For example, a Versioning module allows representing multiple versions of different thematic modules such as buildings and streets. Similarly, a Dynamizer module allows linking a real-time sensor stream to a property of any city object like a building or a city furniture. For applicability with CityGML 2.0, the Versioning and Dynamizer concepts have been implemented as Application Domain Extensions (ADEs). However, they are also planned to become a part of the next version of CityGML (CityGML 3.0) (Kutzner et al. 2020).

### 3.3.2 Data Management

This thesis presents the management of both Versioning and Dynamizer modules using the software 3D City Database (3DCityDB). 3DCityDB is an Open Source software, is maintained regularly, and already supports all the classes and schema for CityGML 2.0. One of the other primary reasons to select 3DCityDB for this thesis is that it includes an ADE Plugin Manager for its Importer/Exporter. The ADE Plugin Manager allows dynamically extending a 3DCityDB instance to facilitate the storage and management of arbitrary ADEs. The Versioning and Dynamizer concepts within this thesis have been proposed for CityGML 3.0. However, they can also be implemented as ADEs for CityGML 2.0. Hence, the 3DCityDB is an ideal software to apply and demonstrate the newly added functionalities. However, 3DCityDB currently manages only static properties of city objects. Part II of the thesis presents concepts for managing time-dependent properties along with static properties using 3DCityDB. This thesis provides a relational database model (c.f. Chapter 6) for storing and managing time-dependent properties for the city objects. Further, the CityGML objects along with their corresponding time-series data (e.g. a building along with the monthly solar irradiation values or monthly energy consumption values) can be imported and exported in standardised ways.



**Figure 3.4:** Data Management of new CityGML modules within 3DCityDB.

One more advantage of using 3DCityDB for this study is that it comes with a Web Feature Service (WFS) interface which allows CityGML features to be accessed in a distributed environment (e.g. in a Spatial Data Infrastructure) as shown in Figure 3.4. The WFS interface is also beneficial for visualisation applications allowing to interpret and visualise geometry and semantics of city objects. However, a WFS is not suitable to query dynamic/time-series data. This thesis also presents a novel concept called InterSensor Service (c.f. Chapter 7), which allows integrating dynamic information with city objects and their properties without needing to store them in 3D city databases. This functionality

is highly suitable for working with sensors and IoT devices, generating a massive amount of time-series data (e.g. observations every minute or every second). The InterSensor Service is a lightweight application and is provided as Open Source software.

### **3.3.3 Proof of concept**

Part III of the thesis provides proofs of concepts that have been developed in the previous two parts. Chapter 8 provides implementations for the integrated and unified visualisation of time-dependent properties along with static properties of city objects. The proposed framework enables applications (such as 3D city model viewers) to access static data and dynamic data in an integrated fashion. Several demonstrations are presented covering the use cases of real-world Smart City projects.

Furthermore, Chapter 9 highlights the fact that security is a crucial component in a distributed environment. The chapter provides solutions for securing the overall access and management of distributed applications and services by giving a demonstration example in a Smart City project. The concept facilitates privacy, security and controlled access to all stakeholders and the respective components by establishing proper authorisation and authentication mechanisms.





## **Part I**

# **Integration of Time-dependent Properties**



## Chapter 4

---

### Modelling Slower Changes

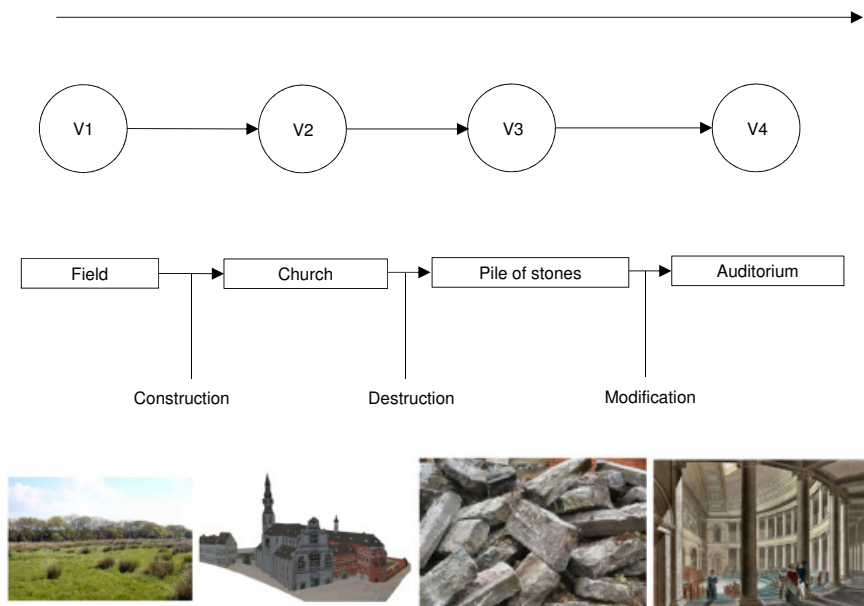
This chapter focuses on changes that are slower such as (i) history or evolution of cities [R6] and (ii) planned alternatives by urban planners [R7]. A novel concept called the Versioning concept is introduced to extend the CityGML data model and its exchange format to support different versions and version transitions to allow identification and organisation of multiple states in a city model. The approach helps to deal with two critical facets of multi-representation of semantic 3D city models. The first facet is the maintenance of the complete history or evolution of the city model, which is supported by version transitions. The transitions include bi-temporal attributes, which help in answering the questions such as “How did the *city* look like at a specific point in time?” and “How did the *city model* look like at a specific point in time?”. The second facet of multi-representation is managing parallel alternative designs of the objects at the same time. The approach allows different versions to be used in an interoperable exchange format and exchanging all the versions of a repository as one dataset. Furthermore, this single dataset can be used by different software systems to visualise, compare, and work with all the versions.

This chapter is based on the published paper and is a joint effort by the authors mentioned as follows:

**Chaturvedi, K.**, Smyth, C. S., Gesquière, G., Kutzner, T. and Kolbe, T. H. (2017). ‘Managing Versions and History Within Semantic 3D City Models for the Next Generation of CityGML’. In: *Advances in 3D Geoinformation*. Ed. by Abdul-Rahman, A. Cham: Springer International Publishing, pp. 191–206. URL: [https://doi.org/10.1007/978-3-319-25691-7\\_11](https://doi.org/10.1007/978-3-319-25691-7_11)

## 4.1 Versioning in semantic 3D city models

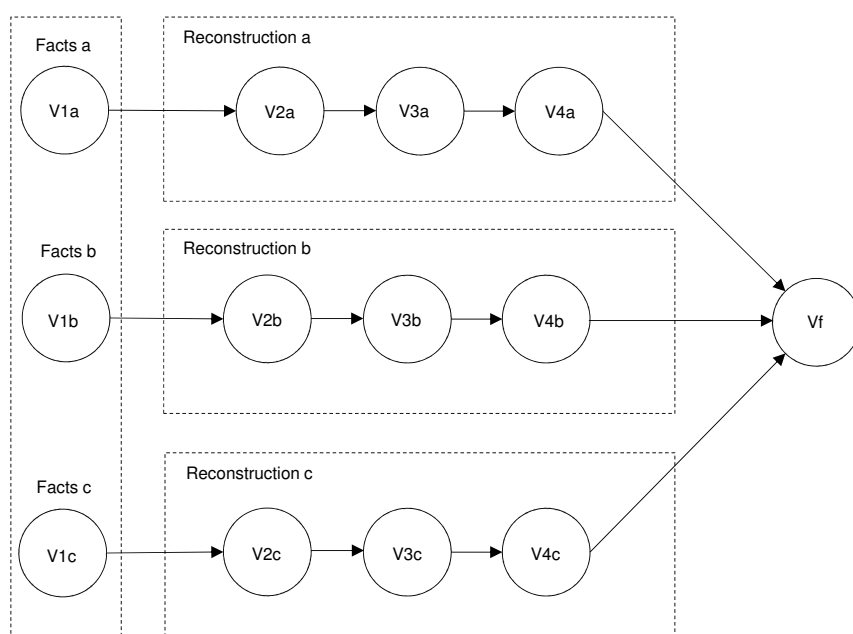
As we learnt previously, semantic 3D city models are an essential source of information in applications related to urban planning, architecture, business development, tourism, history, and archaeology. These areas of application often involve studying the evolution of cities representing how city objects change over a long period. For example, for a time sequence, a building may be constructed, modified, demolished and replaced by other ones. Similar needs can also be identified in serious game projects where objects of a scene may evolve according to a given scenario. For instance, a building may be represented in different states like “destroyed”, “burned” or “partially destroyed” that can be called by the application in coordination with user actions (Figure 4.1). These changes are slower and involve features that begin or cease to exist over different periods. For example, if a new building feature is constructed after a specific time, it is not possible to query the building before the specified time as there was no existence of this feature. Hence, such changes can be managed using historic or linear versions of city models, including up-to-date information about newly added, modified, or demolished objects.



**Figure 4.1:** An illustration of historical succession. Image adapted from (Pfeiffer et al. 2013)

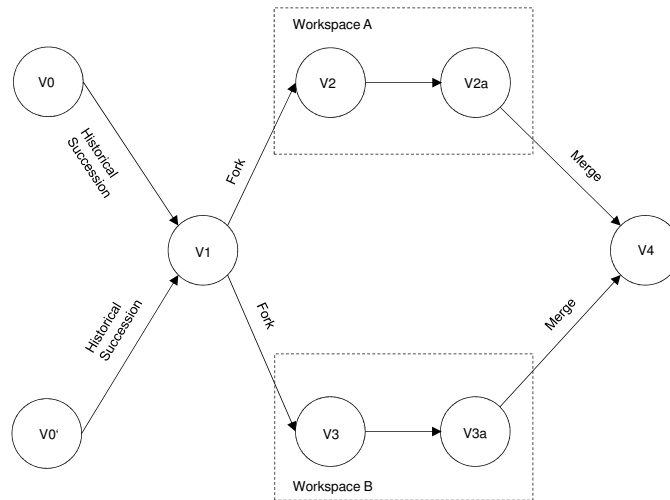
Semantic 3D models also play a crucial role in the documentation and reconstruction of both historical and contemporary events. Examples include crime scene and accident reconstructions (Wolff and Asche 2009), representation of battles and other past events, archival descriptions of ancient structures before demolition, documentation of construction and demolition of buildings (Pfeiffer et al. 2013). Such events involve a sequence of versions of a “reality” at a specific time or in a particular time sequence. Events are often reconstructed from conflicting and incomplete evidence, and a complete reconstruction must allow branching to handle the alternative possibilities. For example, figure 4.2 shows three alternative versions of a city model, labelled within circles as

$V1a$ ,  $V1b$ , and  $V1c$ , of a model based on different facts and information. A version of a city model is a snapshot representing the state of all features of the entire city models at a specific point in time. Based on more information and data collection for each fact, parallel and alternative versions, possibly managed by different authorities, are represented corresponding to the earlier versions. For example, the authority A manages a workspace to reconstruct subsequent versions from 'V1a' and updates them to versions  $V2a$  followed by  $V3a$ , and  $V4a$ . Over a period, changes in these versions are represented by transitions shown by the arrows in the figure. In the end, the alternatives versions, managed by different authorities are merged to form the final version with in the main trunk. More details on version transitions and merging approaches are given in section 4.1.1.



**Figure 4.2:** Reconstruction of events to handle alternative models.

There are also scenarios in Urban Planning requiring backward compatibility to handle multiple representations of the past of a city. A given date may be a starting point to imagine the past and constructing several scenarios. As shown in figure 4.3, the version  $V1$  indicates the current state (version) of the entire city model. Two different past states of the city model are denoted as versions  $V0$  and  $V0'$ . The version transition, in this case, is shown as *Historical Succession* (more details on naming the version transitions is given in section 4.2.3). The different planning authorities can also work with alternative planned versions at the same time to insert a newly generated object or delete or update any existing object. As shown in figure, the main version  $V1$  is *forked* by two different authorities or workers at the same time. After completion of their respective changes (concurrent editing) within their respective workspaces, the versions can be *merged* to form the final version. More details on version transitions and merging approaches are given in section 4.1.1.



**Figure 4.3:** Managing parallel or alternative versions.

#### 4.1.1 Requirements for modelling the new Versioning concept

As already described in Chapter 2, Versioning is not a new concept and there are already several Version Control Systems such as Git, Mercurial, Concurrent Version System (CVS) and Subversion (SVN) (details and references of these systems are given in section 2.2.5). The Version Control Systems (VCS) have the representational power to manage changes, parallel updates, and merges of versions of CityGML models with one exception: versions always represent change in the forward direction of time. The collection maintained by a VCS is rooted in an original version, that is, the versions form a rooted directed acyclic graph (DAG). The original version is the oldest version and it is not possible to create versions earlier than the root. In addition to the problem of forward-only temporality, a VCS also has a designated node in the DAG, usually called the head, and a distinguished path in the DAG, from the root to the head, usually called the "trunk" or "main branch". Neither of these is required for maintaining versions of CityGML models. Although it might be possible to gain the needed representational power by piecing together multiple VCS projects, which share a common root, it would be awkward, at best.

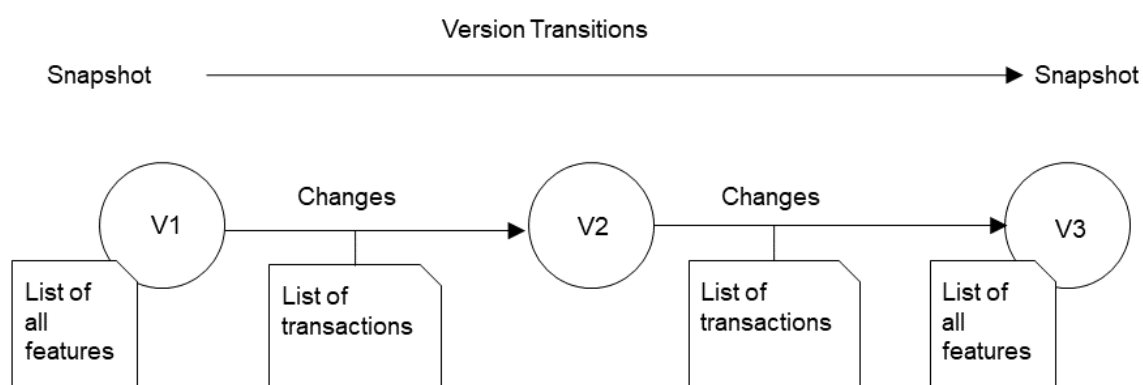
The concept of versions has also successfully been incorporated by database management systems such as Oracle Workspace Manager and ESRI ArcSDE Geodatabases (c.f. section 2.2.5). The Oracle Workspace Manager allows managing multiple versions of the data in the same Oracle Relational Database Management System in the form of workspaces. A workspace is a virtual copy of the data, which separates the collection of changes in the different versions from the live (production) data. The version-enabled data are stored in separate tables with additional columns representing the version metadata. Such additional columns contain the version and workspace of each data row along with the date and time of each update. The database view is created on the version-enabled table and triggers are defined to enable SQL operations such as insert, delete, and update. This approach allows preservation of the structure of the original table and shows the data of only the particular version. The Oracle Workspace Manager also provides the management of historical data by using savepoints and

the means for resolving possible conflicts during the merging of the different versions. Similarly, ESRI ArcSDE Geodatabases also supports managing history, performing “what-if” analysis and conflict detection and resolution. However, there are no interoperable exchange formats related to both Oracle and ESRI which would allow exchanging all versions of a repository as one dataset, nor which would allow the use of the same dataset by both Oracle and ESRI.

The other standards for semantic data models for city objects such as INSPIRE also provide the concept of chronological versioning (details are given in section 2.3). INSPIRE defines several requirements and recommendations for modelling life-cycle information of spatial objects which include UML stereotypes and properties allowing for bi-temporal modelling of geospatial objects. Furthermore, a separate property exists for denoting a specific version of a geospatial object. However, currently only exchanging the last version of spatial objects is supported by INSPIRE; historic versions cannot be provided yet (and especially not within one data file).

Based on the above-mentioned discussions by reviewing other standards, tools, and database management systems, the following requirements were gathered to be included in the proposed new approach:

- None of the standards supports managing multiple historic versions. INSPIRE supports exchanging only the last version of spatial objects. The proposed methodology should allow supporting multiple historic versions within one data file.
- The existing approaches allow only forward-temporality. The proposed approach should allow backward compatibility to handle multiple representations of the past of a city.
- Although the DBMS systems such as Oracle or ESRI ArcSDE Geodatabases already support versions and conflict managements, the proposed approach within CityGML documents would allow exchanging all versions of a repository as one dataset and furthermore, the same dataset would be used by DBMS systems such as Oracle and ESRI.

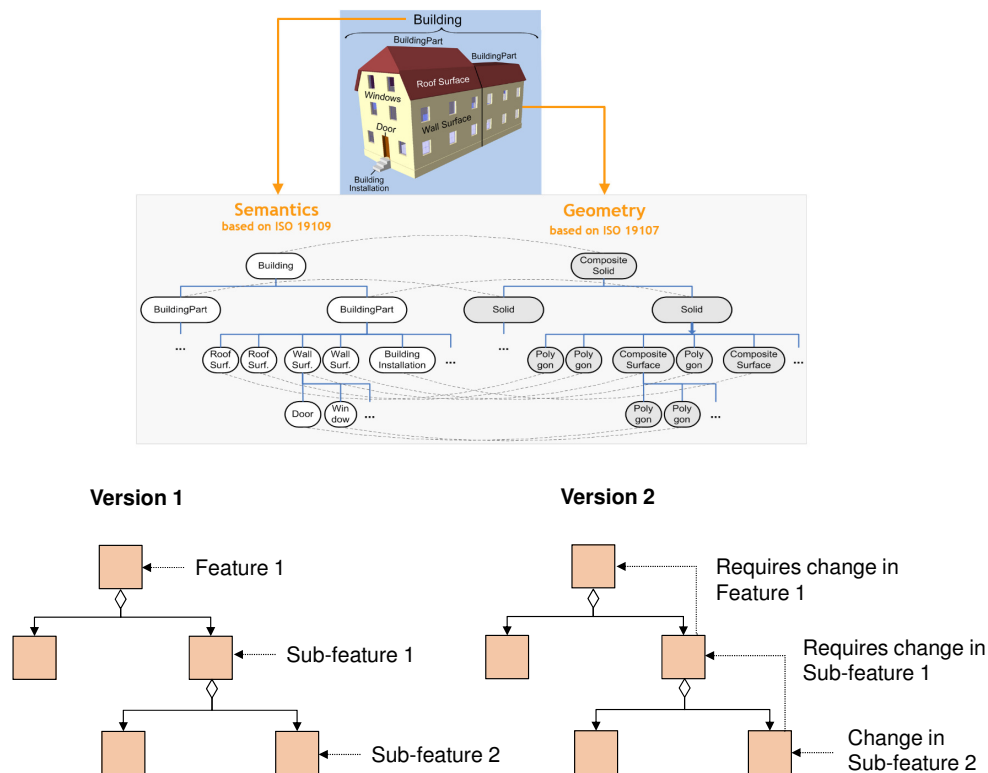


**Figure 4.4:** Representation of Version Transitions.

Another important aspect to be considered is the management of versions and their transitions. As shown in the previous section, a version of a city model is a snapshot representing the state of all features of the entire city model at a specific point in time. Over a period, changes in versions are represented by transitions. Version transitions document the causal relationship between the version snapshots (Figure 4.4). For example, a version transition can be a historical succession representing the evolution of a city over a time period. The transition can also be a planning activity where authorities

work in parallel with different workspaces to insert, delete, or modify objects. Similarly, the transition can also be merge where parallel workspaces are merged to form the final version. Similarly, in the design stages, the transitions can also be planned and realized, indicating the alternative plans suggested to be included in the versions and later realization of the suggestions after the planning. Hence, versions and version transitions together allow identifying and organising multiple states of a city model at different points in time. Each version transition involves a list of transactions representing what kind of changes occur within a transition, for example, whether a new feature is added, replaced, or modified.

A critical aspect with version transitions is the merging of two different versions, which may lead to possible conflicts. For all such convergence situations, it must be ensured that the members of the converged version/state can be determined unambiguously. For example, in the figure 4.3, two authorities work in parallel on their respective workspaces to add, delete, or modify an object. It is important to ensure that both the transitions should be able to detect who has changed an object and whether there are any conflicts. The easiest and safest way within the new modelling approach is to require that at maximum, one of the incoming transitions has transactions. In this way, at most one transaction can be performed at a time for all the incoming transition avoiding any possible conflict.



**Figure 4.5:** Issues with versioning of aggregated features. Image adapted from (Stadler and Kolbe 2007).

Another vital aspect to consider is that 3D city model standards such as CityGML allow features to have aggregated sub-features. For example, a *Building* feature can have boundary surface features



(e.g., *WallSurface*), which may further consist of sub-features such as *WindowSurface* or *DoorSurface*. However, in case of a change in any of the sub-features, the model would require changing all the parent features in the aggregation levels above because aggregate objects point to their parts. If a new version replaces the building part with a new gml:id, the pointer in the aggregate object also will have to be updated. This change will create a new object version for the aggregate object too, and so on, following up the aggregation hierarchy. For example, the window in figure 4.5 has been replaced by a new window with insulated glazing and a new frame. In the updated version, the window will have to be changed along with its parent features. This problem should also be avoided by an appropriate modelling approach.

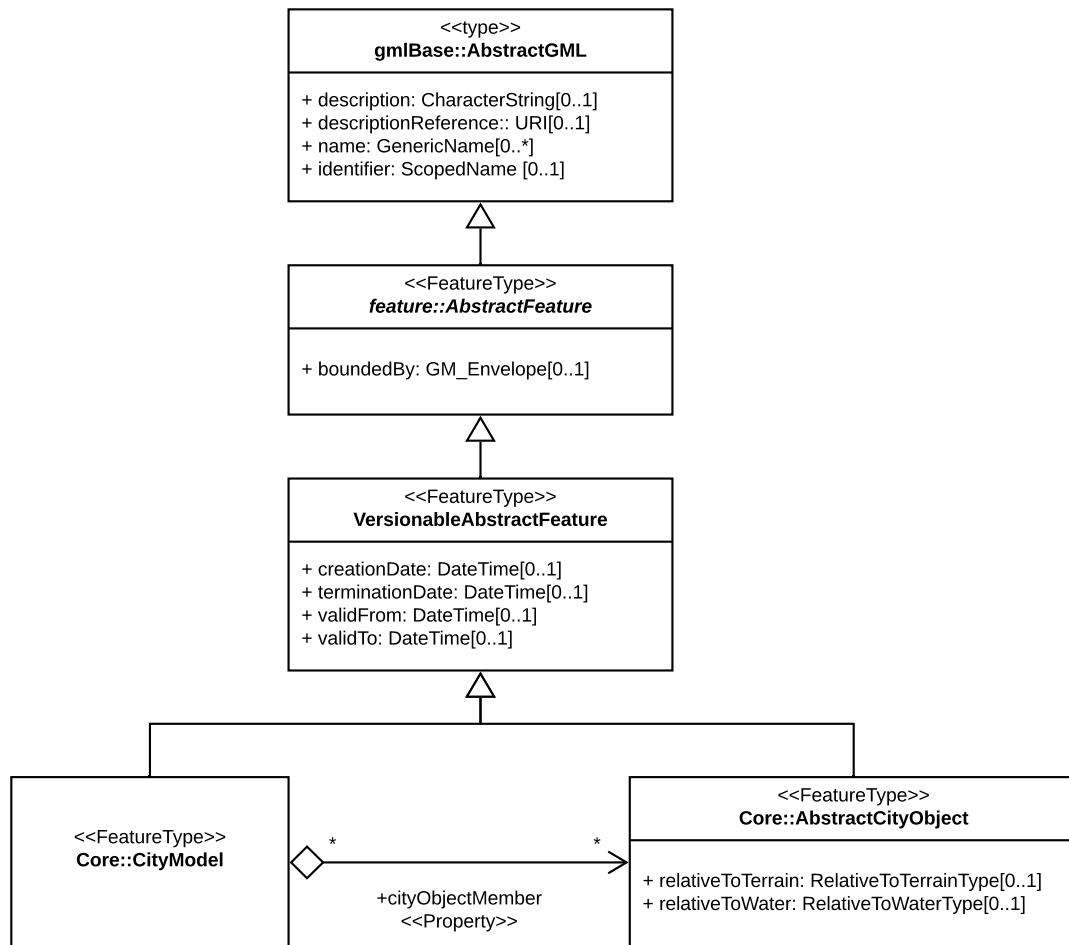
## 4.2 Modelling the Versioning concept within the CityGML standard

The following sub-sections describe the development of the UML model of the Versioning concept based on the gathered requirements.

### 4.2.1 Versionable Features

CityGML 2.0 allows defining the life cycle of objects using the attributes *creationDate* and *terminationDate*. However, these attributes only refer to the period a specific city object is a part of the city model. To represent the complete evolution, it is also essential to document the period spanning the validity of city objects in the real world. The complete history or evolution of the city model can be supported by version transitions having bi-temporal attributes (Jensen and Snodgrass 1999). They may be helpful in answering the questions such as "How did the city look like at a specific point in time?" and "How did the city model look like at a specific point in time?". In the Versioning concept, a new abstract subclass *VersionableAbstractFeature* of the class *AbstractFeature* is introduced allowing all geo-object types to become versionable (Figure 4.6). The class *VersionableAbstractFeature* contains four time attributes for expressing a bi-temporal existence model for versions. Apart from the already existing attributes *creationDate* and *terminationDate* reflecting the transaction time, the class introduces two new attributes *validFrom* and *validTo* for reflecting the actual world time. This approach is similar to the existing INSPIRE model where these attributes can be used to query how the city model looks like at a specific point in time and how the actual city looks like at a specific point in time. These attributes can be defined as an extension to the CityGML core module and can replace the existing attributes *yearOfCreation* and *yearOfDemolition* attributes in the CityGML building module.

In addition, the city object is now assigned a stable object identifier for its entire life-cycle. This stable identifier is termed as "major ID". This stable identifier is supported by GML3.2.1 through the element *gml:identifier* in the class *AbstractGML*, which is used for providing globally unique identifiers. Furthermore, an extension to this "major ID" is given in the form of a "minor ID" to distinguish different versions of the same real-world entity. The combination of "major ID" and "minor ID" allows the representation and exchange of not only the current version but also the entire history of a city model in the same dataset, because it avoids conflicts of having multiple instances with the same object ids (the different versions of the same real-world object) in the same file/database. A separator symbol "\_" is introduced to separate the stable identifier from the individual version. For example, the specific version of a city object can be denoted as *Building1020\_Version1*, where *Building1020* is the *gml:identifier* representing the global unique "major ID" and *Version1* is the "minor ID" to represent the specific version of the building object *Building1020*. This concatenated form (*Building1020\_Version1*) is used as the *gml:id* to distinguish the different versions of the same



**Figure 4.6:** Versionable Features of CityGML.

real-world object. One CityGML instance document can, therefore, include multiple versions of the same real-world object having different *gml:id* but identical *gml:identifier* values. This approach also allows determining the different versions even if they are created in the same dates. For example, if we have 3 alternative versions of the same object at the same point in time, then "majorId\_creationDate" representation would not work. However, the representation "majorId\_creationDate\_Version1" would allow to form a unique *gml:id* of the object even if different versions of this object are created on the same date. The idea of "minor ID" and "major ID" has been adopted from the German AAA model (AAA 2014) and the INSPIRE Data Specifications (INSPIRE 2013). Referencing objects by either their "minor ID" or their "major ID" is inspired from the Solid Earth and Environment GRID (Cox 2006).

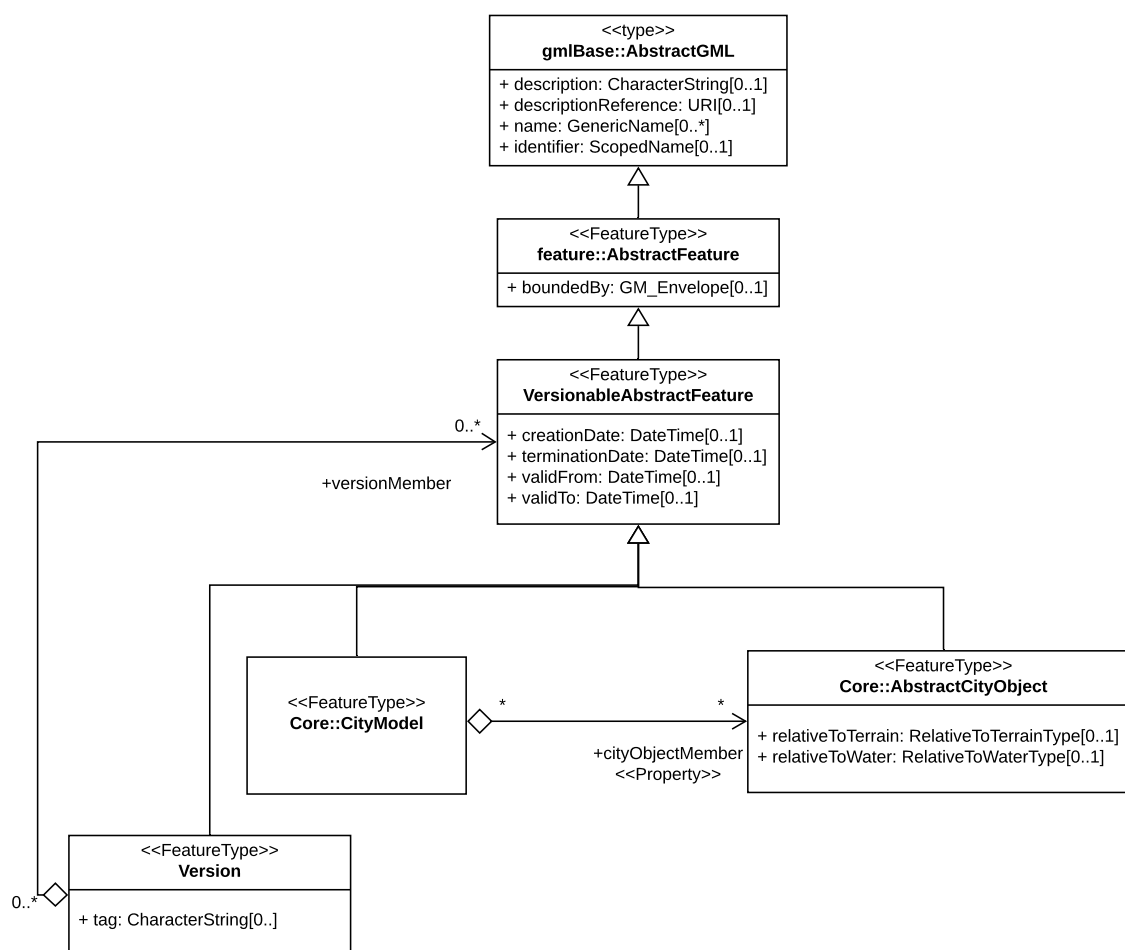


Figure 4.7: Introduction of the new Version feature.

#### 4.2.2 Version - a new Feature Type

The class *Version* is introduced to manage a specific version representing the state of the city model at a specific point in time (Figure 4.7). This class allows each version to be denoted by a set of user-defined keywords named as *tag*. With the help of such *tag* attributes, a user can perform searches based on specific keywords, e.g., "search for a version developed by worker A". Each version contains the city objects or a group of city objects by using the association *versionMember*. By using this association, each city object can be referenced by a specific version. The city objects within each version can be referenced in two ways: (i) by using a simple XLink to the *gml:id* of the referenced object which references a specific version of a real-world object, or by using the XML Path Language (XPath)<sup>76</sup>. XPath in conjunction with XLink allows referencing an object element in a remote XML document (or GML object repository) using the *gml:identifier* property of that object (Cox 2006). The XPath-XLink approach provides a general reference to a real-world object by its "major ID" and does

<sup>76</sup><http://www.w3.org/TR/xpath20/>

not take into account a specific version. This approach allows selecting multiple instances with the same *gml:identifier* value, but with a different *gml:id*. For example, by using a single XPath-XLink query, the user can retrieve multiple versions of the CityGML building parts within the same version of the building. However, the application must determine which specific version of the real world object representation should be used. The attributes *creationDate*, *terminationDate*, *validFrom*, and *validTo* can then be used to choose the appropriate version that was valid at a specific database or real-world time, respectively. Additionally, referencing the "major ID" attribute using the XPath-XLink mechanism also resolves the versioning of aggregated features. An illustration example for managing such versions is given in section 4.3.

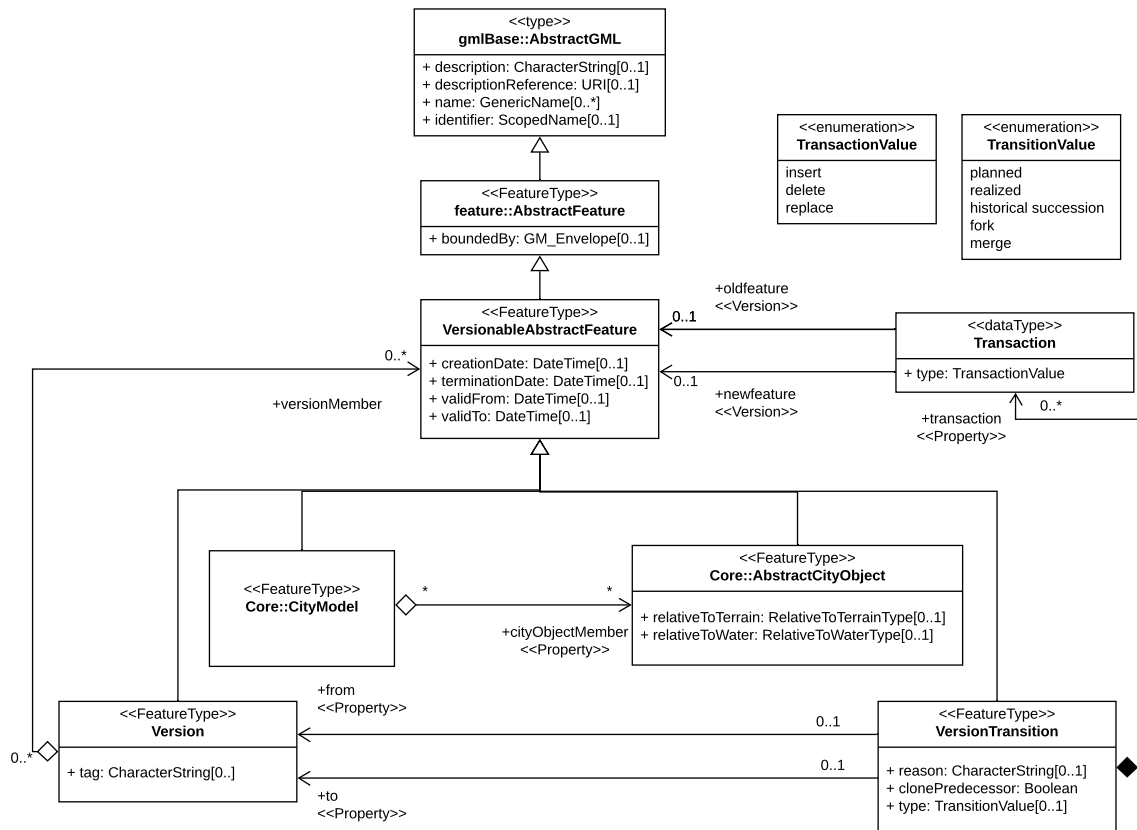


Figure 4.8: Introduction of the new VersionTransition feature.

### 4.2.3 Version Transitions

To represent each transition, they are modelled as a separate feature type named *VersionTransition* (Figure 4.8). The class contains the following attributes:

- *reason* reflecting the reason for the change in version. The reason can be defined as a *CharacterString* and acts as a metadata information about the specific transition. The authorities/users can document here the reason for making the edits in the respective version of the city model.

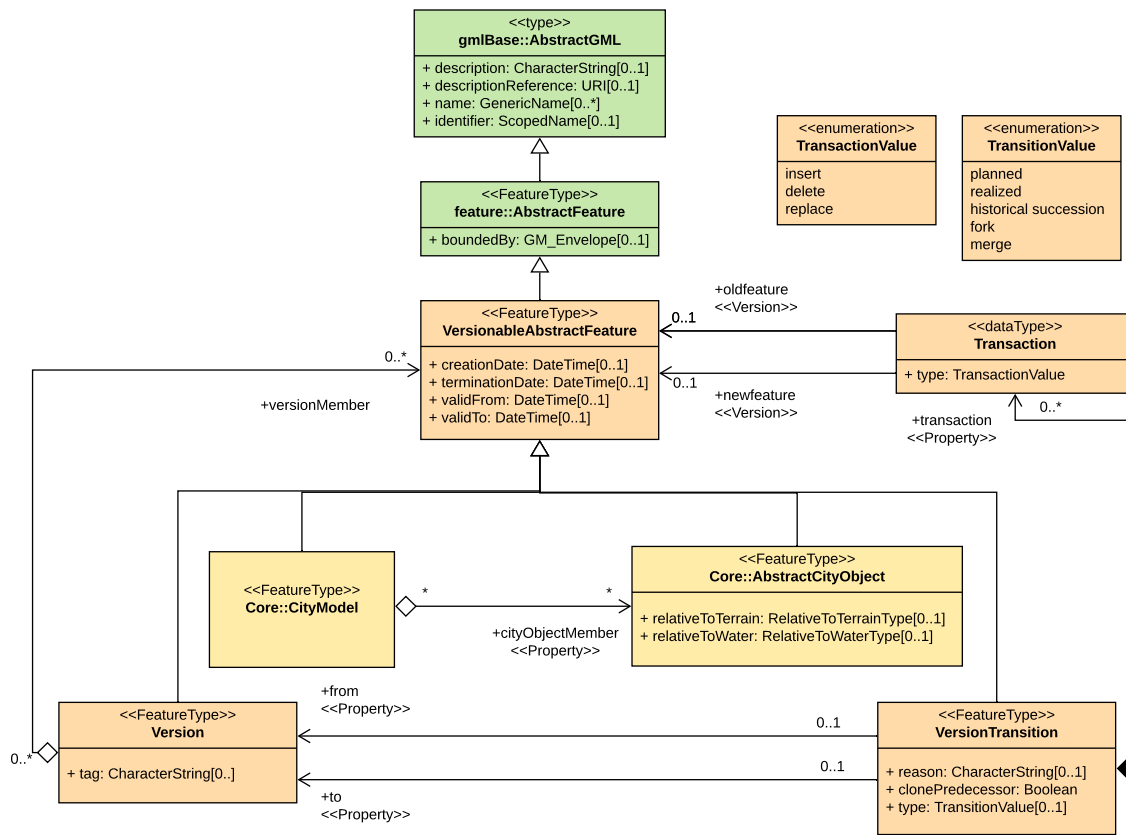
- *clonePredecessor* is of type *Boolean* and accepts values in the form of "true" or "false". If it is "true", it means that all features from the predecessor version are also member of the successor version. Only those features, for which there are additional transactions associated with the *VersionTransition*, will be modified. If this attribute is "false", it means that the set of members of the predecessor version is not copied and that the successor version only will contain those features, that are explicitly enumerated by their *versionMember* associations. The advantage is that with the help of the *clonePredecessor* attribute, we can choose whether the members of the successor version are to be modified from the predecessor version just by some incremental changes, i.e. the transactions. Or if, for the successor version, every member instance will be explicitly enumerated. If *clonePredecessor* is "false", it does not make sense to have transactions listed for that *VersionTransition*.
- *type* indicating what is the type of this transition. As shown in figure 4.3, there may be different types of transitions depending on the requirement of the specific users or authorities. The new model lists different possible types of transitions in the enumeration *TransitionValue*. It accepts the values as (i) *planned* indicating the alternative plans suggested or to be included in the versions, (ii) *realized* indicates the realization of the suggestions after the planning stage, (iii) *historicalSuccession* indicates the changes or modifications in the city objects as a succession of the previous version allowing to represent historical city developments, (iv) *fork* indicates the forking or creating a new sub-branch from the main trunk by the user to perform the required modifications, and (v) *merge* indicates the merging of the branches in the main trunk to form the final version.

Each *VersionTransition* is associated to two specific *Versions* by the associations *from* and *to*. This association allows mapping the relationship between versions within a transition. For each *VersionTransition*, the transaction type can be defined by using the Data Type *Transaction*. The *Transaction* type allows determining the type of changes within the features. The attribute *type* is enumeration *TransactionValue* and accepts values as (i) *insert* indicating addition of new features, (ii) *delete* indicating removal of existing features, and (iii) *replace* indicating modifications of existing features. The *Transaction* type also defines mapping to the affected features by associations *oldFeature* and *newFeature*. Versions and their respective transitions form non-cyclic directed graphs.

The main advantage with *VersionTransition* is that this approach requires low memory and storage requirements. It is similar to the combination of full back-ups and incremental back-ups. Incremental back-ups are the back-ups of all the changes since the last full or incremental back-up. The defined attributes within the class *VersionTransition* allow changes to be expressed incrementally. That means, it is possible to determine the exact changes since the previous version. It helps avoiding creating, and further managing, a complete new version if only parts of a city model or parts of a complex object, like a building or roads, are updated.

#### 4.2.4 Complete UML Model of the Versioning concept

Based on the developments of individual feature types, the new Versioning concept is presented in this thesis work as shown in figure 4.9. Classes shown in green and yellow are from GML and CityGML respectively. Classes shown in orange are newly introduced classes of the Versioning concept. The Versioning concept has been developed for the next version of the CityGML standard (version 3.0) and has been proposed to the OGC CityGML Standard Working Group for its adoption. However, the proposed concept can also be implemented as a CityGML Application Domain Extension (ADE) by using the "hook" mechanism (Gröger et al. 2012). It would allow Versions and Version Transitions to be used with the current version of the CityGML standard (version 2.0).



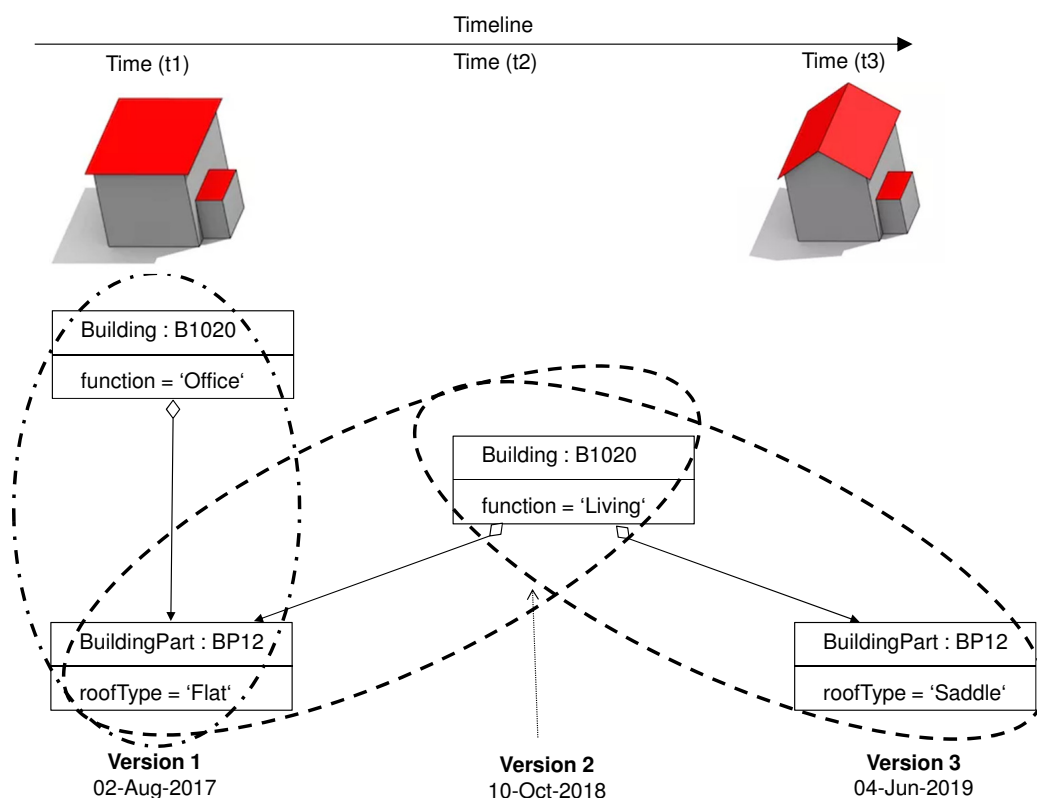
**Figure 4.9:** Complete UML model of the CityGML Versioning concept. Classes shown in green and yellow are from GML and CityGML respectively. Classes shown in orange are newly introduced classes of the Versioning concept.

### 4.3 Illustration of the Concept

This section explains an example scenario for managing different versions using the CityGML Versioning concept. Figure 4.10 shows a simple scenario where a building undergoes changes at different points in times. At time "t1", the building has function property as "Office" and roofType as "Flat". At time "t2", the office property changes to residential property and hence, the function of the building changes to "Living". At time "t3", it is observed that the building exists at the same location; however, after a significant renovation, the roof structure has changed. Hence, the roofType property changes to "Saddle" at the time "t3". Thus, the city object within a city model can be represented as version V1 at the time "t1", version V2 at the time "t2", and version V3 at the time "t3".

#### 4.3.1 Using new CityGML identifiers

Listing 4.1 shows an example representing city objects of different versions using the combination of "major ID" and "minor ID" in a single CityGML instance dataset. As shown in figure 4.10, the *Building* object can be defined with a "major ID" *B1020*. This identifier is stable and is valid for all the different



**Figure 4.10:** An instance example of versions representing modifications of a building.

versions. This *Building* object has a *BuildingPart* object for its roof structure. The *BuildingPart* object is defined with the "major ID" *BP12*. For specific versions, the "minor ID" is associated to the "major ID" which can be considered as *gml:id* for the specific versions. As shown in the below listing, in version 1, the *gml:id* of the *Building* object is "B1020\_version1" with *function* attribute defined as "Office". Similarly, the *BuildingPart* object has *gml:id* "BP12\_version1" with *roofType* attribute as "Flat". In the similar ways, in version 2, the *Building* object as *gml:id* as "B1020\_version2" where the *function* attribute changes to "Living". However, in version 2, there was no change in the *BuildingPart* object which changed in version 3. Hence, there is no *BuildingPart* object associated with the minor ID version 2. Please note that each *Building* object contains a link to the respective *BuildingPart* object using a single XPath-XLink query. In this listing, it is represented as `"/identifier[text()='BP12']"`. In this XPath-XLink query, the expression `"/identifier"` allows selecting nodes in the document from the current node that match the selection *identifier* no matter where they are. Further, the expression `"[text()='BP12']"` allows querying the identifier node whose value string matches with 'BP12'. In this way, this XPath-XLink query allows selecting multiple instances of the same "major ID" ('BP12' in this case) and hence, "minor ID" is not associated with "major ID" in such queries. The applications can determine which specific version of the real world object representation should be used. The attributes *creationDate*, *terminationDate*, *validFrom*, and *validTo* can also be used to choose the appropriate version that was valid at a specific database or real world time, respectively.

**Listing 4.1:** Representation of multiple object versions within one single CityGML dataset

```

<!-- XML namespaces have been omitted in this listing -->
<cityObjectMember>
  <Building gml:id="B1020_version1">
    <identifier>B1020</identifier>
    <consistsOfBuildingPart>
      <BuildingPart xlink:href="//identifier[text()='BP12']"/>
    </consistsOfBuildingPart>
    <creationDate>2017-08-02</creationDate>
    <terminationDate>2018-10-10</terminationDate>
    <function>Office</function>
  </Building>
</cityObjectMember>
<cityObjectMember>
  <Building gml:id="B1020_version2">
    <identifier>B1020</identifier>
    <consistsOfBuildingPart>
      <BuildingPart xlink:href="//identifier[text()='BP12']"/>
    </consistsOfBuildingPart>
    <creationDate>2018-10-10</creationDate>
    <function>Living</function>
  </Building>
</cityObjectMember>
<cityObjectMember>
  <BuildingPart gml:id="BP12_version1">
    <identifier>BP12</identifier>
    <creationDate>2017-08-02</creationDate>
    <terminationDate>2019-06-04</terminationDate>
    <roofType>Flat</roofType>
  </BuildingPart>
</cityObjectMember>
<cityObjectMember>
  <BuildingPart gml:id="BP12_version3">
    <identifier>BP12</identifier>
    <creationDate>2019-06-04</creationDate>
    <roofType>Saddle</roofType>
  </BuildingPart>
</cityObjectMember>

```

### 4.3.2 Using Version and Version Transitions

In the above illustration (as shown in figure 4.10), it is also possible to manage different versions and version transitions within a single CityGML document. For the same *Building* and *BuildingPart* objects, the *Version* and *VersionTransitions* can be represented as shown in listing 4.2.



**Listing 4.2:** Representation of version transitions within one single CityGML dataset. This listing extends Listing 4.1.

```

<!-- XML namespaces have been omitted in this listing -->
<cityObjectMember>
  <ver:Version gml:id="version1">
    <ver:tag>Developed by Worker A</ver:tag>
    <ver:versionMember>
      <bldg:Building xlink:href="//identifier[text()='B1020']"/>
    </ver:versionMember>
  </ver:Version>
</cityObjectMember>
<cityObjectMember>
  <ver:Version gml:id="version2">
    <ver:tag>Developed by Worker A</ver:tag>
    <ver:versionMember>
      <bldg:Building xlink:href="//identifier[text()='B1020']"/>
    </ver:versionMember>
  </ver:Version>
</cityObjectMember>
<cityObjectMember>
  <ver:Version gml:id="version3">
    <ver:tag>Developed by Worker A</ver:tag>
    <ver:versionMember>
      <bldg:Building xlink:href="//identifier[text()='B1020']"/>
    </ver:versionMember>
  </ver:Version>
</cityObjectMember>
<cityObjectMember>
  <ver:VersionTransition gml:id="transition1">
    <ver:reason>Change of Building Function</ver:reason>
    <ver:clonePredecessor>true</ver:clonePredecessor>
    <ver:type>historicalSuccession</ver:type>
    <ver:from xlink:href="#version1"/>
    <ver:to xlink:href="#version2"/>
    <ver:transaction>
      <ver:Transaction>
        <ver:type>replace</ver:type>
        <ver:oldFeature xlink:href="#B1020_version1"/>
        <ver:newFeature xlink:href="#B1020_version2"/>
      </ver:Transaction>
    </ver:transaction>
  </ver:VersionTransition>
</cityObjectMember>
<cityObjectMember>
  <ver:VersionTransition gml:id="transition2">

```

```

<ver:reason>Change of Building Part</ver:reason>
<ver:clonePredecessor>true</ver:clonePredecessor>
<ver:type>historicalSuccession</ver:type>
<ver:from xlink:href="#version2"/>
<ver:to xlink:href="#version3"/>
<ver:transaction>
  <ver:Transaction>
    <ver:type>replace</ver:type>
    <ver:oldFeature xlink:href="#BP12_version1"/>
    <ver:newFeature xlink:href="#BP12_version3"/>
  </ver:Transaction>
</ver:transaction>
</ver:VersionTransition>
</cityObjectMember>
<!-- Building and BuildingPart features from Listing 4.1
are to be added here -->

```

This listing shows different *Versions* and *VersionTransitions* represented within a single CityGML dataset. Each Version is defined using a specific gml:id, e.g. *Version1*. By using the *tag* attribute, we can define a user-defined tag, for example, version developed by a specific worker or department. Each *Version* includes a specific *VersionMember*, which refers to the specific versions of the Building object *B1020* in this case. The *VersionTransition* is also defined with a specific gml:id and includes the appropriate values for the required attributes, for example, *reason*, *clonePredecessor*, and *type* of the transitions as mentioned in the listing. In our example, first transition is defined with the gml:id *transition1* and changes from *version1* to *version2* as defined within the fields *from* and *to*. Each transition has a specific transaction. In the case of *transition1*, the *function* value of the Building object is changed. Hence, the transition takes place from the feature *B1020\_version1* to *B1020\_version2*, which is defined within the fields *oldFeature* and *newFeature*. The attributes for *transition2* are defined in the similar ways.

## 4.4 Discussions

This chapter presents a novel versioning concept supporting the management of historic versions [R6] and alternative versions [R7] within CityGML. The advantage of this approach is that it not only facilitates the data model for supporting different versions but also allows the different versions to be used in an interoperable exchange format and the exchange of all versions of a repository within one dataset. Such a dataset can be used by different software systems to visualise and work with all the versions. The approach not only addresses the implementation of versionable CityGML models but also considers new aspects to previous work, such as managing multiple histories or multiple interpretations of the past of a city. The UML model of the Versioning concept handles versions and version transitions as feature types, which allows the version management to be handled entirely using the OGC Web Feature Service. No extension of other OGC standards is required. The Versioning concept has already been adopted and further extended in the research work by (Samuel et al. 2018).

The concept already addresses the possibility that every feature of CityGML can be made versionable. However, in the future, it might also be required to make individual geometry objects within

city models versionable. It is possible to introduce the versioning of objects at a higher level in the class hierarchy of GML just below *AbstractObject*. In this way, every object of CityGML (and of GML in general) would become versionable. However, this would require changes in the GML specification, which is out of the scope of the OGC CityGML standards working group (CityGML SWG). In the presented form, the concept is entirely modelled within the framework of the CityGML application schema. The CityGML SWG can standardise it without changing other OGC or ISO specifications. For implementation, this concept also does not require a database or GIS with specific version management capabilities.

As mentioned earlier in this chapter, the Versioning concept has been developed for the next version of the CityGML standard (version 3.0) and has been proposed to the OGC CityGML Standard Working Group for its adoption. However, the proposed concept can also be implemented as a CityGML Application Domain Extension (ADE) by using the "hook" mechanism. It would allow Versions and Version Transitions to be used with the current version of the CityGML standard (version 2.0).



## Chapter 5

---

### Modelling Highly Dynamic Changes

This chapter focuses on changes that represent high frequent or dynamic variations of the object properties. Such variations may be related to (i) thematic attributes such as changes of physical quantities (energy demands, temperature, solar irradiation levels), (ii) spatial properties such as change of a feature's position (moving objects), and (iii) appearances such as building textures or colours. In this case, only some of the properties of otherwise static objects need to represent such time-varying values. This chapter presents a new concept called "Dynamizer", which allows extending static 3D city models by supporting variations of individual feature properties and associations over time. The Dynamizer concept fulfils the requirements [R1-R5]. It provides a data structure to represent dynamic values in different and generic ways. Such dynamic values may be given by tabulation of time/value pairs; patterns of time/value pairs; by referencing an external file; or by retrieving observations from sensor and IoT devices. In principle, Dynamizers inject dynamic variations of city object properties into the static representation. These variations are supported for thematic, geometry, and appearance properties of city objects. The conceptual details of Dynamizer represented by the UML model of the CityGML standard are presented in this chapter.

Some of the discussions in this chapter have been presented in the published papers

**Chaturvedi, K.** and Kolbe, T. H. (2017). *Future City Pilot 1 Engineering Report - OGC Doc. No. 16-098*. Tech. rep. Open Geospatial Consortium. URL: <http://docs.opengeospatial.org/per/16-098.html>

**Chaturvedi, K.**, Willenborg, B., Sindram, M. and Kolbe, T. H. (2017). 'Solar Potential Analysis and Integration of the Time-dependent Simulation Results for Semantic 3D City Models using Dynamizers'. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences IV-4/W5*, pp. 25–32. URL: <https://doi.org/10.5194/isprs-annals-IV-4-W5-25-2017>

**Chaturvedi, K.** and Kolbe, T. H. (2016). 'Integrating Dynamic Data and Sensors with Semantic 3D City Models in the context of Smart Cities'. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences IV-2/W1*, pp. 31–38. URL: <https://doi.org/10.5194/isprs-annals-IV-2-W1-31-2016>

**Chaturvedi, K.** and Kolbe, T. H. (2015). 'Dynamizers - Modeling and Implementing Dynamic Properties for Semantic 3D City Models'. In: *Eurographics Workshop on Urban Data Modelling and Visualisation*. Ed. by Biljecki, F. and Tourre, V. The Eurographics Association. URL: <http://dx.doi.org/10.2312/udmv.20151348>

## 5.1 Making 3D City Models Dynamic

As we learnt in Chapter 3, individual city object properties (such as geometry, topology, semantic, and appearance) change over time. In many scenarios, these changes are highly dynamic and associated with only one attribute of city objects. For example, a solar potential simulation generates monthly irradiation values for a building roof or wall surface, for example named "*globalRadMonth*". These monthly values are associated with only one attribute of the building roof surface. Similarly, a Smart Meter API providing real-time energy consumption readings are associated with only one attribute of the building object, for example named "*electricityConsumption*". Most often, these changes are recorded as a time-series represented by a tabulation of timestamps and property values. However, such time-series may belong to different data sources; for example, electricity consumption readings of a building can either be determined from a sensor API or can also be recorded in a CSV file. Similarly, solar potential simulation results can be stored in a database, and a moving vehicle's locations can be mapped in a GPS file. Hence, from the data modelling perspective, the following aspects need to be considered:

- **Representation of time-series data:** As the dynamic data may belong to different sources, the new approach should enable representing the time-series data in different and standardised ways. The simulation-specific time-series data can be represented in-line with city objects allowing the exchange of city objects along with accurate description and metadata of the time-series. In some cases, such simulation results can also be imported in a tabulated file such as CSV. However, in other scenarios, the frequency of time-series can be very high. For example, an indoor sensor, measuring air humidity every 30 seconds inside the living room of a building, produces a considerable amount of observations. It can be cumbersome for the 3D city model management systems. Moreover, the sensor and IoT standardised solutions such as OGC SensorThings API (Liang et al. 2015) and OGC Sensor Observation Service (Bröring et al. 2012) already provide sophisticated data models and management solutions for representing, storing, and querying sensor metadata and raw observations. In such cases, it is not required to represent such huge time-series observations in-line with city objects. However, an explicit link from the city object property to the sensor API would be more efficient. This approach would allow defining a connection of the city object to the physical sensor device, which is associated with the respective city object. At the same time, this link would also make a connection to the city object property, which is being measured by the sensor device and the sensor API querying the raw observations for the respective city object property.
- **Overriding the specific property values:** The dynamic values are associated with a specific object property of a city feature. Therefore, the data model should allow referring to that particular property whose value can be then overridden by the dynamic value specified by the sources as mentioned earlier. For example, a building can have multiple thematic attributes such as an address, owner, number of floors, building height etc. These attributes are mostly static. However, if a new attribute *electricity\_consumption* is defined for the same building and the real-time measurements of this attribute are obtained from a sensor device installed in the building, that would make this attribute dynamic. An energy application may require to perform temporal queries based on the attribute *electricity\_consumption* of the building. For example, "*computing the total electricity consumption of the building between 6 pm and 7 pm on a specific day*", "*comparing the energy levels between weekdays and weekends*", "*notifying the building owner if the energy consumption breaches a specified threshold value*", and so on. Such queries would require that the attribute values of the building property "*electricityConsumption*" can be overridden by the time-series data obtained from its source. Hence, from the modelling perspective, only dynamic attributes would be changed by

this approach, while the static attributes would remain unaffected. In that way, there would be no need to make the entire model dynamic (c.f. section 5.2).

- **Periodic and repetitive patterns:** As highlighted in section 2.1.2.2, the dynamic data are often provided by a means for the tabulation of measured data. However, it is not sufficient in many applications as they may require patterns to represent dynamic variations of properties based on statistics and general rules. For example, energy applications can be used to study patterns in the energy consumption of a building for weekdays, weekends, public holidays, or even customised period (e.g. between 6 pm and 10 pm). In these cases, time may be defined for a non-specific year (e.g. averages over many years), but still classified by the relative time of a year. For example, January monthly summaries for the energy consumption of a building might be described as "all-Januaries 2001-2010". Similarly, the energy consumption values may reflect generic patterns for individual weekdays/weekends in a week or a month. Another example scenario may also be determining patterns for specific seasons (such as spring, summer, autumn and winter) over ten years. In different simulations, such time-series can also be used as a basis for defining schedules. For example, schedules in the energy simulations may be required for specifying setpoint values for the heating and cooling systems, or for setting the operational schedules of energy systems, ventilation, lighting, and electrical appliances. Similarly, in the cases of traffic analysis, a public bus line following a schedule can also have a repeating trajectory. Hence, the new data model should represent complex patterns that could be based on statistics and general rules. A working example is demonstrated in section 5.3.3.

Based on the outlined requirements, the Dynamizer concept has been developed as a part of this thesis work. This concept allows modelling and integrating dynamic properties within semantic 3D city models.

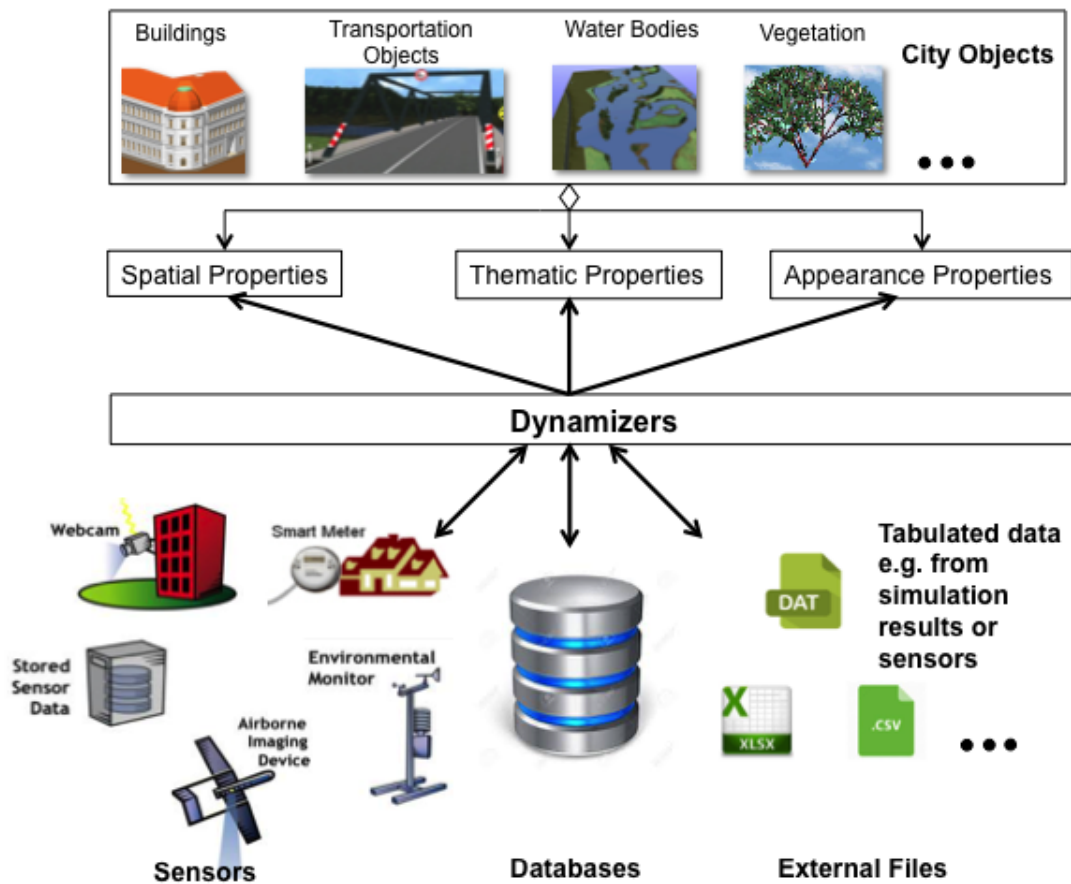
As shown in figure 5.1, Dynamizer serves three main purposes:

1. Dynamizer is a data structure to represent dynamic values in different and generic ways. Such dynamic values may be given by tabulation of time/value pairs; patterns of time/value pairs; by referencing an external file. These values can be obtained from sensors, simulation specific databases, and also external tabulated files such as CSV or Excel sheets.
2. Dynamizer delivers a method to enhance static city models by dynamic property values. It references a specific property (e.g. spatial, thematic or appearance properties) of an object within a 3D city model providing dynamic values overriding the static value of the referenced object attribute.
3. Dynamizer objects establish explicit links between sensor/observation data and the respective properties of city model objects that are measured by them. By making such explicit links with city object properties, the semantics of sensor data become implicitly defined by the city model.

In this way, Dynamizer can be used to inject dynamic variations of city object properties into an otherwise static representation. The advantage of using such an approach is that it allows only selected properties of city models to be made dynamic. If an application does not support dynamic data, it simply does not include these particular types of features.

## 5.2 Modelling the Dynamizer concept within the CityGML standard

The following sub-sections describe the development of the UML model of a Dynamizer based on the gathered requirements. Similar to the Versioning concept, the Dynamizer concept has been developed for the next version of the CityGML standard (version 3.0) and has been proposed to the OGC CityGML Standard Working Group for its adoption. However, the proposed concept can also be implemented as a CityGML Application Domain Extension (ADE) by using the "hook" mechanism



**Figure 5.1:** Conceptual illustration of CityGML Dynamizers. This concept allows (i) the representation of time-variant values from sensors, simulation specific databases, and external files and (ii) enhancing the properties of city objects by overriding their static values.

(Gröger et al. 2012). It allows Dynamizer to be used with the current version of the CityGML standard (version 2.0).

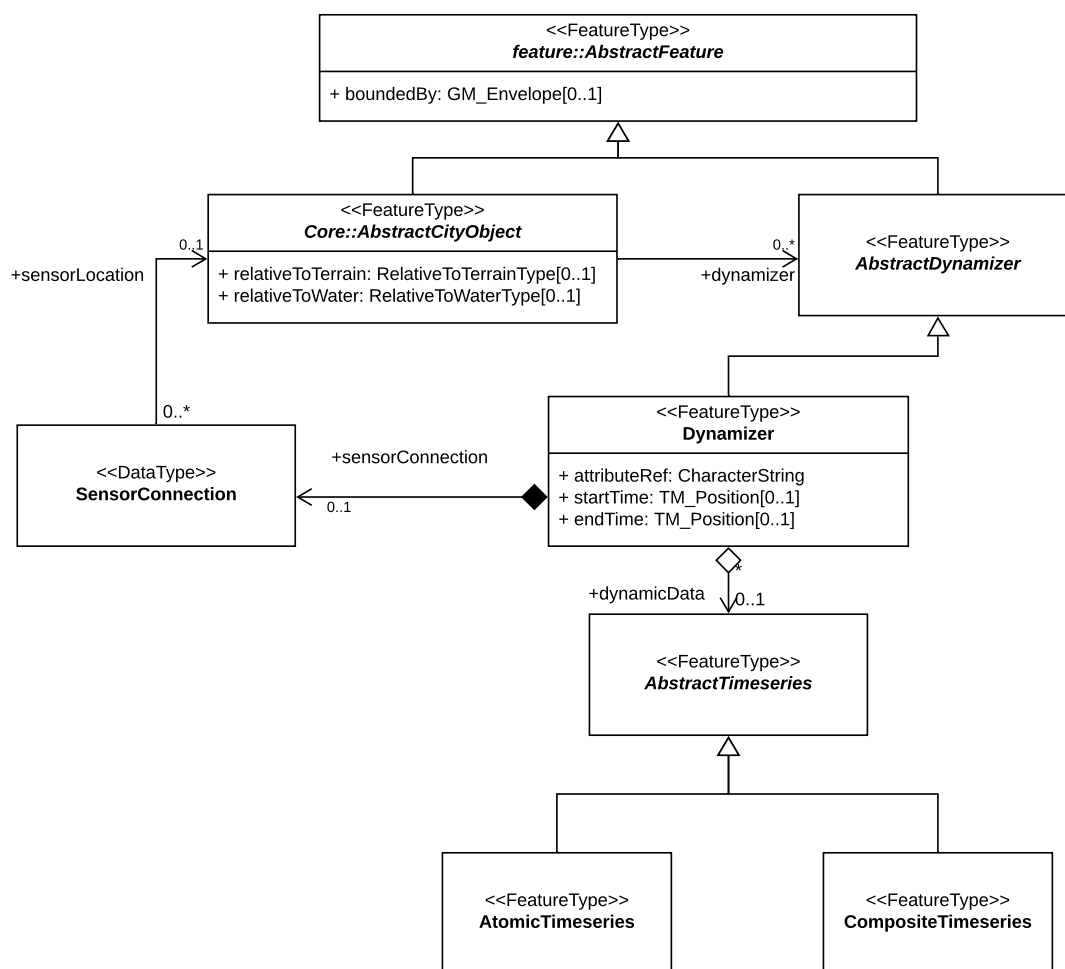
### 5.2.1 Dynamizer - a new Feature Type

CityGML 2.0 is based on the Geography Mark-up Language (GML) version 3.1.1 and CityGML 3.0 will be based on the version GML 3.2. GML 3.2 already allows expressing temporal developments of the features using the so-called *history* property. The *history* property of a dynamic feature contains a sequence of time slices, which captures the evolution of a feature over time. Since CityGML is based on GML, this functionality becomes the obvious choice to include dynamic features in CityGML. However, this would require that we redefine the property data types of all features to use the GML dynamic data mechanism. That would result in a total replacement of the existing CityGML data model. The GML dynamic data schema would have to be generally supported by all systems, even



if none or only very few attributes had time-varying properties. Moreover, GML has no support for representing repetitive patterns.

Hence, *Dynamizer* is defined as a new *FeatureType* which is a sub-class of the GML *AbstractFeature*. In addition, *AbstractCityObject* has an additional association *dynamizer* to the *Dynamizer FeatureType*. This allows all city objects such as buildings, roads, vegetation, etc. to include their *Dynamizer* features either in-line or as links to their respective dynamizer features (see Figure 5.2).



**Figure 5.2:** Dynamizer modelled as a new *FeatureType*.

The class *Dynamizer* consists of three attributes: (i) *attributeRef*, (ii) *startTime*, and (iii) *endTime*. *attributeRef* refers to a specific property of a static CityGML feature which value will then be overridden or replaced by the (dynamic) values specified in the *Dynamizer* feature. In this way, *attributeRef* allows referring to the city object's very specific property which can be related to its geometry, semantics, or even appearances and overriding its values without affecting other static properties. The reference to the specific attribute of a city object is provided by XPath<sup>77</sup>, which is a

<sup>77</sup><https://www.w3.org/TR/xpath-30/>

W3C recommendation used to navigate through the elements and attributes within an XML document. *startTime* and *endTime* are absolute time points denoting the time span for which the Dynamizer provides dynamic values. The time points are modelled as *TM\_Position* defined by the standard ISO 19108<sup>78</sup>, and are referenced to a specific time reference system (e.g. Gregorian Calendar). For implementation purposes, it is a good practice to treat *startTime* and *endTime* using the Half-Open Interval<sup>79</sup> approach where the *startTime* is inclusive and the *endTime* is exclusive. For example, in order to define a range of one calendar year, the *startTime* may be assigned as "2017-01-01T00:00:00" and *endTime* may be assigned as "2018-01-01T00:00:00". The Half-Open approach includes the entirety of 2017-01-01, but excludes 2018-01-01. For date-time with a fractional second, this approach eliminates the problem of trying to capture last moment as various systems may use various granularities such as milliseconds, microseconds, or nanoseconds. With the Half-Open approach, a day, for example, starts at the first moment of the day and runs up to, but does not include, the first moment of the following day. It is also true for other granularities. For example, a week starting on a Monday runs up to, but does not include, the following Monday. A month starts on the 1st and runs up to, but does not include, the first of the following month thereby ignoring the challenge of determining the number of the last day of the month, which may also be the February 29th in a Leap Year.

Dynamizer allows representing dynamic data in the following two ways:

- **Direct links from city object properties to their respective sensor data:** One option is to establish direct links to a sensor or IoT API measuring the required dynamic data using the *Data Type SensorConnection* (c.f. section 5.2.2). *SensorConnection* allows defining the connection parameters to a specific API using its well-defined properties. In this way, Dynamizer establishes a direct link to a specific property measured by a sensor API. At the same time, using the *attributeRef* attribute, Dynamizer overrides the same property defined within the city object. The *SensorConnection* also provides a direct association *sensorLocation* to the *AbstractCityObject*. This association allows specifying to which city object this sensor belongs. For example, a solar panel installed on a building roof surface could be used as a sensor providing the currently generated electrical power. In this case, the *Data Type SensorConnection* would not only provide direct links to the sensor description and observations but also specify the link to the roof surface of the CityGML building object on which the solar panel is installed.
- **Representing time-series in-line with city objects in standardised ways:** The second possibility is to represent a time-series within the Dynamizer Feature. This is modelled as the *AbstractTimeseries* class. The time-series can be modelled in two ways: *AtomicTimeseries* (c.f. section 5.2.3) representing tabulation of timestamps and property values in different and generic ways, and *CompositeTimeseries* (c.f. section 5.2.4) representing repetitive patterns based on statistics and general rules. Based on the defined time-series values, using the *attributeRef* attribute, Dynamizer overrides the property value defined within the city object.

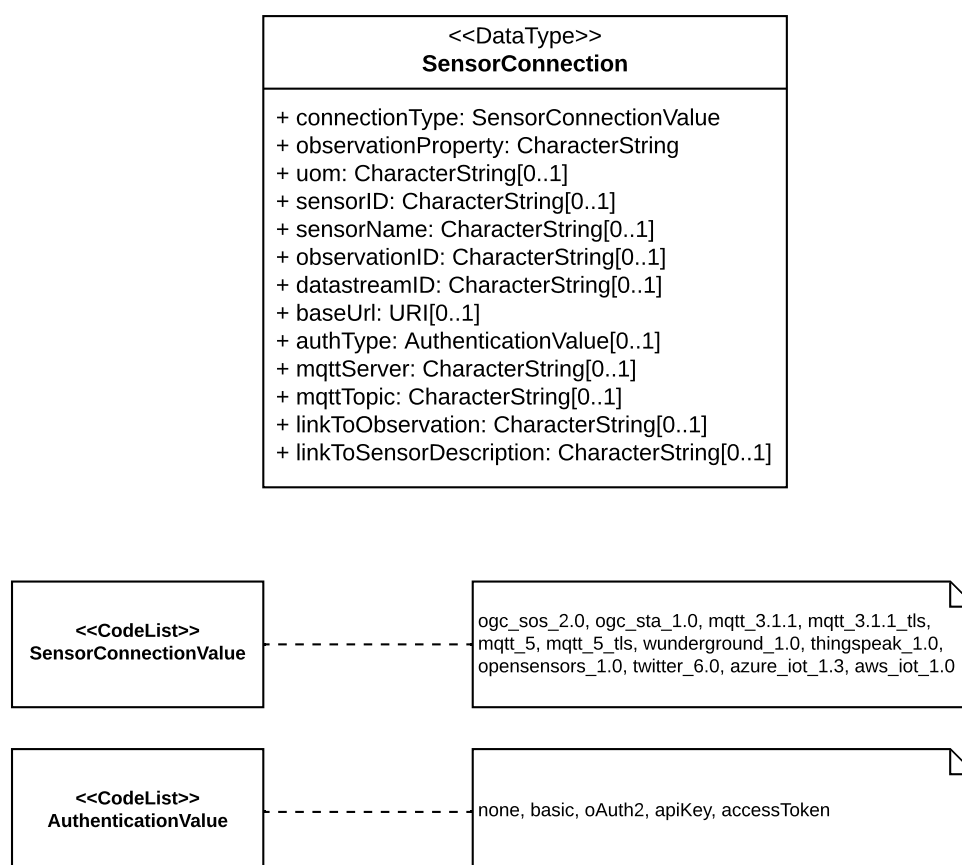
### 5.2.2 SensorConnection

The *Data Type SensorConnection* allows defining connection details to arbitrary APIs managing different kinds of sensor data. These APIs can be open and based on international standards such as OGC SensorThings API and OGC Sensor Observation Service. The APIs can also be proprietary and belong to specific companies such as MathWorks Thingspeak, IBM Weather Underground, Microsoft Azure IoT. The detailed discussion on such APIs is given in section 2.2.1. The *SensorConnection* type

<sup>78</sup><https://www.iso.org/standard/26013.html>

<sup>79</sup><https://mathworld.wolfram.com/Interval.html>

provides different attributes for (i) establishing connections to these APIs based on HTTP requests, (ii) subscribing to a specific data stream using MQTT parameters, and (iii) if required, providing security parameters depending on different authentication protocols. Hence, this class fulfils the requirements [R1-R2].



**Figure 5.3:** Representation of the data type *SensorConnection*, allowing to define the details of sensor based services within the CityGML document.

As shown in figure 5.3, the *sensorConnection* type includes the following key attributes:

1. *connectionType* defines the type of the sensor API. The different possibilities of a connection type are defined within the «CodeList» *SensorConnectionValue* such as OGC Sensor Observation Service, OGC SensorThings API, MQTT v3.1.1 and so on.
2. *observationProperty* defines the name of property which is being measured by the specific API and which the Dynamizer refers. For example, a weather station API usually measures temperature, humidity, wind speed, humidity, and precipitation. Similarly, a Smart Meter installed in a building may record electricity or gas consumption of the building.

3. *uom* is the unit of measurement specified in the API for the defined observed property. For example, the unit of measurement of temperature is Celsius, relative humidity is Percentage, electricity consumption is kWh, and so on.
4. *sensorID* is the unique identifier of the sensor device registered at the specific API. In some cases, this identifier can be defined by a unique number or a digit. In other cases, it can also be defined as a string value. Hence, the data type of this attribute is *CharacterString*.
5. *sensorName* is the name of the sensor device registered with the web service. Some standards (such as OGC Sensor Observation Service) allow querying the observations based on the sensor name.
6. *datastreamID* is the unique identifier of the datastream. The term *Datastream* is defined as a collection of observations measuring the same observed property and produced by the same sensor. For example, an indoor sensor installed in a living room of a building measuring the properties temperature and humidity of the room involves two datastreams for temperature and humidity having a unique identifier for each datastream.
7. *observationID* is the unique identifier for an individual observation within a datastream. Many APIs also allow querying based on individual observation IDs.
8. *baseURL* is the resource locator at the root level. Every API request contains a base URL which is appended by the specific parameters such as sensorID, ObservedProperty, temporal range, and so on. The *baseURL* attribute can be defined once and the subsequent parameters can dynamically be assigned by the users.
9. *authType* is the type of the authentication protocol. The sensor APIs, most often, involve different authentication protocols to ensure secure access to trusted users. Dynamizer already provides different possibilities for authentication to the APIs. Basic Authentication<sup>80</sup> is a simple authentication scheme built into the HTTP protocol. The client sends HTTP requests with the Authorisation header that contains the word 'Basic' followed by space and a base64-encoded string "username:password". OAuth 2.0<sup>81</sup> is an open standard that allows enabling access delegation from the resource owner (i.e. user) for a trusted application to access the protected user resources without disclosing the master credentials. It leverages access tokens for the actual access delegation aspect. OAuth 2.0 is considered to be state-of-the-art for web and mobile applications and is supported by numerous big players of Web 2.0 (e.g. Twitter, Google, and Facebook). Many sensors and IoT APIs have already adopted OAuth 2.0 for authentication. In some cases, APIs also require an "API Key"<sup>82</sup> for accessing its resources. An API Key is a unique identifier or a secret token given to a user to authenticate and generally defines a set of access rights on the API associated with it. The key can usually be appended to the API request or can also be defined as the request header. In some cases, Access Tokens<sup>83</sup> are also used, allowing an application to access an API. The application receives an Access Token after a user successfully authenticates and authorises access, then passes the Access Token as a credential when it calls the target API. The passed token informs the API that the bearer of the token has been authorised to access the API and perform specific actions specified by the scope that was granted during authorisation.
10. *mqttServer* defines the URL of the server where the MQTT Broker runs. An MQTT broker receives messages from the clients and then routes the messages to the appropriate destination client.

---

<sup>80</sup><https://tools.ietf.org/html/rfc7617>

<sup>81</sup><https://oauth.net/2/>

<sup>82</sup><https://swagger.io/docs/specification/authentication/api-keys/>

<sup>83</sup><https://auth0.com/docs/tokens/access-tokens>

11. *mqttTopic* is the name of the topic. In MQTT, the word topic refers to an UTF-8 string that the broker uses to filter messages for each connected client. The topic consists of one or more topic levels (more details are given in example).
12. *linkToObservation* represents the complete URL of the operation requesting for the observations based on the specified parameters.
13. *linkToSensorDescription* represents the complete URL of the operation requesting for the description and metadata of the sensor or IoT device.

Various illustrations of the class *SensorConnection* for linking to numerous sensors and IoT platforms are given in section 5.3.1.

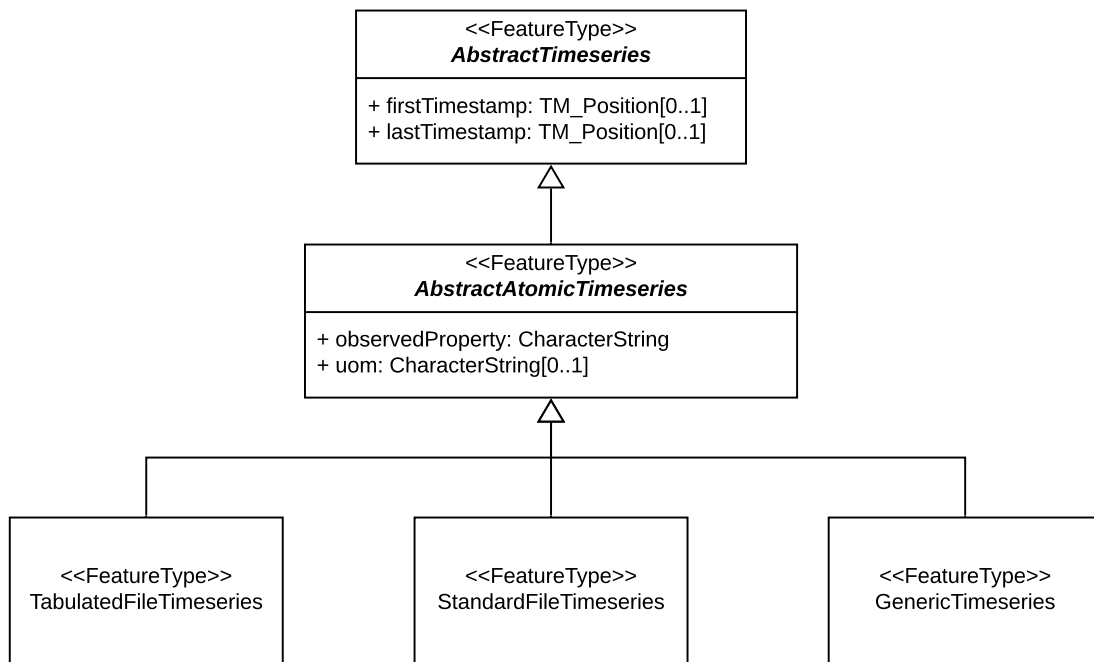
### 5.2.3 Atomic Timeseries

With Dynamizers, it is possible to enrich any CityGML feature by time-series data. The time-series can be (i) the result of some simulation (e.g. the computed solar irradiation values on a building façades over a year, or the traffic density at a road section over a day) and (ii) the input to some simulation (e.g. the standard load profile for electrical energy consumption). In many scenarios, as outlined in [R4], such time-series data is required to be represented in-line with city objects allowing the exchange of city object along with accurate description and metadata of the time-series. As shown in figure 5.4, Atomic Timeseries of Dynamizer is defined as *AbstractAtomicTimeseries* and allows 3 different ways to represent time-series: (i) *StandardFileTimeseries*, (ii) *TabulatedFileTimeseries*, and (iii) *GenericTimeseries*. Each time-series contains common metadata attributes: *observedProperty* indicating the name of the property such as *monthly\_heat\_demand* and *uom* indicating the unit of measurement such as *kWh*. The additional attributes are *firstTimestamp* and *lastTimestamp* representing the temporal range of the individual time-series. However, these two attributes are part of the class *AbstractTimeseries* because these attributes are also useful for *CompositeTimeseries* (c.f. section 5.2.4). By keeping these attributes on a higher generalisation hierarchy, they will be inherited by both Atomic and Composite Timeseries.

#### 5.2.3.1 Standard File Timeseries

The *StandardFileTimeseries* allows representing time-series data utilising the open and international standards such as OGC TimeseriesML 1.0 (Tomkins and Lowe 2016) and OGC Observations & Measurements (O&M) (Cox 2013). As already described in section 2.2, TimeseriesML 1.0 is an OGC standard providing a domain-neutral model for the representation and exchange of time-series data. The TimeseriesML schema supports two types of encodings. First is the interleaved time-value pair encoding, whereby the time and value are coupled together, and the coupling explicitly represents the mapping. The second is the domain-range encoding, where the domain and range are encoded separately, with a mapping function that allows looking up the range value for a given domain value. The advantage with this international standard is that it allows defining interpolation and aggregation types for each point in the time-series. It also helps in mapping missing values or multiple values to specific time points.

Apart from TimeseriesML, Dynamizers also allow referencing a file encoded with the O&M data. O&M is a generic information model for describing observations. It is one of the core standards in the OGC Sensor Web Enablement suite, providing the response model for Sensor Observation Service (SOS) and SensorThings API. Like TimeseriesML, the O&M standard also allows representing rich metadata of time-series.



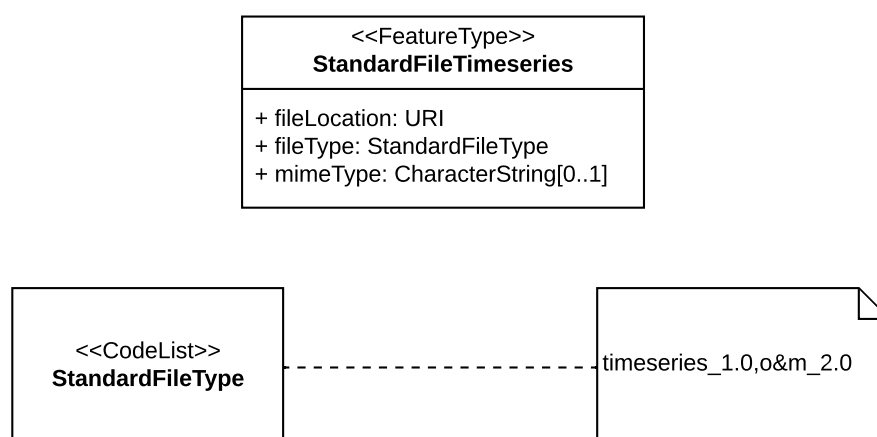
**Figure 5.4:** Representation of the feature type AtomicTimeseries, supporting different representations of time-series in the forms of (i) TabulatedFileTimeseries, (ii) StandardFileTimeseries, and (iii) GenericTimeseries.

The Atomic Timeseries can be represented by embedding the well-defined OGC standards TimeseriesML and O&M. Both the standards provide a comprehensive description and rich metadata for time-series and individual time points. However, their implementation would require to implement the entire concepts of these two OGC standards in the context of management systems as well as visualization applications, which is not practical. Hence, using the class *StandardFileTimeseries*, Dynamizers allow linking to an external file that is compliant to one of these standards.

As shown in figure 5.5, the *StandardFileTimeseries* class includes the following key attributes:

1. *fileLocation* allows defining the location where the file compliant to OGC standards is located.
2. *fileType* represents which type of standard is the file compliant to. The different possibilities of a standard file type are defined within the CodeList *StandardFileType* such as OGC TimeseriesML 1.0 and OGC O&M 2.0.
3. *mimeType* indicates the nature and format of the document or file. MIME Type<sup>84</sup> (known as Multipurpose Internet Mail Extensions) allows defining the type of data in a standardised way so that the applications and software systems know how to handle the data. In general, the MIME-type consists of a *type* and a *subtype* divided by a slash (/) character, for example, the MIME-type *application/pdf* indicates that the file type is pdf and the software can launch an application such as Adobe Reader to open the pdf file. Similarly, the MIME-type *text/html* indicates that this is an HTML document, and the application can render it internally without using any external software.

<sup>84</sup><https://tools.ietf.org/html/rfc6838>



**Figure 5.5:** Representation of the StandardFileTimeseries class. It allows linking to an external file that is compliant to OGC TimeseriesML 1.0 or OGC Observations & Measurements standards.

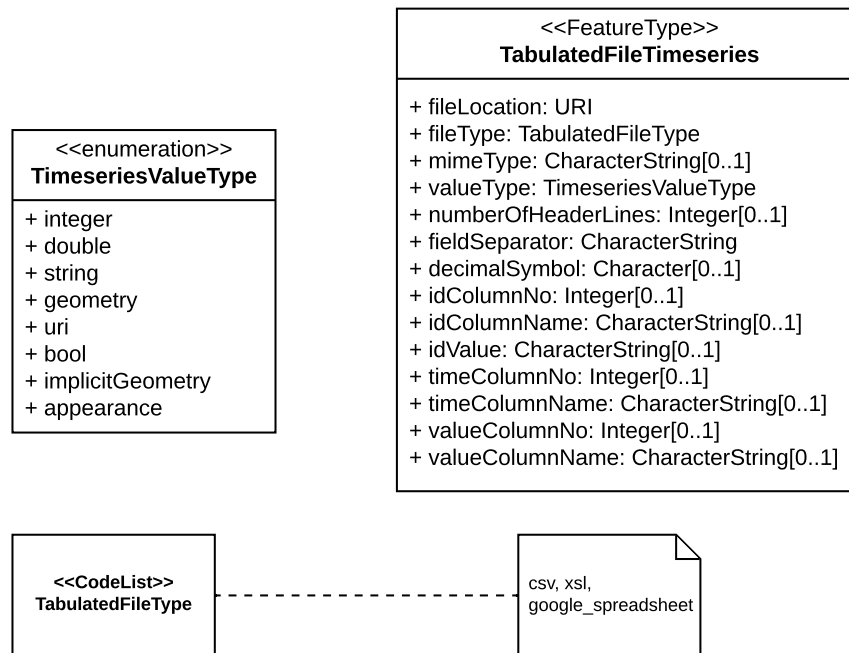
Since TimeseriesML and O&M provide XML encoding for GML Application Schema, their MIME-type is *application/gml+xml*.

### 5.2.3.2 Tabulated File Timeseries

Dynamizer also supports representing time-series data stored in external tabulated data files such as CSV, Microsoft Excel, Google spreadsheets etc. Such files are widely used for storing different kinds of data in a structured way. The *TabulatedFileTimeseries* class allows linking to files that contain time-series data in tabular form (Figure 5.6).

The *TabulatedFileTimeseries* class includes the following key attributes:

1. *fileLocation* defines the location where the file is located.
2. *fileType* indicates the type of the file being used. The different possibilities of the file types are defined using the CodeList *TabulatedFileType* such as CSV, Microsoft Excel, and Google Spreadsheet.
3. *mimeType* indicates the nature and format of the file such as *application/csv*, *application/xls*, and so on.
4. *valueType* represents the data type of the value in the defined time-series. For example, a time-series generated by a weather station for temperature recordings is most likely of datatype double, a time-series from a traffic camera for counting the number of cars at a junction is an integer, and another time-series retrieved from a moving GPS a point object. The different data types are listed in the enumeration *TimeseriesValueType*.
5. *numberOfHeaderLines* allows defining if there are any header lines present in the file. In general, the tabulated files comprise a header line defining the names of each of the columns. However, in a few cases, when a tabulated file is generated from software or an API, the exported file contains only the values and not the header line. Hence, users can specify the presence of header lines by defining the appropriate value in the *numberOfHeaderLines* attribute.



**Figure 5.6:** Representation of the `TabulatedFileTimeseries` class. It allows linking to an external file that contain timeseries data in a tabular form.

6. *fieldSeparator* defines which character is used for field separation. In tabulated files (especially CSV files), field separators are used to separate individual fields and columns. Most often, the comma character (',') is used for this purpose. However, there might also be other characters, such as semi-colon(';'). The type of the field separator character can be defined using the *fieldSeparator* attribute, which may help applications and software systems in interpreting the tabulated file structure correctly.
7. *decimalSymbol* specifies the symbol used to separate the integer part from the fractional part of a number written in decimal form. The most commonly used decimal symbols are dot ('.') and comma (',') characters.
8. *idColumnNo* indicates the column number specifying the identifier (id) of the city object. One single tabulated file may store time-series values for multiple city objects, for example, solar potential simulation's monthly results for 100 buildings in a district. The *idColumnNo* attribute defines which column holds the identifier (e.g. `gml_id`) of the city object.
9. *idColumnName* indicates the column name specifying the identifier (id) of the city object. It allows users the choice to refer to a specific column in a tabulated data file by the name of the column.
10. *idValue* specifies that only those rows from the file are being selected where the value in the *idColumn* matches the specified *idValue*. It allows, for example, to have a large CSV file that has, e.g. the solar irradiation values for all thematic surfaces of a city model for each month in a year. In this case, a Dynamizer can be created for each *RoofSurface* or *WallSurface* feature, which all



refer to the same CSV file, but using a different *idValue* to select the proper rows from the file for the respective *RoofSurface* and *WallSurface*.

11. *timeColumnNo* indicates the column number in the file which specifies timestamps.
12. *timeColumnName* indicates the column name (instead of column number) in the file specifying timestamps.
13. *valueColumnNo* indicates the column number which specifies values corresponding to individual timestamps.
14. *valueColumnName* indicates the column name (instead of column number) in the file specifying values corresponding to individual timestamps.

The representation of attributes within the *TabulatedFileTimeseries* class requires a condition that details about the time column, value column, and id column are specified. These details may be provided either by using the column number or column name. However, one of either field must be provided. This condition can formally be expressed by using the Object Constraint Language (OCL<sup>85</sup>) in the data model. OCL is a declarative language for describing rules that apply to UML models. The constraint can be defined as shown in the listing 5.1.

**Listing 5.1:** OCL expression for defining that either column number or column name must be provided for time and value columns

```
context TabulatedFileTimeseries inv:
  (timeColumnNo->notEmpty() or timeColumnName->notEmpty()) and
  (valueColumnNo->notEmpty() or valueColumnName->notEmpty()) and
  (idValue->notEmpty() implies
    idColumnNo->notEmpty() or idColumnName->notEmpty()
  )
```

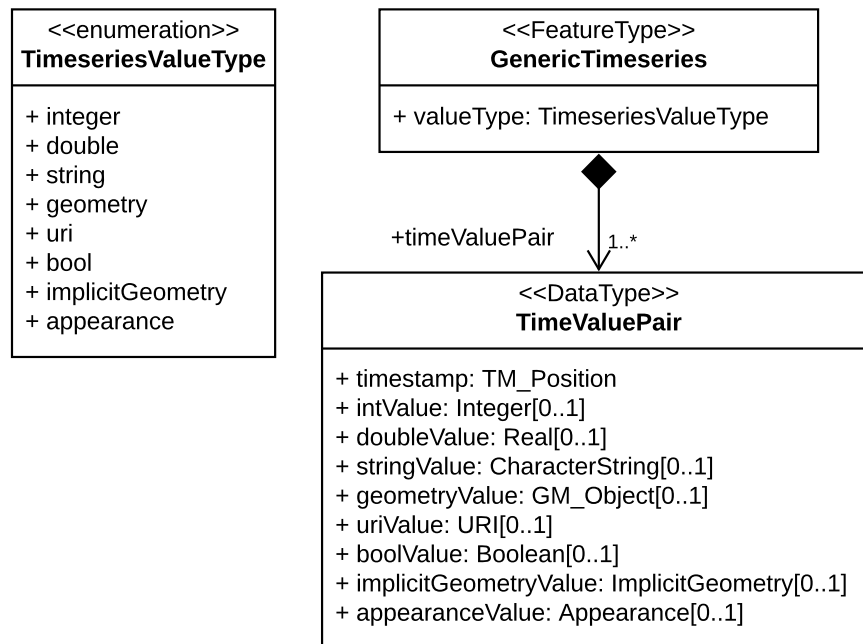
### 5.2.3.3 Generic Timeseries

Dynamizer also allows representing time-series data using a basic structure within the *GenericTimeseries* class. The advantage with *GenericTimeseries* is that a simple time-series data can be encapsulated within the city model dataset, making the dataset complete and self-sufficient. Additionally, it avoids the need for database management systems to store time-series data compliant to the before-mentioned external standards. However, unlike such external time-series standards, *GenericTimeseries* is not capable of mapping missing values or multiple values in time-series using interpolation and aggregation functions. Hence, the intention to use *GenericTimeseries* is to map simple time-series data.

As shown in figure 5.7, each *GenericTimeseries* object allows defining individual time points using the association *timeValuePair*. The type *TimeValuePair* defines timestamp and the corresponding values. Based on the type of the timeseries values defined in the «CodeList» *TimeseriesValueType*, values of the following data types can be defined within the Generic Timeseries:

1. *intValue* defines the values of the type *integer*. Various scenarios require a time-series dealing with an integer value, for example, a time-series from a traffic camera for counting the number of cars at a junction, a time-series representing the number of visitors during a football match in a stadium, and so on.

<sup>85</sup><https://www.omg.org/spec/OCL/>



**Figure 5.7:** Representation of the GenericTimeseries class. It allows representing basic time-series and its metadata data in-line with city objects.

2. *doubleValue* defines the values of the type *double*. For example, a time-series generated by a weather station for temperature recordings is likely of type double. Similarly, a time-series generated by a Smart Meter for gas consumption is of type double.
3. *stringValue* defines the values of the type *string*. It enables representing time-series with qualitative or categorical observations. For example, time-series representing the air quality in a district over a day by using the categories: (i) Very Good, (ii) Good, (iii) Bad, and (iv) Very Bad.
4. *geometryValue* defines the values of the type *geometry*. It allows representing time-series involving any geometric object such as point, line, polygon, and so on. For example, a time-series retrieved from the GPS of a moving car involve varying points (coordinates) of the car object.
5. *uriValue* defines the values of the type *uri*. Such time-series enable representing variations based on the results available on a web repository which can be accessed by a web link (URL). For example, an animation of varying pollutants within a city district over a day may involve a time-series having timestamps and corresponding links to the raster images visualising the pollutant levels.
6. *boolValue* defines the values of the type *boolean*. For example, a time-series for a whole year indicating whether a football match on a specific day in a stadium is scheduled ("true") or not scheduled ("false").
7. *implicitGeometryValue* represents the values of the type *ImplicitGeometry* of CityGML. In some scenarios, it is essential to represent objects changing their locations as well as orientation with time, for example, a moving car turning at junctions. The *ImplicitGeometry* object allows representing the shape of a 3D object as well as instancing at an anchor point plus further transformations like

rotation and scaling using a transformation matrix. The *GenericTimeseries* class allows overriding *ImplicitGeometry* objects over different time intervals.

8. *appearanceValue* defines the values of the type *Appearance* of CityGML. CityGML appearances can be overridden for different time intervals by providing (i) URIs to different texture files within an Appearance object, or (ii) completely different Appearance objects.

The *GenericTimeseries* class requires a condition that only one type of time-series can be represented within a single feature. This condition can formally be expressed by OCL as shown in the listing 5.2.

**Listing 5.2:** OCL expression for defining that only one type of time-series value can be represented within a single *GenericTimeseries* and only one type of encoding is represented by *TimeValuePair*

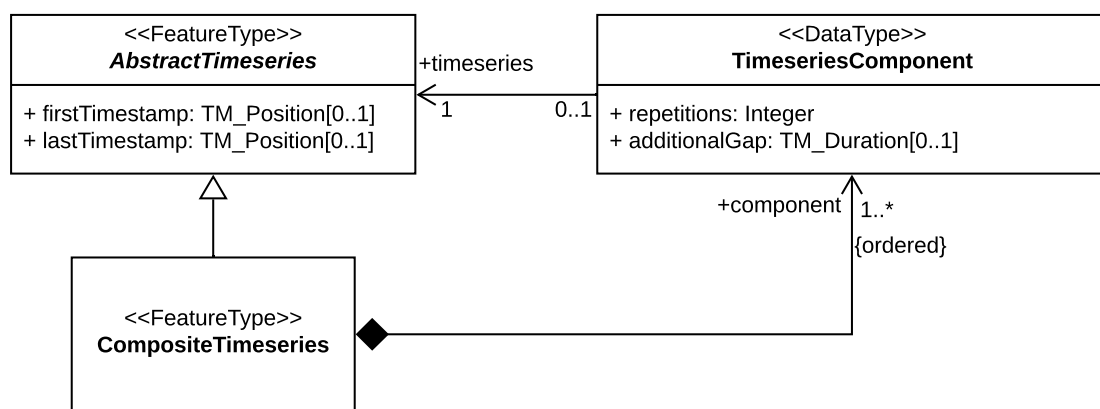
```
context GenericTimeseries inv:
  if valueType = TimeseriesValueType::integer
    then timeValuePair->forall(c|c.intValue->size()=1)
  else if valueType=TimeseriesValueType::double
    then timeValuePair->forall(c|c.doubleValue->size()=1)
  else if valueType=TimeseriesValueType::string
    then timeValuePair->forall(c|c.stringValue->size()=1)
  else if valueType=TimeseriesValueType::geometry
    then timeValuePair->forall(c|c.geometryValue->size()=1)
  else if valueType=TimeseriesValueType::uri
    then timeValuePair->forall(c|c.uriValue->size()=1)
  else if valueType=TimeseriesValueType::bool
    then timeValuePair->forall(c|c.boolValue->size()=1)
  else if valueType=TimeseriesValueType::implicitGeometry
    then timeValuePair->forall(c|c.implicitGeometryValue->size()=1)
  else if valueType=TimeseriesValueType::appearance
    then timeValuePair->forall(c|c.appearanceValue->size()=1)
  endif

context TimeValuePair inv:
  self.intValue->size()+
  self.doubleValue->size()+
  self.stringValue->size()+
  self.geometryValue->size()+
  self.uriValue->size()+
  self.boolValue->size()+
  self.implicitGeometryValue->size()+
  self.appearanceValue->size()=1
```

#### 5.2.4 Composite Timeseries

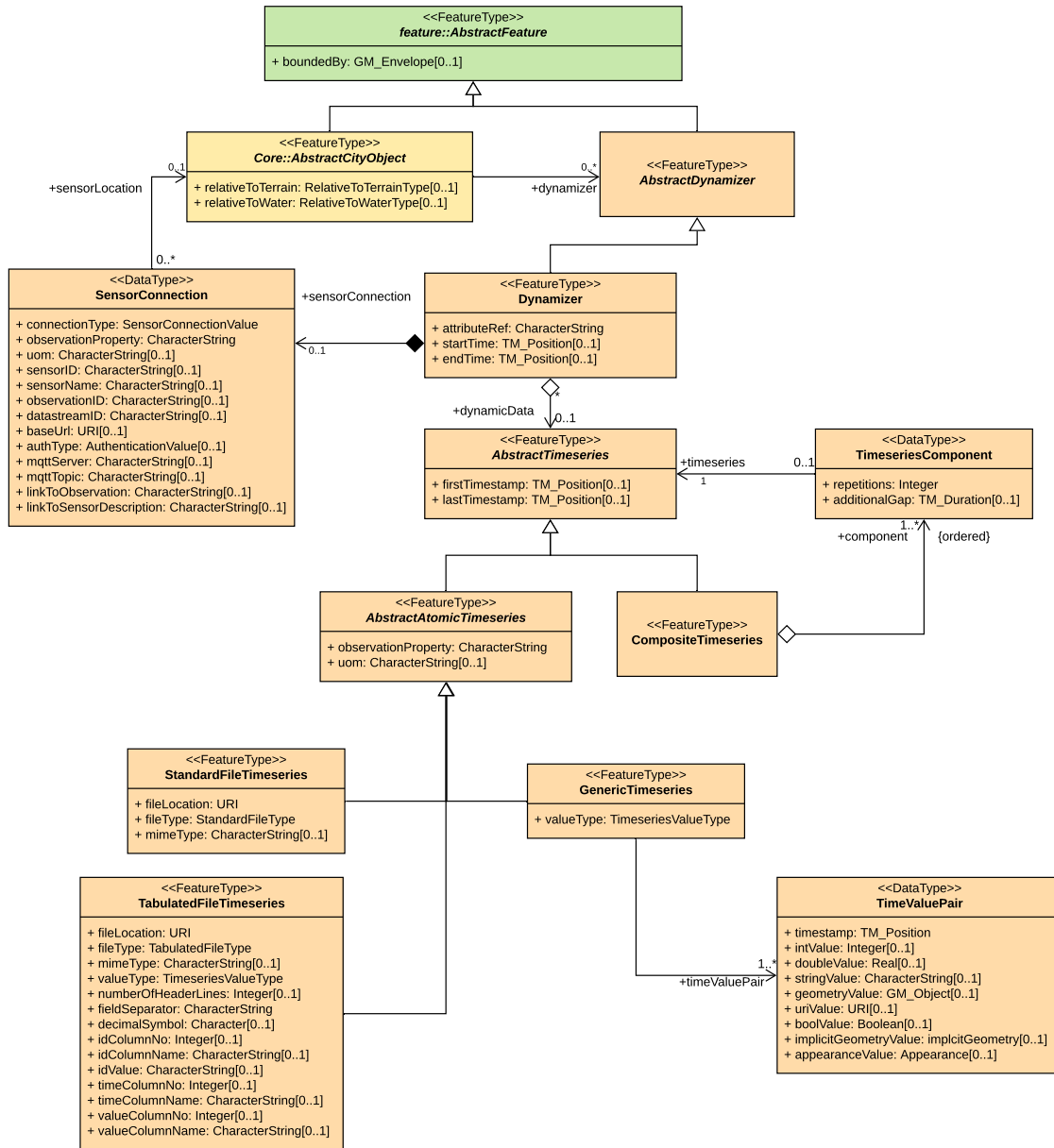
Dynamizers support absolute start and end time points referencing a specific time reference system. The absolute time points can be mapped to the attribute values and can be represented as a tabulation

of the measured data. One typical example illustrating such a scenario is mapping of the energy consumption values of a building for every hour in a day. However, in many applications, it is not sufficient to provide a means for the tabulation of time-value pairs. The applications may require patterns to represent dynamic variations of properties based on statistics and general rules. In such scenarios, time cannot be described by absolute positions, but relative to the absolute positions. In these cases, time may be defined for a non-specific year (e.g., averages over many years), but still classified by the relative time of a year. For example, January monthly summaries for the energy consumption of a building might be described as "all-Januaries 2001-2010". Similarly, the energy consumption values may reflect generic patterns for individual weekdays/weekends in a week or a month. Another example scenario may also be determining patterns for specific seasons (such as spring, summer, autumn, and winter) over ten years.



**Figure 5.8:** Representation of Composite Timeseries, allowing representing patterns of dynamic variations of properties.

To support such patterns, Dynamizer includes the concept of Composite Timeseries, as shown in figure 5.8. The *CompositeTimeseries* class is modelled in such a way that it composes of an ordered list of *AbstractTimeseries*. *CompositeTimeseries* includes a component called *component*, which denotes the number of repetitions for a time-series component. *repetitions* is an integer type which determines how many times the nested time-series requires to be iterated. For example, to determine the pattern of a building's electricity consumption for weekdays, a *CompositeTimeseries* may include five repetitions of *AtomicTimeseries* of a single weekday consumption. It also contains an attribute *additionalGap*, which is of type *TM\_Duration*. It allows defining customised patterns by providing the gaps within the existing time-series. For instance, for an entire monthly time-series of energy consumption for all days of a week, the gaps can be provided for the weekends to define the patterns of energy consumption only for the weekdays. Furthermore, this attribute also allows connecting non-overlapping time-series that have been separately collected to make a single time-series. For example, if the latest two months of time-series data is transferred from one system to a major archive, the series must be connected to make a full series over which the patterns can be determined (e.g., to determine yearly patterns). However, for a *CompositeTimeseries*, it is necessary to model the time positions according to a relative time reference system. According to ISO 19108, they may be defined as *TM\_OrdinalEras* within the *TM\_OrdinalReferenceSystems*. The use of *CompositeTimeseries* allows



**Figure 5.9:** Complete UML model of the Dynamizer concept. Newly introduced classes are shown in orange. Classes shown in green and yellow are from GML and CityGML respectively. The OCL expressions and Code Lists (mentioned in the respective sub-sections) are not shown in this figure due to the limited page width.

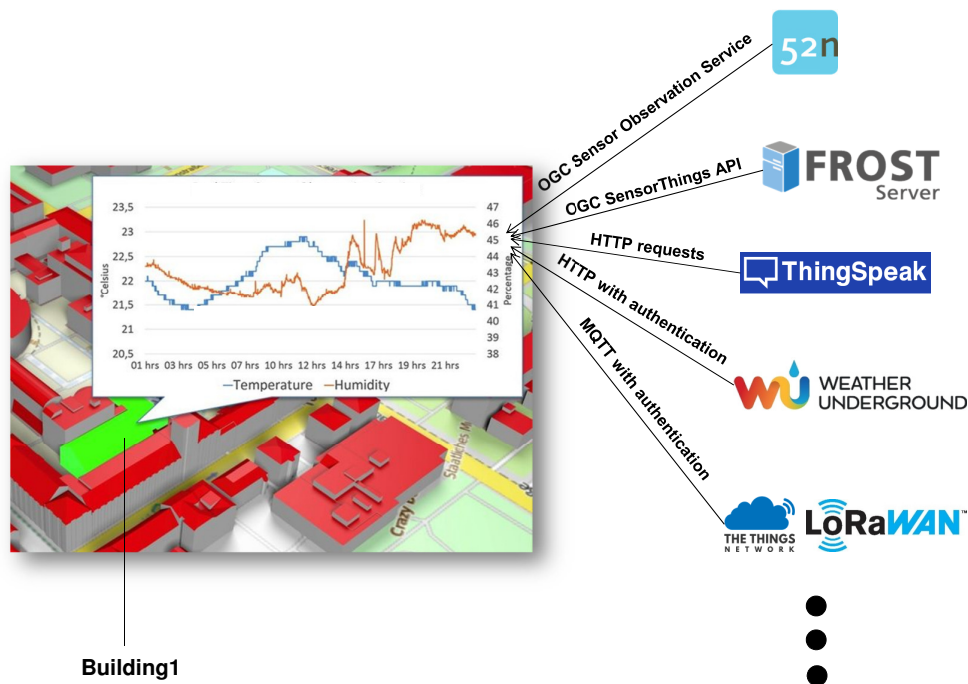
defining local reference systems for the specific use cases. An illustration of the *CompositeTimeseries* concept is shown in section 5.3.3.

### 5.2.5 Complete UML Model of Dynamizer

Based on the developments of individual features, the complete UML class diagram of the Dynamizer concept is shown in figure 5.9. Classes shown in green and yellow are from GML and CityGML respectively. The OCL expressions and Code Lists (mentioned in the respective sub-sections) are not shown in this figure due to the limited page width. The Dynamizer concept has been developed for the next version of the CityGML standard (version 3.0) and has been proposed to the OGC CityGML Standard Working Group for its adoption. However, the proposed concept can also be implemented as the CityGML Application Domain Extension (ADE) by using the "hook" mechanism. It would allow Dynamizers to be used with the current version of the CityGML standard (version 2.0).

## 5.3 Illustration of the Concept

This section describes the illustration of different functionalities of the Dynamizer concept based on the CityGML version 3.0. The following sub-sections show different scenarios for using CityGML Dynamizers for (i) linking city object properties with their respective sensor data, (ii) representing timeseries and its metadata in-line with a city object, and (iii) representing repetitive patterns within CityGML objects.



**Figure 5.10:** Dynamizer *SensorConnection* linking to different sensor platforms.

### 5.3.1 Integrating city object properties with real-time sensors

Dynamizer allows integrating city object properties with real-time sensor data streams available at numerous platforms (figure 5.10). At the same time, it allows overriding the property values based on the dynamic sensor values. For this purpose, Dynamizer *SensorConnection* type is used.

#### 5.3.1.1 Example 1: ThingSpeak platform

This example illustrates a scenario in which a room within a building has an indoor sensor DHT22<sup>86</sup> installed. This sensor measures air temperature, and humidity for the room and the observations are available on the ThingSpeak platform (c.f. section 2.2.1). As shown in figure 5.10, this building is represented as a CityGML object having a unique building id "*building1*" and the specific room, where the sensor is located, is defined using a unique id "*room1*". The object may have several static attributes such as building function, roof type, address, and so on. However, this room has a generic attribute named *temperature*, which is dynamic and changes with time (see listing 5.3). The value of the *temperature* attribute is retrieved from the ThingSpeak platform. CityGML Dynamizer can be used to override the value of the *temperature* attribute based on the direct queries from the ThingSpeak platform shown in listing 5.4.

**Listing 5.3:** Illustration of a CityGML Building object having a generic attribute 'temperature'

```
<core:cityObjectMember>
  <bldg:Building gml:id="building1">
    <bldg:buildingRoom>
      <bldg:BuildingRoom gml:id="room1">
        <core:genericAttribute>
          <gen:DoubleAttribute>
            <gen:name>temperature</gen:name>
            <gen:value>20.2</gen:value>
          </gen:DoubleAttribute>
        </core:genericAttribute>
      </bldg:BuildingRoom>
    </bldg:buildingRoom>
  </bldg:Building>
</core:cityObjectMember>
```

An example query to request data from the ThingSpeak platform looks like as follows (the request is a single line without any linebreak and spaces):

```
https://thingspeak.com/channels/64242/fields/1.json?start=
2019-10-08T00:00:00Z&end=2019-10-09T00:00:00Z
```

The request above retrieves all the observations between '2019-10-08T00:00:00Z' and '2019-10-09T00:00:00Z' for a specific field ID ('1') from a registered ThingSpeak channel. ThingSpeak

<sup>86</sup><https://www.adafruit.com/product/385>

manages each registered sensor using a unique channel ID (e.g. 64242 in this example). The query to each channel ID responds with the details of the sensor device as well as available datastreams. In this case, the datastreams available are temperature and humidity registered with field ID 1 and 2, respectively. Hence, to link the building property to the temperature observation of the specific ThingSpeak datastream, the Dynamizer is defined as follows:

**Listing 5.4:** Dynamizer defined within the CityGML 3.0 document having direct links to sensor observations available at ThingSpeak platform

```
<bldg:Building gml:id="building1">
  <core:dynamizer>
    <dyn:Dynamizer gml:id="room1_Dynamizer">
      <dyn:attributeRef>
        <!--Single line without any linebreak and space-->
        //bldg:BuildingRoom[@gml:id='room1']/core:genericAttribute
        /gen:DoubleAttribute[gen:name='temperature']/gen:value
      </dyn:attributeRef>
      <dyn:startTime>2019-10-08T00:00:00Z</dyn:startTime>
      <dyn:endTime>2019-10-09T00:00:00Z</dyn:endTime>
      <dyn:sensorConnection>
        <dyn:SensorConnection>
          <dyn:connectionType>thingspeak_1.0</dyn:connectionType>
          <dyn:observationProperty>Temperature</dyn:observationProperty>
          <dyn:uom>Celsius</dyn:uom>
          <dyn:sensorID>64242</dyn:sensorID>
          <dyn:sensorName>DHT22</dyn:sensorName>
          <dyn:datastreamID>1</dyn:datastreamID>
          <dyn:baseURL>https://thingspeak.com</dyn:baseURL>
          <dyn:authType>none</dyn:authType>
          <dyn:linkToObservation>
            <!--Single line without any linebreak and space-->
            %baseURL%/channels/%sensorID%/fields/%datastreamID%.json
            ?start=%startTime%&end=%endTime%
          </dyn:linkToObservation>
          <dyn:sensorLocation>
            xlink:href="#room1"></dyn:sensorLocation>
          </dyn:SensorConnection>
        </dyn:sensorConnection>
      </dyn:Dynamizer>
    </core:dynamizer>
    <!-- The element <bldg:BuildingRoom>...</bldg:BuildingRoom>
    from Listing 5.3 has to be embedded here -->
  </bldg:Building>
```

As shown in listing 5.4, various parameters to link with the ThingSpeak platform can be defined using the Dynamizer *SensorConnection*. The listing shows that the Dynamizer "room1\_Dynamizer"



can be defined for the specific room for which the the real-time observations are retrieved from the ThingSpeak platform. The *attributeRef* shows an XPath expression, which refers to the specific generic attribute *temperature* in the BuildingRoom with the gml:id *room1*. The generic attribute *temperature* is dynamic in nature and requires to be overridden by time-dependent values retrieved from the specific ThingSpeak channel. The attributes *startTime* and *endTime* represent the entire temporal extent for the dynamic values. The attributes defined within *SensorConnection* allow connecting to an arbitrary ThingSpeak datastream. The *connectionType* indicates the type of the connection, which is *thingspeak\_1.0* in this listing. The values of different types of connections are encoded in the CodeList *SensorConnectionValue* as shown in Figure 5.3. As mentioned above, the *observationProperty* of our interest in this listing is *Temperature*, its associated *datastreamID* is *1*, and the *uom* is *Celsius*. The *baseURL* for all the channels available on the ThingSpeak platform is *https://thingspeak.com*. Since the mentioned *sensorID* is publicly available and requires no authentication, the attribute *authType* is mentioned as *none*. The different types of authentication values are mentioned in the CodeList *AuthenticationValue* as shown in Figure 5.3. Furthermore, the attribute *linkToObservation* allows the possibility to encode the URL string for establishing direct links to the observations of the specific datastream. The URL strings are generated using the combination of the attributes of the Dynamizer *SensorConnection* class and any other static strings. For example, in this listing, the dynamic variables are represented as *%baseURL%*, *%sensorID%*, *%datastreamID%*, *%startTime%*, and *%endTime%*. The values of these variables defined in the *SensorConnection* class are used in combination with the static strings to generate the complete URL string for the specific datastream. For example, the attribute substitution in the given listing corresponds to the URL string "https://thingspeak.com/channels/64242/fields/1?start=2019-10-08T00:00:00Z&end=2019-10-09T00:00:00Z". The attribute *SensorLocation* refers to the specific BuildingRoom id *room1*, indicating that the sensor from the ThingSpeak channel is located within this specific room.

### 5.3.1.2 Example 2: OGC SensorThings API

This example illustrates the same building room as shown in listing 5.3 with the same indoor sensor DHT22 installed in it. This sensor measures air temperature, and humidity for the room. However, the sensor observations are stored in a FROST Server and, hence, can be queried using the OGC SensorThings API. More details on OGC SensorThings API are given in section 2.2.1. An example request using the OGC SensorThings API looks like as follows (the request is a single line without any linebreak and spaces):

```
http://127.0.0.1:8080/FROST-Server/v1.0/Datastreams(1)/Observations
?$filter=during(phenomenonTime,2019-04-12T10:00:00.000Z/
2019-04-12T10:30:00.000Z)
```

The above-mentioned request retrieves all the observations between '2019-04-12T10:00:00.000Z' and '2019-04-12T10:30:00.000Z' for a specific Datastream ID ('1') registered at the FROST Server installed on the machine '127.0.0.1:8080'. The Datastream ID ('1') represents the temperature property in our case. According to the SensorThings API, Observations are always aligned with the Datastreams. Based on a Datastream ID, all the other properties can be queried, e.g.

```
http://127.0.0.1:8080/FROST-Server/v1.0/Datastreams(1)/Observations
```

The above-mentioned request allows us retrieving all the Observations associated to the specific Datastream ID ('1'). Similarly, the below-mentioned request allows us retrieving the details of the Sensor associated to the Datastream ID ('1'). Many more possibilities of such queries over SensorThings API have been presented in detail by (Liang et al. 2015).

```
http://127.0.0.1:8080/FROST-Server/v1.0/Datastreams(1)/Sensor
```

Hence, in order to link the building property to the temperature observation available over the specific FROST Server, the Dynamizer is defined as shown in listing 5.5.

**Listing 5.5:** CityGML Dynamizer having direct links to sensor observations available at the FROST Server

```
<bldg:Building gml:id="building1">
  <core:dynamizer>
    <dyn:Dynamizer gml:id="room1_Dynamizer">
      <dyn:attributeRef>
        <!--Single line without any linebreak and space-->
        //bldg:BuildingRoom[@gml:id=' room1' ]/core:genericAttribute
        /gen:DoubleAttribute[gen:name=' temperature' ]/gen:value
      </dyn:attributeRef>
      <dyn:startTime>2019-04-12T10:00:00.000Z</dyn:startTime>
      <dyn:endTime>2019-04-12T10:30:00.000Z</dyn:endTime>
      <dyn:sensorConnection>
        <dyn:SensorConnection>
          <dyn:connectionType>ogc_sta_1.0</dyn:connectionType>
          <dyn:observationProperty>Temperature</dyn:observationProperty>
          <dyn:uom>Celsius</dyn:uom>
          <dyn:datastreamID>1</dyn:datastreamID>
          <dyn:baseURL>
            http://127.0.0.1:8080/FROST-Server/v1.0
          </dyn:baseURL>
          <dyn:authType>none</dyn:authType>
          <dyn:linkToObservation>
            <!--Single line without any linebreak and space-->
            %baseURL%/Datastreams (%datastreamID%) /Observations?
            $filter=during(phenomenonTime,%startTime%/%endTime%)
          </dyn:linkToObservation>
          <dyn:linkToSensorDescription>
            %baseURL%/Datastreams (%datastreamID%) /Sensor
          </dyn:linkToSensorDescription>
          <dyn:sensorLocation xlink:href="#room1"></dyn:sensorLocation>
        </dyn:SensorConnection>
      </dyn:sensorConnection>
    </dyn:Dynamizer>
  </core:dynamizer>
</bldg:Building>
```

```
<!-- The element <bldg:BuildingRoom>...</bldg:BuildingRoom>
from Listing 5.3 has to be embedded here -->
</bldg:Building>
```

The above-mentioned listing shows how a Dynamizer *SensorConnection* class can be used to define parameters to connect to a specific datastream over the FROST Server. The *attributeRef* refers to the generic attribute *temperature* in the *BuildingRoom* with *gml:id room1* as shown in Listing 5.3. The time-depending values for the *temperature* property are retrieved from the Datastream ID ('1') of the before-mentioned FROST Server. Within the class *SensorConnection*, the required parameters can be defined. The *connectionType* in this case is *ogc\_sta\_1.0* representing the OGC SensorThings API version 1.0. This value is encoded in the CodeList *SensorConnectionValue* as shown in Figure 5.3. Similar to Listing 5.4, the *observationProperty* is *Temperature*, its associated *datastreamID* is *1*, and the *uom* is *Celsius*. The *baseURL* for the running FROST Server in this listing is *http://127.0.0.1:8080/FROST-Server/v1.0*. It indicates that the FROST Server version 1.0 is running on a machine 127.0.0.1 at port 8080. This base URL may differ for FROST Servers running on different machines or Cloud environment. Unlike a ThingSpeak channel, it is not mandatory to mention a Sensor ID to retrieve observations of a specific Datastream. The Datastream ID is sufficient for this purpose. Hence, in this listing, the Sensor ID is not mentioned. Since there is no authentication required for accessing the service in this listing, the attribute *authType* is mentioned as *none*. The *linkToObservation* attribute allows specifying the URL string with the combination of dynamic variables and static strings in order to retrieve observations from the specified Datastream. For example, the attribute substitution within the *linkToObservation* attribute in this listing corresponds to the URL string "http://127.0.0.1:8080/FROST-Server/v1.0/Datastreams(1)/Observations?\$filter=during(phenomenonTime,2019-04-12T10:00:00.000Z/2019-04-12T10:30:00.000Z)". Similarly, the attribute substitution within the *linkToSensorDescription* attribute in this listing corresponds to the URL string "http://127.0.0.1:8080/FROST-Server/v1.0/Datastreams(1)/Sensor". It allows us to retrieve the sensor details corresponding to the specified Datastream. The attribute *sensorLocation* refers to the specific *BuildingRoom* id *room1*, indicating that the sensor from the FROST Server is located within this specific room.

### 5.3.1.3 Example 3: Weather Underground API with authentication

This example illustrates the building shown in listing 5.6. A personal weather station is installed on top of the building, measuring the weather-related properties (e.g. *outside\_temperature*, relative humidity, wind speed, etc.). The personal weather station is registered on the Weather Underground platform (c.f. section 2.2.1) with an identifier "MyPWS1".

**Listing 5.6:** Illustration of a CityGML *RoofSurface* object having a generic attribute named 'outside\_temperature'

```
<core:cityObjectMember>
<bldg:Building gml:id="building1">
  <con:RoofSurface gml:id="building1_roof">
    <core:genericAttribute>
      <gen:DoubleAttribute>
        <gen:name>outside_temperature</gen:name>
```

```

    <gen:value>23.3</gen:value>
  </gen:DoubleAttribute>
</core:genericAttribute>
</con:RoofSurface>
</bldg:Building>
</core:cityObjectMember>

```

An example request to the Weather Underground platform looks like as follows:

```
http://api.wunderground.com/api/*****/conditions/q/pws:MyPWS1.json
```

Weather Underground API requires a unique API Key for every user allowing to authenticate to a running sensor ID. The building object has a generic attribute *outside\_temperature* which is dynamic. To link the generic attribute to the outside\_temperature observation from the weather station registered on the Weather Underground, the Dynamizer is defined as shown in listing 5.7.

**Listing 5.7:** CityGML Dynamizer having direct links to sensor observations available at the Weather Underground platform

```

<bldg:Building gml:id="building1">
  <core:dynamizer>
    <dyn:Dynamizer gml:id="building1_roof_Dynamizer">
      <dyn:attributeRef>
        <!--Single line without any linebreak and space-->
        //con:RoofSurface[@gml:id='building1_roof']/core:genericAttribute
        /gen:DoubleAttribute[gen:name='outside_temperature']/gen:value
      </dyn:attributeRef>
      <dyn:startTime>2019-01-01T10:00:00.000Z</dyn:startTime>
      <dyn:endTime>2019-02-01T00:00:00.000Z</dyn:endTime>
      <dyn:sensorConnection>
        <dyn:SensorConnection>
          <dyn:connectionType>wunderground_1.0</dyn:connectionType>
          <dyn:observationProperty>Temperature</dyn:observationProperty>
          <dyn:uom>Celsius</dyn:uom>
          <dyn:sensorID>MyPWS1</dyn:sensorID>
          <dyn:baseUrl>http://api.wunderground.com/api</dyn:baseUrl>
          <dyn:authType>apiKey</dyn:authType>
          <dyn:linkToObservation>
            %baseUrl%/%authType%/conditions/q/pws:%sensorID%.json
          </dyn:linkToObservation>
          <dyn:sensorLocation>
            xlink:href="#building1_roof"></dyn:sensorLocation>
          </dyn:SensorConnection>
        </dyn:sensorConnection>
      </dyn:Dynamizer>
    </core:dynamizer>
  </bldg:Building>

```

```
</dyn:Dynamizer>
</core:dynamizer>
<!-- The element <con:RoofSurface>...</con:RoofSurface>
from Listing 5.6 has to be embedded here -->
</bldg:Building>
```

The above-mentioned listing shows a Dynamizer *SensorConnection* class for connecting to the Weather Underground API for retrieving the observations of the property *outside\_temperature* of the personal weather station with the identifier "MyPWS1". The attribute *attributeRef* represents an XPath expression for referring to the value of the generic attribute *outside\_temperature* of the RoofSurface with the gml:id *building1\_roof*. The *connectionType* in this case is *wunderground\_1.0* (also encoded in the CodeList *SensorConnectionValue*). The *observationProperty* is indicated as *Temperature* as it is also defined with the same name in the Weather Underground channel. The *uom* is *Celsius* and the *SensorID* is *MyPWS1*. The *baseURL* is *http://api.wunderground.com/api*. Since the API requires a unique API Key to connect to the server, the attribute *authType* is mentioned as *apiKey*. This is also indicated in the CodeList *AuthenticationValue*. The attribute *linkToObservation* allows users to specify the attribute values with the help of dynamic variables. For example, the variable *%authType%* allows users to assign their secret unique key dynamically without representing it within the CityGML Dynamizers. With the help of the combination of such dynamic variables and static strings, the complete URL string can be formed allowing users to provide the attribute values on request. The attribute *sensorLocation* refers to the specific RoofSurface id *building1\_roof*, indicating that the sensor from the Weather Underground API is located on this specific roof surface.

#### 5.3.1.4 Example 4: Subscribing to real-time datstreams using the MQTT protocol

This example illustrates the same building room as shown in listing 5.3 with the same DHT22 sensor installed in it; however, the observations are available on The Things Network platform (c.f. section 2.2.1). The Things Network supports the MQTT protocol and allows users to subscribe to its data streams. An example request to connect to the Things Network via MQTT looks like as follows:

```
Server: eu.thethings.network
Topic: +/devices/tumgis-dragino-shield-with-gps/up/temperature
Username: *****
Password:*****
```

Here, *Server* represents the URL of the machine where the server is deployed. MQTT requires *Topic* to connect to the individual data stream (in some cases, all the datastreams). The authentication details are provided as username and password. These details are required as separate attributes by an MQTT broker and hence, cannot be provided within the single URL. To link the generic attribute to the temperature observation using MQTT, the Dynamizer is defined as shown in listing 5.8.

**Listing 5.8:** CityGML Dynamizer subscribing to a sensor data stream using the MQTT protocol

```

<bldg:Building gml:id="building1">
  <core:dynamizer>
    <dyn:Dynamizer gml:id="room1_Dynamizer">
      <dyn:attributeRef>
        <!--Single line without any linebreak and space-->
        //bldg:BuildingRoom[@gml:id='room1']/core:genericAttribute
        /gen:DoubleAttribute[gen:name='temperature']/gen:value
      </dyn:attributeRef>
      <dyn:startTime>2019-01-01T00:00:00.000Z</dyn:startTime>
      <dyn:endTime>2020-01-01T00:00:00.000Z</dyn:endTime>
      <dyn:sensorConnection>
        <dyn:SensorConnection>
          <dyn:connectionType>mqtt_3.1.1</dyn:connectionType>
          <dyn:observationProperty>Temperature</dyn:observationProperty>
          <dyn:uom>Celsius</dyn:uom>
          <dyn:authType>basic</dyn:authType>
          <dyn:mqttServer>eu.thethings.network</dyn:mqttServer>
          <dyn:mqttTopic>
            +/devices/tumgis-dragino-shield-with-gps/up/temperature
          </dyn:mqttTopic>
          <dyn:sensorLocation xlink:href="#room1"></dyn:sensorLocation>
        </dyn:SensorConnection>
      </dyn:sensorConnection>
    </dyn:Dynamizer>
  </core:dynamizer>
  <!-- The element <bldg:BuildingRoom>...</bldg:BuildingRoom>
  from Listing 5.3 has to be embedded here -->
</bldg:Building>

```

The above-mentioned listing shows a Dynamizer *SensorConnection* class for connecting to an MQTT Server subscribing to the observations of the property *temperature* available over The Things Network. The attribute *attributeRef* represents an XPath expression for referring to the value of the generic attribute *temperature* of the *BuildingRoom* with the gml:id *room1*. The *connectionType* in this case is *mqtt\_3.1.1* (also encoded in the CodeList *SensorConnectionValue*). It shows the version of the MQTT protocol as 3.1.1. The *observationProperty* is indicated as *Temperature* and the *uom* is *Celsius*. The attribute *authType* is indicated as *basic*, which means that a basic authentication (with a username and a password) is required to establish this connection. For establishing a connection to an MQTT Server, the *SensorConnection* class provides two attributes *mqttServer* and *mqttTopic* for providing the server and topic details respectively. The server and topic names are mentioned before in this sub-section. The attribute *sensorLocation* refers to the specific *BuildingRoom* id *room1*, indicating that the time-dependent values from the MQTT Topic correspond to this specific room.

### 5.3.2 Representing timeseries values in-line within city objects

As discussed in section 2.1.2, most of the simulations involve time-dependent attributes, for example, monthly values of solar irradiations or energy demand estimations for a building. To perform detailed realistic simulations, for instance, cross-domain exchanging of simulation results with city objects for enhancing disaster management or energy assessment, it is essential to model the precise description of the time-series. The Dynamizer *AtomicTimeseries* allows time-series data to be exchanged with appropriate metadata. It enables correct machine interpretation and thus proper use for further analysis.

This section illustrates examples of using Dynamizer *AtomicTimeseries* for representing time-series in-line with city objects. For the illustration, the results of the solar potential simulation on building surfaces are shown. The Solar Potential Analysis Tool developed by the Chair of Geoinformatics, Technical University of Munich is widely used for assessing and estimating solar energy production for the roofs and façades of 3D building objects in different ways. The simulation tool operates on the semantic 3D city models defined according to the CityGML standard (Willenborg et al. 2017). By combining a transition model, sun position calculation, and an approximation of the skydome, the solar power from direct, diffuse, and global sunlight irradiation are estimated for individual months and years.

As shown in listing 5.9, a building wall surface (having an id "*building1\_wall1*") has a generic attribute named "*globalRadMonth*" representing the global irradiation value for different months. The attribute "*globalRadMonth*" is dynamic in nature and changes its value every month in the simulation results.

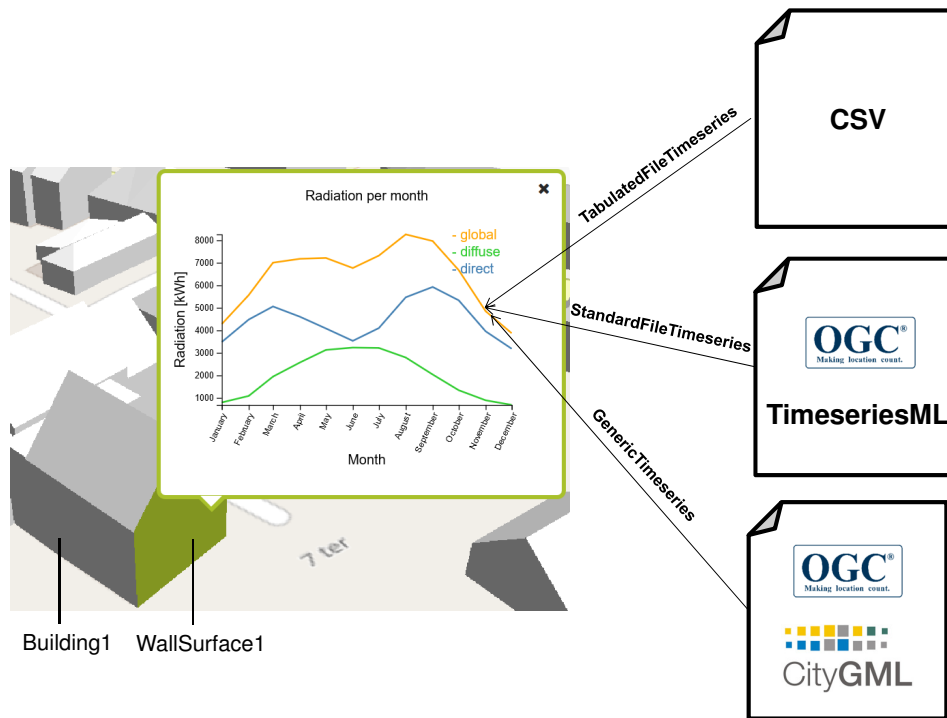
**Listing 5.9:** Illustration of a CityGML Building WallSurface having a generic attribute to record monthly solar irradiation value

```
<con:WallSurface gml:id="building1_wall1">
  <core:genericAttribute>
    <gen:DoubleAttribute>
      <gen:name>globalRadMonth</gen:name>
      <gen:value>4293.446</gen:value>
    </gen:DoubleAttribute>
  </core:genericAttribute>
</con:WallSurface>
```

The results of the solar potential simulation can be stored in other sources such as an external CSV file or can also be represented in-line. The following sub-sections show different possibilities of managing the solar potential simulation results in various data sources as shown in figure 5.11.

#### 5.3.2.1 Example 1: TabulatedFileTimeseries

This example illustrates that the monthly simulation results for the building roof surface are exported in a CSV file named *results.csv*. The CSV file includes a header line showing the column names. The first column is "Surface\_ID" mentioning the *gml:id* of the building surfaces. The second column is "Time" showing the timestamp for each irradiation value. The timestamp includes year followed by month digits (YYYY-MM). For example, 2015-01 represents the irradiation value computed for January 2015. The next column is "Uom" representing the unit of measurement, which is Kilowatt-



**Figure 5.11:** Representation of time-series in-line using Atomic Timeseries.

hour (kWh) in this example. The next three columns represent the computed Diffuse, Direct, and Global irradiation levels for the individual building surfaces. In the given CSV file, the field separator character is the comma (',') and the decimal symbol is the dot ('.').

```
Surface_ID,Time,Uom,Diffuse,Direct,Global
building1_wall1,2015-01,kWh,1454.653,3315.214,4293.446
building1_wall1,2015-02,kWh,1866.883,4002.232,5563.502
.....
building1_wall1,2015-12,kWh,1341.543,3001.412,4010.239
building1_wall12,2015-01,kWh,1313.344,3112.122,4109.742
.....
```

Hence, to override the generic attribute "globalRadMonth" of the building wall surface "building1\_wall1" based on the results stored in the given CSV file, the Dynamizer can be defined as shown in listing 5.10.



**Listing 5.10:** CityGML Dynamizer `TabulatedFileTimeseries` referring to time-series stored in an external CSV file

```

<dyn:Dynamizer gml:id="global_irradiation_Dynamizer">
  <dyn:attributeRef>
    <!--Single line without any linebreak and space-->
    //con:RoofSurface[@gml:id='building1_wall1']
    /core:genericAttribute
    /gen:DoubleAttribute[gen:name='globalRadMonth']
    /gen:value
  </dyn:attributeRef>
  <dyn:startTime>2015-01-01T00:00:00Z</dyn:startTime>
  <dyn:endTime>2016-01-01T00:00:00Z</dyn:endTime>
  <dyn:dynamicData>
    <dyn:TabulatedFileTimeseries>
      <dyn:firstTimestamp>2015-01-01T00:00:00Z</dyn:firstTimestamp>
      <dyn:lastTimestamp>2016-01-01T00:00:00Z</dyn:lastTimestamp>
      <dyn:observationProperty>
        GlobalIrradiationPerMonth
      </dyn:observationProperty>
      <dyn:uom>kWh</dyn:uom>
      <dyn:fileLocation>
        file:///C:/Folder1/results.csv
      </dyn:fileLocation>
      <dyn:fileType>csv</dyn:fileType>
      <dyn:mimeType>application/csv</dyn:mimeType>
      <dyn:valueType>double</dyn:valueType>
      <dyn:numberOfHeaderLines>1</dyn:numberOfHeaderLines>
      <dyn:fieldSeparator>,</dyn:fieldSeparator>
      <dyn:decimalSymbol>.</dyn:decimalSymbol>
      <dyn:idColumnNo>1</dyn:idColumnNo>
      <dyn:idValue>building1_wall1</dyn:idValue>
      <dyn:timeColumnNo>2</dyn:timeColumnNo>
      <dyn:valueColumnNo>6</dyn:valueColumnNo>
    </dyn:TabulatedFileTimeseries>
  </dyn:dynamicData>
</dyn:Dynamizer>

```

In the listing 5.10, the attribute *attributeRef* refers to the generic attribute *globalRadMonth* of the *RoofSurface* with the *gml:id* *building1\_wall1*. The attributes *startTime* and *endTime* represent the entire temporal extent for the dynamic values. The *TabulatedFileTimeseries* allows us to provide details of the CSV file which includes the simulation results. The attributes *firstTimestamp* and *lastTimestamp* represent the temporal extent specified within the CSV file. The *observationProperty* is specified as *GlobalIrradiationPerMonth* and *uom* is *kWh*. The attribute *fileLocation* specifies the location where the CSV file is stored. The *fileType* is *csv* and the associated *mimeType* is *application/csv*. The

simulation results recorded in the CSV file are of type double. Hence, the *valueType* is mentioned as *double*. In the mentioned CSV file, there is one header line indicating the names of the columns. Hence, the attribute *numberOfHeaderLines* is 1. In case, there is no header line in the CSV file, it can be shown as 0. The attributes *fieldSeparator* and *decimalSymbol* represent the type of separator and decimal symbol in the file, which are ',' and '.' respectively in this case. This listing uses the column numbers for retrieving the values from the fields Surface\_ID, Time, and Global value. Hence, the specific values for column numbers are given in the attributes *idColumnName*, *timeColumnName*, and *valueColumnName*. Alternatively, it is possible to use column names instead of column numbers which is illustrated in listing 5.11. The attribute *idValue* indicates the value of the id required for querying. For example, in the mentioned CSV file, the simulation results are stored for different building ids such as *building1\_wall1*, *building1\_wall2*, *building1\_wall3* etc. Since the *attributeRef* refers to the RoofSurface *building1\_wall1* in this example, the *idValue* is set to *building1\_wall1*. It allows to define that only the Surface\_ID *building1\_wall1* is used from this *TabulatedFileTimeseries*.

**Listing 5.11:** Alternative representation of CityGML Dynamizer TabulatedFileTimeseries

```
<dyn:Dynamizer gml:id="global_irradiation_Dynamizer">
  <!-- same as Listing 5.10 -->
  <dyn:dynamicData>
    <dyn:TabulatedFileTimeseries>
      <dyn:firstTimestamp>2015-01-01T00:00:00Z</dyn:firstTimestamp>
      <dyn:lastTimestamp>2016-01-01T00:00:00Z</dyn:lastTimestamp>
      <dyn:observationProperty>
        GlobalIrradiationPerMonth
      </dyn:observationProperty>
      <dyn:uom>kWh</dyn:uom>
      <dyn:fileLocation>
        file:///C:/Folder1/results.csv
      </dyn:fileLocation>
      <dyn:fileType>csv</dyn:fileType>
      <dyn:mimeType>application/csv</dyn:mimeType>
      <dyn:valueType>double</dyn:valueType>
      <dyn:numberOfHeaderLines>1</dyn:numberOfHeaderLines>
      <dyn:fieldSeparator>,</dyn:fieldSeparator>
      <dyn:decimalSymbol>.</dyn:decimalSymbol>
      <dyn:idColumnName>Surface_ID</dyn:idColumnName>
      <dyn:idValue>building1_wall1</dyn:idValue>
      <dyn:timeColumnName>Time</dyn:timeColumnName>
      <dyn:valueColumnName>Global</dyn:valueColumnName>
    </dyn:TabulatedFileTimeseries>
  </dyn:dynamicData>
</dyn:Dynamizer>
```

### 5.3.2.2 Example 2: Generic Timeseries

The solar potential simulation results (as shown in the previous listings) can also be represented in-line using the *GenericTimeseries* class. It allows defining the necessary metadata of the time-series. The Dynamizer *GenericTimeseries* can be defined as shown in listing 5.12.

**Listing 5.12:** CityGML Dynamizer *GenericTimeseries* representing time-series in-line

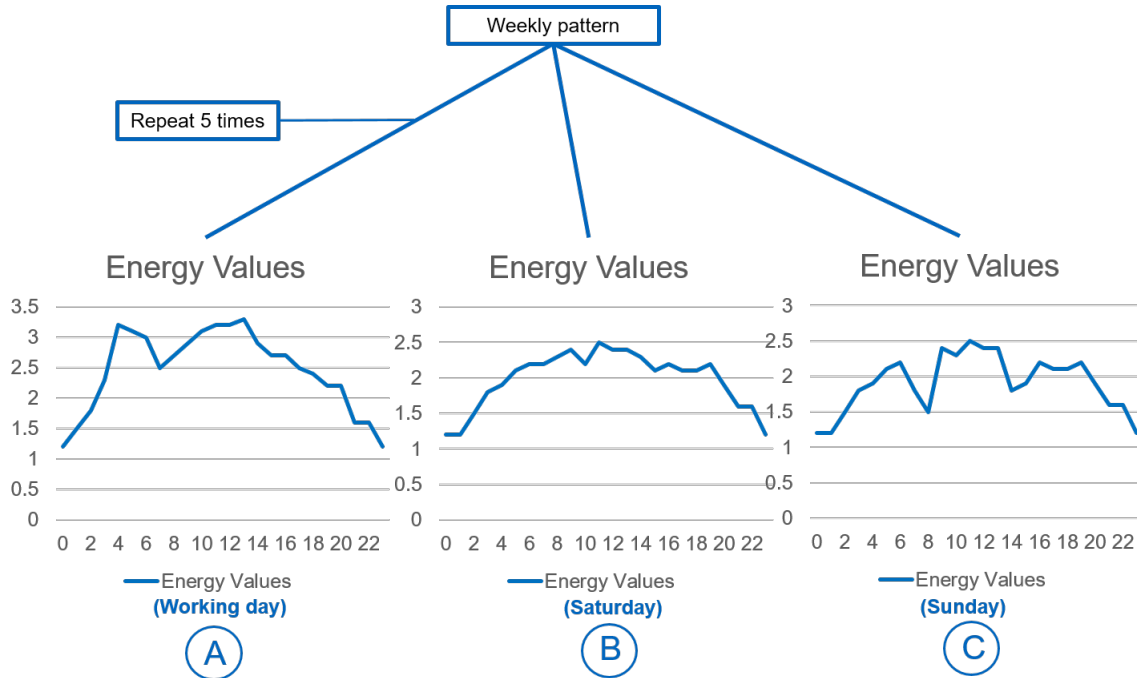
```
<dyn:Dynamizer gml:id="global_irradiation_Dynamizer">
  <!-- same as Listing 5.10 -->
  <dyn:dynamicData>
    <dyn:GenericTimeseries>
      <dyn:firstTimestamp>2015-01-01T00:00:00Z</dyn:firstTimestamp>
      <dyn:lastTimestamp>2016-01-01T00:00:00Z</dyn:lastTimestamp>
      <dyn:observationProperty>
        GlobalIrradiationPerMonth
      </dyn:observationProperty>
      <dyn:uom>kWh</dyn:uom>
      <dyn:valueType>double</dyn:valueType>
      <dyn:timeValuePair>
        <dyn:TimeValuePair>
          <dyn:timestamp>2015-01</dyn:timestamp>
          <dyn:doubleValue>4293.446</dyn:doubleValue>
        </dyn:TimeValuePair>
      </dyn:timeValuePair>
      <dyn:timeValuePair>
        <dyn:TimeValuePair>
          <dyn:timestamp>2015-02</dyn:timestamp>
          <dyn:doubleValue>5563.502</dyn:doubleValue>
        </dyn:TimeValuePair>
      </dyn:timeValuePair>
      <!-- TimeValuePair for all 12 months are shown here -->
    </dyn:GenericTimeseries>
  </dyn:dynamicData>
</dyn:Dynamizer>
```

It is also possible to represent the time-series values according to international standards such as OGC TimeseriesML 1.0 using the *StandardFileTimeseries* class. A working illustration is shown in Chapter 8.

### 5.3.3 Representing complex periodic patterns using Dynamizers

Dynamizers already support *AtomicTimeseries* based on absolute time points within which the values of the attributes can be mapped and can be represented as a tabulation of measured data. One typical example illustrating such a scenario is the mapping of energy values of a building for every hour in a

day. The hourly consumption readings for a working day, a Saturday, and a Sunday can be represented as individual *AtomicTimeseries* (represented by A, B, and, C respectively in figure 5.12).



**Figure 5.12:** Example of composing *AtomicTimeseries* to a pattern.

Now, a *CompositeTimeseries* may contain five repetitions of *AtomicTimeseries* A to reflect a pattern of energy consumption values for all weekdays. However, energy demand estimations may require separate patterns for weekdays and weekends requiring the composition of multiple variation behaviours. Such complex behaviours may be expressed using the *CompositeTimeseries*, wherein a weekly pattern can be defined containing the energy values for all seven days of a week (represented as 'AAAAABC'). The advantage in using such *CompositeTimeseries* is that it allows the study of customised patterns; for example, patterns for only weekdays ('AAAAA') or patterns of only weekends for four weeks ('BCBCBCBC'). Similarly, more complex patterns of arbitrary depths such as weekly, monthly and yearly, can be defined using this approach (as shown later in figure 5.13). It is also possible to express time-varying data for an arbitrarily long period by combining various patterns. For example, if we need to determine the pattern for energy consumption values for evening hours on weekdays, an *AtomicTimeseries* can be defined for 6 hours (e.g. from 6 PM to midnight). Further, the *CompositeTimeseries* allows connecting an additional 6 hours for all weekdays in a week. To represent a weekly pattern (shown above as 'AAAAABC') based on five repetitions of weekday values followed by one repetition of each of Saturday and Sunday values, the Dynamizer *CompositeTimeseries* can be defined as shown in listing 5.13.

**Listing 5.13:** CityGML Dynamizer CompositeTimeseries representing periodic patterns of energy consumption values

```

<!-- Only the Dynamizer feature is shown in this listing -->
<dyn:Dynamizer gml:id="electricity_consumption">
  <dyn:attributeRef>
    <!-- XPath expression to the city object property -->
  </dyn:attributeRef>
  <!-- ISO 8601 Week Date Representation showing absolute
    timestamps-->
  <dyn:startTime>2015-W01-01</dyn:startTime>
  <dyn:endTime>2015-W02-01</dyn:endTime>
  <dyn:dynamicData>
    <dyn:CompositeTimeseries>
      <!-- Component for Weekdays-->
      <dyn:component>
        <dyn:TimeseriesComponent>
          <dyn:repetitions>5</dyn:repetitions>
          <dyn:timeseries>
            <dyn:GenericTimeseries gml:id="Weekdays">
              <!-- ISO 8601 Time Representation showing relative
                timestamps-->
              <dyn:firstTimestamp>T00:00:00</dyn:firstTimestamp>
              <dyn:lastTimestamp>T24:00:00</dyn:lastTimestamp>
              <dyn:observationProperty>ElecConsump</dyn:observationProperty>
              <dyn:uom>kWh</dyn:uom>
              <dyn:valueType>double</dyn:valueType>
              <dyn:timeValuePair>
                <dyn:TimeValuePair>
                  <dyn:timestamp>T00:00:00</dyn:timestamp>
                  <dyn:doubleValue>1.32</dyn:doubleValue>
                </dyn:TimeValuePair>
              </dyn:timeValuePair>
              <dyn:timeValuePair>
                <dyn:TimeValuePair>
                  <dyn:timestamp>T01:00:00</dyn:timestamp>
                  <dyn:doubleValue>1.41</dyn:doubleValue>
                </dyn:TimeValuePair>
              </dyn:timeValuePair>
              <dyn:timeValuePair>
                <dyn:TimeValuePair>
                  <dyn:timestamp>T02:00:00</dyn:timestamp>
                  <dyn:doubleValue>1.53</dyn:doubleValue>
                </dyn:TimeValuePair>
              </dyn:timeValuePair>
              <!-- values for all 24 hours in a weekday -->
            </dyn:GenericTimeseries>
          </dyn:timeseries>
        </dyn:TimeseriesComponent>
      </dyn:component>
    </dyn:CompositeTimeseries>
  </dyn:dynamicData>
</dyn:Dynamizer>

```

```

    </dyn:GenericTimeseries>
  </dyn:timeseries>
</dyn:TimeseriesComponent>
</dyn:component>
<!-- Component for Saturday-->
<dyn:component>
  <dyn:TimeseriesComponent>
    <dyn:repetitions>1</dyn:repetitions>
    <dyn:timeseries>
      <dyn:GenericTimeseries gml:id="Saturday">
        <!-- ISO 8601 Time Representation showing relative
             timestamps-->
        <dyn:firstTimestamp>T00:00:00</dyn:firstTimestamp>
        <dyn:lastTimestamp>T24:00:00</dyn:lastTimestamp>
        <dyn:observationProperty>ElecConsump</dyn:observationProperty>
        <dyn:uom>kWh</dyn:uom>
        <dyn:valueType>double</dyn:valueType>
        <dyn:timeValuePair>
          <dyn:TimeValuePair>
            <dyn:timestamp>T00:00:00</dyn:timestamp>
            <dyn:doubleValue>1.39</dyn:doubleValue>
          </dyn:TimeValuePair>
        </dyn:timeValuePair>
        <dyn:timeValuePair>
          <dyn:TimeValuePair>
            <dyn:timestamp>T01:00:00</dyn:timestamp>
            <dyn:doubleValue>1.44</dyn:doubleValue>
          </dyn:TimeValuePair>
        </dyn:timeValuePair>
        <dyn:timeValuePair>
          <dyn:TimeValuePair>
            <dyn:timestamp>T02:00:00</dyn:timestamp>
            <dyn:doubleValue>1.52</dyn:doubleValue>
          </dyn:TimeValuePair>
        </dyn:timeValuePair>
        <!-- values for all 24 hours in a weekday -->
      </dyn:GenericTimeseries>
    </dyn:timeseries>
  </dyn:TimeseriesComponent>
</dyn:component>
<!-- Component for Sunday-->
<dyn:component>
  <dyn:TimeseriesComponent>
    <dyn:repetitions>1</dyn:repetitions>
    </dyn:timeseries>
    <dyn:GenericTimeseries gml:id="Sunday">

```

```

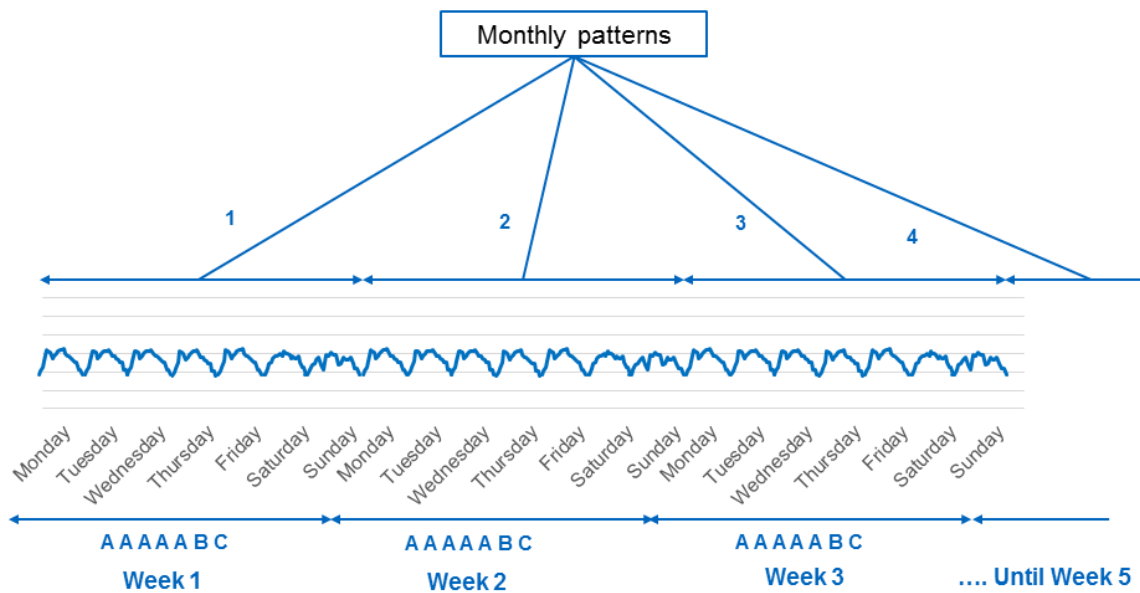
<!-- ISO 8601 Time Representation showing relative
      timestamps-->
<dyn:firstTimestamp>T00:00:00</dyn:firstTimestamp>
<dyn:lastTimestamp>T24:00:00</dyn:lastTimestamp>
<dyn:observationProperty>ElecConsump</dyn:observationProperty>
<dyn:uom>kWh</dyn:uom>
<dyn:valueType>double</dyn:valueType>
<dyn:timeValuePair>
  <dyn:TimeValuePair>
    <dyn:timestamp>T00:00:00</dyn:timestamp>
    <dyn:doubleValue>1.30</dyn:doubleValue>
  </dyn:TimeValuePair>
</dyn:timeValuePair>
<dyn:timeValuePair>
  <dyn:TimeValuePair>
    <dyn:timestamp>T01:00:00</dyn:timestamp>
    <dyn:doubleValue>1.46</dyn:doubleValue>
  </dyn:TimeValuePair>
</dyn:timeValuePair>
<dyn:timeValuePair>
  <dyn:TimeValuePair>
    <dyn:timestamp>T02:00:00</dyn:timestamp>
    <dyn:doubleValue>1.59</dyn:doubleValue>
  </dyn:TimeValuePair>
</dyn:timeValuePair>
<!-- values for all 24 hours in a weekday -->
</dyn:GenericTimeseries>
</dyn:timeseries>
</dyn:TimeseriesComponent>
</dyn:component>
</dyn:CompositeTimeseries>
</dyn:dynamicData>
</dyn:Dynamizer>

```

The listing shows that a Dynamizer *CompositeTimeseries* can be defined including the individual *AtomicTimeseries* for weekdays, Saturdays, and Sundays. The attribute *attributeRef* can be defined to refer to a specific generic attribute which requires to be overridden by the dynamic time-dependent values as shown in the previous listings. The attributes *startTime* and *endTime* represent the overall temporal extent of the Dynamizer feature. In this example, the Dynamizer *CompositeTimeseries* includes the overall extent of one week. This is represented using the ISO 8601 Week Date Representation<sup>87</sup>. This representation is defined using the string "YYYY-Www-D", where [YYYY] is the year, [Www] is the week number prefixed by the letter W, from W01 through W53, and [D] is the weekday number, from 1 through 7, beginning with Monday and ending with Sunday. Therefore, the string "2015-W01-01" in this listing indicates Monday of the week number 1 of the year 2015. Similarly,

<sup>87</sup><https://www.iso.org/iso-8601-date-and-time-format.html>

the string "2015-W01-07" indicates Sunday of the week number 1 of the year 2015. In this way, the values of attributes *startTime* and *endTime* represent the temporal extent of one week including all the days of that particular week. As already mentioned in section 5.2.1, the *startTime* and *endTime* follow the Half-Open interval, which means the *startTime* is inclusive and the *endTime* is exclusive. The attribute *CompositeTimeseries* includes 3 *components* in this listing, representing Weekdays, Saturdays, and Sundays respectively. The *component* for Weekdays includes a *GenericTimeseries* with the *gml:id Weekdays* indicating all the time/value pairs for electricity consumption values for every hour in a day. Within this *GenericTimeseries*, the attributes *firstTimestamp* and *lastTimestamp* are relative timestamps, and represent the temporal extent for all the weekdays, starting at 0 am and ending at 0 am the next day. The duration of the *GenericTimeseries* is 24 hours, and this is repeated five times leading to a total duration of five days. Each observation is represented using *TimeValuePair* where the *timestamp* is a time value relative to the absolute value defined within the temporal extent. For each iteration, the combination of this relative timestamp and the absolute timestamp give a complete timestamp including the date and time values. The attribute *repetitions* has a value 5 for this *component* to represent 5 iterations for all the weekdays in the week. In a similar way, the other components for Saturday and Sunday can be defined using the *repetitions* value as 1 to ensure only 1 iteration.



**Figure 5.13:** Example of complex CompositeTimeseries.

Listing 5.13 is just a representation for a *CompositeTimeseries* representing periodic patterns of electricity consumption values for a week. Depending on the use cases, more complex *CompositeTimeseries* can be defined in similar ways (as shown in figure 5.13). For example, the *CompositeTimeseries* defined in Listing 5.13 can be repeated 52 times to represent such patterns for the entire year. Similarly, the *CompositeTimeseries* can have the components only for weekdays to represent the patterns only for weekdays.



## 5.4 Discussions

This chapter presents the novel Dynamizer concept extending static 3D city models by supporting variations of individual feature properties and associations over time. It provides a data structure to represent dynamic values in different and generic ways. Such dynamic values may be given by retrieving observations from sensor-based services [R1]; subscribing to a real-time data streams for alerts [R2]; tabulation of time/value pairs for varying geometries, topologies, semantic, and appearance attributes [R3-R4]; and patterns of time/value pairs [R5].

The Dynamizer concept has been developed for the next version of the CityGML standard (version 3.0) and has been proposed to the OGC CityGML Standard Working Group for its adoption. However, the proposed concept can also be implemented as a CityGML Application Domain Extension (ADE) by using the "hook" mechanism. The implementation and examples of CityGML Dynamizers as an ADE have already been shown in the Engineering Report of the project OGC Future City Pilot Phase 1 (Chaturvedi and Kolbe 2017). This implementation allows using such Dynamizer features with the current version of CityGML 2.0. The advantage of this approach is that it enables selected properties of city models to become dynamic without changing the original CityGML data model. If an application does not support dynamic data, it does not include these particular types of features. Dynamizer is a feature type, which even allows users and applications to use it with the OGC Web Feature Service. No extension of other OGC standards is required.



## **Part II**

# **Management of Time-dependent Properties**



## Chapter 6

---

### Management on the level of databases

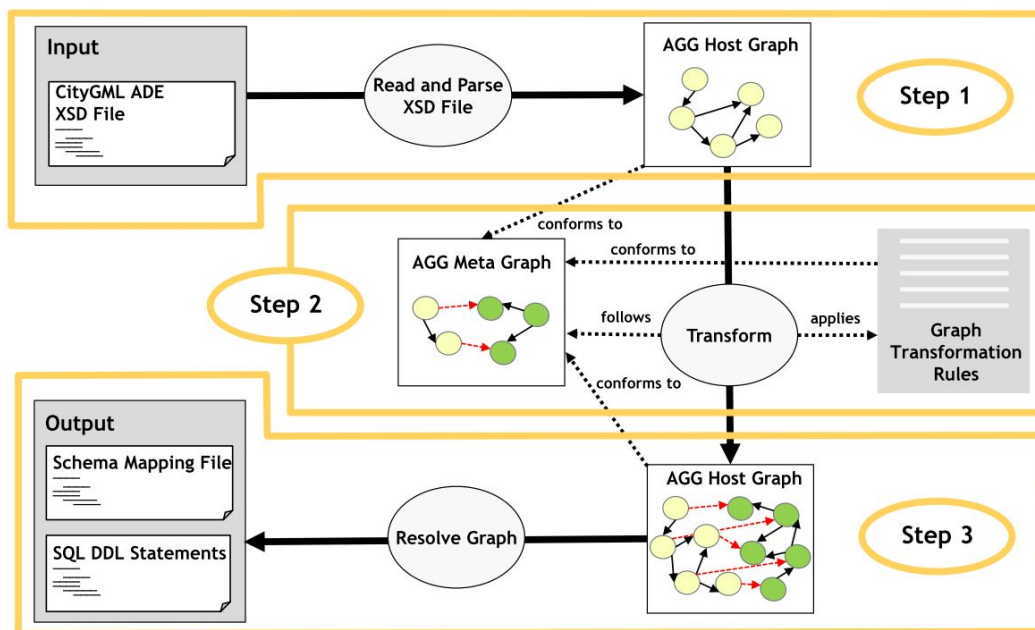
This chapter discusses the approaches for managing the newly added modules of CityGML in database management systems such as the 3DCityDB. The Versioning and Dynamizer concepts are developed for CityGML 3.0, however, they can also be implemented as CityGML ADEs. In order to manage the ADEs in the 3DCityDB, the 3DCityDB ADE Plugin Manager is utilised. The extended schema of 3DCityDB includes the relevant tables and relations for storing Versioning and Dynamizer modules. However, the current version of 3DCityDB manages only static properties of city objects. The dynamic properties such as time-series are not supported. Hence, this chapter presents a new relational database model for storing and managing dynamic properties along with static properties within the 3DCityDB.

This chapter is based on the published paper

**Chaturvedi, K.,** Yao, Z. and Kolbe, T. H. (2019). ‘Integrated Management and Visualization of Static and Dynamic Properties of Semantic 3D City Models’. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLII-4/W17*, pp. 7–14. URL: <https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-4-W17/7/2019/>

## 6.1 Managing CityGML ADEs within databases

The 3D City Database (3DCityDB) software suite comprises (i) a database schema built on top of standard relational database schema such as PostgreSQL and Oracle, (ii) an Importer/Exporter allowing importing and exporting of CityGML data to/from the database schema, and (iii) a Web Feature Service allowing to retrieve CityGML features using standardised web requests. However, CityGML also provides an extension mechanism called “Application Domain Extension (ADE)” . The ADE mechanism allows third parties to dynamically and systematically extend the existing CityGML data models by attaching new application schemas which can contain additional feature classes and attribute properties defined for specific application domains. For example, the Energy ADE (Agugiaro et al. 2018) introduces new attributes specific to energy use cases. Similarly, the UtilityNetwork ADE (Kutzner et al. 2018) provides new features for mapping underground structures. Since ADEs are diverse and completely different from each other, it has been a challenge to extend the database schema to newly defined feature classes and attribute properties belonging to individual ADEs. For this purpose, the current release of the 3D City Database software suite (version 4.2) includes a tool called "ADE Plugin Manager". The ADE Plugin Manager allows dynamically extending a 3DCityDB instance to facilitate the storage and management of arbitrary CityGML ADEs. Its concept was developed by (Yao and Kolbe 2017). The ADE Plugin Manager provides two-step solutions:



**Figure 6.1:** Transformation Workflow of the 3DCityDB ADE Plugin Manager. Image taken from (Yao and Kolbe 2017).

- **Dynamically extending the 3DCityDB to manage ADEs.** This implementation is based on the Open Source Attributed Graph Grammar (AGG) transformation engine<sup>88</sup>. As shown in figure 6.1, the AGG transformation engine enables automatic transformation from the XML application schema (XSD) of a CityGML ADE to a compact relational database schema which includes necessary

<sup>88</sup><https://www.user.tu-berlin.de/~o.runge/agg/agg-docu.html>

tables, indexes, and constraints. Besides, an XML-based schema mapping file is also automatically generated, which contains the relevant meta-information about the derived database schema. The mapping file also defines explicit mapping relationships between the source and target schemas. It allows developers to implement applications for managing and processing the ADE data contents stored in a 3DCityDB instance.

- **Extending the Importer/Exporter to import and export ADE data.** Once the ADE is registered with the 3DCityDB, the Importer/Exporter requires an extension to import or export ADE data to/from the new ADE tables. The Importer/Exporter does not provide generic ADE support yet; hence, an ADE extension<sup>89</sup> needs to be implemented for the ADE API manually for each ADE.

In this way, the ADE Plugin manager not only extends the database schema but also allows importing the ADE elements in the database tables (see figure 6.2). The existing Web Feature Service extension of the 3DCityDB already allows retrieving ADE elements using standardised requests.

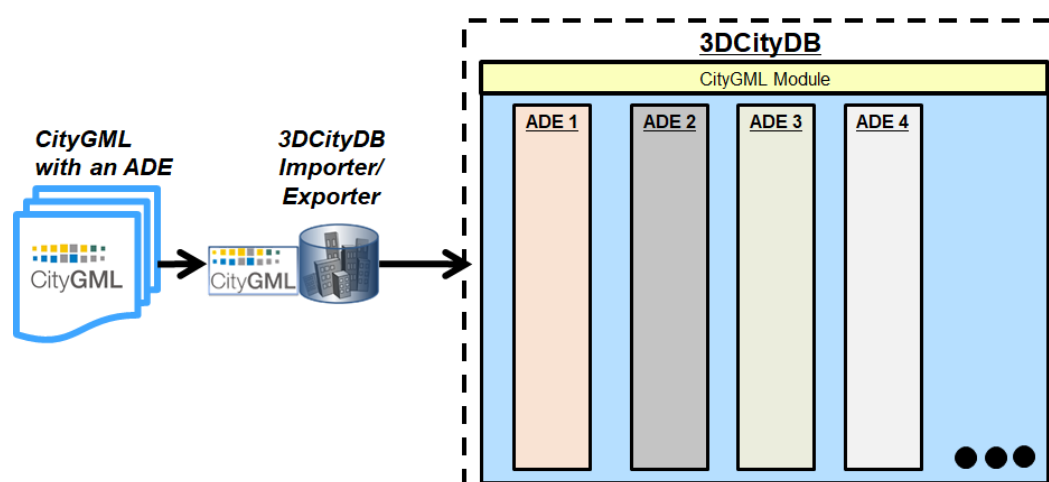


Figure 6.2: Management of CityGML ADEs within the 3DCityDB.

Using the concepts and implementations developed within the ADE Plugin Manager, the newly developed ADEs, Versioning ADE and Dynamizer ADE, can also be managed dynamically in the 3DCityDB. The following sub-sections discuss approaches to extend the 3DCityDB for storing both ADEs and retrieving the newly developed features using web services.

### 6.1.1 Managing the Versioning ADE within the 3DCityDB

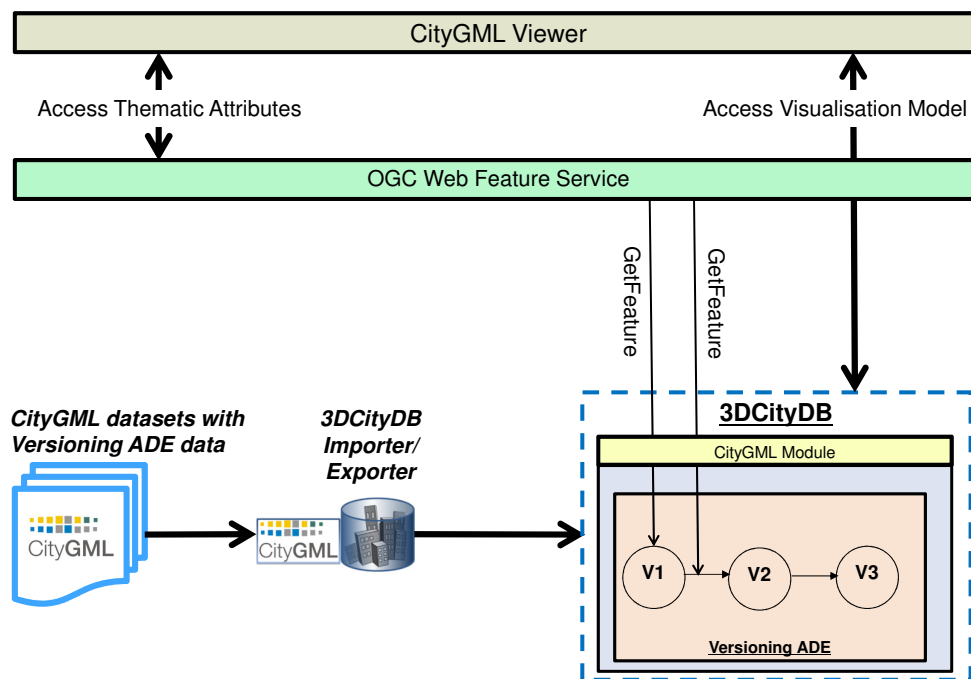
The extension of the 3DCityDB for a specific ADE requires the XML Application Schema (XSD) for that particular ADE. The ADE is developed using the Unified Modelling Language (UML), and the UML model can be transformed into the XSD file dynamically using the tool ShapeChange<sup>90</sup>. ShapeChange is an open-source Java tool, which can process UML models for geographic information according to the ISO 19100 standards family and derives the GML application schemas (and other transfer formats) from these UML models. The tool can directly read the UML models created using the software Enterprise Architect (EA) via the EA Java API. The guidelines and examples for how

<sup>89</sup><https://github.com/3dcitydb/extension-test-ade>

<sup>90</sup><https://shapechange.net/>

to derive the XML schemas from the EA file and for how to create the ADEs and to derive the corresponding XML schema using ShapeChange are given by (Kutzner 2016). The CityGML instance files can be validated against the XSD file using tools such as FME<sup>91</sup> and NetBeans<sup>92</sup>.

Once the XSD file is available, the ADE Plugin Manager can be used to extend the 3DCityDB schema to create tables and relations dynamically. In the case of the Versioning ADE, new tables and relations are created for *Version* and *VersionTransition* features. Once the database schema is extended, the Importer/Exporter tool is configured to allow the import and export of valid CityGML files using the Versioning ADE. The guidelines for using the ADE Plugin Manager for arbitrary ADEs are provided in the 3DCityDB official documentation<sup>93</sup>.



**Figure 6.3:** Version Management in the 3DCityDB using the Versioning ADE.

Figure 6.3 shows a conceptual illustration of how CityGML instance datasets with three versions of a city model (same as listing 4.1) can be managed in the 3DCityDB and further queried using an OGC Web Feature Service by external applications. As shown in listing 4.1, Version 1 contains a building with roof type "Flat" and function "Office". In Version 2, the function changes to "Living". In version 3, the roof type changes to "Saddle". The ADE Plugin Manager of the Importer/Exporter allows the possibility of deriving the relational schema of the Versioning ADE and registering them in the 3DCityDB. As described by (Yao and Kolbe 2017), an additional code has to be programmed further to implement the import/export capabilities for the ADE data. After performing this implementation, the CityGML file with all the versions can be imported to the 3DCityDB where all versions with links

<sup>91</sup> <https://www.safe.com/>

<sup>92</sup> <https://netbeans.org/>

<sup>93</sup> <https://www.3dcitydb.org/3dcitydb/documentation/>



to their city objects and properties and version transitions are stored in the respective tables. This enables performing queries (e.g. SQL operations) not only on city objects and properties but also on the versions and their transitions. Since all the Versions and Version Transitions are accessible as *FeatureType*, they can also be accessed by the OGC Web Feature Service of the 3DCityDB. It is possible to perform queries, e.g. "retrieve all available versions of the city model". With the Versioning concept, each object now also supports bi-temporal attributes: *creationDate* and *terminationDate* reflecting database transaction time and *validFrom* and *validTo* reflecting actual world time. Hence, these attributes could be used by a WFS query to choose the appropriate version that was valid at a specific database or real world time respectively. It enables querying "How did the city look like at a specific point in time?" or "How did the city model look like at a specific point in time?". The appropriate query can determine all the features and attributes based on the temporal attributes defined for each attribute.

### 6.1.2 Managing the Dynamizer ADE within the 3DCityDB

Using the ADE Plugin Manager, the Dynamizer ADE can also be registered in the 3DCityDB. As described in Chapter 5, Dynamizer is defined as a *FeatureType*, which (i) has a direct link to an external sensor-based web service using the *SensorConnection* type or (ii) has time-series represented in-line with city objects using *AtomicTimeseries* or *CompositeTimeseries* or (iii) uses external files. The Dynamizer concept was initially implemented as an ADE within the project OGC Future City Pilot Phase 1 (Chaturvedi and Kolbe 2017). Similar to the Versioning ADE (as described in section 6.1.1), the UML model of the Dynamizer ADE can be transformed into valid XSD files using the ShapeChange tool. Further, using the ADE Plugin Manager of the Importer/Exporter, it is possible to derive the relational schema of the Dynamizer ADE and to register them in the 3DCityDB. This allows extending the 3DCityDB schema to support tables and relations for new Dynamizer features including (i) *SensorConnection*, (ii) *AtomicTimeseries*, and (iii) *CompositeTimeseries*. In order to better manage and query on specific time-value pairs, new modules have been introduced for the relational data model for the Dynamizer ADE (more details in section 6.2). Furthermore, after the database schema is extended, using an additional programmable code as mentioned by (Yao and Kolbe 2017), the Importer/Exporter tool can be configured to allow import and export of the CityGML datasets with the Dynamizer ADE data (more details in section 6.3).

The benefit with the proposed methodology is that it enables the management of time-series information represented within the Dynamizer data at the database level. For example, in the case of solar potential simulation results represented in-line as *GenericTimeseries* within a building wall surface (as shown in section 5.3.2), an SQL query on the 3DCityDB can determine the irradiation value for a specific month or the average of irradiation values in summer over the last five years.

This methodology also allows retrieving Dynamizer features using the OGC Web Feature Service. For example, it is possible to determine "which city objects have Dynamizers associated with them?", "which rooms have sensors of type Smart Meters installed?", "which building roofs have solar panels on top of them", "which traffic junctions have inductive loops located beneath them?" and so on. However, problems arise when we access such time-value pairs using the Web Feature Service as shown in Figure 6.4. The issue is that the Web Feature Service is not suitable for querying time-series information. For example, in the case of solar potential simulation results represented in-line as *Generic Timeseries* within a building wall surface, it is not optimal for a WFS to query the irradiation values for a specific month. Although CityGML applications can access thematic information of a CityGML data using a Web Feature Service, there is no concrete way for such applications to access

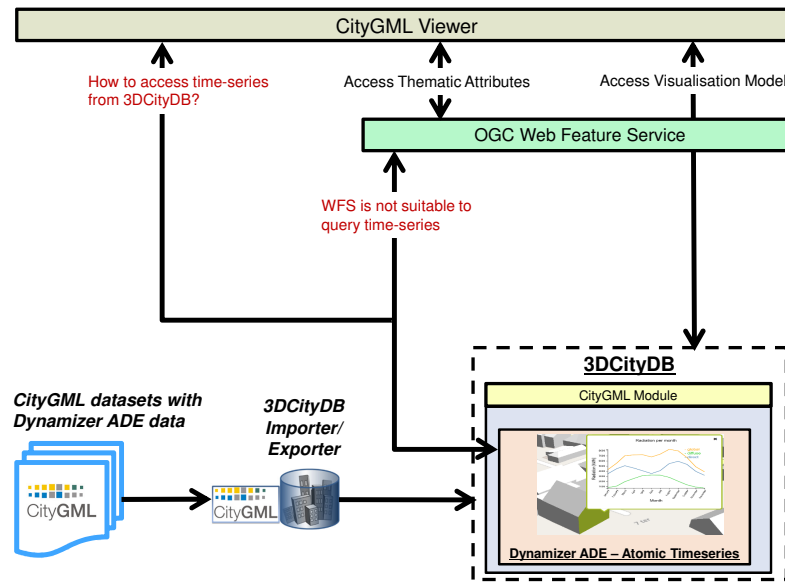


Figure 6.4: Issues with the access of Dynamizer AtomicTimeseries by CityGML Viewers.

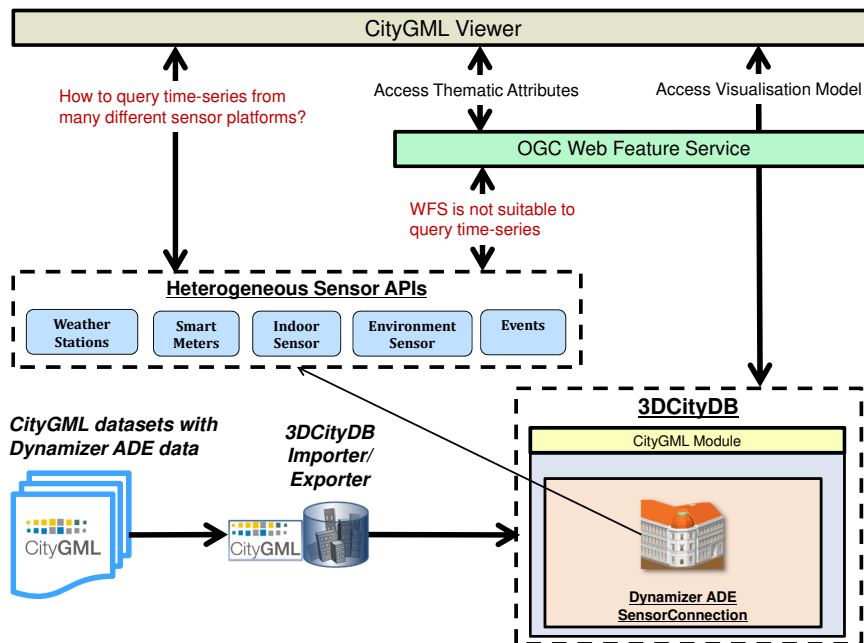


Figure 6.5: Issues with the access of Dynamizer SensorConnection by CityGML Viewers.

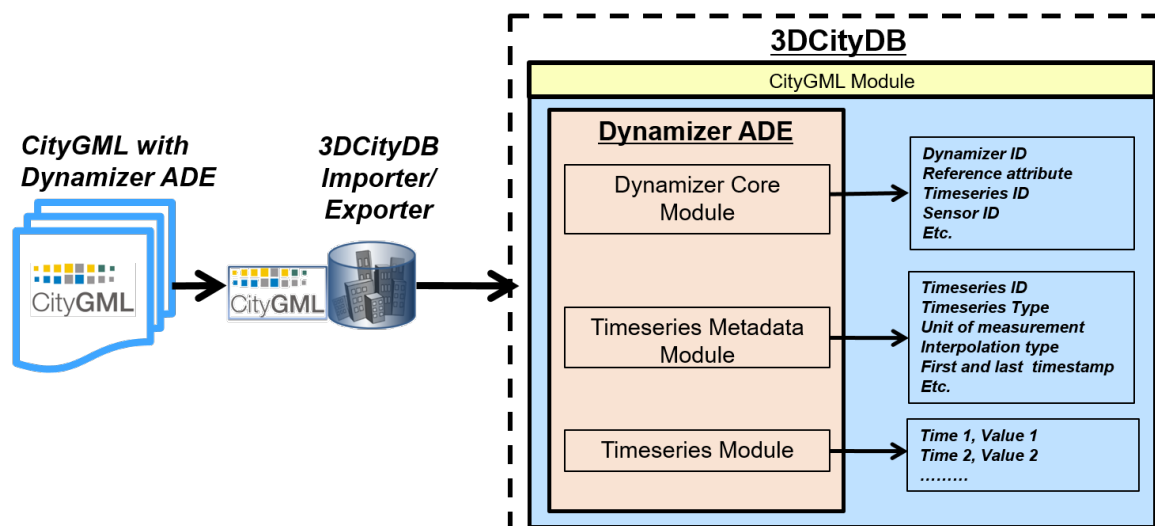
and interpret time-series information from the same CityGML dataset. Similarly, if Dynamizer has a *SensorConnection*, although it avoids storing time-series values in the database, objects can be linked

with any external API. It is still an issue of how results from different APIs can be interpreted in common ways by CityGML applications (as shown in Figure 6.5). These two issues are discussed in detail and solutions are provided accordingly in Chapter 7.

The following section introduces a relational database model for storing and managing the Dynamizer ADE along with its time-series within the 3DCityDB. Further, the research work proposes an extension of the 3DCityDB Importer/Exporter to import and export CityGML documents, including Dynamizer ADE data. It enables managing dynamic data (such as time points within time-series) associated with city objects, which can further be queried and used with standard SQL operations. The concepts have also been developed for querying sensor data from heterogeneous APIs in common and standardised ways (discussed in Chapter 7).

## 6.2 New Relational Data Model for the Dynamizer ADE

This section proposes a high-level architecture for managing and visualising the time-dynamic properties along with static properties of Semantic 3D City Models. The architecture extends the 3D City Database (3DCityDB) to store and manage dynamic properties encoded within the CityGML Dynamizer ADE by defining a new relational data model.



**Figure 6.6:** High Level Overview of the implementation of the Dynamizer ADE within the 3DCityDB.

As shown in figure 6.6, the 3DCityDB is extended for supporting the Dynamizer ADE. The implementation employs the 3DCityDB ADE Plugin Manager, which provides an automatic way for dynamically extending the 3DCityDB to support the storage and management of CityGML models with ADEs. However, to improve the query performance, the relational data model of Dynamizer ADE has been developed by defining three separate modules. The first module is the Dynamizer Core Module for storing the core attributes of Dynamizer (such as unique ID, reference attribute, etc.). The second module is the Timeseries Metadata Module for storing the metadata of Timeseries such as Timeseries type, Unit of Measurement etc. The third module is the Timeseries Module for storing raw time point values. The advantage of keeping the Timeseries module separate from the Dynamizer module is that this approach allows making Timeseries module re-usable, for example, by storing

time-series from other ADEs such as the Energy ADE. Furthermore, the 3DCityDB Importer/Exporter is extended to facilitate import and export of CityGML documents with Dynamizer ADE data. It enables managing dynamic data (such as time points within time-series) associated with city objects, which can further be queried and used using standard SQL operations. The framework also allows accessing and retrieving static and dynamic data in an integrated way.

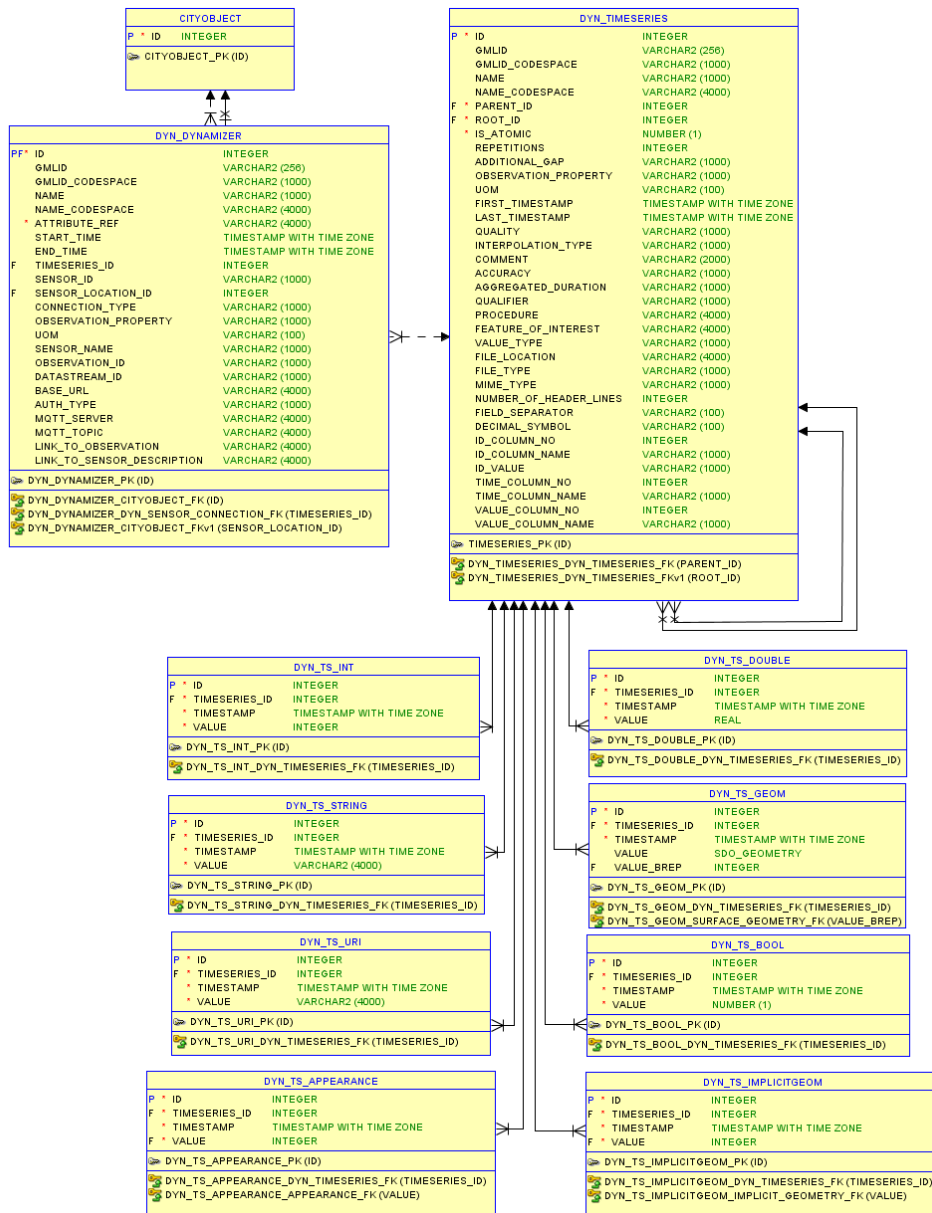
Figure 6.7 shows the relational logical model for the Dynamizer ADE. The relational schema is designed in a way to unify the different representations of time-series into a single representation. As mentioned in Chapter 5, Dynamizers support various possibilities in the forms of *SensorConnection*, *AtomicTimeseries*, and *CompositeTimeseries*. Furthermore, it allows various representations of *AtomicTimeseries* in the form of *GenericTimeseries*, *TabulatedFileTimeseries*, and *StandardFileTimeseries*. The unified representation within the relational schema allows importing and managing different kinds of metadata and time-value pairs using the same structure. More details on which attributes from the Dynamizer schema are mapped onto which attributes in the relational schema are described in the following sub-sections:

### 6.2.1 Dynamizer Core Module

The class *Dynamizer* from the UML model consists of the following core attributes: (i) *Dynamizer\_id*, (ii) *attributeRef*, (iii) *startTime*, and (iv) *endTime*. *attributeRef* refers to a specific (dynamic) attribute of a city object property by using an XPath expression. *startTime* and *endTime* are absolute time points denoting the time span for which the Dynamizer provides dynamic values. These core attributes are stored in the table DYN\_DYNAMIZER. In addition, Dynamizer also provides direct explicit links to external sensor and IoT based services by using the class *SensorConnection*. As already described in detail in section 5.2.2, the *SensorConnection* includes the following attributes (i) *connectionType* defines the type of the sensor API (ii) *observationProperty* defines the name of property which is being measured by the specific API and which the Dynamizer refers, (iii) *uom* is the unit of measurement specified in the API for the defined observed property, (iv) *sensorID* is the unique identifier of the sensor device registered at the specific API, (v) *sensorName* is the name of the sensor device registered with the web service, (vi) *datastreamID* is the unique identifier of the datastream, (vii) *observationID* is the unique identifier for an individual observation within a datastream, (viii) *baseURL* is the resource locator at the root level, (ix) *authType* is the type of the authentication protocol, (x) *mqttServer* defines the name of the server where the MQTT Broker runs, (xi) *mqttTopic* is the name of the topic, (xii) *linkToObservation* represents the complete URL of the operation requesting for the observations based on the specified parameters, and (xiii) *linkToSensorDescription* represents the complete URL of the operation requesting for the description and metadata of the sensor or IoT device. These attributes of the *SensorConnection* class are also stored in the table DYN\_DYNAMIZER.

### 6.2.2 Timeseries Metadata Module

Dynamizer also supports defining time-series in-line within city objects. The in-line time-series within Dynamizers can be modelled in two ways: (i) *AtomicTimeseries*, and (ii) *CompositeTimeseries*. As we learnt in chapter 5, an *AtomicTimeseries* allows 3 different ways to represent time-series: (i) *StandardFileTimeseries* for representing OGC standards such as TimeseriesML and Observations & Measurements, (ii) *TabulatedFileTimeseries* for representing time-series data stored in external tabulated files such as CSV, and (iii) *GenericTimeseries* for representing time-series data using a basic structure. Based on the type of *AtomicTimeseries*, the respective metadata of the



**Figure 6.7:** Relational Logical Model of the Dynamizer ADE. The relational model shows the relations between entities. Each entity box represents different sections indicating (from top to bottom): (i) name of the table, (ii) column names and their data types, (iii) primary key, and (iv) foreign keys. The letters 'P' and 'F' against specific column names indicate Primary and Foreign Keys. The relationship between each entity is shown by association arrows according to the Bachman notation. Due to limited page-width, the complete representations for *ImplicitGeometry*, *SurfaceGeometry*, and *Appearances* are not shown in the figure; only their references are shown using respective Foreign Keys.

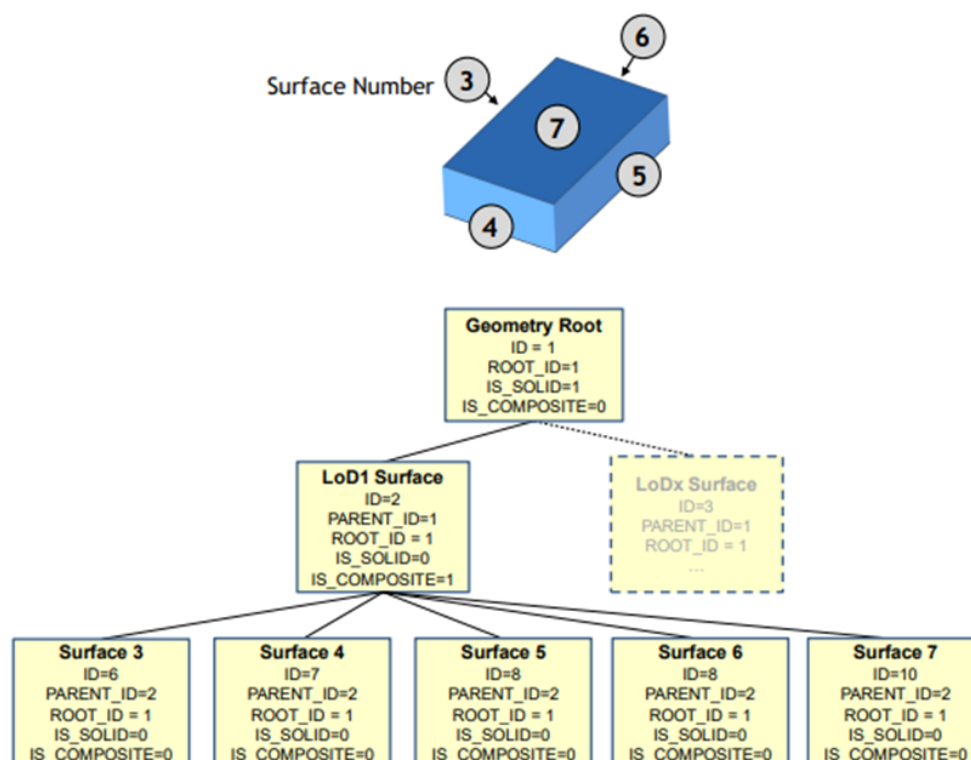
time-series are mapped to the table DYN\_TIMESERIES. For example, if the *StandardFileTimeseries* includes the data according to the TimeseriesML standard, the attributes stored in the table are: FILE\_LOCATION, FILE\_TYPE, and MIME\_TYPE. There may also be other metadata associated to a TimeseriesML file. For example, if it contains a specific interpolation type, its values can be mapped to the attribute INTERPOLATION\_TYPE. Similarly, in the case of *TabulatedFileTimeseries*, the attributes stored in the table are: (i) FILE\_LOCATION, (ii) FILE\_TYPE, (iii) MIME\_TYPE, (iv) VALUE\_TYPE, (v) NUMBER\_OF\_HEADER\_LINES, (vi) FIELD\_SEPERATOR, (vii) DECIMAL\_SYMBOL, (viii) ID\_COLUMN\_NO, (ix) ID\_COLUMN\_NAME, (x) ID\_VALUE, (xi) TIME\_COLUMN\_NO, (xii) TIME\_COLUMN\_NAME, (xiii) VALUE\_COLUMN\_NO, and (xiv) VALUE\_COLUMN\_NAME. In the case of *GenericTimeseries*, the attributes stored in the table is: VALUE\_TYPE. The flag IS\_ATOMIC is used to determine whether the time-series is atomic or composite.

In order to manage *CompositeTimeseries*, the database structure is inspired from the already existing SURFACE\_GEOMETRY table in the 3DCityDB. The representation of the geometry stored in the table SURFACE\_GEOMETRY differs substantially from the UML chart explained in the CityGML specification; however, it offers about the same functionality. It contains various attributes to manage aggregations of multiple surfaces. For example, in case of a LoD1 building, a closed volume is bounded by a *CompositeSurface* which consists of single polygons. As shown in figure 6.8, the aggregation of multiple surfaces, e.g.  $F_1$  to  $F_n$  (IDs 6 to 10) is realised in a way that the newly created surface tuple  $F_{n+1}$  (ID 2) is not assigned a geometry. Instead, the PARENT\_ID of the surfaces  $F_1$  to  $F_n$  refer to the ID of  $F_{n+1}$ . In addition, a further tuple (ID 1) is introduced, which represent the solid and defines the root element of the whole aggregation structure. Each surface references to its root, using the ROOT\_ID attribute. Apart from that, the flags IS\_SOLID distinguishes between surface (0) and solid (1), and IS\_COMPOSITE defines whether this is an aggregate (e.g. *MultiSolid*, *MultiSurface*) or a composite (e.g., *CompositeSolid*, *CompositeSurface*).

In a similar way, since *CompositeTimeseries* composes of an ordered list of *AbstractTimeseries*, several Atomic or Composite Timeseries can be aggregated to form a *CompositeTimeseries*. Each nested time-series references to its root using the ROOT\_ID attribute. This information has a big influence on the system performance, as it allows to avoid recursive queries. If, e.g. the retrieval of all time-series forming a specific *CompositeTimeseries* is of importance, simply those IDs have to be selected which contain the related PARENT\_ID and ROOT\_ID. For instance, in energy applications, an *AtomicTimeseries* may be defined for a working day, a Saturday, and a Sunday (represented by A, B, and C respectively). Now, to reflect a pattern of energy consumption of the entire week (represented as Wx), a *CompositeTimeseries* may contain five repetitions of *AtomicTimeseries* 'A' followed by single representations of *AtomicTimeseries* B and C (represented as 'AAAAABC'). Similarly, for reflecting a pattern of the entire month (Mx), the *CompositeTimeseries* may contain four representations of the time-series W (represented as W1, W2, W3, W4). And lastly, for reflecting a pattern of the entire year (Yx), the *CompositeTimeseries* may contain 12 repetitions of the time-series M (represented as M1, M2,...M12). Hence, in this case, the Timeseries Y would have ID = 1 and ROOT\_ID = 1; Timeseries M1 would have ID = 2, PARENT\_ID = 1, and ROOT\_ID = 1; Timeseries W1 would have ID = 3, PARENT\_ID = 2, and ROOT\_ID = 1; and so on.

### 6.2.3 Timeseries Module

This module is responsible for storing the raw time-series values (time-value pairs). Depending on the source and type, time-series can be represented according to different data types. For example, a



**Figure 6.8:** Geometry hierarchy managed within the table `SURFACE_GEOMETRY` for a LoD1 solid geometry. Image taken from the 3DCityDB official documentation version 4.2 available at <https://www.3dcitydb.org/3dcitydb/documentation/>

time-series generated by a weather station for temperature recordings is a *double*, a time-series from a traffic camera for counting the number of cars at a junction is an *integer*, and another time-series retrieved from a moving GPS a *Geometry* (e.g. a Point).

In order to manage timeseries of different types, the 3DCityDB is extended by individual tables: `DYN_Ts_INT` (Timeseries Integer), `DYN_Ts_DOUBLE` (Timeseries Double), `DYN_Ts_STRING` (Timeseries String), `DYN_Ts_GEOM` (Timeseries Geometry), `DYN_Ts_URI` (Timeseries External Link), `DYN_Ts_BOOL`, (Timeseries Boolean), `DYN_Ts_APPEARANCE` (Timeseries Appearance), and `DYN_Ts_IMPLICITGEOM` (Timeseries Implicit Geometry). The tables `DYN_Ts_APPEARANCE` and `DYN_Ts_IMPLICITGEOM` include the attribute `VALUE`, which acts a Foreign Key to the 3DCityDB tables `APPEARANCE` and `IMPLICIT_GEOMETRY` respectively. Similarly, the table `DYN_Ts_GEOM` also includes an attribute `VALUE_BREP`, which acts a Foreign Key to the 3DCityDB table `SURFACE_GEOMETRY`. In this way, either attribute `VALUE` has a non-B-Rep geometry or `VALUE_BREP` has a reference to the `SURFACE_GEOMETRY` table in each row of the table. What is not shown in figure 6.7 (due to limited page-width) are the complete representations of the 3DCityDB tables `IMPLICIT_GEOMETRY`, `SURFACE_GEOMETRY`, and `APPEARANCE`. The table `APPEARANCE`

in the 3DCityDB contains attributes for managing the information about the appearances. Since each city model or city object may store its own appearance data, this table is put in relation to the base classes *CityObject* and *CityModel* by two foreign keys which may be used alternatively. An appearance is composed of data for each surface geometry object. Information on the data types and its appearance are stored in the table SURFACE\_DATA. Similarly, attributes for mapping textures to objects (point list or transformation matrix) which are defined by the CityGML classes *\_TextureParameterization*, *TexCoordList*, and *TexCoordGen* are stored in the table TEXTUREPARAM. Apart from appearances, in the database schema, the geometry consists of planar surfaces which correspond each to one entry in the table SURFACE\_GEOMETRY. The surface-based geometry is stored as attribute GEOMETRY. The implicit geometry is stored as attribute IMPLICIT\_GEOMETRY. The volumetric geometry is stored as attribute SOLID\_GEOMETRY and its boundary surfaces (outer shell) are stored as attribute GEOMETRY as well. More details on these specific tables are given in the 3DCityDB official documentation version 4.2 available at <https://www.3dcitydb.org/3dcitydb/documentation/>.

### 6.3 Import and Export of Dynamizer ADE data to/from the 3DCityDB

Once the relational database model is developed, the next step is to extend the Import and Export functionality of the 3DCityDB to map CityGML documents with Dynamizer ADE contents onto the appropriate tables. The CityGML Dynamizer ADE files can be imported to the 3DCityDB by following four major steps.

1. Mapping the XML Schema definition of the ADE to a relational schema that integrates with the 3DCityDB core schema
2. Creating an XML-based schema mapping file that captures the mapping between elements of the XML schema and elements of the relational schema
3. Registering the ADE with the metadata tables of the 3DCityDB
4. Implementing specific import/export stubs in Java for the Dynamizer ADE elements

The ADE Plugin Manager automates these steps. It reads the XML schema and applies a rule-based transformation to derive a relational schema for the ADE that seamlessly integrates with the 3DCityDB. In other words, the ADE Plugin Manager automatically creates the tables and joins based on the classes and their relations defined in the GML application schema/XSD files. Users can redefine default rules or even add new rules, and thus have full control over the mapping result. However, as mentioned in the previous section, to improve querying efficiency, the relational database model of the Dynamizer ADE involves only three independent modules: Dynamizer Core Module, Timeseries Metadata Module and Timeseries Module. This approach gives the flexibility to re-use existing time-series modules with other ADEs such as Energy ADE and UtilityNetwork ADE. Hence, the structure of the relational database model is different from the UML model. For this reason, it is proposed to include the Timeseries and Metadata modules as an integrated part of 3DCityDB. However, in the future, the ADE Plugin Manager will be extended to map the Dynamizer UML model in such a way that the Dynamizer core attributes are mapped onto the Dynamizer core table, and the associated time-series are mapped onto the Timeseries and Metadata modules.

Once the ADE is registered with the 3DCityDB by performing the above steps, the 3DCityDB Importer/Exporter tool requires extensions to (i) import time-series data from Dynamizer ADE to the new Dynamizer ADE tables, and (ii) export time-series data from Dynamizer ADE tables to the CityGML documents. For this purpose, the 3DCityDB provides the Importer/Exporter tool. Since the Importer/Exporter does not provide generic ADE support yet, the Dynamizer ADE extension



is required to be developed against the ADE API of the Importer/Exporter. The Dynamizer ADE extension can be developed by performing the following steps:

1. Creating an ADE module for citygml4j for parsing and writing CityGML with Dynamizer ADE.
2. Implementing the ADEExtension interface of the ADE API and providing code for reading and writing data into the ADE tables.

By performing these steps, the functionalities of the Importer/Exporter can be extended for importing and exporting the time-series data from the CityGML Dynamizer ADE. The implementation of this code was beyond the research frame of this thesis and these steps are mentioned here as a starting point for a future implementation.

## 6.4 Discussions

In Chapter 6, it was shown how to extend the 3DCityDB for managing the Versioning as well as the Dynamizer ADE. The management of the Versioning schema can be done with the ADE Plugin Manager, which allows dynamically extending the relational database schema for managing versions and version transitions. Since *Version* and *VersionTransitions* are defined as *FeatureTypes*, the WFS can retrieve each version and version transition, which can be accessed and visualised by CityGML applications. Unlike the Versioning schema, the management of the Dynamizer ADE is not straightforward. The ADE Plugin manager can extend the relational database schema for managing Dynamizer features. However, Dynamizers may represent time-series, and it requires to store, manage, and perform queries based on the time-value pairs. Hence, the relational database model has been extended for supporting time-series, including metadata and raw time-value pairs. In this way, the extended relational database schema allows performing queries on time-series data. However, as shown in figure 6.4 and figure 6.5, the WFS is not capable of querying time-series values stored in the 3DCityDB. Furthermore, a Dynamizer can also contain links to external sensor APIs, which may be diverse and heterogeneous. It is an issue of how CityGML applications can interpret such diverse and heterogeneous sensor observations in unified ways. The solutions for this issue are discussed in Chapter 7.



## Chapter 7

---

### Management of Dynamic City Models on the level of SDIs

This chapter discusses how time-dependent properties of city objects can be retrieved along with the static properties using web services. For this purpose, the chapter presents the concept of Spatial Data Infrastructures (SDI) and describes ways to access time-series data using open and international standards. Further, the chapter introduces a new concept called InterSensor Service allowing to retrieve time-series data from CityGML Dynamizers and translates them "on-the-fly" according to the international standards. Besides, the service can also retrieve time-series data from arbitrary sensor and IoT platforms, databases, and external tabulated files and perform such translations. In this way, time-series data from heterogeneous data sources can be retrieved and accessed in a unified way.

Some of the discussions in this chapter have been presented in the following published papers:

**Chaturvedi, K.** and Kolbe, T. H. (2019). 'Towards Establishing Cross-Platform Interoperability for Sensors in Smart Cities'. In: *Sensors* 19.3. URL: <https://www.mdpi.com/1424-8220/19/3/562>

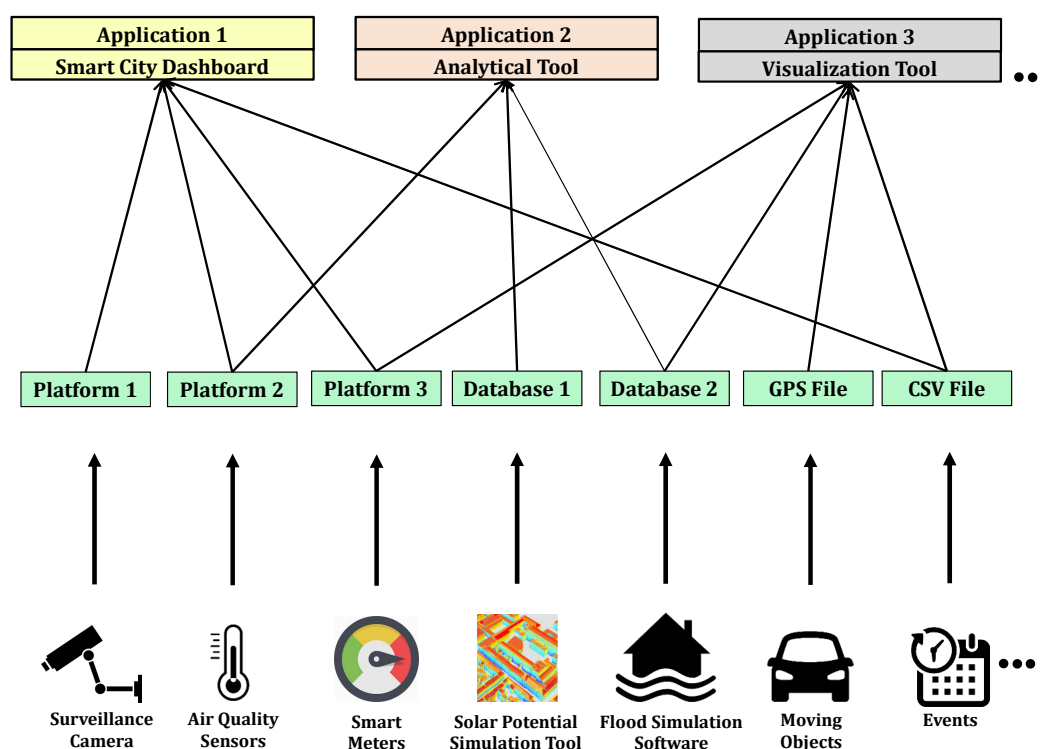
**Chaturvedi, K.** and Kolbe, T. H. (2018). 'InterSensor Service: Establishing Interoperability over Heterogeneous Sensor Observations and Platforms for Smart Cities'. In: *2018 IEEE International Smart Cities Conference (ISC2)*, pp. 1–8. URL: <https://doi.org/10.1109/ISC2.2018.8656984>

## 7.1 Spatial Data Infrastructures (SDI)

CityGML Dynamizers allow representing time-series data in-line with city objects as well as enable establishing direct links to heterogeneous sensor and IoT platforms and APIs. Although the proposed extensions to database management systems such as 3DCityDB allow managing such dynamic information, access and retrieval of such dynamic information is still a significant issue. As described in the previous chapter, the OGC Web Feature Service can be used to request static information of city objects; however, it is not suitable to query time/value pairs. As we have already learned before, city objects can be linked to multiple sensors and IoT platforms depending on the use cases. In most scenarios, these sensors and IoT platforms belong to various stakeholders and companies. These stakeholders usually are interested in specific applications or simulations and collect data for their purposes. For example, an energy provider company participating in a project owns the energy consumption data for buildings. In general scenarios, this data is meant to be used with the application owned by the same energy provider company. Most often, the structure of the data is not standardised and lacks explicit semantics. Hence, its structure might be different from other datasets, making it difficult to interpret by common applications. It is typically also the case for sensor and IoT platforms being used in such projects. In most scenarios, stakeholders use their sensors which are built for specific purposes and are based on particular platforms. These platforms may be open or proprietary; however, most of the time, they are not standardised. Another challenge is that the APIs associated with these platforms are subsequently changed often without notifying the users. Moreover, the observations retrieved from these sensors are not always associated with an API. In many scenarios, such time-series observations are the results of simulations (Willenborg et al. 2017) which are stored in databases or even simple files. It leads to a significant challenge to work in unified ways with a wide variety of data sources and their data types which are entirely different from each other.

It shows that such distributed systems are complex involving multiple stakeholders, diverse applications, a multitude of sensor and IoT platforms and data sources. It is imperative to achieve a proper data integration strategy for making well-informed decisions. Such integration strategies must allow working with heterogeneous data sources and platforms in a common operational framework. In this way, joint analytics can be performed to manage aspects of how a city functions and is managed, e.g. by using smart city dashboards (Kitchin 2014) as shown in Figure 7.1. However, as highlighted by (Moshrefzadeh et al. 2017), due to data privacy concerns and competition between several stakeholders, it does not make sense to try to collect all available data resources within a central data repository. Instead, the data should remain with their owners and should be combined flexibly according to specific applications or stakeholders.

Spatial Data Infrastructures (SDI) (Aalders and Moellering 2001) play an essential role in linking and integrating various distributed data and systems. SDIs facilitate the discovery, access, management, distribution and reuse of digital geospatial resources. In general, SDIs establish service-oriented architectures (SOA), allowing unified access to distributed resources using well-defined web services and interfaces. Such service-oriented SDIs are essential for distributed systems. They allow the data to remain with their respective owners and stakeholders and to be accessed by applications and users via well-defined interfaces. However, it requires interoperability over the connected components and systems to deal with the different types of data and systems. Interoperability can be achieved by using open and international standards. These standards, on the one hand, allow modelling and representing the data sources and, on the other hand, provide interfacing with the distributed components that give access to data, visualisations, and analytical tools.

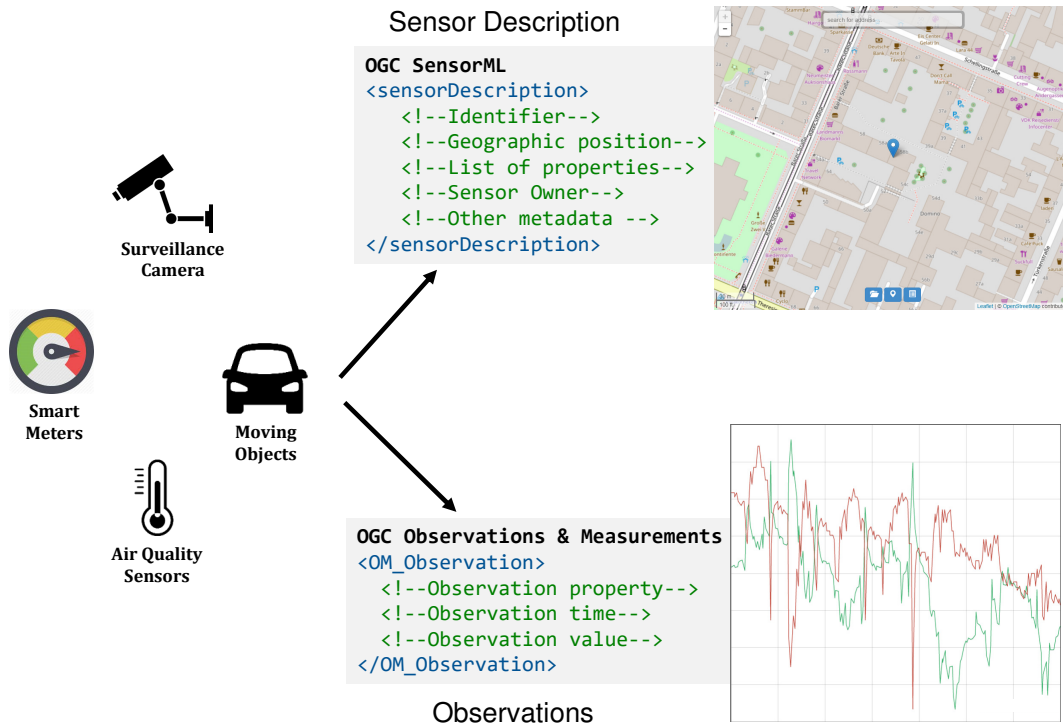


**Figure 7.1:** Illustration of heterogeneous data sources for sensor and time-series data. The requirement is to integrate and use them in a unified way over different applications.

## 7.2 Establishing cross-platform interoperability for sensor and time-series data

The interoperability over the heterogeneous sensor and time-series data can be achieved using international standards. As discussed in section 2.2.1, there are many standard frameworks like OGC Sensor Web Enablement (SWE) (Bröring et al. 2011), FIWARE (FIWARE 2018), BIG-IoT (Bröring et al. 2017), VICINITY (Mynzhasova et al. 2017), and SenML (Jennings et al. 2018). Since this thesis is based on the OGC CityGML standard, it discusses the approaches for establishing cross-platform sensor interoperability using the OGC Sensor Web Enablement (SWE) standard suite. However, interoperability using other mentioned initiatives can also be achieved in similar ways.

As already described before, the OGC SWE standards suite comprises well-defined information models such as (i) SensorML (Botts 2014), which not only represents sensor description and metadata, but also sensor calibration records and accuracy and precision information, and (ii) Observations and Measurements (O&M) (Cox 2013) for describing real-time sensor observations. The SWE also provides comprehensive interface models and web services such as the Sensor Observation Service (SOS) (Bröring et al. 2012) and the SensorThings API (Liang et al. 2015) for retrieval of sensor descriptions and observations with the help of standardised requests. In comparison to SOS, SensorThings API is a relatively new standard, which is REST-ful, lightweight, and based on JSON.



**Figure 7.2:** Interoperability of sensor and time-series data using the OGC Sensor Web Enablement standard suite.

Such sensor web infrastructures play an essential role in establishing interoperability for heterogeneous sensors. They allow encoding sensor description and observations using well-defined standards as well as accessing them using standardised interfaces. In this way, applications and tools can be developed based on these standards without worrying about what different kinds of sensors and interfaces they use (Figure 7.2). Multiple sensors can be attached to these infrastructures, and their interfaces will always be common for different applications.

The following sections describe ways to achieve interoperability using OGC SWE standards.

### 7.2.1 OGC Sensor Observation Service

The Sensor Observation Service (SOS) is one of the oldest standards, which defines a web service interface for querying observations, sensor metadata, as well as representations of observed features. Further, this standard supports transactional operations allowing new sensors to be registered and the existing ones to be removed. Also, it defines operations to insert new sensor observations depending on the frequency of the property observed. The SOS standard defines these functionalities according to two bindings: a Key-Value-Pair (KVP) binding and a Simple Object Access Protocol (SOAP) binding.

The SOS standard structures all the operations in three major categories:

**Core Operations (core profile)**

- *GetCapabilities* returns the description of the service, which includes information about the interface and available sensor data. For example, the response includes the period for which sensor data is available, sensors that produce the measured values, and phenomena that are observed like humidity and temperature.
- *GetObservation* returns the observed values, along with their metadata. The response is encoded according to the Observations and Measurements format (O&M).
- *DescribeSensor* provides the sensor description. The response includes information about the identifier, its location and the list of phenomena the sensor observes. It also includes the details of calibration data. The response is encoded according to the SensorML standards.

**Transactional operations (transactional profile)**

- *RegisterSensor* allows to register a new sensor in a deployed SOS.
- *InsertObservation* can be used to insert data for already registered sensors in the SOS.

**Extended operations (enhanced profile)**

- *GetResult* provides the ability to query for sensor readings without the metadata given consistent metadata (e.g. sensor, observed object).
- *GetFeatureOfInterest* returns the geobject whose properties are monitored by sensors in Geography Markup Language encoding.
- *GetFeatureOfInterestTime* provides time periods in which measurements of an observed object in the SOS are available.
- *DescribeFeatureType* returns the type of the observed geobjects (XML Schema).
- *DescribeObservationType* returns the type of observation (XML Schema), such as om:Measurement).
- *GetObservationById* allows to query a specific observation using an identifier returned by the service as response to an *InsertObservation* operation.
- *DescribeResultModel* provides the XML Schema of the measured value, which is particularly important for complex measurements, such as multi-spectral data.

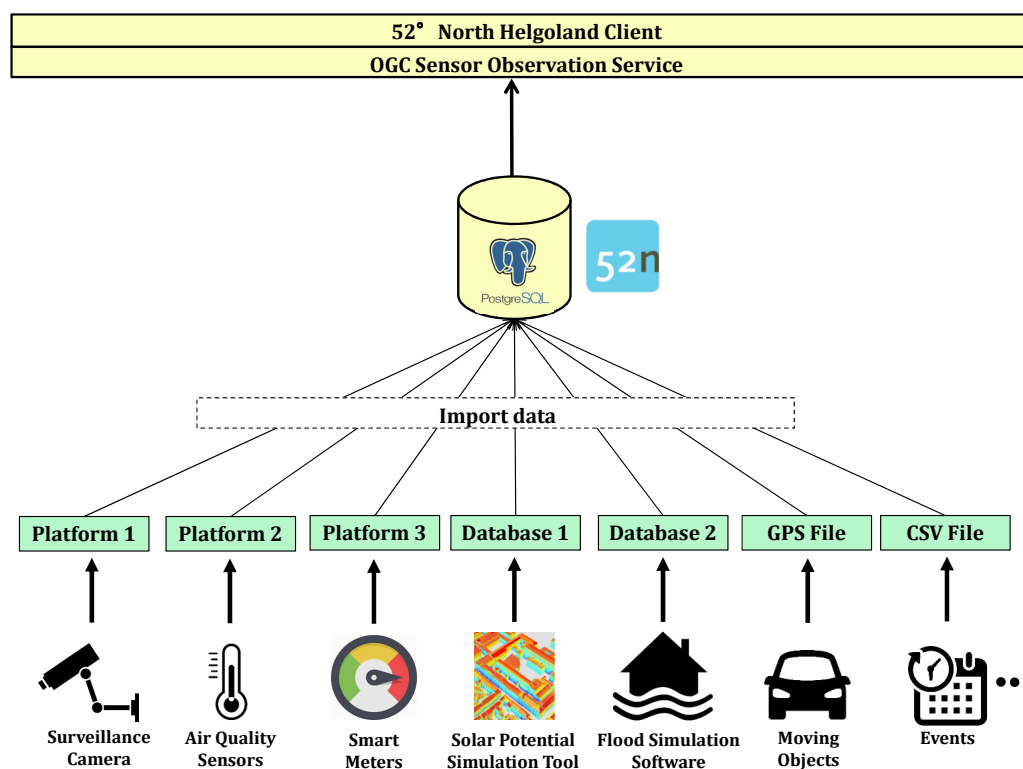
**7.2.1.1 Tools for supporting the OGC SOS standard**

52°North GmbH is a company based in Germany, which provides a complete SOS implementation. The implementation is free and available as Open Source software<sup>94</sup>. The software provides the following key components:

- *52°North SOS*, a Java-based reference implementation for the OGC SOS standard. It enables users to perform all the SOS operations in a standardised way. The implementation can very easily be deployed using a WAR file on a web application server.
- *Relational Data Model* allowing creating tables and relations for managing sensor and time-series data on top of standard relational database management systems like PostgreSQL/PostGIS, Oracle, and MySQL.
- *Timeseries API* provides a REST-ful web binding to the OGC Sensor Observation Service to be easily queried and visualised by lightweight web and mobile clients.

---

<sup>94</sup><https://github.com/52North/SOS>



**Figure 7.3:** Interoperability of sensor and time-series data using the OGC SOS standard.

- *Helgoland Client* is a lightweight web application to explore, analyse, and visualise running SOS instances via the Timeseries API. The client allows visualising sensor locations on a map as well as time-series graphs and charts for the sensor observations. It also supports temporal zooming and panning.

In a nutshell, the 52°North SOS implementation provides a complete suite of applications allowing to bring multiple sensors and their observations to a common operational framework. As shown in figure 7.3, heterogeneous sensor observations can be imported to a running SOS server. 52°North already provides a free tool called SOS Importer<sup>95</sup> for this purpose. It allows inserting sensor observations from CSV files available locally or remotely (e.g. provided via FTP/HTTP Server). Alternatively, the schedulers such as FME Server<sup>96</sup> and NodeRED<sup>97</sup> can also be used to prepare workflows. These workflows enable (i) retrieving the observations directly from a sensor platform, (ii) creating the appropriate *InsertObservation* requests, and (iii) inserts them into a running SOS server in regular intervals. Once the observations are stored in the database, the implementation provides an automated Timeseries API interface for visualising observations in unified ways on client applications like the Helgoland client.

<sup>95</sup><https://52north.org/software/software-projects/sos-importer/>

<sup>96</sup><https://www.safe.com/fme/fme-server/>

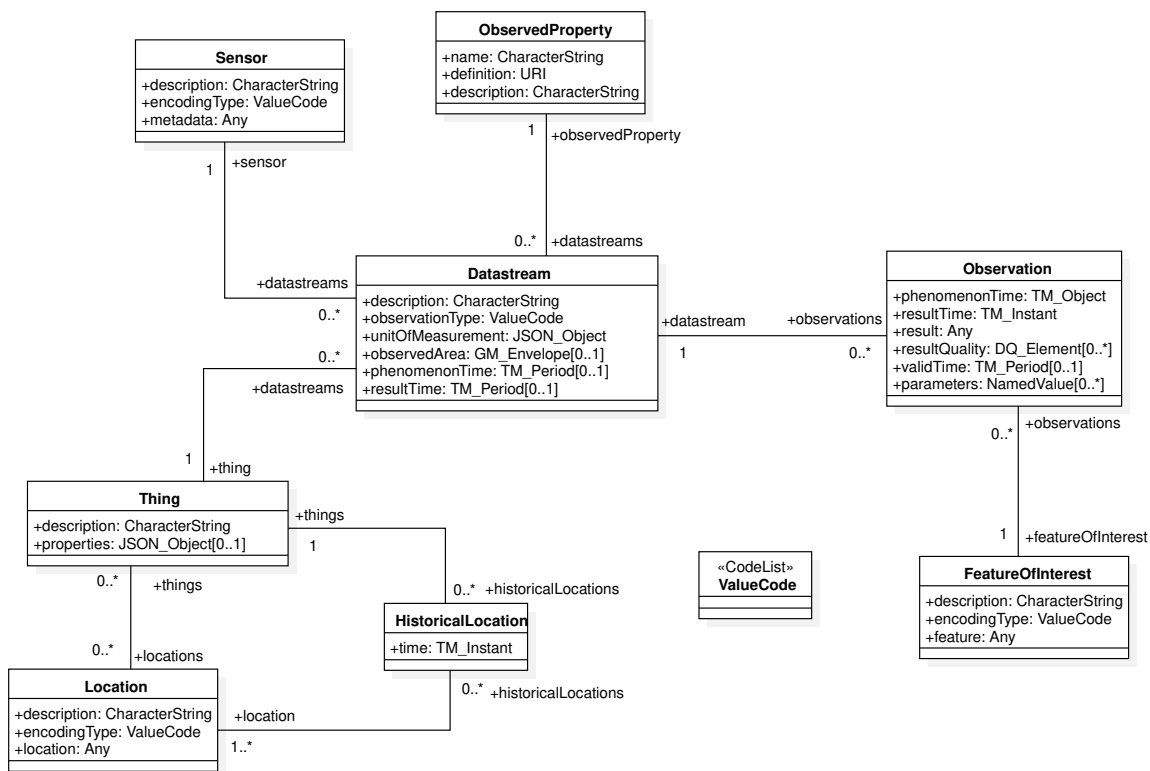
<sup>97</sup><https://nodered.org/>



## 7.2.2 OGC SensorThings API

OGC SensorThings API is a relatively new standard, which provides an open and unified framework to interconnect IoT sensing devices and their real-time observations over the web. The standard is lightweight, follows REST principles, the JSON encoding, and the OASIS OData protocol<sup>98</sup> and URL conventions. In addition to HTTP, it also has an MQTT extension allowing users/devices to publish and subscribe updates from devices.

The SensorThings API specification defines two parts for handling two main functionalities. The two profiles are (i) the Sensing part and (ii) the Tasking part. The Sensing part manages and retrieves observations and metadata from heterogeneous IoT sensor systems in a standardised way. Like the Sensor Observation Service, the SensorThings API is also based on the OGC Observations and Measurements (O&M) model. The Tasking part is used for parametrising - also called tasking - IoT devices, such as sensors or actuators.



**Figure 7.4:** Official UML Data Model of the OGC SensorThings API standard (Liang et al. 2015).

As shown in figure 7.4, SensorThings API defines several resources for retrieving different information. SensorThings is a RESTful web service; hence, each entity can be created, retrieved, deleted, and modified using different HTTP operations such as POST, GET, PATCH, and DELETE.

- *Thing*: A physical or a virtual object capable of being identified and integrated into communication networks.
- *Locations*: Locates the Thing or the Things it associated with.

<sup>98</sup><https://www.oasis-open.org/committees/odata/>

- *HistoricalLocations*: A set providing the current (i.e., last known) and previous locations of the Thing with their time.
- *Datastream*: A collection of observations measuring the same observed property produced by the same sensor.
- *ObservedProperty*: Specifies the phenomenon of an observation
- *Sensor*: A device that observes a property with the goal of producing an estimate of the value of the property.
- *Observation*: Act of measuring or otherwise determining the value of a property.
- *FeatureOfInterest*: An observation results in a value being assigned to a phenomenon. The phenomenon is a property of a feature, the latter being the feature of interest of the observation.

### 7.2.2.1 Tools for supporting the OGC SensorThings API standard

SensorThings API is widely gaining its popularity due to its lightweight nature. There are several implementations available, allowing users to install and work with the SensorThings API.

- SensorUp Inc.<sup>99</sup> provides a complete implementation of the SensorThings API. It makes information from all different kinds of sensors accessible in a single platform, by using open standards to connect the sensors. Although this implementation is not Open Source, SensorUp provides free deployment platforms for testing and demonstration of the API.
- FROST Server<sup>100</sup> is an Open Source server implementation of the OGC SensorThings API developed by the Fraunhofer Institut IOSB in Germany. It is a full implementation of the entire specification including the extensions for HTTP and MQTT protocols. The application is written in Java and can be easily deployed in Tomcat or Wildfly. The implementation is also available as a Docker image which can be deployed in the cloud environment.
- GOST<sup>101</sup> (Go-SensorThings) is an IoT platform written in Golang (Go). It implements the Sensing profile (Part 1) of the OGC SensorThings API, including the MQTT extension. It also provides an in-built dashboard application for visualising data streams in a real-time manner.
- Whiskers<sup>102</sup> is an OGC SensorThings API framework consisting of a JavaScript client and a lightweight server for IoT gateways (e.g., Raspberry Pi). Besides the client library, Whiskers, provides a SensorThings server module for IoT gateways, running e.g. on a Raspberry Pi. It enables developers to make several IoT gateways compliant according to the SensorThings API standard so that they can easily be connected with spatial data servers worldwide that implement the full array of OGC Sensor Web Enablement (SWE) standards.
- CGI Inc. provides a server implementation of the SensorThings API named Kinota Server<sup>103</sup>. Kinota Server mostly focuses on Big Data applications allowing to store sensor data from an arbitrary number of sensors collecting data up to every 500 milliseconds. Kinota allows a cloud-friendly architecture, which is horizontally scalable and provides flexibility to choose between cloud providers like Microsoft Azure, Amazon Web Service, and Google Cloud.

---

<sup>99</sup><https://sensorup.com/>

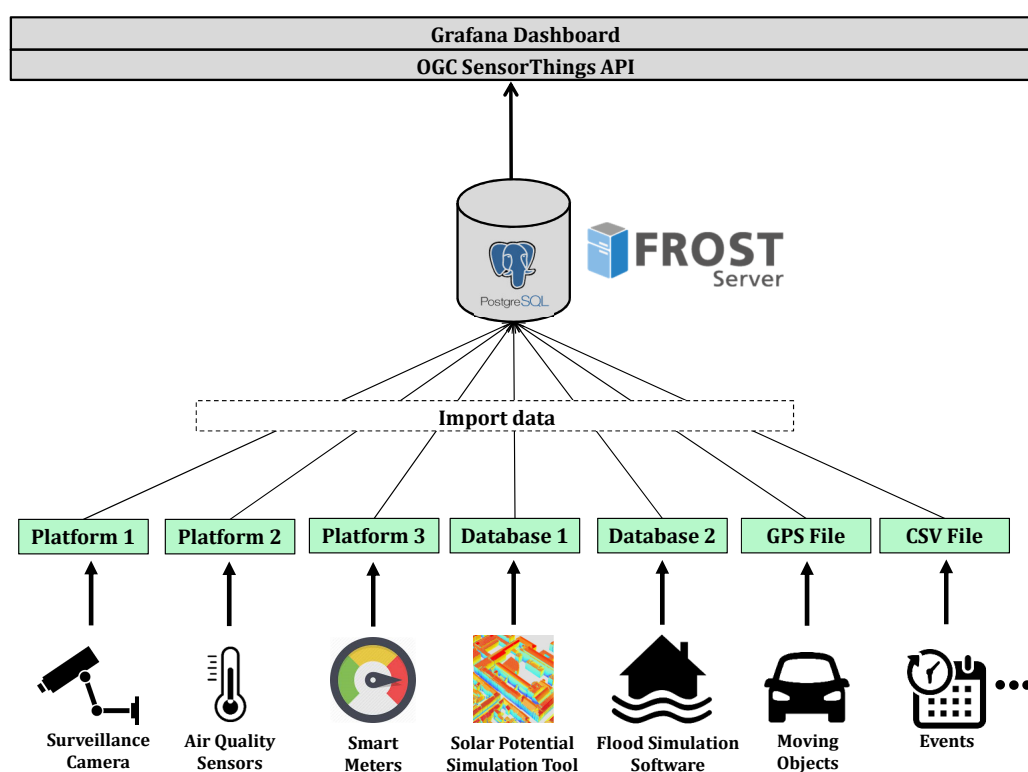
<sup>100</sup><https://github.com/FraunhoferIOSB/FROST-Server>

<sup>101</sup><https://www.gostserver.xyz/>

<sup>102</sup><https://github.com/eclipse-archived/whiskers.js>

<sup>103</sup><https://github.com/kinota/SensorThingsServer>

- SensorThings Dashboard <sup>104</sup> provides easy-to-use client-side visualisation of IoT sensor data retrieved using OGC SensorThings API requests. The dashboards can be created by arranging various types of widgets. It is a web application and can be embedded into any website.
- Grafana <sup>105</sup>, a widespread dashboard application, provides a plugin <sup>106</sup> enabling the visualisation and location of sensor data from running OGC SensorThings API servers.



**Figure 7.5:** Interoperability of sensor and time-series data using the OGC SensorThings API standard.

As shown in figure 7.5, heterogeneous sensor observations can be imported to a running SensorThings API server and can further be queried and visualised on common dashboard applications. There are several importer tools provided, for example, by Fraunhofer<sup>107</sup> and HfT Stuttgart<sup>108</sup>, allowing to insert sensor observations from CSV files. Alternatively, schedulers such as FME Server, NodeRED, or Mosquitto MQTT Broker can also be used to set up workflows. Once the observations are stored in the database, they can be retrieved and visualised on dashboard applications like Grafana. The complete description of this workflow is provided in a GitHub repository <https://github.com/tum-gis/iot-frost-ecosystem> prepared by the author of this thesis. This workflow

<sup>104</sup> <https://github.com/SensorThings-Dashboard/SensorThings-Dashboard>

<sup>105</sup> <https://grafana.com/>

<sup>106</sup> <https://grafana.com/grafana/plugins/linksmart-sensorthings-datasource>

<sup>107</sup> <https://github.com/FraunhoferIOSB/SensorThingsImporter>

<sup>108</sup> <https://github.com/JoeThunyathep/SensorThings-Importer>

is already being used in several projects such as Digital Twin Munich <sup>109</sup>, Smart District Data Infrastructure <sup>110</sup>, and the Digital Twin of the Agricultural Landscape (Moshrefzadeh et al. 2020).

### 7.2.3 Further recommendations on working with the OGC SWE standards

Both SOS and SensorThings APIs cover a wide range of applications and are successfully used in many industries and applications. New sensors can be attached to the implementations and observations can be visualised in simple ways using common applications and dashboards. However, while working with distributed smart city applications, there are a few observations:

- The implementations of all of the approaches (e.g. 52°North SOS Server and FROST Server) always require a data storage (e.g. a database repository) for storing sensor description and their observations. The interfaces and web services can query and retrieve sensor data and observations from such data storages. In a distributed environment, multiple stakeholders and sensor owners are involved with proprietary sensors. However, in some cases, not all of the stakeholders would be willing to inject their proprietary data into such third-party data storages in the sensor web. One such example of a proprietary Smart Meter platform is shown in section 8.2. Moreover, the need of always having another data storage for the sensor web will require regular maintenance of the storage infrastructure. It will also increase complexity while moving the infrastructure to different locations, for example, from one server to another or into the cloud.
- The implementations require importing the observations from the original platform and storing them in their data storages. The issue with such an approach is that it leads to data redundancy. The respective platform, such as ThingSpeak already stores observations in its own data storage. Another challenge is that in some cases, these platforms may also be proprietary. In this case, the owners would like to avoid storing their proprietary data in third-party data storage.
- Currently, there is no Sensor Web Enablement implementation which allows retrieving time-series data directly from external files such as CSV files and CityGML Dynamizer files. The 52°North SOS Implementation and FROST Server support importing the time-series data from a CSV file; however, the data is first imported to their data storage which again leads to the above two issues.

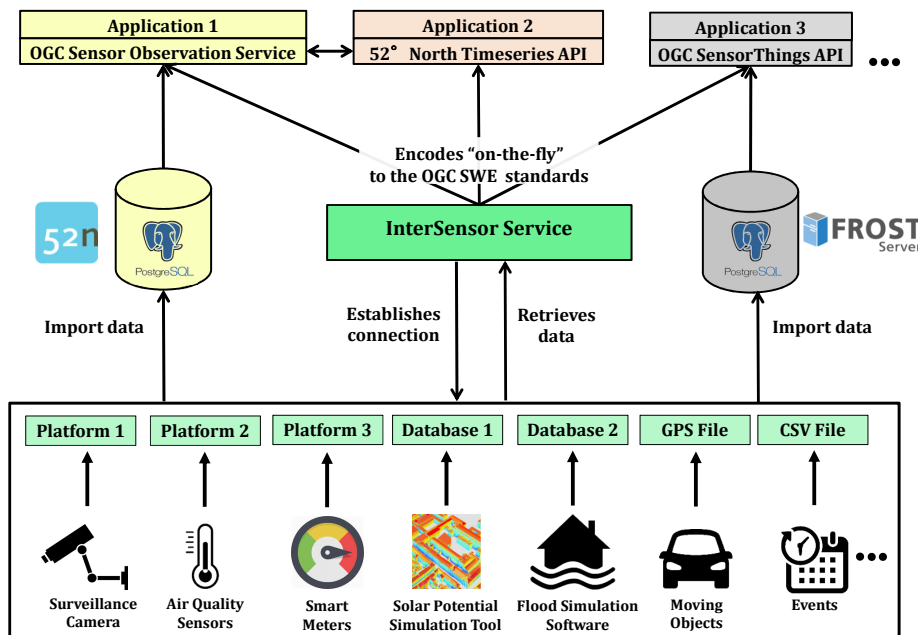
In such cases, it is essential to have an intermediate service which can connect to a specific data source and encodes the observations "on-the-fly" according to the standardised OGC SWE interfaces without worrying about the data storage and multitude of data sources (see Figure 7.6). In other words, this intermediate service should be like a "Babel Fish" from The Hitchhiker's Guide to the Galaxy (Adams 1979) which is a *"universal translator that neatly crosses the language divide between any species"*.

## 7.3 Introduction to the InterSensor Service

This thesis provides solutions for the issues named in section 7.2.3 by introducing the lightweight InterSensor Service. This service offers several data adapters for establishing connections to different data sources such as IoT platforms, external databases, CSV files, Cloud-based spreadsheets, GPS feeds, and real-time Twitter feeds. Once the connection is established, the service allows users to retrieve the observations directly from the data source based on query parameters. Furthermore, the observations are encoded "on-the-fly" according to the international standardised interfaces like the OGC Sensor Observation Service and OGC SensorThings API. In this way, applications compliant to

<sup>109</sup><https://muenchen.digital/twin/>

<sup>110</sup><https://www.lrg.tum.de/en/gis/projects/smart-district-data-infrastructure/>



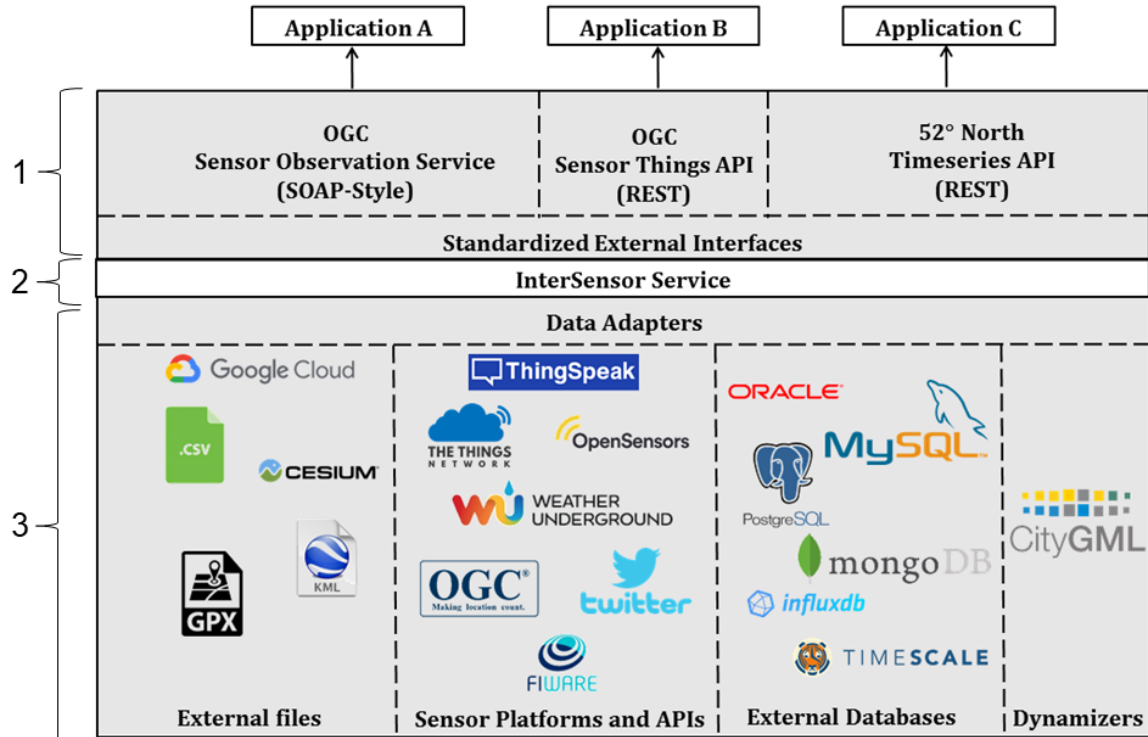
**Figure 7.6:** Motivation of developing the InterSensor Service. The implementations of OGC Sensor Observation Service and SensorThings API require importing the observations to their respective data storages. This thesis introduces a lightweight intermediate service named InterSensor Service (shown in dark green) allowing connecting to the respective platform, retrieving observations and encoding them "on-the-fly" according to the OGC SWE standardised interfaces.

such OGC standardised interfaces can interact with these heterogeneous observations without worrying about their data storage. The main reasons for initial development for the responses according to the OGC SWE interfaces are as follows. First, the OGC SWE framework is completely based on released and published Open Standards adopted internationally. When implementing something that is not standardised, there is a high risk that the developed encodings or APIs will be abandoned, replaced, or vanish after the project is over. Second, in distributed infrastructures, a lot of other data and presentation services such as web maps, 3D visualisations, data with geographic coverages like weather data, air quality, wind fields etc. are provided by Spatial Data Infrastructures (SDIs). All of these services are also offered using OGC standards. Hence, sensor and IoT based interfaces add another category of web service to SDIs, and it is beneficial to make the IoT service compliant to SDIs such that they can be used with similar protocols and tools already used in the framework of SDIs. Third, it also makes the observations suitable to be visualised and managed with the other numerous OGC geospatial standards such as CityGML (Gröger et al. 2012). Also, no other implementation yet provides such "on-the-fly" interfaces for international OGC SWE standards. However, the concept is not limited to only the OGC standards. In the future, interfaces can also be developed according to other standards/protocols such as FIWARE. The InterSensor Service is a Java-based application and is available for free as Open Source software<sup>111</sup>.

<sup>111</sup><http://www.intersensorservice.org/>

### 7.3.1 Architecture

As shown in Figure 7.7, the architecture comprises of three layers:



**Figure 7.7:** The three-layer architecture of the InterSensor Service. The service can be instantiated for individual data sources using adapters and provides standardised external interfaces.

#### 7.3.1.1 Data Adapters

The Data Adapter layer establishes the connection to multiple data sources. The data sources can be the existing sensor and IoT platforms such as ThingSpeak, OpenSensors, The Things Network, and OGC SWE standards. Users can also connect to running databases such as Oracle, PostgreSQL, and TimescaleDB. Similarly, time-series data associated with any external files can also be used for such connections. These external files may be CSV and Excel sheets, GPX file (or GPS feeds embedded in a KML or CZML file), Google Fusion Tables, and CityGML Dynamizer files. These files can be located on a local machine, remote server, or cloud environments.

#### 7.3.1.2 Standardised External Interfaces

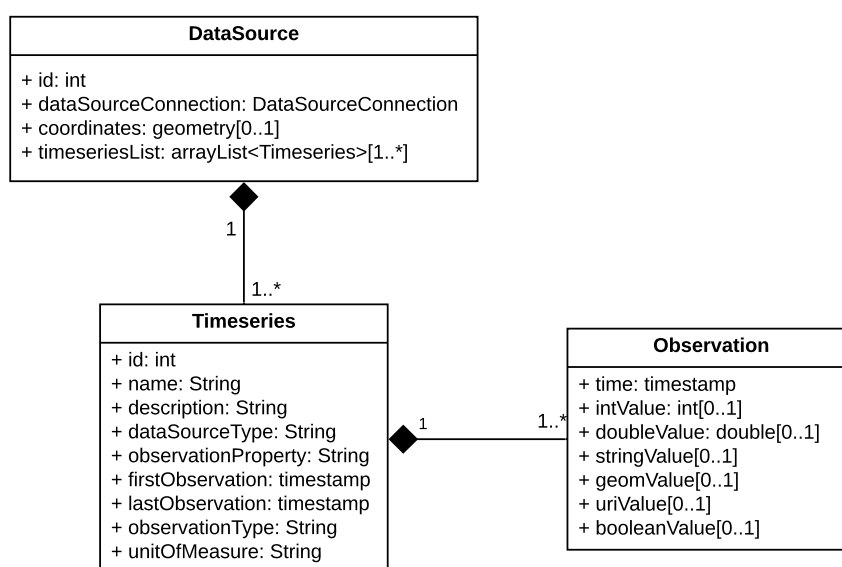
This layer encodes the queried observations according to well-defined interfaces such as the OGC Sensor Observation Service (SOS) and the SensorThings API. As we learned before, the SOS interface includes several operations for querying sensor-related data, e.g. *DescribeSensor* to retrieve sensor metadata according to the SensorML standard and *GetObservation* to query sensor observations according to the O&M format. This layer of the InterSensor Service encodes the queried data

"on-the-fly" according to the responses of *DescribeSensor* and *GetObservation* requests. In this way, an application compliant to the SOS standard can interpret and visualise the data queried by the InterSensor Service. Another interface provided by the InterSensor Service is the Timeseries API (TimeseriesAPI 2018), which is not an international standard but was specified by 52°North providing a RESTful web binding to the SOS. Using the Timeseries API interface, the sensor data and observations retrieved using the InterSensor Service can be queried and visualised over the so-called Helgoland web client (Helgoland 2018). Similarly, the InterSensor service also allows observations to be encoded and queried according to the SensorThings API interface. In this way, all applications compliant to the OGC SensorThings API (such as Grafana Dashboard) can be used for visualising the data obtained using the InterSensor Service. The illustrations of each of the interfaces are shown in section 7.4.

### 7.3.1.3 InterSensor Service

This is an intermediate layer acting as a "Babel Fish" between the data sources and the interfaces. This middle layer allows establishing connections to the individual data sources using the specified data adapters. Once the connection is successfully established, the service maps the data according to the resources defined in the data model (c.f. section 7.3.2). Furthermore, multiple interfaces can read the observations from this layer and encodes the data according to the desired interface. In this way, the InterSensor Service can query observations from heterogeneous and distributed data sources and map them using common and simple objects. At the same time, the service encodes observations using standardised interfaces to analyse and visualise them together in a unified way.

## 7.3.2 Data Model

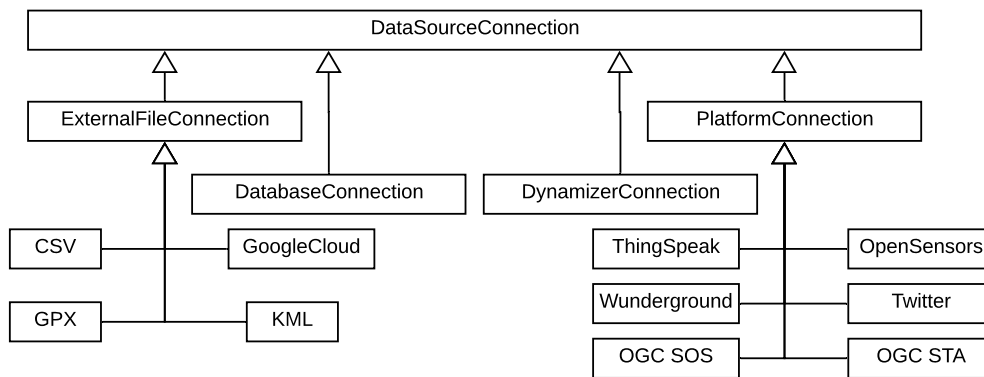


**Figure 7.8:** Key resources of the InterSensor Service.

The InterSensor Service includes distinct classes to connect to the individual data sources. These classes contain specific attributes which can be used to connect to a particular data source. After successful connection to the data sources, the InterSensor Service forms three resources named *DataSource*, *Timeseries*, and *Observation* as shown in Figure 7.8. *DataSource* contains all the details of a specific data source whose link can be established using *DataSourceConnection*. The details of each class are mentioned as follows:

### 7.3.2.1 DataSourceConnection

As shown in Figure 7.9, this class allows users to specify parameters to connect to individual data sources. It contains metadata attributes such as name and description of the data source, what type of connection it is (e.g., a CSV file, JDBC connection, a web service, etc.). Further, it contains subclasses to connect to different resources. *ExternalFilesConnection* provides connection details to external files such as CSV, GPX, KML and CZML, and also to Cloud-based documents such as a Google Fusion Table. *DatabaseConnection* contains parameters to connect to a specific database. In similar ways, the InterSensor Service can also be used to connect to CityGML Dynamizer datasets using *DynamizerConnection*.



**Figure 7.9:** Representation of types of data sources which can be used by the InterSensor Service.

*PlatformConnection* is designed for connecting to different sensor and IoT platforms. This class has further subclasses for each platform, for example, *ThingSpeak*, *OpenSensors*, *OGC SensorThings*, *OGC Sensor Observation Service*, and *Twitter*. Each subclass contains specific properties for the connection to be established. For example, in case of the SensorThings API, *ThingId* is a unique ID to determine the details and metadata of a “Thing” (e.g., a weather station) such as [https://example.sensorup.com/v1.0/Things\(8774755\)](https://example.sensorup.com/v1.0/Things(8774755)). One “Thing” can deliver different observations (e.g. temperature, humidity etc.). Each observation can be determined by a *DatastreamId* such as [https://example.sensorup.com/v1.0/Datastreams\(8774757\)](https://example.sensorup.com/v1.0/Datastreams(8774757)).

Hence, in order to add a timeseries property from the above-mentioned data stream from the SensorThings API, the minimal inputs required are: a *baseURL* such as <https://example.sensorup.com/v1.0>, *ThingId* such as 8774755 and *DatastreamId* such as 8774757. Similarly, a valid ThingSpeak channel consists of a *baseURL*, a *channelID*, and *fieldID*. By providing these details, the InterSensor Service generates valid request calls and establishes a connection to the ThingSpeak channel.



### 7.3.2.2 DataSource

After configuring the data source connection details, the InterSensor Service validates the connection and instantiates three resources. *DataSource* creates a unique ID for the data source and contains the details of *DataSourceConnection*. It also contains a list of available time-series associated with it.

### 7.3.2.3 Timeseries

Each *DataSource* can have multiple *Timeseries*. For example, if a data source is a ThingSpeak channel with two time-series associated with it: (i) temperature and (ii) humidity. In this case, the InterSensor Service creates two time-series (one for temperature and the other for humidity) with two unique time-series IDs associated with a common data source ID. However, depending on the requirements, it is also possible to establish a connection to a specific time-series from a data source connection (for example, only to the temperature stream on the ThingSpeak channel).

### 7.3.2.4 Observations

Both *DataSource* and *Timeseries* classes contain properties to connect to the data source. By providing query filters such as a period between two timestamps, the InterSensor Service establishes the connection to the data source, retrieves observations depending on the filter and maps them using the *Observation* class. It means that for a single query, relevant observation objects are dynamically created without storing them in a local data storage. This functionality allows encoding all kinds of observations in a common way irrespective of the type of the data source. Such common representations of the observations can be used further by the individual interfaces allowing joint analysis and visualisations.

The observations from a sensor can be of different data types depending on the sensor type, scenario, use case, and application. For example, a temperature observation is a number while a single observation from a GPS feed is a location. As shown in Figure 7.8, the *Observations* class allows encoding observations with different data types and hence providing flexibility to users to encode many possible kinds of observations.

## 7.4 Illustration of the concept

The InterSensor Service is a Java application based on the Spring framework<sup>112</sup>. It has been released as Open Source software, and is available at <https://github.com/tum-gis/InterSensorService>. It includes distinct classes for each of the mentioned resources. The service can be installed easily as a standalone application using JAVA JAR commands and can also be deployed on a running server using WAR files.

### 7.4.1 Adding a data source

The deployment of an instance of the InterSensor service requires establishing a data source connection as a first step. The connection details can be provided in a configuration file. These configuration files allow defining all the required parameters to connect to a specific data source. For example,

---

<sup>112</sup><https://spring.io/>

one publicly available ThingSpeak channel is <https://thingspeak.com/channels/64242>, which can be connected to the InterSensor Service as shown in listing 7.1.

**Listing 7.1:** Example of configuring the data source connection to a ThingSpeak channel

```
{
  datasource-connection:
    name: "Thingspeak_Humidity_Sensor"
    description: "New thingspeak connection recording Humidity"
    connectionType: "Thingspeak"
    observationProperty: "Humidity_Thingspeak"
    observationType: "OM_Measurement"
    unitOfMeasure: "Percent"
    serviceName: "Thingspeak"
    serviceType: "JSON"
    channel: 64242
    field: 2
}
```

It shows a DHT22 sensor located in Munich, Germany and comprises two observation properties: Field 1 (Temperature) and Field 2 (Humidity). The configuration mentioned above allows adding a specific property (e.g. Field 2 - Humidity) from the Thingspeak channel (with the id 64242) to the InterSensor Service.

**Listing 7.2:** Example of configuring the data source connection to a Twitter channel

```
{
  datasource-connection:
    name: "TwitterConnection"
    description: "Geo-Tagged Tweets around a location"
    connectionType: "Twitter"
    observationType: "JsonString"
    unitOfMeasure: "Tweet"
    serviceName: "Twitter API"
    serviceType: "JSON"
    baseUrl: "https://api.twitter.com/1.1/search/tweets.json"
    apiKey: "*****"
    apiSecret: "*****"
    accessToken: "*****"
    accessTokenSecret: "*****"
    latitude: 51.54347 #Location of a point
    longitude: -0.01652 #Location of a point
    radius: 1 #Radius in km
}
```

Some of the data sources may also require authentication parameters such as username/passwords or an OAuth 2.0 access token. The listing 7.2 shows an example of a connection to the Twitter API, which requires authentication parameters such as *apiKey*, *apiSecret*, *accessToken*, and *accessTokenSecret* in order to retrieve the tweets. The Twitter API supports querying geotagged tweets using the *geocode* parameter. It requires a point location (latitude, longitude) and a radius (e.g. 1 km) around that point. Additionally, even a search keyword can also be provided; however, it can be left blank for retrieving all the tweets. Such parameters can directly be provided in the configuration files.

**Listing 7.3:** Example of configuring the data source connection to a Dynamizer stored in 3DCityDB

```
{
  datasource-connection:
    name: "DynamizerConnection"
    description: "Connection to a Dynamizer AtomicTimeseries"
    connectionType: "Dynamizer"
    databaseType: "PostgreSQL"
    ipAddress: "127.0.0.1"
    port: 5432
    databaseName: "3DCityDB"
    username: "user"
    password: "*****"
    dynamizerId: "building_01_dyn_01"
}
```

In similar ways, the InterSensor Service can also be used to connect to the CityGML Dynamizer time-series stored in a 3DCityDB. Assuming that the 3DCityDB is installed on top of PostgreSQL, listing 7.3 shows the configuration parameters to connect to the Dynamizer time-series. Likewise, connections to arbitrary data sources such as external databases, different IoT platforms (for example, OpenSensors, Weather Underground, SensorThings API) and various file systems can also be established in more straightforward ways using the pre-defined configuration files. There might be scenarios (c.f. section 8.2), where the time-series data is stored in simple tabulated files such as CSV. In these cases, the configuration details can be specified accordingly by providing the file path, and the columns for timestamps and their respective values. Additionally, the information can also be given for other metadata such as the unit of measurement being used and geo-location of the sensor device.

Alternatively, new data sources can be added to a running instance of the InterSensor Service using an HTTP POST request. Such requests can be performed with the help of any REST client, cURL commands or using software systems such as "HTTP Caller" from the ETL software Feature Manipulation Engine (FME) being very popular in the geospatial domain.

## 7.4.2 Automated generation of the standardised interfaces

Upon successful establishment of the connection to a data source, the InterSensor Service generates instances of the three primary classes *DataSource*, *Timeseries*, and *Observation*. These three classes act as an intermediate layer to connect to a data source, retrieve observations and encode observations "on-the-fly" according to the standardised interfaces OGC SensorThings API and OGC Sensor Observation

Service, and the open-source Timeseries API. Upon a successful connection, the interfaces for the standards as mentioned above with appropriate classes are automatically generated.

Assuming the server hostname is 127.0.0.1, and the port is 8080, the three classes can be accessed and queried with the help of the following HTTP GET requests:

**Listing 7.4:** Illustration of the InterSensor Service resource endpoints generated for each data source connection

```
#Base URL
http://127.0.0.1/inter-sensor-service/

#Accessing DataSource details
http://127.0.0.1/inter-sensor-service/datasources/{id}

#Accessing Timeseries metadata
http://127.0.0.1/inter-sensor-service/timeseries/{id}

#Accessing Observations
http://127.0.0.1/inter-sensor-service/timeseries/{id}/observations
```

#### 7.4.2.1 OGC SensorThings API

The SensorThings API includes a well-defined data model (figure 7.4) with different resources such as *Thing*, *Locations*, *Datastream* etc. The InterSensor Service translates the connected data source details according to the SensorThings API data model, which can simply be accessed as follows:

**Listing 7.5:** Illustration of the OGC SensorThings API endpoints automatically generated for each data source connection

```
#Base URL for the SensorThings API standard
http://127.0.0.1/OGCSensorThingsApi/v1.0
#Thing
http://127.0.0.1/OGCSensorThingsApi/v1.0/Things
#Location
http://127.0.0.1/OGCSensorThingsApi/v1.0/Locations
#HistoricalLocation
http://127.0.0.1/OGCSensorThingsApi/v1.0/HistoricalLocations
#Datastream
http://127.0.0.1/OGCSensorThingsApi/v1.0/Datastreams
#Sensor
http://127.0.0.1/OGCSensorThingsApi/v1.0/Sensors
```

### 7.4.2.2 OGC Sensor Observation Service

In similar ways, the InterSensor Service generates the interfaces for the Sensor Observation Service (SOS). The SOS comprises of operations such as *DescribeSensor* to retrieve sensor description according to the SensorML standard and *GetObservation* to retrieve real-time observations according to the Observations and Measurements (O&M) standard. For example, the observations from an established InterSensor Service can be queried according to the O&M format by simply using the following *GetObservation* request:

**Listing 7.6:** Illustration of the OGC Sensor Observation Service endpoints automatically generated for each data source connection

```
#GetObservation Request interface generated by InterSensor Service
http://127.0.0.1/OGCSensorThingsApi/v1.0/ogc-sos-webapp/service?
service=SOS&version=2.0.0
&request=GetObservation
&temporalFilter=om:phenomenonTime,
2018-08-05T00:00:00/2018-08-05T18:00:00
```

### 7.4.2.3 52°North Timeseries API

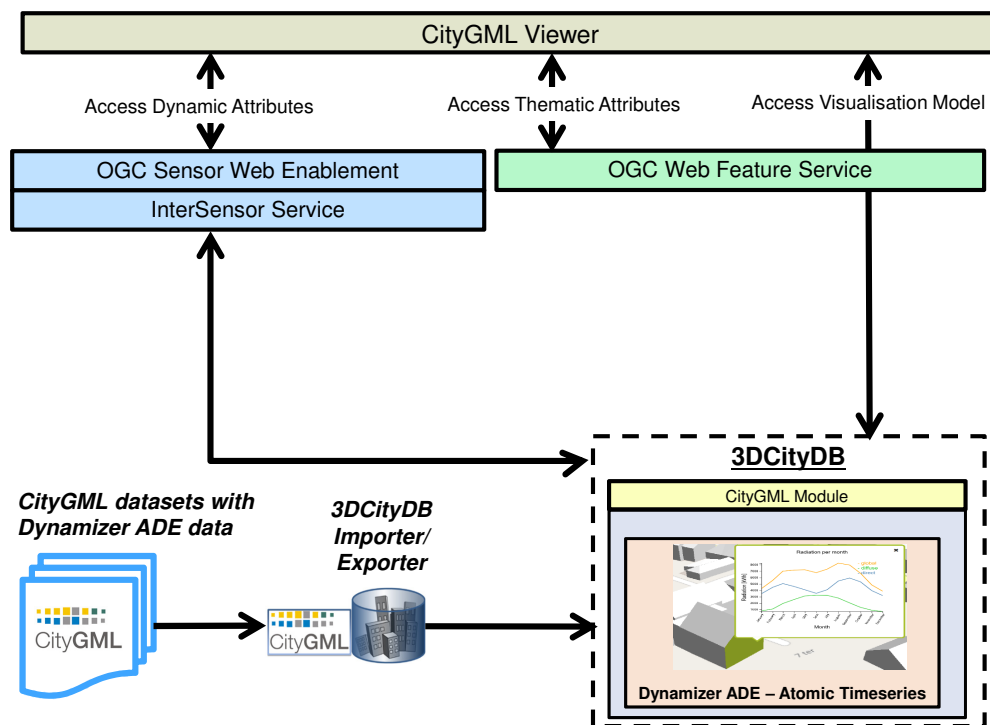
**Listing 7.7:** Illustration of the 52°North Timeseries API endpoints automatically generated for each data source connection

```
#Base URL for the Timeseries API
http://127.0.0.1/52n-rest-api/
#Services
http://127.0.0.1/52n-rest-api/services
#Stations
http://127.0.0.1/52n-rest-api/stations
#Timeseries
http://127.0.0.1/52n-rest-api/timeseries
#Offerings
http://127.0.0.1/52n-rest-api/offerings
#Procedures
http://127.0.0.1/52n-rest-api/procedures
#Features
http://127.0.0.1/52n-rest-api/features
#Phenomena
http://127.0.0.1/52n-rest-api/phenomena
```

The Timeseries API developed by 52°North is a REST-ful web binding to the OGC Sensor Observation Service. While it is not a standard, we decided to support this API, because it allows

querying and visualising sensor locations and their observations using the so-called Helgoland Open Source web client. Like the SensorThings API, the Timeseries API also comprises of a well-defined data model. The observations from an established InterSensor Service can be queried according to the Timeseries API by using the standardised requests shown in listing 7.7.

The querying of the data using the standardised requests and responses allow them to be used on the OGC SWE compliant applications. For example, by providing these requests to the Helgoland application, the sensor information can be visualised irrespective of its platform. This is later illustrated in figure 8.8.



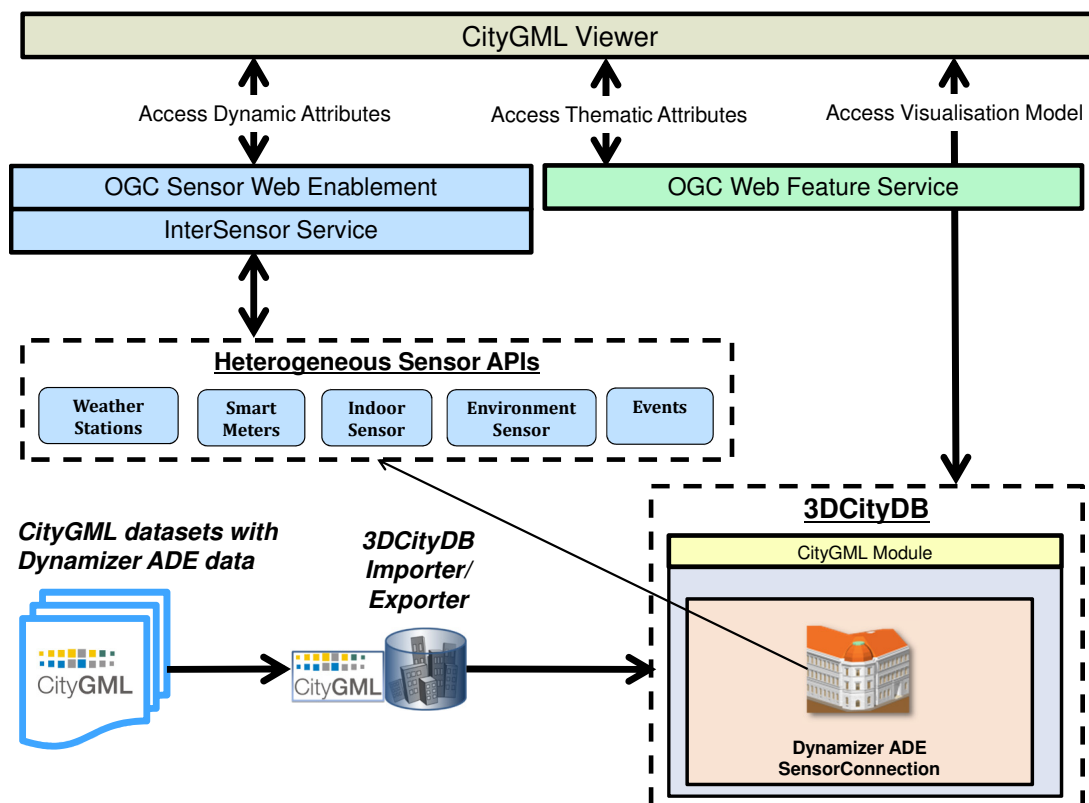
**Figure 7.10:** Resolving the issue of accessing Dynamizer AtomicTimeseries shown in figure 6.4 by the InterSensor Service.

## 7.5 Discussions

In this way, the OGC SWE standards can be used with the extensions of CityGML models for supporting time-dynamic properties. As discussed in chapter 6, since WFS is not suitable for retrieving and querying time-series values, the OGC SWE standards can be used for that. In the case of sensors, they can be attached to the OGC SWE implementations such as the OGC Sensor Observation Service and the OGC SensorThings API. If data storage is an issue, the InterSensor Service can be used to directly connect to the respective APIs and retrieve their observations without storing their results in a database. The InterSensor Service is also beneficial for working with virtual sensors such as Twitter

API, or time-series stored in CityGML Dynamizer files. As shown in figure 7.10, the adapters of InterSensor Service can be developed for CityGML Dynamizer representing an AtomicTimeseries. The time-series data and its metadata can be stored in the 3DCityDB using a new relational database model (c.f. chapter 6). The InterSensor Service can read the Dynamizer time-series data and encodes them "on-the-fly" according to the OGC SWE interfaces.

In another illustration (as shown in figure 7.11), a Dynamizer can have links to different types of sensor and IoT platforms. These sensor platforms may completely be different from each other, and may run on different APIs. The InterSensor Service can be used with such heterogeneous sensor platforms and provide unified access according to the OGC SWE interfaces. These concepts have been applied in real-world Smart City projects. The results are mentioned in chapter 8. As the InterSensor Service establishes direct connections to the data source, a limitation is that it is not suitable for performing very large queries on the source data (e.g. retrieving the readings per second for the duration one year). It may affect the performance of the application. Hence, it is also not suitable to perform analytics on such large datasets. Such analytics should directly be managed within the data sources. The current implementation of the InterSensor Service is read only. The write access could be developed too, however that would require additional data fields in the data model of the data sources and their implementations.



**Figure 7.11:** Resolving the issue of accessing observations from heterogeneous sensor platforms shown in figure 6.4 by the InterSensor Service.





## **Part III**

### **Proof of Concept**



## Chapter 8

---

### Using Dynamic 3D City Models in Smart Cities

This chapter discusses the implementations of the developed concepts such as Dynamizers and InterSensor Service in the context of two real-world Smart City projects: OGC Future City Pilot Phase 1 and the Smart District Data Infrastructure Demonstrator. The chapter describes the goals and objectives of these two projects and demonstrates how Dynamizers and InterSensor Service are used for supporting their use cases. In the OGC Future City Pilot Phase 1, CityGML Dynamizer *AtomicTimeseries* is used for representing solar potential simulation results with the 3D city model of the district Bruz in Rennes, France. In another use case, Dynamizer *SensorConnection* is used for linking with real-time sensor data with the 3D city model of the Greenwich district in London, U.K. However, the use case requires a connection to only a single sensor device accessible using the OGC Sensor Observation Service. The complexity arises when dealing with multiple heterogeneous sensors and IoT platforms, belonging to numerous stakeholders and running on different platforms. Such cross-platform interoperability of time-series data is achieved using the InterSensor Service within the project Smart District Data Infrastructure (SDDI) Demonstrator. Such unified representations improve decision-making in Smart City scenarios. Various demonstrations in the Queen Elizabeth Olympic Park, London, U.K. are shown in this chapter.

Some of the results in this chapter have been presented in the published papers

- Chaturvedi, K.,** Yao, Z. and Kolbe, T. H. (2019). ‘Integrated Management and Visualization of Static and Dynamic Properties of Semantic 3D City Models’. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLII-4/W17*, pp. 7–14. URL: <https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-4-W17/7/2019/>
- Chaturvedi, K.** and Kolbe, T. H. (2019). ‘Towards Establishing Cross-Platform Interoperability for Sensors in Smart Cities’. In: *Sensors* 19.3. URL: <https://www.mdpi.com/1424-8220/19/3/562>
- Chaturvedi, K.** and Kolbe, T. H. (2017). *Future City Pilot 1 Engineering Report - OGC Doc. No. 16-098*. Tech. rep. Open Geospatial Consortium. URL: <http://docs.opengeospatial.org/per/16-098.html>
- Chaturvedi, K.,** Willenborg, B., Sindram, M. and Kolbe, T. H. (2017). ‘Solar Potential Analysis and Integration of the Time-dependent Simulation Results for Semantic 3D City Models using Dynamizers’. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences IV-4/W5*, pp. 25–32. URL: <https://doi.org/10.5194/isprs-annals-IV-4-W5-25-2017>

## 8.1 OGC Future City Pilot Phase 1

The Future City Pilot Phase 1 (FCP1) is an OGC Interoperability Program initiative in collaboration with buildingSMART International (bSI). The pilot aimed at demonstrating and enhancing the ability of spatial data infrastructures to support the quality of life, civic initiatives, and urban resilience. The objective of the OGC pilot project was to demonstrate how the use of geospatial technologies including international standards such as CityGML (Gröger et al. 2012) and Industry Foundation Classes (IFC) (IFC 2016) can provide stakeholders with information, knowledge and insight which enhances financial, environmental, and social outcomes for citizens living in cities. During the pilot, a series of scenarios were set up based on real-world requirements that can serve as blueprints for other cities to modify and apply in their context. The pilot focused on four scenarios related to the following areas: Urban Planning, Urban Flood Mapping, Adult Social Care, and Dynamic Resource Modelling. These scenarios were put forward by the pilot sponsors: Sant Cugat del Vallès (Barcelona, Spain), Ordnance Survey Great Britain (UK), virtualcitySYSTEMS GmbH (Germany), and Institut National de l'Information Géographique et Forestière - IGN (France). Scientific solutions for the proposed scenarios were developed and demonstrated by the pilot participants: University of Melbourne (Australia), Remote Sensing Solutions, Inc. (USA), and Chair of Geoinformatics, Technische Universität München (Germany). The pilot was considered a big success in utilising international standards for various demonstrations. The demonstrations of the OGC Future City Pilot Phase 1 are documented in several Engineering Reports available at <https://www.ogc.org/projects/initiatives/fcp1> and are presented in a YouTube Video <https://youtu.be/aSQFIPwf2oM>.

The Dynamizer concept was implemented as an Application Domain Extension (ADE) for CityGML and used for two scenarios in this pilot. The following sub-sections describe the details of the use cases, challenges, and solutions by using CityGML Dynamizers. The details of the solutions are documented in an OGC Engineering Report (Chaturvedi and Kolbe 2017).

### 8.1.1 Integrating city object properties with real-time sensor data

One of the objectives of the pilot was to demonstrate “*how dynamic city models can provide better services to the citizens as well as can help to perform the better analysis?*”. This objective was set up by Ordnance Survey, Great Britain’s National Mapping Agency, together with the Royal Borough of Greenwich. The goals were to demonstrate (i) how the use of dynamic data can support the provision of services to citizens and stakeholders; (ii) how officials and those who deliver and coordinate services can share and coordinate data more effectively and use smarter working practices; and (iii) how agencies/boroughs can improve collaboration through the creation and use of interoperable data, content, and insight. The pilot, in general, highlighted that the use of interoperable data, content, and insight built upon open standards has potential to create opportunities for cross-department collaboration and sharing, and this could be achieved through the development of platforms, portals and data exchanges. This particular use case was aimed at developing a Digital Twin of the Greenwich area by including the city’s static data such as buildings or houses with elderly citizens having special needs integrated with dynamic data such as outside temperature or air humidity. Such potential integration within council-owned assets can lead to better decision making in case of extreme weather or other emergency scenarios matching human needs to the right housing/resources.

### 8.1.1.1 Creation of CityGML datasets

The 3D building objects were created according to the CityGML LoD1 specification. The dataset includes 265,000 building objects generated using the Ordnance Survey MasterMap<sup>113</sup> building footprints. The dataset was further enriched by various thematic properties such as building address, details of residents living in buildings, adult care, and housing stock information. The CityGML documents were validated and stored in a database using the software 3DCityDB.

### 8.1.1.2 Working with sensors

For the demonstration purpose, the real-time observations were retrieved from a weather station installed in the area. The weather stations are named after Intel Collaborative Research Institute (ICRI)<sup>114</sup> as ICRI\_0001, ICRI\_0002, and ICRI\_0003 and measure 15 properties in the area including temperature, humidity, wind speed, etc. The frequency of each observation is one minute. The sensor details and observations are accessed from Intel's platform via a Hypercat registry<sup>115</sup> and encoded using the SenML format (Jennings et al. 2018). The sensors and observations were brought to the OGC Sensor Web Enablement environment for working with them in an interoperable way. In this project, the 52°North SOS server implementation was used, and the observations were imported to the SOS server in regular intervals (by using the steps mentioned in section 7.2). In this way, the observations could be accessed using standardised SOS operations such as *DescribeSensor* and *GetObservation*.

The *GetObservation* request looks like as follows (the request should be formed in a single line without any linebreak and spaces):

```
http://127.0.0.1:8080/weather-sensors-sos-webapp/service
?SERVICE=SOS
&VERSION=2.0.0
&REQUEST=GetObservation
&PROCEDURE=ILONDON577
&OBSERVEDPROPERTY=Temp
&TEMPORALFILTER=om:phenomenonTime,
    2019-10-08T00:00:00Z/2019-10-09T00:00:00Z
```

The above request retrieves all the temperature observations from the sensor "ILONDON577" between the timestamps "2019-10-08T00:00:00Z" and "2019-10-09T00:00:00Z". The SOS server runs on the machine 127.0.0.1:8080. Similarly, an example *DescribeSensor* request for the same sensor looks like as follows (the request should be formed in a single line without any linebreak and spaces):

```
http://127.0.0.1:8080/weather-sensors-sos-webapp/service
?REQUEST=DescribeSensor
&SERVICE=SOS
&VERSION=2.0.0
&PROCEDURE=ILONDON577
&procedureDescriptionFormat=http://www.opengis.net/sensorML/1.0.1
```

<sup>113</sup><https://www.ordnancesurvey.co.uk/business-government/products/mastermap-topography>

<sup>114</sup><https://www.icri-cars.org/>

<sup>115</sup><https://www.iotone.com/organisation/hypercat/o153>

**Listing 8.1:** Illustration of Dynamizer SensorConnection to link to a weather station sensor running over the OGC SOS

```

<bldg:Building gml:id="building1">
  <core:dynamizer>
    <dyn:Dynamizer gml:id="building1_Dynamizer">
      <dyn:attributeRef>
        <!-- Single line without linebreak and space -->
        //bldg:Building[@gml:id='building1']
        /core:genericAttribute
        /gen:DoubleAttribute[gen:name='temperature']
        /gen:value
      </dyn:attributeRef>
      <dyn:startTime>2019-01-01T00:00:00Z</dyn:startTime>
      <dyn:endTime>2020-01-01T00:00:00Z</dyn:endTime>
      <dyn:sensorConnection>
        <dyn:SensorConnection>
          <dyn:connectionType>ogc_sos_2.0</dyn:connectionType>
          <dyn:observationProperty>Temp</dyn:observationProperty>
          <dyn:uom>Celsius</dyn:uom>
          <dyn:sensorID>ILONDON577</dyn:sensorID>
          <dyn:baseURL>
            http://127.0.0.1:8080/weather-sensors-sos-webapp/service
          </dyn:baseURL>
          <dyn:authType>none</dyn:authType>
          <dyn:linkToObservation>
            <!-- Single line without linebreak and space -->
            %baseURL%
            ?service=SOS
            &version=2.0.0
            &request=GetObservation
            &procedure=%sensorID%
            &observedProperty=%observationProperty%
            &temporalFilter=om:phenomenonTime,%startTime%/%endTime%
          </dyn:linkToObservation>
          <dyn:linkToSensorDescription>
            <!-- Single line without linebreak and space -->
            %baseURL%
            ?REQUEST=DescribeSensor
            &SERVICE=SOS
            &VERSION=2.0.0
            &PROCEDURE=%sensorID%
            &procedureDescriptionFormat
            =http://www.opengis.net/sensorML/1.0.1
          </dyn:linkToSensorDescription>
        </dyn:SensorConnection>
      </dyn:sensorConnection>
    </dyn:Dynamizer>
  </core:dynamizer>
</bldg:Building>

```


```

    <dyn:sensorLocation
      xlink:href="#building1"></dyn:sensorLocation>
  </dyn:SensorConnection>
</dyn:sensorConnection>
</dyn:Dynamizer>
</core:dynamizer>
.....
.....
</bldg:Building>

```

### 8.1.1.3 Representing direct sensor links using CityGML Dynamizers

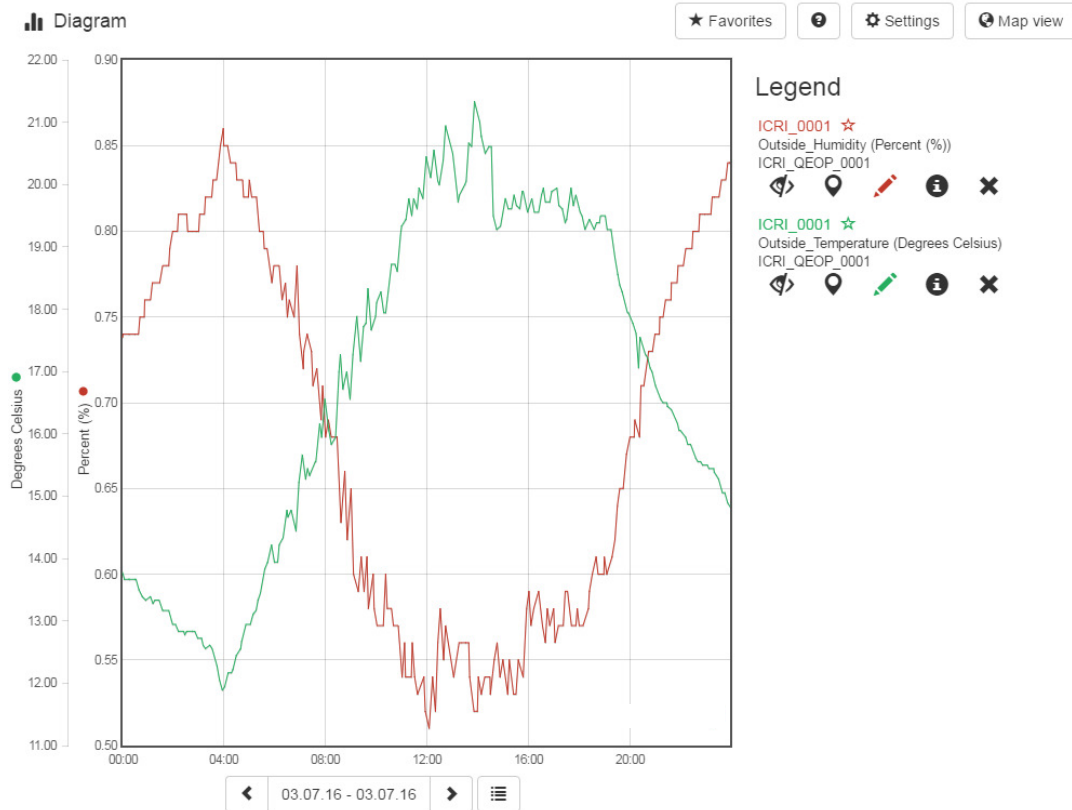
In order to link the building attribute to the running Sensor Observation Service, the Dynamizer can be defined as shown in listing 8.1. The illustration shows a Dynamizer *SensorConnection* linking to a running SOS by providing the required parameters. At the same time, the reference to a specific attribute of the CityGML building object can also be defined, allowing to override this attribute based on the values retrieved from the sensor connection.



Layer Settings   GSpreadsheet   <b>GFusionTable</b>		
Show in ▾   Generate Report		
<input type="checkbox"/>	Fieldname	Value
<input type="checkbox"/>	GMLID	osgb_7ea64278-a92f-459a-9bea-f905bbcb63e3
<input type="checkbox"/>	TOID	osgb1000041344855
<input type="checkbox"/>	SensorID	ICRI_0001
<input type="checkbox"/>	ServiceType	SOS 2.0
<input type="checkbox"/>	LinkToSensorML	<a href="http://129.187.38.201:8080/52n-sos-webapp-geop/servi...">http://129.187.38.201:8080/52n-sos-webapp-geop/servi...</a>
<input type="checkbox"/>	LinkToObservation	<a href="http://129.187.38.201:8080/52n-sos-webapp-geop/servi...">http://129.187.38.201:8080/52n-sos-webapp-geop/servi...</a>
<input type="checkbox"/>	LinkToSensorClient	<a href="http://129.187.38.201:8080/52n-sos-webapp-geop/static...">http://129.187.38.201:8080/52n-sos-webapp-geop/static...</a>

**Figure 8.1:** Thematic attributes of a buiding including description and links for sensor based services.

Figure 8.1 shows different panels of the 3DCityDB Web-map-client Pro (Chaturvedi et al. 2015), allowing users to visualise and interact with 3D building geometries as well as thematic attributes of



**Figure 8.2:** Timeseries graph visualisation of real-time sensor observations. Screenshot taken from the Helgoland client. The graph shows outside temperature and humidity retrieved from the weather station on the 3rd of July 2016 overlaid within the same view.

the buildings. As shown, the *GMLID* is the unique ID of the building which is based on the TOID provided by the Ordnance Survey Mastermap. The sensor-related attributes have been defined based on the dynamizer class. *Sensor\_ID* is the unique ID for the specific sensor. *ServiceType* shows the type of service being used by the sensor (SOS 2.0 in this case). Further, *LinkToSensorDescription* is the request URL for the SOS *DescribeSensor* operation, which returns the sensor descriptions and metadata encoded in the SensorML format. *LinkToObservation* is the request URL for the SOS *GetObservation* operation, which returns the sensor observations encoded in the OGC O&M format. *LinkToSensorClient* is an additional generic attribute showing the URL for the sensor visualisation client, which shows the sensor observations in the form of timeseries graphs as shown in figure 8.2.

The Dynamizer implementation is considered as a successful first attempt in integrating real-time sensor observations with static city objects. However, the pilot focused only on one sensor data source which could easily be brought to the OGC SWE environment. There are other scenarios which require dealing with not only one sensor but multiple heterogeneous sensors belonging to different stakeholders, different APIs, and different platforms. Such complex scenarios have been observed in other projects described in section 8.2.



## 8.1.2 Enriching city object properties with solar potential simulation time-series

This use case was set up by IGN and virtualcitySYSTEMS GmbH and was based in the commune of Bruz, located 11 km southwest of Rennes in Brittany, France and is a part of Rennes Metropole. The main objective was to provide interactive and detailed information on solar irradiation of buildings to the citizens and the energy planners by making sophisticated solar potential analysis. The use case aimed at answering questions such as (i) "How many buildings have solar irradiation of more than a specific unit (e.g. 5000 kWh)?", (ii) "Which buildings are well suited for installing solar panels?", and (iii) "How does the irradiation for individual buildings varies through the year?".

### 8.1.2.1 Creation of CityGML datasets

During the project, 3D building objects were created based on the CityGML standard according to Ref3DNat recommendations <sup>116</sup>, which is an IGN reference specification proposed for 3D city models in France. The CityGML dataset includes approximately 5500 building objects in the Level of Detail 2 (LoD2). The dataset was further enriched by solar irradiation values computed by a Solar Potential Analysis tool for semantic 3D city models (Zahn 2015) developed at the Chair of Geoinformatics at the Technical University of Munich. The simulation tool (c.f. section 2.1.2) estimates the solar power from direct, diffuse, and global sunlight irradiation for individual months of the year.

**Listing 8.2:** Representation of monthly solar irradiation values as individual generic attributes in the current version of CityGML 2.0

```
<bldg:WallSurface gml:id="building1_wall1">
  <gen:doubleAttribute name="globalRadYear">
    <gen:value>77004.913</gen:value>
  </gen:doubleAttribute>
  <gen:doubleAttribute name="globalRadMonth_01">
    <gen:value>4293.446</gen:value>
  </gen:doubleAttribute>
  <gen:doubleAttribute name="globalRadMonth_02">
    <gen:value>5563.502</gen:value>
  </gen:doubleAttribute>
  <gen:doubleAttribute name="globalRadMonth_03">
    <gen:value>7010.33</gen:value>
  </gen:doubleAttribute>
  <gen:doubleAttribute name="globalRadMonth_04">
    <gen:value>7180.839</gen:value>
  </gen:doubleAttribute>
  <!-- all months in a year -->
  <!-- ..... -->
  <gen:doubleAttribute name="globalRadMonth_12">
    <gen:value>4010.239</gen:value>
  </gen:doubleAttribute>
  <!-- diffuse radiation -->
```

<sup>116</sup>[http://professionnels.ign.fr/doc/DC\\_Ref3DNat\\_1-0-2\\_charteIGN-1.pdf](http://professionnels.ign.fr/doc/DC_Ref3DNat_1-0-2_charteIGN-1.pdf)

```
<!-- direct radiation --->
</bldg:WallSurface>
```

To be able to perform profound analyses, the values are stored as attributes in the city model in addition to the purely visual representation of the solar irradiation values as textures. Based on the point grid results, the different temporal resolutions for direct, diffuse and global irradiation are computed and stored as generic attributes for each spatial aggregation level (wall, roof surfaces and building) in the city model. Currently, the temporal classification of a simulation result parameter is encoded in the attribute name as a suffix. For instance, an attribute named *globalRadMonth\_01* denotes the aggregate global irradiation estimate on a specific feature for January. Similarly, individual generic attributes can be defined for storing the irradiation values for individual months. In the same way, the attributes can be defined for the aggregate global irradiation estimate on a specific feature for the entire year such as *globalRadYear*. The code representation in listing 8.2 is an excerpt of a wall surface of a building in a CityGML instance file for the corresponding attributes. However, this illustration shows that the monthly results are represented as multiple static values. The attributes are not dynamic. The following subsections show how a Dynamizer can be used to replace the Listing 8.2. Utilising the Dynamizer *AtomicTimeseries* class, it is possible to override one single static CityGML attribute based on the dynamic solar potential simulation results.

As shown below, a building wall surface (having an id "*building1\_wall1*") has a *genericAttribute* named "*globalRadMonth*" representing the global irradiation value for different months. In this case, the attribute "*globalRadMonth*" is dynamic and changes its value every month depending on the simulation results. The following sub-sections show how a Dynamizer can be used to replace Listing 8.2.

```
<bldg:WallSurface gml:id="building1_wall1">
  <core:genericAttribute>
    <gen:DoubleAttribute>
      <gen:name>globalRadMonth</gen:name>
      <gen:value>4293.446</gen:value>
    </gen:DoubleAttribute>
  </core:genericAttribute>
</bldg:WallSurface>
```

### 8.1.2.2 Representing in-line time-series using international standards

The timeseries generated by the solar potential simulation results can either be stored in external files such as CSV or can also be represented along with its rich metadata using international standards such as OGC TimeseriesML 1.0 (as shown in section 5.3.2). During the pilot, the simulation results were encoded according to the TimeseriesML standard.

Listing 8.3 illustrates that the monthly simulation results for the building wall surface are represented according to the OGC TimeseriesML 1.0 standard. In this illustration, the TimeseriesML file represents the time-series for the simulation results using well-defined metadata attributes. The temporal extent of the time-series is "2015-01-01T00:00:00Z" and "2016-01-01T00:00:00Z". Since the time-series is regularly spaced, it is defined by the combination of the attributes *tsml:baseTime* and *tsml:spacing*.

The values indicate that starting from the timestamp "2015-01-01T00:00:00Z", the values are regularly spaced by "1 Month". Besides, the standard also allows defining other metadata such as the unit of measurement (UoM), quality of time-series, and interpolation type. After specifying the required metadata attributes, the values of the time-series are defined using the *tsml:point* object. In the mentioned example, the values are of type *tsml:MeasurementTVP* and contain values for each month of the specific year. The advantage in using well-defined standards like TimeseriesML is that it allows providing rich metadata and accurate description of the time-series enabling better interpretation for users and applications.

**Listing 8.3:** Illustration of monthly solar irradiation values represented according to the OGC TimeseriesML 1.0 standard

```
<tsml:TimeseriesTVP>
  <gml:description>
    Example of RegularTimeSeries object with values every month
  </gml:description>
  <tsml:metadata>
    <tsml:TimeseriesMetadata>
      <tsml:temporalExtent>
        <gml:TimePeriod>
          <gml:beginPosition>2015-01-01T00:00:00Z</gml:beginPosition>
          <gml:endPosition>2016-01-01T00:00:00Z</gml:endPosition>
        </gml:TimePeriod>
      </tsml:temporalExtent>
      <tsml:baseTime>2015-01-01T00:00:00Z</tsml:baseTime>
      <tsml:spacing>P1M</tsml:spacing>
    </tsml:TimeseriesMetadata>
  </tsml:metadata>
  <tsml:defaultPointMetadata>
    <tsml:PointMetadata>
      <tsml:quality xlink:title="Good"/>
      <tsml:uom code="kWh"/>
      <tsml:interpolationType xlink:title="Continuous"/>
    </tsml:PointMetadata>
  </tsml:defaultPointMetadata>
  <tsml:point>
    <tsml:MeasurementTVP>
      <tsml:value>4293.446</tsml:value>
    </tsml:MeasurementTVP>
  </tsml:point>
  <tsml:point>
    <tsml:MeasurementTVP>
      <tsml:value>5563.502</tsml:value>
    </tsml:MeasurementTVP>
  </tsml:point>
  <!-- ..... -->
```

```
<!-- ..... -->
</tsml:TimeseriesTVP>
```

### 8.1.2.3 Interpreting time-series data using CityGML Dynamizers

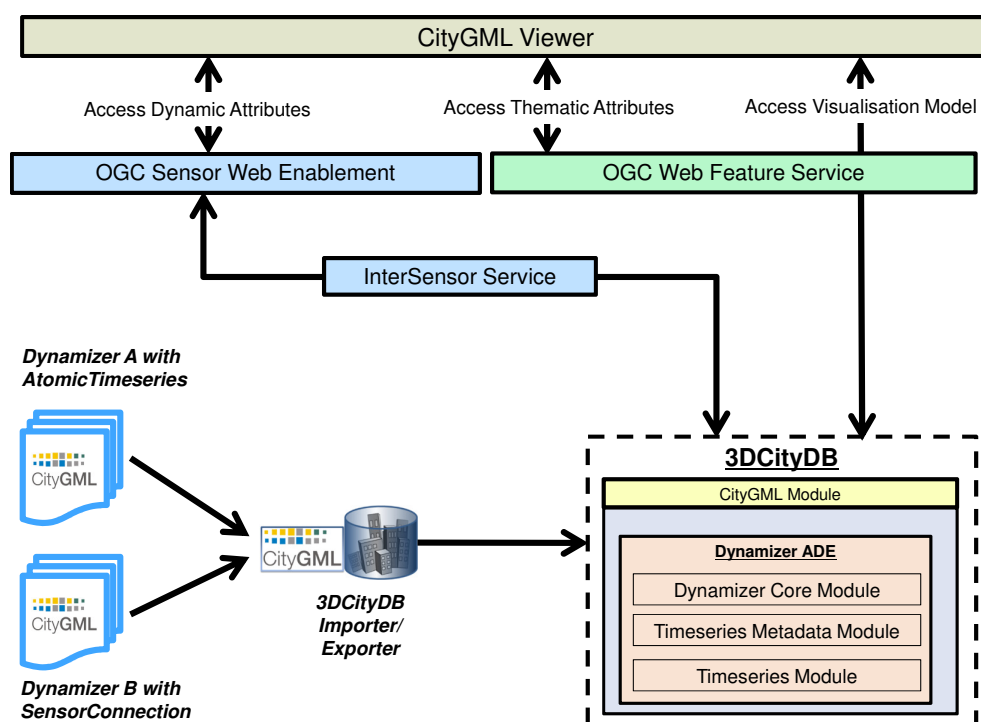
The TimeseriesML data (as mentioned in the listing 8.3) are stored in a file named *results.xml*. With the help of a Dynamizer Atomic Timeseries, it is possible to override the static attribute "globalRadMonth" of the building wall surface "building1\_wall1" based on the simulation results stored in an external TimeseriesML file. Dynamizer can be defined for this purpose as shown in listing 8.4. This example shows that reference to the file *results.xml* is given using the attribute *fileLocation*, which allows interpreting the TimeseriesML file accordingly.

**Listing 8.4:** Illustration of a Dynamizer AtomicTimeseries representing dynamic solar irradiation values for a specific Building Wall Surface

```
<dyn:Dynamizer gml:id="global_irradiation_Dynamizer">
  <dyn:attributeRef>
    <!-- Single line without linebreak and space -->
    //con:WallSurface[@gml:id='building1_wall1']
    /core:genericAttribute
    /gen:DoubleAttribute[gen:name='globalRadMonth']
    /gen:value
  </dyn:attributeRef>
  <dyn:startTime frame="#ISO-8601">
    2015-01-01T00:00:00Z
  </dyn:startTime>
  <dyn:endTime frame="#ISO-8601">
    2016-01-01T00:00:00Z
  </dyn:endTime>
  <dyn:dynamicData>
    <dyn:StandardFileTimeseries>
      <dyn:firstTimestamp>2015-01-01T00:00:00Z</dyn:firstTimestamp>
      <dyn:lastTimestamp>2016-01-01T00:00:00Z</dyn:lastTimestamp>
      <dyn:observationProperty>
        GlobalIrradiationPerMonth
      </dyn:observationProperty>
      <dyn:uom>kWh</dyn:uom>
      <dyn:fileLocation>
        file:///C:/Folder1/results.xml
      </dyn:fileLocation>
      <dyn:fileType>timeseriesml_1.0</dyn:fileType>
      <dyn:mimeType>application/xml</dyn:mimeType>
    </dyn:StandardFileTimeseries>
  </dyn:dynamicData>
</dyn:Dynamizer>
```

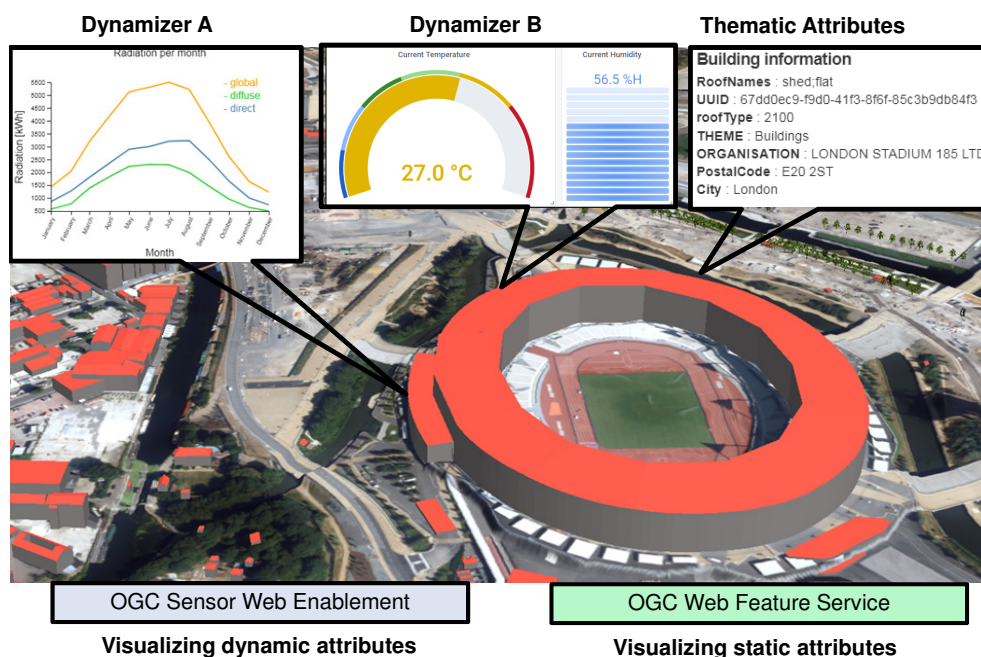
### 8.1.3 Integrated management and visualisation of static and dynamic properties

Although the Dynamizer ADE data were not managed in the 3DCityDB during the pilot, the CityGML files can be imported and stored in the 3DCityDB using the concepts developed in Chapter 6 and Chapter 7. As discussed in previous sub-sections, two CityGML files were created during the OGC Future City Pilot Phase 1. One file for the Greenwich district contains Dynamizer *SensorConnection* for linking to real-time sensor observations retrieved from OGC Sensor Observation Service (SOS). The other CityGML data for the Bruz district includes the Dynamizer *AtomicTimeseries* for representing the monthly solar irradiation values using the standard OGC TimeseriesML 1.0. The 3DCityDB can be extended for the Dynamizer ADE using the ADE Plug-in Manager. As shown in figure 8.3, both the CityGML files can be imported to the 3DCityDB and the relevant features and attribute values can be mapped to the respective tables and relations in 3DCityDB according to the new relational data model proposed in Chapter 6.



**Figure 8.3:** Management of CityGML Dynamizer ADEs for Future City Pilot Phase 1.

Once the CityGML file is imported into the 3DCityDB, its thematic attributes can be retrieved using the OGC Web Feature Service. Further, its time-series and dynamic attributes can be retrieved using the OGC Sensor Web Enablement standards with the help of the Open Source InterSensor Service. The data source connections to the individual Dynamizers can be configured as shown in listing 7.3. Figure 8.4 shows an illustration of a CityGML Viewer (3DCityDB Web Map Client), allowing to retrieve and visualise static and dynamic data from CityGML file in an integrated fashion.



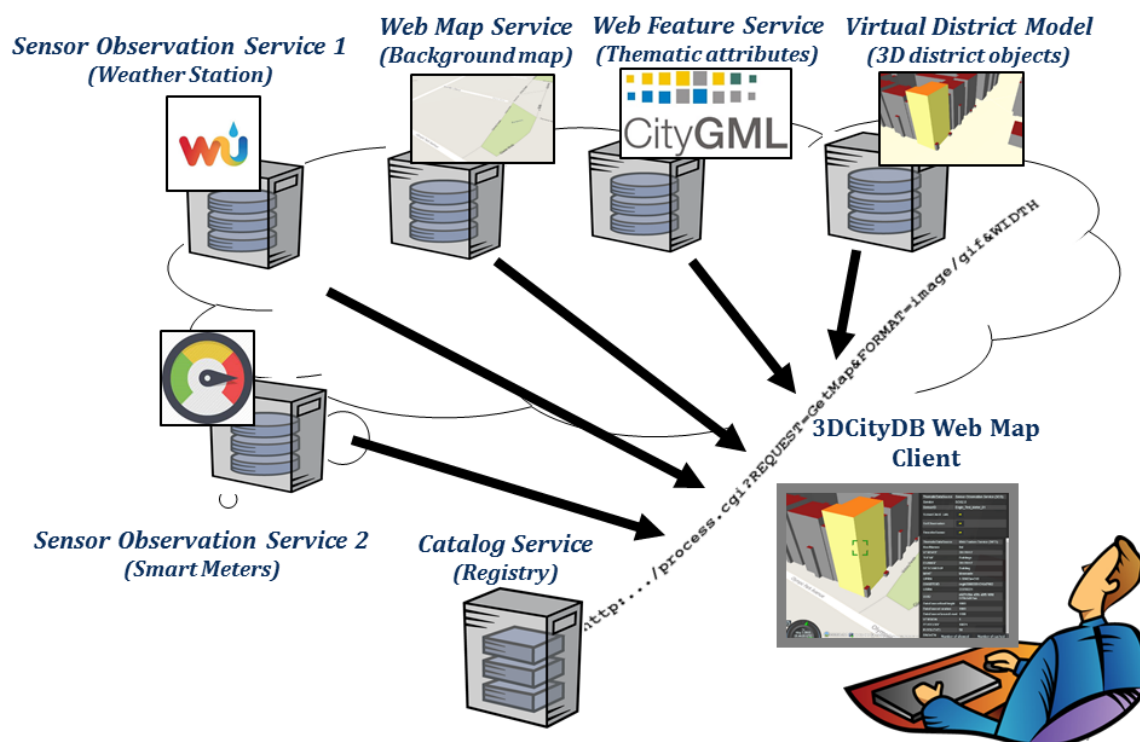
**Figure 8.4:** Integrated management and visualisation of static and dynamic properties of CityGML Dynamizers.

## 8.2 Smart District Data Infrastructure (SDDI)

The Smart District Data Infrastructure (SDDI) is a framework coined by (Moshrefzadeh et al. 2017) developed within the Smart Sustainable Districts Program of the Climate-KIC of the European Institute for Innovation and Technology (EIT). The framework focuses on district-level solutions and provides a way to integrate heterogeneous resources such as actors and stakeholders, applications, urban analytic toolkits, sensors and IoT devices with a Virtual District Model (VDM). The VDM is a 3D spatial and semantic representation of the physical reality of the district. It consists of relevant objects like buildings, streets, vegetations, water bodies and networks based on the CityGML standard. The SDDI framework is designed based on open and international OGC standards which comprise well-defined information models such as CityGML for semantic 3D city models and SensorML for defining sensor and IoT devices. The SDDI framework also utilises mature and well-supported web services according to OGC standards such as the Web Feature Service (WFS) (Vretanos 2010) to retrieve CityGML objects and other object-based datasets, the Sensor Observation Service (SOS) (Bröring et al. 2012) and Sensor-Things API (Liang et al. 2015) to retrieve real-time sensor observations, and (iv) the Catalogue Service for the Web (Douglas et al. 2014) allowing registering and discovering registered resources using standardised metadata (Figure 8.5). The use of international standards allows linking and managing different components in a unified and stable way and across manufacturers.

One of the first implementations of the SDDI framework was developed for Queen Elizabeth Olympic Park in London. Within the project, the London Legacy Development Corporation (LLDC<sup>117</sup>),

<sup>117</sup><https://www.queenelizabetholympicpark.co.uk/our-story/the-legacy-corporation>



**Figure 8.5:** Joint usage of standardised web services in the Smart District Data Infrastructure.

have identified different use cases related to the reduction of resource and energy usage, reduction of waste, reduction of emissions, improvements of well-being, mobility, and in general concerning efficiency.

As shown in Figure 8.6, for different use cases, the district has access to multiple sensors and IoT devices owned by various stakeholders and partners. For example, two weather stations are located in the park determining the real-time environmental properties such as outside temperature, humidity, wind speed etc. These weather stations are registered with the Weather Underground platform<sup>118</sup>. As a part of the Nature-Smart Cities program<sup>119</sup>, a network of 15 bat monitors is installed across the Olympic Park. The program assumes that bats are considered to be a good indicator species, reflecting the general health of the natural environment. So, a healthy bat population correlates with healthy biodiversity in the local area. Hence, the smart bat monitors are installed in different habitats across the park and continuously capture data on bat species and activity levels. The observations from these bat monitors are accessed using another platform called the OpenSensors<sup>120</sup>. There are smart meters installed in important buildings such as the Aquatic Center and the Copper Box Arena. These Smart Meters are used for determining real-time energy consumption (e.g. electricity and gas usage) for the buildings. These meters belong to the company Engie and are managed within a proprietary platform called C3NTINEL<sup>121</sup>. Similarly, a use case also requires to gather the visitor's

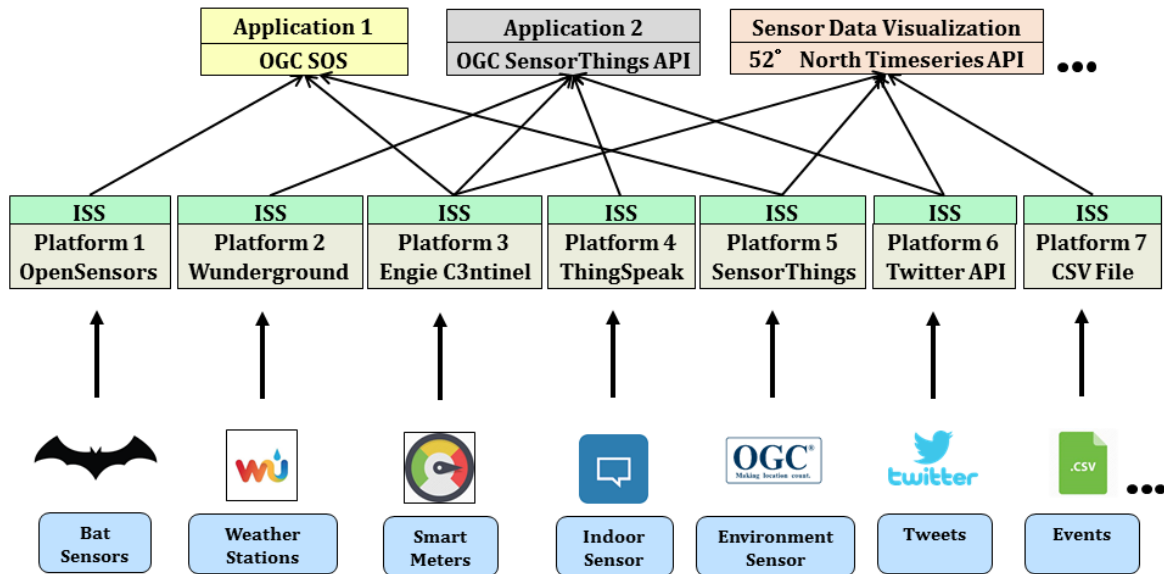
<sup>118</sup><https://www.wunderground.com/>

<sup>119</sup>[www.naturesmartcities.com](http://www.naturesmartcities.com)

<sup>120</sup><https://www.opensensors.com/>

<sup>121</sup><https://www.c3ntinel.com/>

sentiments or experiences by studying the Twitter activity around the park. For this use case, access to the Twitter API<sup>122</sup> was required to retrieve real-time geo-tagged tweets around the park. For another use case, the park administrators need to assess the impact of scheduled events in the park on the other properties. For example, "if a football match is scheduled in the stadium, what is its impact on the gas consumption of the stadium on that particular day?". The information of such scheduled events is listed in external CSV files, which can also be treated as a data source with a time-series in this context.



**Figure 8.6:** Implementation scenario of the InterSensor Service (ISS) establishing interoperability for different sensor platforms and observations in the district Queen Elizabeth Olympic Park, London.

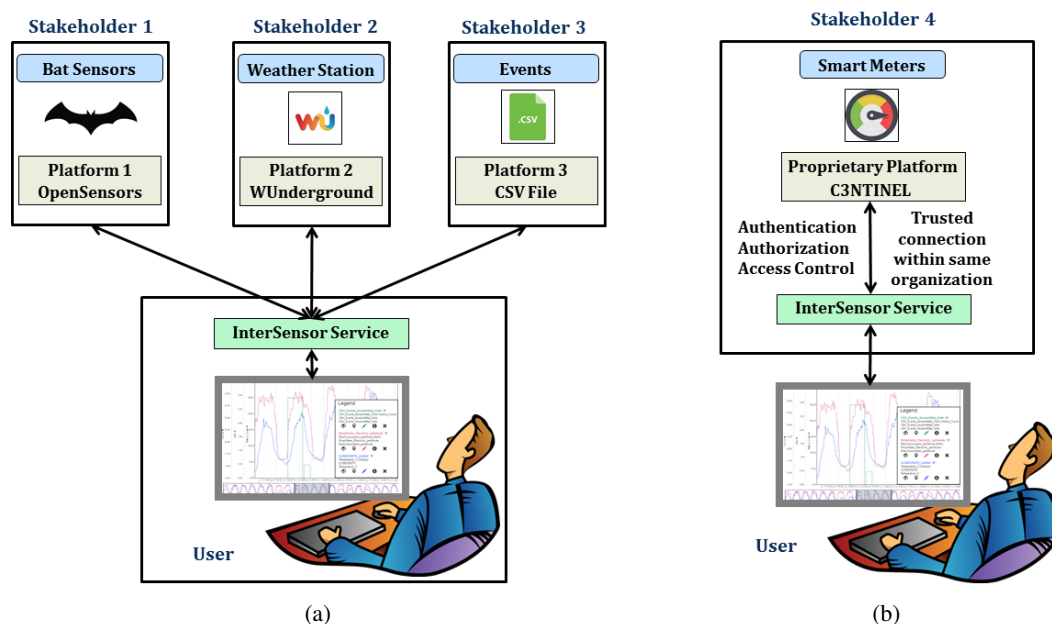
As mentioned, these data sources are heterogeneous in a way that they (i) belong to different stakeholders, (ii) are used for various purposes, (iii) based on different platforms and APIs, and (iv) provide different types of observations. However, it is essential to analyse them together for making well-informed decisions. To bring all of them within a common operational framework, the InterSensor Service (c.f. Chapter 7) is used to connect to all of them. It allows encoding all sensors, their descriptions, metadata and recorded real-time observations using common and mature standards, as well as querying and analysing them using common interfaces on the OGC Sensor Web Enablement (SWE) compliant applications.

### 8.2.1 Deployment options for the InterSensor Service

In the distributed working scenarios having many stakeholders, it is crucial to consider the interests of the stakeholders and types of their platforms before connecting them to the InterSensor Service. It is essential to determine whether (i) the platform is open or proprietary, (ii) the platform requires

<sup>122</sup><https://developer.twitter.com/en/docs>





**Figure 8.7:** Deployment of the InterSensor Service. An InterSensor Service can be deployed (a) by a user connecting to different data sources, as well as (b) by a stakeholder by setting up a trusted connection within same organisation.

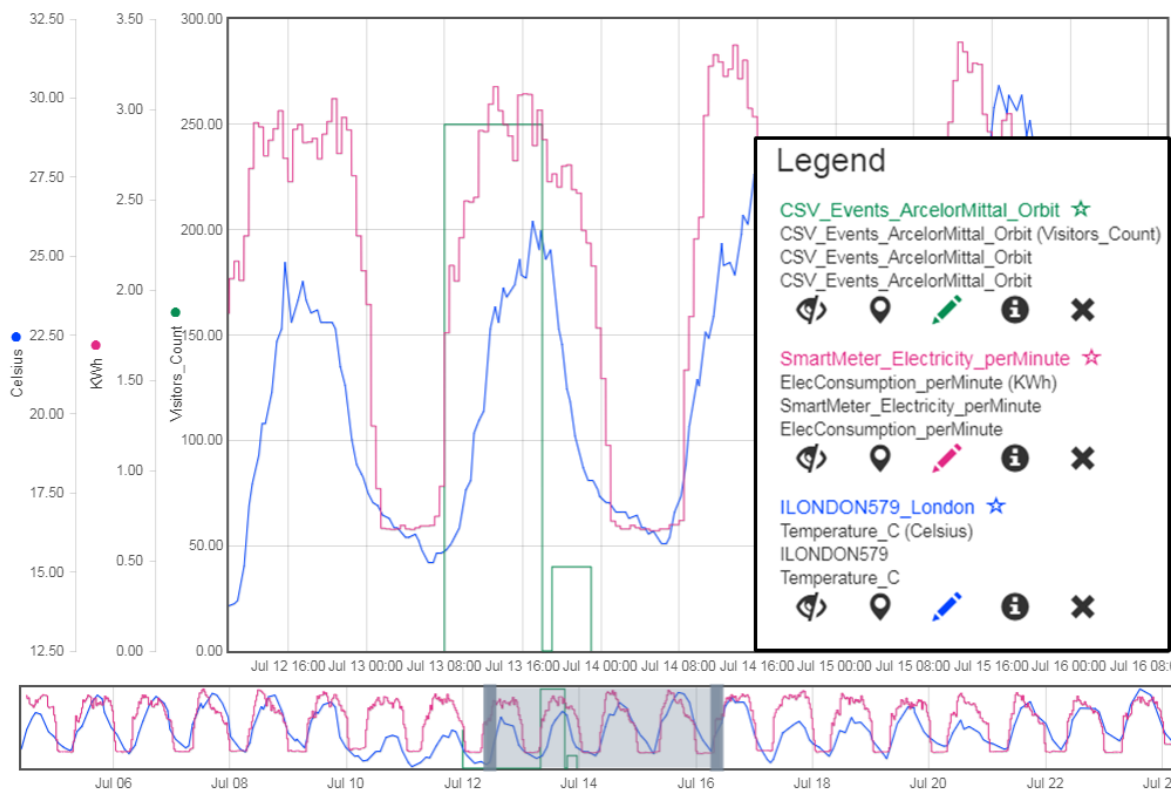
establishing trust by authentication mechanisms, (iii) the stakeholder is willing to share their information to all the users or only to a specific group of users, and so on. The InterSensor Service provides several deployment possibilities to meet the interests of different types of stakeholders. The InterSensor Service allows users to configure a data source connection by extending the existing or a new IoT platform by using simple Java classes. A medium skilled Java programmer is capable of implementing a new adapter within a day based on the already provided examples. It would allow the user retrieving sensor observations from all the connected data sources. The provision of the additional standardised interfaces by the InterSensor Service enables users to visualise and analyse the various sensor locations and observations within an application in a homogeneous and integrated way as shown in figure 8.7(a). Such implementations are ideal for scenarios where the involved platforms are open and do not require establishing a trusted, i.e. a secured, connection between the stakeholder and the user.

However, there might be scenarios when a data source, e.g. the C3NTINEL platform, is proprietary and contains confidential and secure information. In such cases, it is necessary to establish trust between the stakeholder and the user. The platform requires secure credentials which may be in the form of username/password or OAuth 2.0 access tokens. Due to privacy concerns, the stakeholder would like to avoid revealing the security credentials to the users of the InterSensor Service. In such cases, the respective stakeholder can configure an instance of the InterSensor Service by using the appropriate security credentials and allow real-time observations to be accessed by the standardised interfaces (as shown in figure 8.7(b)). In this case, without revealing the credentials to a user, the observations can jointly be analysed with other properties. In this way, it is also possible to configure an additional layer of security facade for providing the appropriate access control. This access control

layer allows the stakeholder to configure whether a set of users are allowed to retrieve the observations or not. Such additional security layers can be set up on the OGC based web services by using an approach proposed in Chapter 9.

### 8.2.2 Joint visualisation and analysis of heterogeneous sensor data

After establishing the connections to multiple heterogeneous data sources, the sensor data and observations could be retrieved according to the external interfaces such as OGC Sensor Observation Service, OGC SensorThings API, and 52°North Timeseries API. It allows applications supporting such OGC SWE interfaces to retrieve the sensor information being retrieved from multiple sensors and IoT platforms.



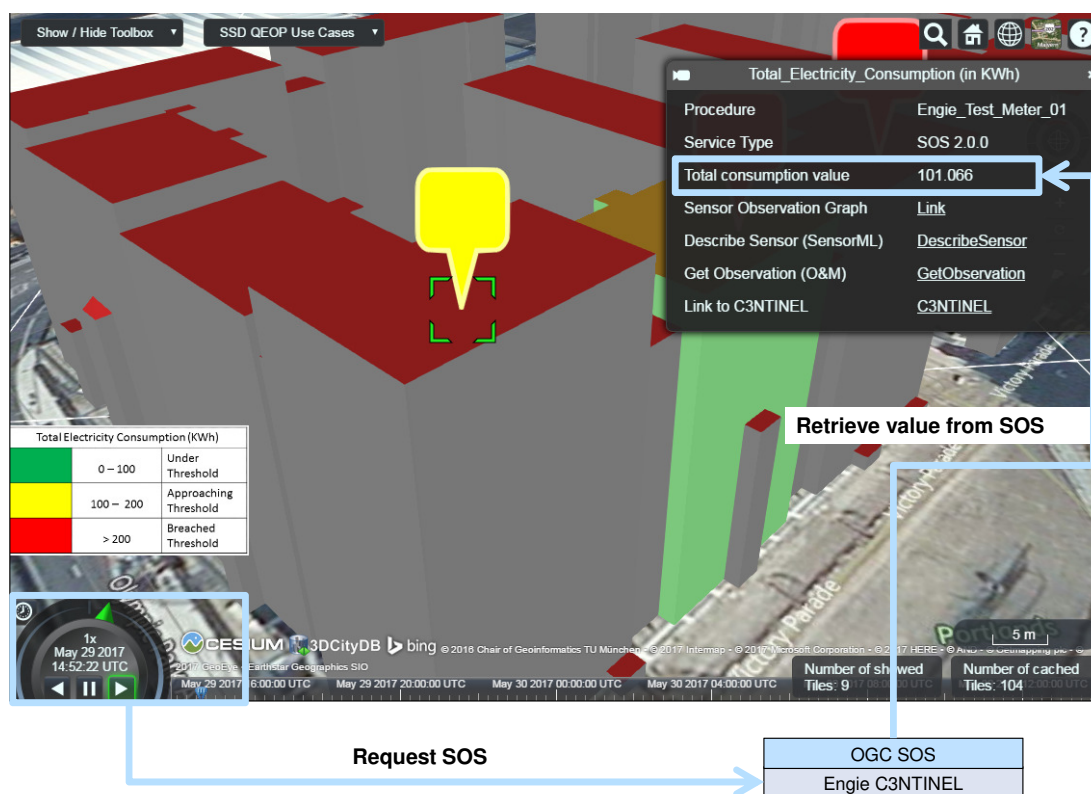
**Figure 8.8:** Joint visualisation of observations being retrieved directly from heterogeneous data sources: (i) electricity consumption from the proprietary C3NTINEL platform in pink, (ii) Outside Temperature readings from the Weather Underground platform in blue, and (iii) scheduled event and visitor count from a CSV file in green. Screenshot taken from the Helgoland web client application.

Figure 8.8 shows a screenshot taken from the Helgoland application developed for visualising and interacting with sensor data based on the 52°North Timeseries API. The interface from the InterSensor Service can directly be used with the Helgoland application allowing us to interact with observations being retrieved directly from a weather station (outside temperature retrieved from Weather Underground platform), Smart Meter located in a prominent building (electricity consumption per minute retrieved from the proprietary C3NTINEL platform) and scheduled events in the same

important building (visitor counts during the scheduled event retrieved from a CSV file). Such joint visualisations help to determine the correlation between different properties, e.g., *"what is the impact of the weather or any scheduled event on the electricity consumption of a building?"*. Of course, such a common standard-based API will also be very valuable for any other kind of application or analysis tool.

### 8.2.3 Integrated management and visualisation of static and dynamic properties

The real-time observations from different sensor platforms can also be visualised along with static properties of 3D city models. One use case in the project requires the demonstration of an Energy Management System with the 3D city model of the park. As mentioned earlier, Smart Meters, running on the Engie C3NTINEL platform, are installed in some of the buildings in the park. The administrators required a real-time notification system alerting to the fact that an energy efficiency threshold has been breached.



**Figure 8.9:** Real-time energy notification system for buildings within Queen Elizabeth Olympic Park. The green colour indicates that the total energy consumption is under threshold. The yellow colour indicates that the energy consumption is approaching a threshold. The red colour shows that the total consumption has breached the threshold. Screenshot taken from the 3DCityDB-Web-Map application.

As shown in figure 8.9, the green colour indicates that the total energy consumption is below the threshold. The yellow colour indicates that the energy consumption is approaching a threshold. The

red colour shows that the total consumption has breached the threshold. For this demonstration, the InterSensor Service was used to connect to the Engie C3NTINEL platform. The consumption values can be determined by the web application in regular intervals. The screenshot shows that the values are retrieved using the OGC Sensor Observation Service. However, using the InterSensor Service, other interfaces such as OGC SensorThings API can also be used. Similarly, an MQTT protocol can also be used in such scenarios for subscribing to the data stream and be notified as soon as the specified threshold is breached.



**Figure 8.10:** Joint visualisation of geo-tagged tweets retrieved by the InterSensor Service along with CityGML based 3D building objects in the district Queen Elizabeth Olympic Park, London. Screenshot taken from the 3DCityDB-Web-Map application.

For a different use case in the Queen Elizabeth Olympic Park, it is required to visualise real-time twitter feeds around the park to study sentiments and experience of visitors. Using the InterSensor Service, a secure connection to the Twitter API could be established to retrieve geo-tagged tweets around the park. The response, according to the OGC SWE interfaces, makes it suitable to be visualised together with other OGC standards compliant datasets. One such example is shown in figure 8.10, where the geo-tagged tweets are retrieved using the InterSensor Service and visualised along with the 3D city objects which are represented according to the OGC CityGML standard. This figure is a screenshot taken from the 3DCityDB-Web-Map application, which allows visualising and interacting with large-scale CityGML based objects directly within web browsers. The 3DCityDB-Web-Map

application is extended for this work to support the OGC SWE interfaces making the application more dynamic. In this way, arbitrary sensor observations can be visualised along with city objects to which they are associated with.



**Figure 8.11:** Joint visualisation of available rental car information being retrieved by the InterSensor Service along with CityGML based 3D building objects in the city of Augsburg in Germany. Screenshot taken from the 3DCityDB-Web-Map application.

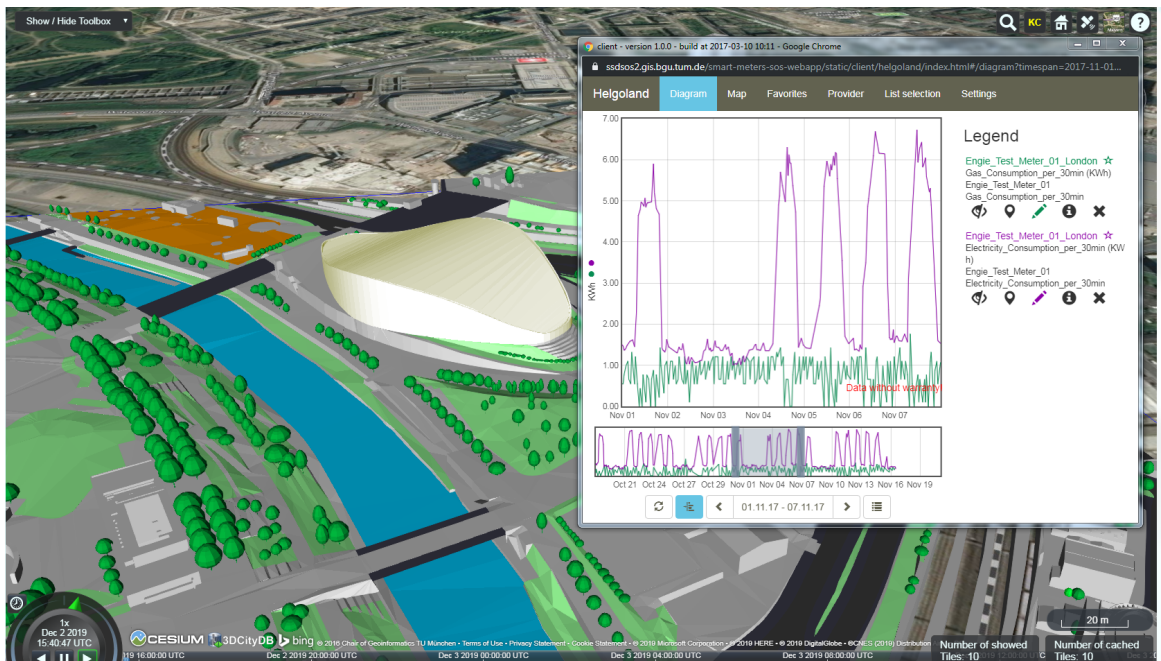
Another similar implementation has been done for the city of Augsburg in Germany where the InterSensor Service is used to connect to a proprietary car sharing application. The car-sharing application is based on an open interface called the Interface for X-Sharing Information (IXSI)<sup>123</sup>. With the help of its defined API, it is possible to retrieve information about available rental cars throughout the city in a real-time manner. The InterSensor Service is used to connect to this interface and retrieve the responses according to the OGC SWE interfaces. The standardised response by using the InterSensor Service made it suitable to be visualised along with CityGML based 3D objects using the 3DCityDB-Web-Map application (see figure 8.11).

#### 8.2.4 Easy deployment of interoperable solutions

Another use case within the project is to demonstrate the developed concepts within SDDI along with a highly detailed visualisation models of the Olympic Park in London. For this purpose, an external

<sup>123</sup><https://github.com/RWTH-i5-IDSG/ixsi>

visualisation solution, offered by the company VU.City<sup>124</sup>, is used. The use case involves in-depth discussions between the SDDI Demonstrator team and VU.City for identifying potential synergies, leading to two parallel approaches. The first approach is to bring the VU.City high resolution models on the already developed SDDI web-based platform. VU.City provides models in game engine formats such as Unity. By bringing the VU.City model on the existing SDDI web-based platform, the high resolution objects can be used with other thematic and real-time sensor information, as shown in figure 8.12. For this purpose, the VU.City model is converted into the CityGML standard using FME workbenches. It allows the models to be used with the 3D CityDB Web Map client, which is the primary viewer for 3D city models in the SDDI Demonstrator project.



**Figure 8.12:** Illustration of the SDDI functionalities with highly detailed models provided by external visualisation solution providers. The CityGML-based 3D model is converted from a game engine format provided by VU.City. Screenshot taken from the 3DCityDB-Web-Map application.

The second approach is to access the SDDI components by the proprietary VU.City platform. VU.City software runs as a standalone application (using a game engine like Unity). Hence, it needs to be installed locally on a machine. Advantages of the VU.City platform are that it can be used with gaming applications, Virtual Reality (VR) applications, and interactive touch screens. The platform can, thus, run some of the use cases with external park stakeholders, particularly those related to planning and public outreach. The solutions developed within the SDDI Demonstrator project utilise open, international, and interoperable standards. As discussed previously, the 3D city model is based on the OGC CityGML standard. Its thematic properties can be retrieved using the OGC Web Feature Service standard. The real-time information from numerous sensors, IoT platforms, and other data sources can be interpreted according to the OGC Sensor Web Enablement interfaces utilising the InterSensor Service. The standards and their respective implementations are open-source, and can

<sup>124</sup><https://vu.city/>

easily be deployed to the servers and cloud environments. In this way, the web services can easily be interpreted by the 3rd party proprietary software systems (VU.City platform in this case). As shown in figure 8.13, the thematic attributes of city objects can easily be retrieved by the VU.City platform.



**Figure 8.13:** Illustration of the SDDI functionalities with an external visualisation platform. The thematic attributes of the CityGML model are parsed and retrieved by the VU.City platform. Screenshot taken from the VU.City application.





## Chapter 9

---

### Securing Data Infrastructures for Smart Cities

Based on the open and international standards of the Open Geospatial Consortium (OGC), the Smart District Data Infrastructure (SDDI) concept integrates different sensors, IoT devices, simulation tools, and 3D city models within a common operational framework. However, such distributed systems, if not secured, may cause a significant threat by disclosing sensitive information to untrusted or unauthorised entities. Also, various users and applications prefer to work with all the systems in convenient ways using Single-Sign-On. This chapter presents a novel concept for securing distributed applications and services in such data infrastructures for Smart Cities. The concept facilitates privacy, security and controlled access to all stakeholders and the respective components by establishing proper authorization and authentication mechanisms. The approach facilitates Single-Sign-On (SSO) authentication by a novel combination in the use of the state-of-the-art security concepts such as OAuth2 access tokens, OpenID Connect user claims and Security Assertion Markup Language (SAML). This chapter shows an implementation of the concept for the district Queen Elizabeth Olympic Park, London, U.K. All of the implemented components are now available as Open Source software at <https://github.com/tum-gis/sddi-security-federation-framework>. The implementation is also available as an online demonstration at <https://www.3dcitydb.org/demos/sddi-security-demo>.

This chapter is based on published papers and is a joint effort by the authors mentioned as follows:

**Chaturvedi, K.**, Matheus, A., Nguyen, S. H. and Kolbe, T. H. (2019). ‘Securing Spatial Data Infrastructures for Distributed Smart City applications and services’. In: *Future Generation Computer Systems* 101, pp. 723 –736. URL: <http://www.sciencedirect.com/science/article/pii/S0167739X18330024>

**Chaturvedi, K.**, Matheus, A., Nguyen, S. H. and Kolbe, T. H. (2018). ‘Securing Spatial Data Infrastructures in the Context of Smart Cities’. In: *2018 International Conference on Cyberworlds (CW)*, pp. 403–408. URL: <https://doi.org/10.1109/CW.2018.00078>

## 9.1 Securing the Smart District Data Infrastructure (SDDI)

Smart City data infrastructures such as the SDDI can increase productivity and efficiencies for citizens and governments. However, they may have a severe problem when they lack proper security mechanisms. Smart City solutions can facilitate access to sensitive information from different stakeholders and citizens, and hence are vulnerable to information and privacy leakage by outside attackers (Zhang et al. 2017). For example, Smart Meters and other types of ubiquitous sensors, pedestrian/traffic movement, simulation databases must be considered confidential data. It would cause a significant threat to disclose such information to untrusted or unauthorised entities in both the physical and communication worlds. Another challenging issue is data sharing and access control. For example, within a common infrastructure with various stakeholders, it is crucial to establish appropriate access policies and enable privacy-preserving data sharing among the collaborators. It also requires proper identity and privacy management to authorise only trusted users to access the system (Bartoli et al. 2011). As local governments pursue Smart City initiatives realising the full potential of these digitally connected communities, it is critical to implement security best practices by extending existing systems. Partners and stakeholders will only conduct business if their rights, trust and security requirements are met. There are also several other studies, such as proposed by (Cui et al. 2018; Sookhak et al. 2019; Gharaibeh et al. 2017; Biswas and Muthukkumarasamy 2016), which highlight various security and privacy-related issues in the context of Smart Cities.

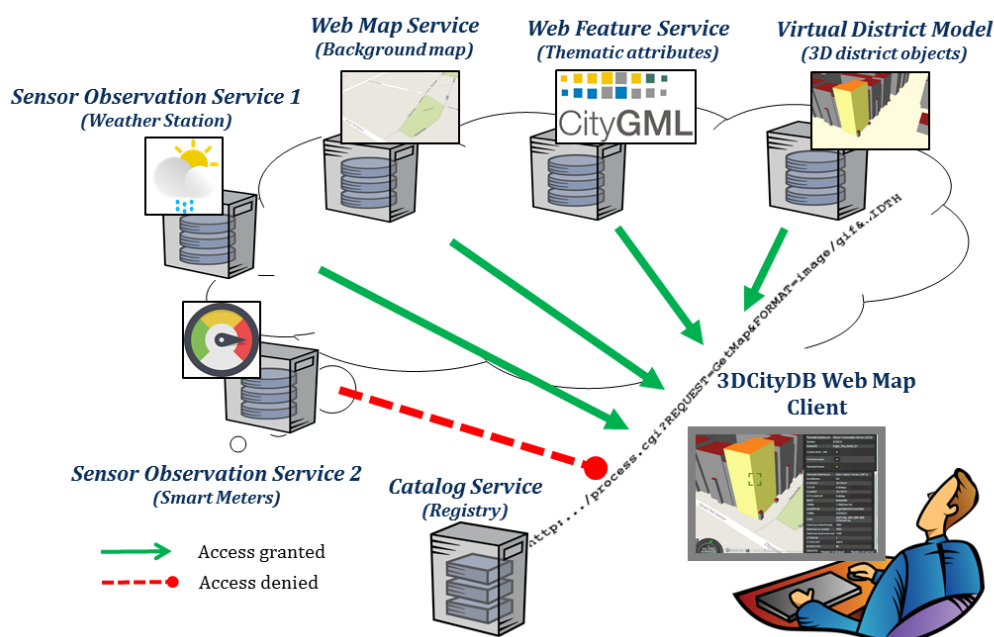
This chapter identifies key requirements of developing and securing Spatial Data Infrastructures (SDI) for Smart City scenarios based on the proposed Smart District Data Infrastructure (SDDI) framework. Furthermore, it presents a novel concept for securing the data access and integration of distributed Smart City applications, services, simulation and analytical tools, sensors and IoT devices, and geographic information to meet the identified key requirements. The concept facilitates privacy, security and controlled access and provides ways to authorise and authenticate these distributed components without the need for repetitive logins. At the highest level, the approach combines the use of modern standards such as OAuth2 (Hardt 2012) access tokens, OpenID Connect user claims (Sakimura et al. 2014) and Security Assertion Markup Language (SAML) (Cantor et al. 2005) based Single-Sign-On (SSO) authentication. The combination of such best practice security standards also enables easy integration with external authentication services such as publicly accessible providers like Google and Facebook as well as with Academic Federations (allowing the solutions to be used by academic users). For modern, security-aware spatial data infrastructures, this is a state-of-the-art concept.

The chapter also shows a demonstration implementation of the concept for a specific scenario carried out within the district Queen Elizabeth Olympic Park in London. The demonstrator application is conformant to the EU General Data Protection Regulation (GDPR)<sup>125</sup> and allows to link different components such as 3D buildings with semantic information, weather stations, and Smart Meters in open, standardised and secure ways. This demonstration application supports individual access rights for different types of user groups including (i) public Google account, and (ii) academic users from universities and research institutes. It allows handling of various identity providers.

As described in Chapter 8, since SDDI is a complex distributed system involving heterogeneous resources, this demonstration aims to establish a proper security layer for all the components to ensure authorisation, authentication and Single-Sign-On capabilities. Such security layer enables secure and

---

<sup>125</sup><https://eur-lex.europa.eu/eli/reg/2016/679/oj>



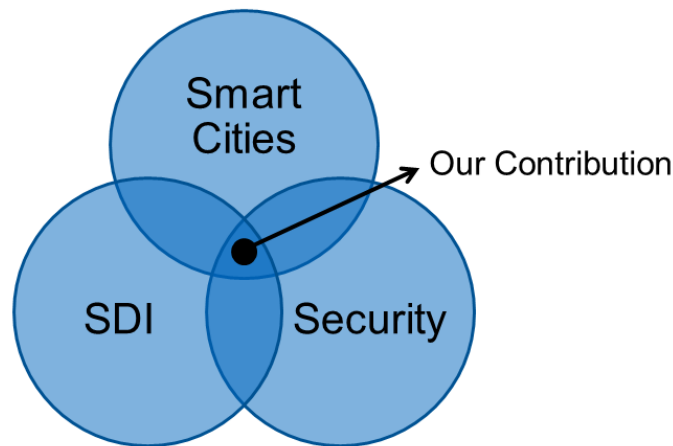
**Figure 9.1:** Illustration of secure and controlled access to the distributed applications and services within the SDDI framework.

controlled access to the distributed applications and services, as shown in figure 9.1. For illustration purposes in a simplified scenario, only a small subset is shown as follows:

- **Virtual District Model** based on the CityGML standard. It comprises of semantic 3D building and street models with spatial and thematic information stored in a 3D geodatabase.
- **Web Feature Service** allowing users to retrieve as well as modify objects from the Virtual District Model using interoperable interfaces.
- **Sensor Observation Service 1** retrieving real-time observations from a weather station installed in the park. The weather station records properties such as temperature, humidity, wind speed etc.
- **Sensor Observation Service 2** retrieving real-time observations from Smart Meters installed in important buildings such as the stadium and the Aquatic Center. The Smart Meters are managed within a proprietary platform of the company Engie and record electricity and gas consumptions for the buildings.
- **3DCityDB Web Map Client** is a web-based front-end for the 3D City Database for 3D visualisation and interactive exploration of large semantic 3D city models in CityGML.

## 9.2 Gathering requirements for securing the infrastructure

Before establishing such security layers, this chapter identifies requirements which are critical for securing the SDIs for Smart Cities. The focus of this chapter lies in the intersection of the fields Smart Cities, Spatial Data Infrastructures (SDI) and Security (see figure 9.2). Hence, the significant requirements considered for this study are as follows:



**Figure 9.2:** Venn Diagram illustrating the key focus of the research contribution.

### 9.2.1 Smart Cities

**Requirement SR1: Different stakeholders.** Typically, Smart City infrastructures involve distributed systems which may have different stakeholders or end-users such as citizens, municipalities, utility and transportation service providers, real estate firms etc. These stakeholders are usually the group of people and organisations for which the infrastructure offers services and applications. The infrastructure must consider the needs and requirements of these different stakeholders, and as a consequence, not all data can and will be stored/maintained within a single system/platform.

**Requirement SR2: Distributed applications.** It should be possible for stakeholders to register and interact with distributed applications. These applications usually implement the logics according to specific tasks and make use of different sets of data, sensor observations or simulation results involved, for example, City Dashboards, Energy Portals, Mobility Applications, and Disaster Management Portals.

**Requirement SR3: Simulation/Analytical Tools.** There may be simulation tools or analytical toolkits, which are software components developed for specific scenarios. These scenarios may include, for example, estimating the energy demands or potentials of solar energy production for all buildings, simulating road traffic and pedestrian flows, or performing noise propagation or flooding simulations. The results of these simulations can not only be provided to the applications but also be used for planning and forecasting. Furthermore, the results of one simulation can be used by multiple applications, or one application can use results from various simulations. Hence, such simulation tools should be registered and operated separately from the applications.

**Requirement SR4: Sensors and IoT.** Ubiquitous sensors and IoT devices are essential parts of several Smart City infrastructures providing detailed information by (real-time or near real-time) sensing the environment. These sensors can be stationary such as Smart Meters and weather stations. Some of the sensors can also be mobile such as moving sensors for measuring air quality. It is essential to register such sensors and IoT devices in the infrastructure enabling their observations to be integrated with applications or analytical tools.

**Requirement SR5: Inclusion of geographic information.** Nearly all Smart City concepts focus on mainstream Information and Communication Technologies (ICT) such as the Internet of Things (IoT), Big Data, Cloud Computing, and so on. However, it is also important to consider geographic

information as a key element. Many of the simulations or planning scenarios for the cities need to work with models of the physical reality. Hence, semantic 3D city models (Kolbe 2009) act as an important complementary asset. These 3D city models represent both spatial and semantic information of physical objects such as buildings, roads, water bodies etc. Furthermore, semantic 3D city models provide a means for interactive and spatio-semantic queries and aggregations. It is important to consider other geographic data such as maps and coverages too, but also Building Information Models (BIM) (Borrmann et al. 2015).

### 9.2.2 Spatial Data Infrastructures (SDI)

**Requirement SR6: Interoperability.** To deal with the different and heterogeneous data, applications, sensor and IoT devices, and simulation tools within a common operational framework, the interoperability over various connected components and systems is essential. Interoperability facilitates accessing and retrieving data, services, and applications by using standardised and, therefore, stable interfaces.

**Requirement SR7: Open International Standards.** The information models and interface models must be based on Open Standards adopted internationally, for example, standards issued by the Open Geospatial Consortium (OGC). In the case of non-standardised Open APIs and models, there is a high risk that the encodings/APIs will be abandoned, replaced by, e.g. big Internet Companies, or vanish after the project that suggested them is over.

**Requirement SR8: Linked Components.** The use of standardised interfaces such as the ones issued by the OGC also allows managing and accessing different components linked to each other. Dealing with such linked components is also an essential requirement for a distributed system. For example, if a Smart Meter is installed in a building, a web service (such as the Web Feature Service) can retrieve the building's semantic information. This semantic information may further include a link to the running web service (such as the Sensor Observation Service) of the Smart Meter, which is measuring real-time gas consumption.

### 9.2.3 Security

**Requirement SR9: Authentication and Authorisation.** In a complex distributed infrastructure, the most basic requirement is to protect access to the data and functionalities. Thus, authentication and authorisation of users play an essential role. The term authentication means that an individual identifies himself/herself unambiguously. Typically, a username and password are used for authentication. Authorisation describes the process of checking whether a user has access rights to a specific resource. However, it is not practical to use different login credentials for various resources.

Hence, modern standards, such as OAuth2 (Hardt 2012) are used to secure applications. OAuth2 allows enabling access delegation from the resource owner (i.e. user) for a trusted application to access the protected user resources without disclosing the master credentials. It leverages access tokens for the actual access delegation aspect. OAuth2 is considered to be state-of-the-art for web and mobile applications and is supported by numerous big players of Web 2.0 (e.g. Twitter, Google, and Facebook). However, authentication and exchange of user assertions are out of scope for the OAuth2 framework.

**Requirement SR10: User Information.** Another important aspect is user privacy. The OpenID Connect community standard (Sakimura et al. 2014) has been designed as an extension to the OAuth2 framework to be able to link user assertions (user claims) with access tokens. The granularity of personal information included in the claims, linked to an access token, depends upon the user's

approval, which is a crucial aspect of being compliant with user privacy. Moreover, the implementation using OpenID Connect allows the easy integration with external authentication services such as Google and Facebook making the application suitable for usage involving plenty of users worldwide.

**Requirement SR11: Single-Sign-On.** In the cases of distributed systems where resources are linked, setting up a security facade for each component is cumbersome. It is not user friendly to authenticate every interface separately. Thus, it is an essential requirement to have the Single-Sign-On (SSO) functionality, which allows a user to access different applications and services without the need for repetitive logins.

An example of achieving Single-Sign-On is by the unique identification of users in a distributed system, for example, as implemented in Academic Federations like eduGAIN<sup>126</sup>. The eduGAIN federation is based on the international standard Security Assertion Markup Language version 2 (SAML2) (Cantor et al. 2005), which is an OASIS standard to define assertion structures and protocols for exchanging assertions about users between trusted entities in a distributed system. The asserting party is the Identity Provider (IdP), and the relying party is the Service Provider (SP). Attribute assertions allow exchanging personal information about a user and authorisation assertions can describe the access rights of a user on a given resource.

**Requirement SR12: Single-Sign-On with delegated authorisation.** Modern standards such as OAuth2 already support delegated authorisation allowing a trusted application to access a protected resource without disclosing the master credentials. However, to achieve Single-Sign-On, federated authentication is required, which is not supported by the OAuth2 framework. Hence, it is essential to integrate OAuth2 with the popular standards for federated authentication such as SAML2 and OpenID. An extension to the OAuth2 framework is already available as OpenID Connect user claims, allowing easy integration with authentication services like Facebook and Google. However, large Academic federations such as eduGAIN are based on SAML2. Hence, it is necessary to combine SAML2 authentication with the OAuth2 Authorisation Server.

This powerful combination enables to operate an OpenID Connect compliant Authorisation Server to honour the needs for modern security and web applications but also create and maintain an identity federation as operated in Academic Federations worldwide each day with hundreds of millions of users.

To the best of the knowledge of the author of this thesis, no research work fulfils all the requirements listed previously. A comprehensive literature review for this aspect is already given in the published paper (Chaturvedi et al. 2019). The modern standards such as SAML, OAuth, and OpenID are being used in different studies for authorisation, authentication, and Single-Sign-On capabilities. However, it is equally crucial to ensure that such security mechanisms can be established for distributed and heterogeneous resources in a unified and standardised way.

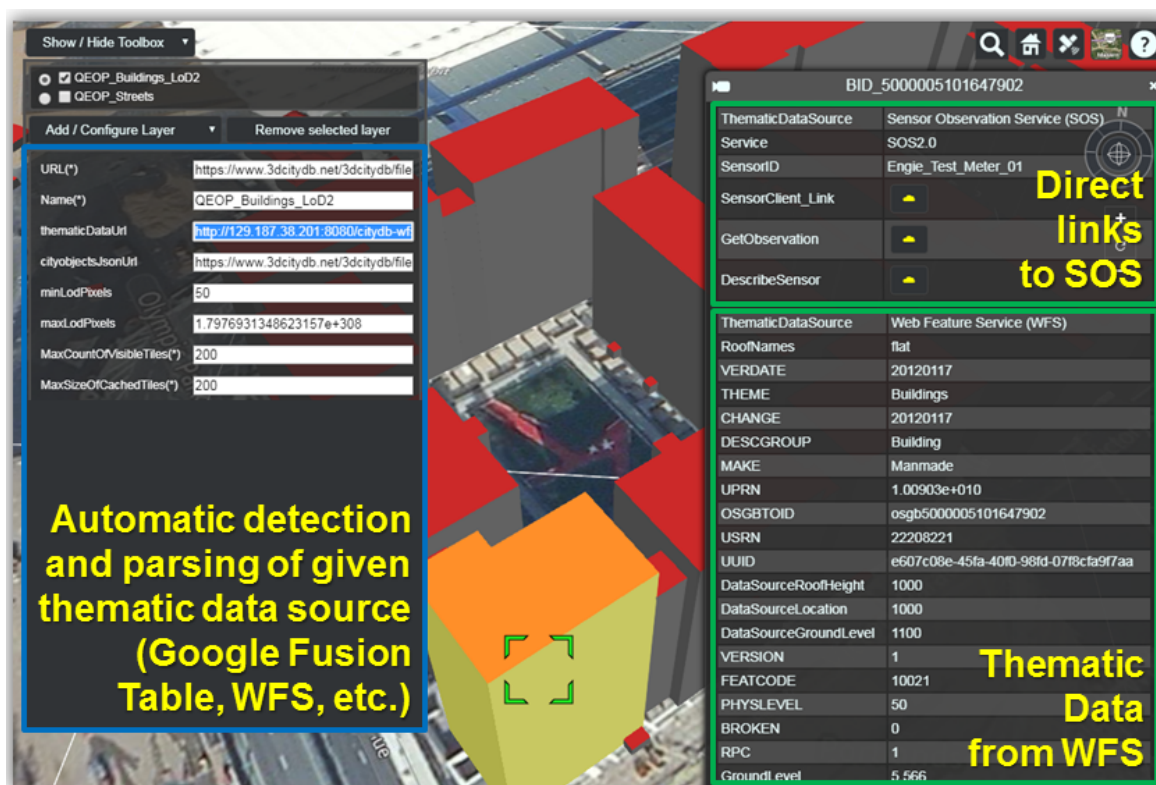
### 9.3 Demonstration scenario for securing the SDDI framework

Based on the requirements listed in the previous section, this chapter focuses on securing the Smart District Data Infrastructure (SDDI) framework implemented in the Queen Elizabeth Olympic Park, London. In this scenario (as shown in figure 9.3), all the services and resources are combined in an integrated application within the 3DCityDB Web Map Client (Yao et al. 2018). The building objects of the Olympic Park are represented according to the CityGML standard. When the user clicks on a building, its thematic data such as the building name and address are retrieved from

---

<sup>126</sup><https://edugain.org/>

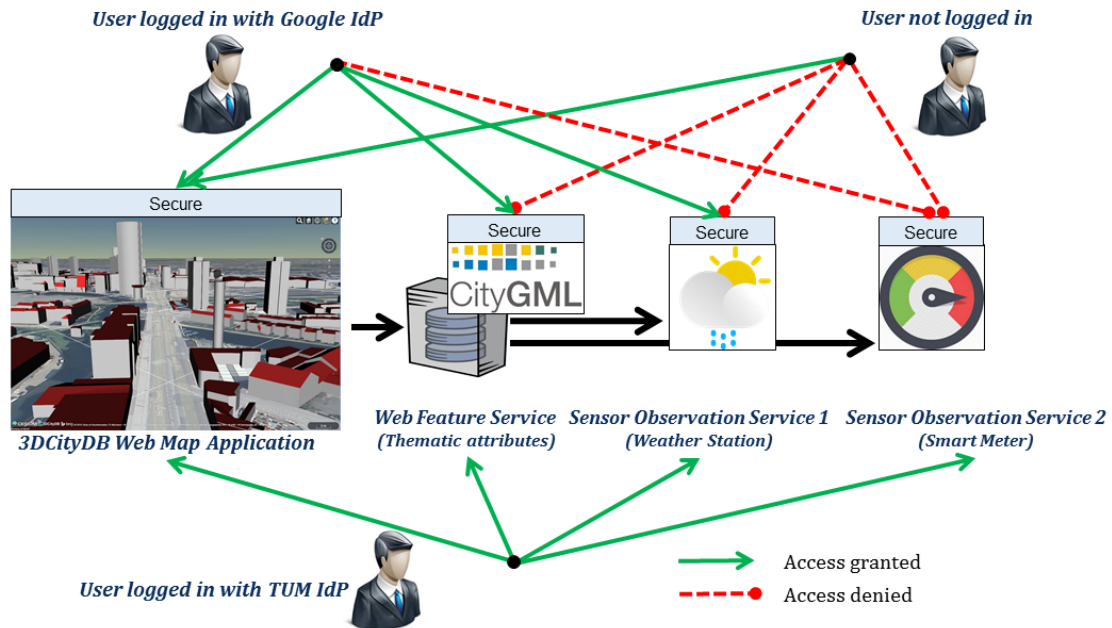
the Web Feature Service (WFS). The WFS response also includes direct links to both the Sensor Observation Services giving access to Smart Meters and weather stations. For accommodating the security demonstration scenario, the infrastructure involves multiple distributed resources which are linked together in different ways.



**Figure 9.3:** Representation of chaining of distributed resources in the SDDI framework. Image taken from (Chaturvedi et al. 2019).

This concept aims to fulfil the security requirements [SR9-SR12] by providing

- security layers to all of the resources, so that no resource can be accessed without proper authentication and authorisation,
- federated login and Single-Sign-On access using different Identity Providers. This functionality is demonstrated by showing that users can log in using academic identity federations (such as eduGAIN service supporting approximately 2758 university identity providers worldwide) and public accounts (such as Google accounts) to all the secured resources hosted on distributed systems without repetitive logins, and
- access control to all the secured resources. Users can log in via two different classes of identity providers: (i) a valid public account (e.g. Google) and (ii) a valid academic organisation account linked to eduGAIN. In the scenario, if a user is not logged in, he or she can view the 3D models but cannot connect to any further resource. Users logged in using the Google Identity Provider can access all resources except Sensor Observation Service 2 for Smart Meters. In contrast, users logged in using an eduGAIN based research organisation's Identity Provider will be able to access all resources. In the illustrations, users from the Technical University of Munich (TUM), also linked to eduGAIN, can access all the resources (see figure 9.4).



**Figure 9.4:** Illustration of the security demonstration scenario showing that users identified by different identity providers can access the distributed components. Green arrows mean "Access Granted" and red dashed arrows mean "Access Denied" to specific components.

## 9.4 Implementations

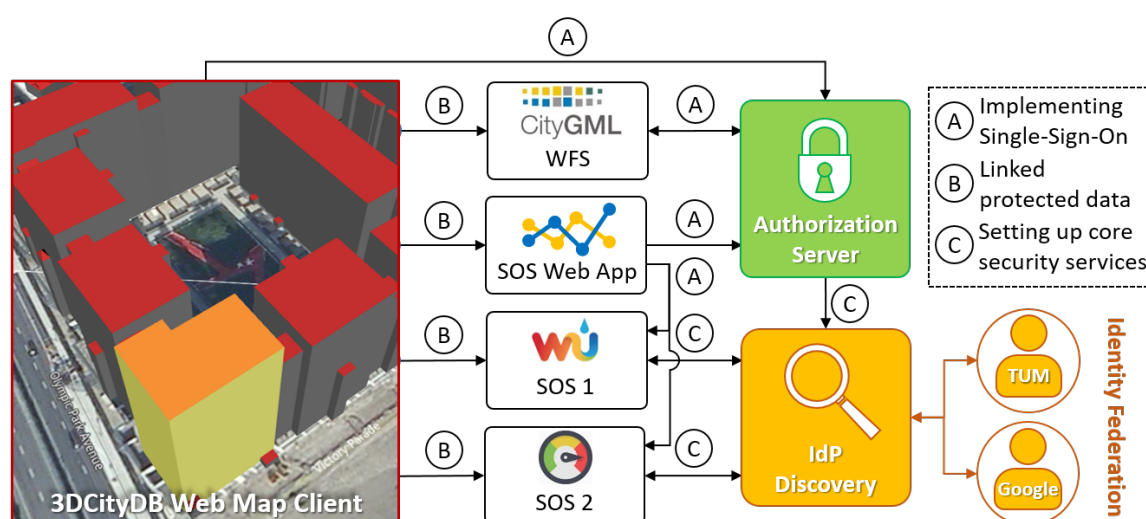
Figure 9.5 gives an overview of the security demonstrator architecture. For simplicity, the figure illustrates only the relevant components including (i) the 3DCityDB Web Map Client, (ii) Web Feature Service, (iii) Sensor Observation Service 1 for a weather station, and (iv) Sensor Observation Service 2 for a proprietary Smart Meter platform. However, other applications and web services can also be secured by using the steps mentioned in the following sub-sections.

### 9.4.1 Implementing Single-Sign-On

SAML2 specifies a web browser Single-Sign-On (SSO) Profile which involves an Identity Provider (IdP) and a Service Provider (SP). As described by (Cantor et al. 2005), the session initiation is triggered by the Service Provider (SP) based on HTTP redirects. This way of initiating a session is limited to native web browser interactions. It is important to note that redirects require that a session cookie is transported with the interactions. The first request from the client to the SP creates a temporary session which gets referred to in a cookie. It is also essential that this cookie is sent on the second redirect to ensure that the SP can create a real session and issue another cookie, referencing the full session. Afterwards, the session is referred to by all requests initiated by the web browser that contains the session cookie.

As we know, the 3DCityDB Web Map Application is written in the JavaScript programming language. Trying to adopt this SAML2 protocol behaviour for a JavaScript-based web application may result in a conflict with the Same Origin Policy. This policy safeguards the content loading in a web





**Figure 9.5:** An overview of the security demonstrator architecture. The notations "A", "B" and "C" in the figure refer to sections 9.4.1, 9.4.2 and 9.4.3 respectively. Image taken from (Chaturvedi et al. 2019).

browser by intercepting network requests initiated by JavaScript and XMLHttpRequest object or Asynchronous JavaScript and XML (AJAX). Regardless of the technology, the web browser verifies the conditions under which the network request is initiated based on the W3C Cross-Origin Resource Sharing (CORS) (Kesteren 2014) recommendation. Without going into details, any JavaScript application gets loaded from a web server of which its hostname is considered the origin of the code. If a network request gets initiated to another hostname which is not in the same domain or sub-domain, the receiving web server must reply with particular HTTP headers. According to the W3C CORS recommendation, the origin changes to the literal 'null' after any HTTP redirect. It means that for the SAML2 session initiation via the web browser SSO Profile, the redirect ending at the IdP will carry origin 'null'. It disables the intended use of that HTTP header, i.e. determining the trust of the JavaScript code based on the hostname from which the code is loaded. Based on 'null', the typical whitelisting can no longer be applied. Therefore, the IdP blindly trusts the redirected request, which it should not. At this point, the interaction to initiate a new session with the SAML2 SP fails to assume a proper validation of the origin.

The OAuth2 framework general protocol (Hardt 2012) suggests that the session initiation is different compared to SAML2. In particular, no two-way HTTP round-trip is required to instantiate a new session. A session is referred to via an access token. The application does interact with the Authorisation Server to obtain an access token leveraging one of the different protocols (grant types) that are designed to work well with the web browser Same Origin Policy and web applications. Once the application has received an access token, it can be used for any calls to the protected resources hosted at the Resource Servers.

### 9.4.2 Linked Protected Data

As shown in figure 9.4, the major interaction takes place between the user and the 3DCityDB Web Map Client. The client loads the CityGML based 3D city model and renders the information. Also,

the application extracts URLs to additional resources (such as the Web Feature Service and Sensor Observation Services), linked from the CityGML response. For example, the 3D building model data contain the links to those sensor services giving access to the sensors operated within the building. However, these links to other resources are protected. Such a connection cannot contain any security context as that would be insecure. Therefore, it must either be the application or the web browser that can add information to the link when it is followed. At this point, it is crucial to know whether the web browser or the web application is going to follow the link. The first link is from the rendered 3D model to additional information about each building. This information is available to the user by clicking on each building. The 3DCityDB Web Map Client initiates the network call, which causes the web browser to inspect the call towards CORS. This is the first fact to note when implementing the security to the Web Feature Service (WFS).

#### 9.4.2.1 Securing the WFS Interface

The securing of the WFS must leverage OAuth2 access tokens as the 3DCityDB Web Map Client follows the link to fetch the information from the WFS. Therefore, the WFS must be protected as an OAuth2 Resource Server (RS) accepting OAuth2 access tokens. The interface behaviour for an RS is defined in the OAuth2 Bearer Token Usage (Jones 2012). According to that specification, the RS must accept the access token either as part of the URL (parameter `access_token`) or as part of the HTTP header named `Authorisation` using the scheme `'Bearer'`. After the access token is isolated from the incoming request, the RS must validate the access token. Because access tokens are of type `bearer`, the RS must request validation by the Authorisation Server (AS) that issued the token. For supporting this interaction in an interoperable fashion, the AS for this prototype implements the OAuth2 Token Introspection (Richer 2015).

Assuming the RS has successfully verified the access token, it could undertake access control based on the token metadata received from the introspection endpoint or based on the user information that the RS can request from the OAuth2 assuming it is OpenID compliant. For the implementation of this prototype, no further access control is implemented. It means that the detailed building information from the WFS can be obtained only by the authenticated users.

#### 9.4.2.2 Securing the SOS Interface

The next level of linking is based on the links included in the WFS response: the `FeatureCollection`. Each geographic feature contains detailed information about a building including different types of links, but all are pointing to the protected endpoints. For this level of linked data, the URIs can resolve to different resource types. The first kind of link would return a web application which is used to visualise sensor readings. The second kind of link would return the responses of sensor description and sensor observations in simple XML format.

The link that refers to the sensor visualisation application (Helgoland Client, in this case) must be resolved directly by the web browser. Therefore, the SAML2 session initiation must be implemented on this endpoint. The link that refers to sensor observations connects to an OGC SOS initiating the *GetObservation* operation. The SAML2 session management is also sufficient for this sensor visualisation application, as it is loaded from the same hostname and path as the actual sensor readings. In the general case, where the sensor visualisation application and the sensor readings are not hosted on the same machine, the session and access management can be based on OAuth2 access tokens.

The access controls were implemented based on the login origin of the user. As described earlier, when a user is logged in using a Google account, he or she can access only the sensor reading from Sensor Observation Service 1. When the user is logged in using an arbitrary eduGAIN account (like the user account from TU Munich), he or she can access sensor readings from both the Sensor Observation Services.

### 9.4.2.3 Modifying the Web3D Application

Based on the chosen security for the WFS and SOS interfaces, the 3DCityDB Web Map Client need only to be enabled to use OAuth2. It can be achieved by integrating any open source library that supports OAuth2. The library chosen in this work is HelloJS<sup>127</sup>. The application is registered to enable the 3DCityDB Web Map Client to obtain access tokens from the Authorisation Server. Since the application is considered 'non-confidential', it must leverage the OAuth2 Implicit Grant.

### 9.4.2.4 Modifying the Sensor Visualisation Application

This application was not modified, as SAML2 session instantiation is implemented with the SOS interfaces. As discussed, the session initialisation is done by the web browser itself following the SAML2 SSO web browser profile when loading the application. Once the session is established, the application is loaded and can then fetch sensor readings from the same SOS leveraging the existing session referenced by the HTTP cookie.

## 9.4.3 Setting up the core security services

In addition to adapting the web application and the services to support the required security interfaces, there need to be 'core' services as illustrated in fig. 9.5. First of all, there needs to be Identity Providers to allow the user to log in with, e.g. Google and TUM. For the latter one, the TUM Identity Provider registered with eduGAIN is used. The IdP for supporting Google login is a SAML2 gateway that is based on a standard SimpleSAMLPHP<sup>128</sup> deployment which is Open Source. In case there is more than one IdP, an IdP Discovery Service must implement the SAML2 IdP Discovery Profile. To support Single-Sign-On, the central Discovery Service<sup>129</sup> is used. A Coordination Centre is also created that maintains and signs the SAML2 metadata representing this mini federation. The IdP bridge is registered as an application with Google in order to work with the Google IdP bridge.

The protection of the SOS is based on the Shibboleth Service Provider implementation that is open source and also commonly used in the Academic Federations around the world.

All of the deployments for creating the federation are Open Source and are available at the GitHub repository <https://github.com/tum-gis/sddi-security-federation-framework>. The basic OAuth2 / OpenID Connect library, available from Github (Shaffer 2018), was extended to support the SAML2 federation login.

Setting up the Resource Server to protect the access to the WFS, a typical web server stack is used: the Internet-facing web server is an Apache that is also configured to support HTTPS. It is important to note that all communication is via HTTPS. The actual services got deployed on Tomcat, the defacto default hosting for Java-based services beside Jetty and JBoss. For protecting the service endpoints, a

<sup>127</sup><https://github.com/MrSwitch/hello.js>

<sup>128</sup><https://simplesamlphp.org/>

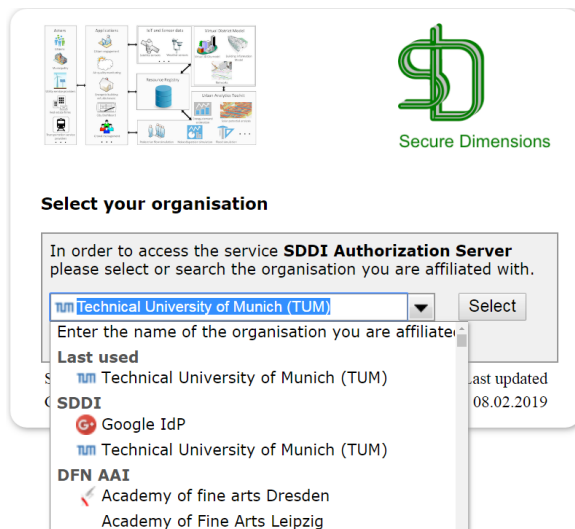
<sup>129</sup><https://www.switch.ch/aai/support/tools/wayf/>

simple PERL handler is created that implements the OAuth2 Bearer Token Usage (Jones 2012). The handler, loaded as an Apache module, intercepts service requests and interacts with the Authorization Server to validate the received access token.

Finally, the support for the W3C CORS recommendation is implemented as another module inside the Apache 2.4 deployment. For returning HTTP headers to support CORS, the whitelisting for the JavaScript code is not used. The reason is that the service endpoints are protected as OAuth2 Resource Servers and can only be accessed with a valid access token. However, one must keep in mind that an HTTP request with submitting the access token as an HTTP Bearer header causes the web browser to execute a so-called pre-flight request to check the response headers before actually submitting the intercepted request. The pre-flight request is an HTTP OPTIONS request, and even though a GET and POST request requires an access token, the Options request does not. This specific CORS behaviour was configured into the Apache webserver.

## 9.5 Illustration of the Concept

Based on the methodology and implementations described in the previous section, the security and access control layers are successfully set up on all implemented resources. The secured interfaces are developed for the 3DCityDB Web Map application, the 3DCityDB Web Feature Service (WFS), Sensor Observation Service (SOS1) for the weather station, and Sensor Observation Service (SOS2) for a proprietary Smart Meter platform. The additional security facades allow ensuring that (i) no resource can be accessed without proper authentication, (ii) federated login and Single-Sign-On access to all the secured resources hosted on distributed systems with one login, and (iii) access control with proper rights, roles, and grants to all the secured resources.



**Figure 9.6:** Selection of the appropriate Identity Provider to access the resources.

The Security demonstrator described in this chapter is publicly available<sup>130</sup>. The application utilises the powerful combination of SAML2 and OAuth2. It enables to operate an OpenID Connect compliant Authorisation Server to login using valid public accounts (in this case, Google). At the same time, it

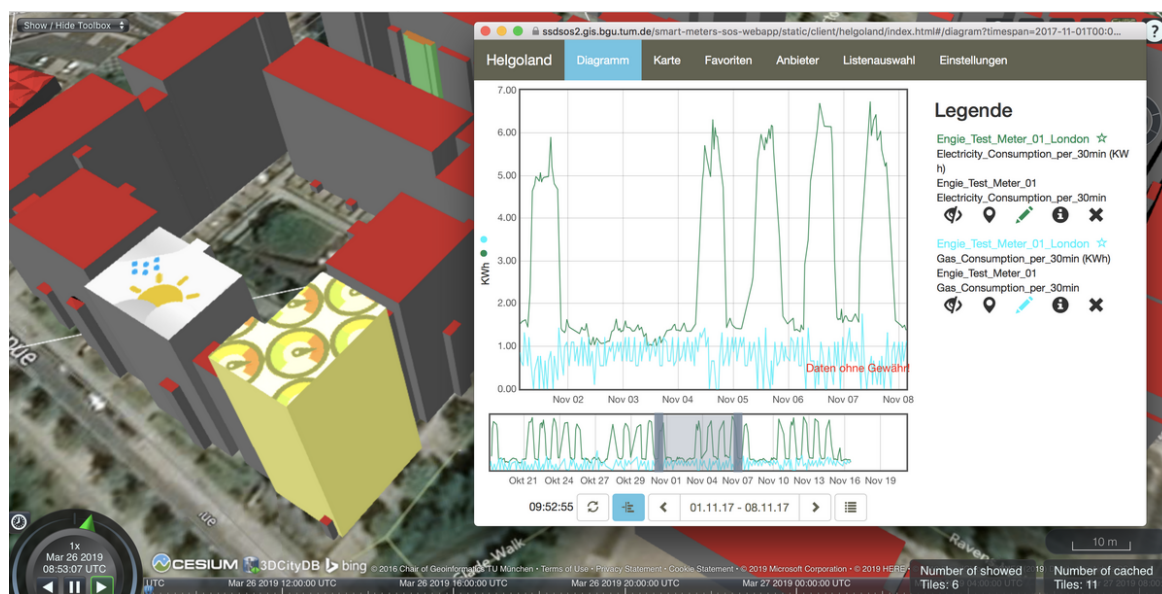
<sup>130</sup>[www.lrg.tum.de/en/gis/projects/smart-district-data-infrastructure/#c604](http://www.lrg.tum.de/en/gis/projects/smart-district-data-infrastructure/#c604)

also creates and maintains an identity federation as practised in Academic Federations worldwide (e.g. organisational accounts through eduGAIN service) as shown in figure 9.6.

The login access can then be provided based on required authorisation. When a user logs in using a specific credential, based on the valid authentication and authorisation rules, access tokens are generated for the protected services. The Resource Server requests validation by the Authorisation Server that issued the token. Upon successful validation of the access token, appropriate access control is given based on user information that the Resource Server requests from the OpenID Connect User Information endpoint which is a part of the OAuth2 Authorization Server operated for the demonstration. In the case of the WFS, there is no further access control, which means any authenticated user can obtain building information from the WFS. In the cases of SOS1 and SOS2, access controls were implemented based on the login origin of the user.

According to the proposed scenarios, the demonstrator application showcases three scenarios:

1. When a user is not logged in, he/she can view the 3D city model, but cannot connect to the protected WFS and further SOS1 and SOS2 to retrieve thematic and sensor data.
2. When a user is logged in using a valid Google account, he/she can access the WFS and SOS1 (which is a service running on a public weather station). However, SOS2 (which is a service running on a proprietary Smart Meter) is not accessible to this user group.
3. When a user is logged in using an organisational account (in this case any account supported through eduGAIN service), he/she can access the WFS as well as both SOS1 and SOS2 as shown in figure 9.7.



**Figure 9.7:** SOS2 can only be accessed by the user with a valid eduGAIN login.

Upon successful validation of credentials and access control roles, the user can connect to the respective resource and retrieve the information. In this way, the application fulfils the Requirements [SR9-SR12]. It ensures (i) that no resource can be accessed without proper authentication, (ii) federated login and Single-Sign-On access to all the secured resources hosted on distributed systems with one login, and (iii) access control with proper rights, roles, and grants to all the secured resources can be performed.

Besides, the application is compliant with the new EU General Data Protection Rights (GDPR) to regulate the processing of personal data. The amount of personal data that can be collected by the application can be configured while registering the application at the Authorisation Server. It is possible to display the amount of personal data collected by the application by (i) registering with a particular level, and (ii) by choosing a login for a specific level. Personal data is only processed after the user's approval.

## Chapter 10

---

### Conclusions and future work

#### 10.1 Thesis Summary

The work described in this thesis successfully extends current generation semantic 3D city models by providing explicit support of various kinds of time-dependent properties. The thesis, first of all, reviews multiple application domains of semantic 3D city models and identifies seven key requirements for temporal extensions of city object properties. The research classifies the listed requirements according to two broad categories: slower changes and highly dynamic changes. Accordingly, two new concepts: (i) the Versioning concept, and (ii) the Dynamizer concept are introduced. The Versioning concept deals with slower changes and allows representing historic and parallel versions of 3D city models. The Dynamizer concept deals with highly dynamic changes and allows representing as well as linking city object properties with numerous sources of highly dynamic time-dependent properties, including real-time sensor and IoT devices. The Dynamizer concept also provides a method for injecting dynamic variations of city object properties into the static representation making city objects truly dynamic. Considering the practical requirements, the thesis provides a complete "ecosystem", which not only allows extending semantic 3D city models using the conceptual data models, but also offers ways to (i) implement, (ii) manage, and (iii) use them in the applications. For this purpose, the thesis is divided into three separate parts.

Part I provides the in-depth details of the Versioning concept (chapter 4) and the Dynamizer concept (chapter 5). The discussions include the details of the UML models incorporating the listed requirements and several illustrations of how these concepts can be represented within CityGML files. Part II describes novel approaches for managing the extended 3D city models within a database management system (in this thesis, the open-source software 3DCityDB is used). For storing the time-series and time-series metadata associated with the Dynamizers, a new extension of the relational data model for the 3DCityDB is introduced (chapter 6). For the interoperable access and retrieval of the time-series and sensor observations associated with Dynamizers, a new concept called Inter-Sensor Service is introduced (chapter 7). This services retrieves time-series data (either stored in the 3DCityDB or managed using arbitrary sensor platform) and translates them "on-the-fly" according to the standardised interfaces such as OGC Sensor Observation Service and OGC SensorThings API. The successful demonstrations of the CityGML Dynamizers and the InterSensor Service are applied in real-world Smart City projects such as OGC Future City Pilot Phase 1 and the Smart District Data Infrastructure (SDDI) Demonstrator as shown in Part III. The results are discussed in chapter 8. Furthermore, chapter 9 highlights the importance of security in distributed Smart City projects (such as SDDI) and demonstrates a scenario of how individual components including the 3D city model

and all the sensors can be accessed using proper authorisation, authentication with Single-Sign-On functionality.

The concepts presented in the thesis are developed for the CityGML standard. However, they can also be applied to other GML-based application schemas including the European INSPIRE data themes and national standards for topography and cadasters like the British Ordnance Survey Mastermap or the German cadaster standard ALKIS. The concepts can also be applied to other standards such as IFC and IndoorGML.

## 10.2 Discussion of the results

After this general overview summarising the individual contributions achieved in each chapter, this section presents answers to the research questions and hypotheses stated in section 1.3.

**Question 1.1:** What are time-dependent properties in the context of semantic 3D city models?

Semantic 3D city models allow defining classes and relations to represent each topographic object for its geometrical, topological, thematic, and appearance properties. This thesis highlights that any such property, which changes with time, becomes a time-dependent property. For example, as described in section 3.1, a building may undergo several transformations, which may result in changes in (i) geometrical or spatial properties such as the addition of a new floor in the building, (ii) topology within a building (e.g. division of a big room into two smaller rooms by the creation of a new wall), (iii) appearance of a building (such as materials, textures, and colours on the wall and roof surfaces), and (iv) semantics of a building (e.g. changing the building's type from residential to commercial). Furthermore, this section outlines that these changes may occur in different frequencies. Some of the changes can be sudden. For instance, changing the owner of a house is a sudden event. On the other hand, changes may also be gradual and progressive. The demolition of a building is a gradual but comparatively shorter event. But the construction of a Gothic cathedral is a very long event that lasts several centuries. Similarly, historical building deteriorations may take centuries or millennia. The changes in city object properties can also be discrete or continuous. The evolution of a city can be represented as a series of time-stamped snapshots whereby each snapshot represents the state of city features at a specific point in time. Such snapshots are linear and discrete. On the other hand, rising water during a flood event is continuous. Therefore, changes in cities can be classified according to the rate at which they change. This thesis classifies such changes according to two broad categories: (i) slower changes, and (ii) highly dynamic changes.

**Question 1.2:** What kinds of time-dependent properties are required in various applications of semantic 3D city models?

The thesis provides a comprehensive review (c.f. chapter 2) of significant application domains of semantic 3D city models including (i) Smart Cities and Digital Twins, (ii) Urban simulations, (iii) Mobility, and (iv) Urban development. Numerous use cases of these application domains are studied to gather essential requirements for dealing with different types of time-dependent properties. Based on the systematic analysis, seven key requirements are identified including (i) establishing direct links of city object properties with external sensors and IoT platforms, (ii) enabling city object properties to link with alerts and event management systems, (iii) integration and overlay of static city objects with



the dynamics of moving objects, (iv) representing and exchanging time-series and its metadata in-line with city objects, (v) representing and exchanging complex patterns based on statistics and general rules, (vi) managing and exchanging alternative versions, and (vii) managing and exchanging historic versions. The identified requirements with several use cases are further categorised to determine (i) whether they are slower changes or highly dynamic changes, and (ii) which city object property gets affected by such changes. Such analysis helps to develop the extensions for semantic 3D city models in a systematic way.

**Question 1.3:** How can existing data models be extended to support the identified time-dependent properties?

The discussions in chapter 2 and chapter 3 help concluding that both slower and highly dynamic changes are fundamentally different from each other. Slower changes involve features that begin or cease to exist over different time intervals. For example, if a new building is added in the city model at a certain point or period in time, it is not possible to query it before that specified time as there was no existence of the feature. Similarly, the planning scenarios involve a comparison of multiple versions of the same city model by different planners. Hence, such changes require different versions of the city models having completely new or modified features. On the other hand, highly dynamic changes are mostly associated with city object properties and can be defined as a function of time. In this case, only some of the properties of otherwise static objects need to represent such time-varying values. For example, the energy consumption of a building determined by a Smart Meter installed in the same building requires only one specific property (e.g. *"energy\_consumption"*) of the building to be dynamic, while other properties (e.g. *"building\_roof\_type"*) remain static. Hence, this thesis considers both slower and highly dynamic changes separately and proposes different extensions to represent and manage them.

Based on this classification, this thesis recommends extending semantic 3D city models for dealing with both types of changes in two different ways. Part I of the thesis introduces conceptual data models for the CityGML standard and describes how these data models support the requirements [R1-R7] identified in chapter 2. The Versioning concept (c.f. chapter 4) deals with slower changes and allows representing historic and parallel versions of 3D city models. The Dynamizer concept (c.f. chapter 5) deals with highly dynamic changes and allows representing as well as linking city object properties with numerous sources of highly dynamic time-dependent properties. The Dynamizer concept also provides a method for injecting dynamic variations of city object properties into the static representation making city objects truly dynamic.

Depending on the requirements, these two different types of changes can also be represented together. For example, there are two indoor sensors from two different sensor providers installed in the same room, and both of them measure the same property within the building (e.g. *"indoor\_air\_quality"*). In this case, two alternative versions of the dataset can represent building objects referring to alternative sensor sources. In similar ways, one version may link to a sensor source and the other one may link to simulation results.

**Hypothesis 1.4:** Existing modelling standards for representing various time-dependent properties can be utilised in extending semantic 3D city models.

Section 2.2 includes a comprehensive literature review and lists the already existing standards for each gathered requirement [R1-R7]. For example, in relation to sensors and IoT data access and management, this section reviews well-established standards such as the OGC Sensor Web Enablement, FIWARE, BIG\_IoT, VICINITY, and so on. Similarly, for representing time-series and its metadata, the section reviews popular international standards such as OGC TimeseriesML and OGC Observations & Measurements. Since these standards are stable, well-defined, and are used worldwide, they have been utilised in the developed data models. For instance, CityGML Dynamizer provides functionality to link a specific city object property directly with arbitrary sensor and IoT web services such as the OGC Sensor observation Service, the OGC SensorThings API, ThingSpeak, The Things Network, and so on. Furthermore, such links can be established using either HTTP or MQTT protocols. Similarly, Dynamizer Atomic Timeseries also utilise OGC TimeseriesML 1.0 for representing and exchanging city objects along with the associated time-series and time-series metadata.

During the development of extensions of the CityGML standard, the author of this thesis participated in multiple meetings and conferences with the standardisation committees including (i) regular OGC Technical Committee Meetings for the standards CityGML and TimeseriesML 1.0, (ii) OGC SensorThings API Summit (in 2018) for the SensorThings API, and (iii) Geospatial Sensor Web Conference (in 2016 and 2018) for discussions for the OGC Sensor Observation Service.

**Question 1.5:** How can we manage time-dependent properties along with static properties of 3D city models?

There are already sophisticated database management systems such as the 3DCityDB, which allow importing, managing, and exporting semantic 3D city models based on the CityGML standard. They enable applications to query as well as to access city objects and their properties using query languages and standardised web services. As highlighted in the thesis, current versions of such databases manage only the static properties of city objects. Part II introduces several ways for managing time-dependent properties of city objects along with their static properties. Chapter 6 presents how the newly introduced modules Versioning and Dynamizer can be managed within the 3DCityDB using its ADE Plugin Manager. It allows storing as well as retrieving the new features. Since Dynamizers represent time-series data associated with city object properties, Chapter 6 presents a new extension of the relational database model for 3DCityDB. This relational model offers new tables and relations for mapping time-series and its metadata values. In this way, it is also possible to perform temporal queries associated with city object properties.

**Hypothesis 1.6:** It is not always required to store time-dependent properties along with the static properties of city objects in database management systems.

In many scenarios (especially in distributed applications), the dynamic and time-series data may belong to different stakeholders and companies. These stakeholders may be owners, sensor operators, solution providers, citizens, and visitors. The data may also belong to numerous data sources such as databases, sensor web services, IoT platforms, and simple tabulated files. To make well-informed decisions using such distributed and heterogeneous dynamic data, it is crucial to achieving a proper data integration strategy. This strategy must allow working with heterogeneous data sources and platforms in a common operational framework, which requires interoperability. Such interoperability

can be achieved by using international open standards. These standards allow modelling and representing the data using well-defined information models. At the same time, such standards allow interfacing the distributed components using well-defined interface models (web services). In such complex distributed systems, the concept of Spatial Data Infrastructures (SDIs) is often utilised. With the help of such infrastructures, it is possible to retrieve the time-varying information from remote and distributed resources such as external databases, Application programming interfaces (APIs) or files using web services. This approach avoids data to be stored in centralised databases together with the 3D city models. It is also beneficial, especially in the cases of sensors and IoT data, when the frequency of time-series data is very high (e.g. up to milliseconds). Hence, the hypothesis is *true* that it is not always required to store such time-series and dynamic data in the 3D city modelling database management systems. Chapter 7 highlights the importance of interoperable OGC Sensor Web Enablement standards for retrieving time-series information associated with city object properties.

**Question 1.7:** How can we achieve cross-platform interoperability for heterogeneous data sources of time-dependent properties?

SDIs play an important role in establishing interoperability for heterogeneous data sources and are considered as one of the keys to work in distributed scenarios. They allow encoding sensor descriptions and observations using well-defined standards as well as accessing them using standardised interfaces. In this way, applications and tools can be developed based on these standards without worrying about what different kinds of data sources they use. Multiple sources can be attached to these infrastructures, and their interfaces will always be common for different applications. Chapter 7 demonstrates several approaches for establishing cross-platform interoperability using the OGC SWE standards such as the Sensor Observation Service (SOS) and the SensorThings API.

However, Chapter 7 also highlights that both the SOS and SensorThings API always involve data storage to store their metadata and time-series observations, based on which web services can query and retrieve data and observations. The issue, in such cases, is that in a distributed environment, where multiple stakeholders and owners are involved with proprietary platforms, not all of them would be willing to inject their proprietary data into a third-party data storage. Moreover, in a running distributed system having another data storage will require regular maintenance. It can also be a complex affair while moving the infrastructure to different locations, for example, from one server to another or into the cloud. In such cases, it is essential to have an intermediate service which can connect to a specific data source and encodes the observations "on-the-fly" according to the standardised OGC SWE interfaces without worrying about the data storage and multitude of data sources. For this purpose, chapter 7 introduces the lightweight InterSensor Service. This service provides several data adapters which can be used for establishing connections to not only different IoT platforms, but also to external databases, CSV files, CityGML Dynamizers, Cloud-based spreadsheets, GPS feeds, and real-time Twitter feeds. While querying, the service opens a data source connection and retrieves the observations based on querying parameters directly from the data source. The service encodes these observations "on-the-fly" according to the international standardised interfaces such as the OGC Sensor Observation Service and OGC SensorThings API. In this way, applications compliant to such OGC standardised interfaces can be used to interact with heterogeneous observations without worrying about their data storage.

**Question 1.8:** How can city modelling applications achieve integrated access to static as well as dynamic properties of city models?

The InterSensor Service plays a crucial role in accessing dynamic properties from the time-series values stored, (i) in a CityGML database such as 3DCityDB, (ii) in an external Sensor or IoT platform, (iii) in an external file such as a CSV. As demonstrated in chapter 8, the extensions of the InterSensor Service allow to establish connections to the specific Dynamizers stored in the 3DCityDB and querying and visualising the dynamic properties using international standards such as the OGC SensorThings API and OGC SOS. At the same time, the static features from the same 3DCityDB can be accessed with the existing OGC Web Feature Service. In this way, one application can access and visualise both the static and dynamic properties from one database instance using open and international standards.

**Question 1.9:** How can we have a unified visualisation for multiple heterogeneous time-dependent properties using the same framework?

Chapter 8 demonstrates a scenario using the InterSensor Service to access and visualise multiple and heterogeneous data sources. As described in the architecture, the service provides several data adapters for establishing connections to different IoT platforms, external databases, CSV files, Cloud-based spreadsheets, GPS feeds, and real-time Twitter feeds. While querying, the service opens a data source connection and retrieves the observations based on querying parameters directly from the data source. The service encodes these observations "on-the-fly" according to the international standardised interfaces such as the OGC Sensor Observation Service and OGC SensorThings API. It enables observations from heterogeneous data sources to be visualised and interpreted on common dashboard applications (e.g. the Helgoland client for the OGC SOS and Grafana Dashboards for the OGC SensorThings API). Chapter 8 demonstrates a number of scenarios in the project Smart District Data Infrastructure (SDDI) Demonstrator for visualising and interacting with sensor data based on the 52°North Timeseries API. The interface from the InterSensor Service can directly be used with the Helgoland application allowing users to interact with observations being retrieved directly from (i) a weather station (outside temperature retrieved from Weather Underground platform), (ii) Smart Meter located in a prominent building (electricity consumption per minute retrieved from the proprietary platform), and (iii) scheduled events in the same important building (visitor counts during the planned event extracted from a CSV file). Such joint visualisation helps to determine the correlation between different properties, e.g., "*what is the impact of the weather or any scheduled event on the electricity consumption of a building?*". Of course, such a common standard-based API will also be very valuable for any other kind of application or analysis tool. Similarly, another data adapter developed for the Twitter API allows visualising geotagged tweets with the static 3D city model.

**Question 1.10:** What are secure ways allowing users and applications to access static and dynamic properties of semantic 3D city models?

Chapter 9 provides solutions for securing the overall access and management of distributed applications and services. The proposed concept facilitates privacy, security and controlled access to all stakeholders and the respective components by establishing proper authorisation and authentication mechanisms.

The approach facilitates Single-Sign-On (SSO) authentication by a novel combination in the use of the state-of-the-art security concepts such as OAuth2 access tokens, OpenID Connect user claims and Security Assertion Markup Language (SAML). The chapter also shows a demonstration of implementation for the concept for a specific scenario carried out within the district Queen Elizabeth Olympic Park in London. The demonstrator application is conformant to the EU General Data Protection Regulation (GDPR). It allows linking different components such as 3D buildings with semantic information, weather stations, and Smart Meters in open, standardised and secure ways. The demonstration application supports individual access rights for different types of user groups including (i) public Google account, and (ii) academic users from universities and research institutes.

### 10.3 Scientific Contributions

The research work in this thesis makes contributions in the field of 3D Geoinformation Science and moves the entire field forward in multiple ways. The developed concepts have opened doors to many new research fields that were not available before:

1. **Versioning in semantic 3D city models.** Versioning has been a well-established term in the field of computer programming. There are several Version Control Systems such as Git, Mercurial, Concurrent Versions Systems, and Subversion. Similarly, the concept of versions has been incorporated successfully in database management systems such as Oracle Workspace Manager and ESRI ArcSDE Geodatabases. However, they lack significant functionalities from the perspective of semantic 3D city models. The INSPIRE and German AAA standards support versioning in its basic form. For example, INSPIRE supports exchanging only the last version of spatial objects. However, none of the standards supports managing multiple historical versions. The Versioning concept (c.f. chapter 4) developed within this thesis allows supporting multiple historical versions within one data file. Similarly, the earlier approaches allowed only forward-temporality. The Versioning concept allows backward compatibility to handle multiple representations of the past of a city. Although the DBMS systems such as Oracle or ESRI ArcSDE Geodatabases already support versions and conflict management, the CityGML Versioning approach allows exchanging all versions of a repository in an interoperable way, even within a single dataset.
2. **Dynamic aspects in semantic 3D city models** This thesis introduces the Dynamizer concept enabling 3D city objects to override object properties dynamically. These properties may be spatial, thematic, or even appearances. The Dynamizer approach allows dynamics to be explicitly represented and managed in a coherent framework. Hence, 3D city models with Dynamizers can be used in multiple application domains, including (i) Smart Cities and Digital Twins by establishing explicit links with real-time sensors and IoT devices using HTTP and MQTT protocols, (ii) mobility applications by enabling a moving object to communicate with static objects in a real-time manner, (iii) numerous urban simulations such as representing and exchanging solar irradiation levels, complex periodic patterns of energy usage, and many more.
3. **Spatial Data Infrastructures** This thesis also contributes to the field of Spatial Data Infrastructures (SDI): First of all, it highlights the importance of SDIs in managing complex distributed datasets, including semantic 3D city models, sensors and IoT devices and simulations in Smart City applications. The thesis describes the Smart District Data Infrastructure (SDDI) concept, which

was developed by (Moshrefzadeh et al. 2017). The author of the thesis was responsible for implementing the SDDI in the district Queen Elizabeth Olympic Park, London (as described in Chapter 8). The thesis describes different ways of managing and accessing time-series data along with the static data of the city objects using open, international, and interoperable standards. The thesis introduces the concept of InterSensor Service for this purpose. The InterSensor Service has been implemented by the author of this thesis and is provided as a free and Open Source application.

Furthermore, this thesis also highlights the importance of security in the SDIs for Smart Cities (such as SDDI). Chapter 9 introduces a concept that facilitates privacy, security and controlled access to all stakeholders and the respective components by establishing proper authorisation and authentication mechanisms. The approach facilitates Single-Sign-On (SSO) authentication by a novel combination in the use of the state-of-the-art security concepts such as OAuth2 access tokens, OpenID Connect user claims and Security Assertion Markup Language (SAML). An implementation of this concept for the district Queen Elizabeth Olympic Park in London is shown in this thesis and is also provided as an online demonstration. The thesis provides references to a comprehensive literature review indicating that such access control and security federation based realisation has not been considered in spatial data infrastructures for Smart Cities before.

## 10.4 Outlook and future prospects

From the discussions and outcomes of this thesis, the following possible topics for future research can be derived:

1. **Indoor modelling** The concepts developed in this thesis are implemented for the OGC CityGML standard. The new data models are defined using the Unified Modelling Language (UML), and the new datasets are represented using Geography Markup Language (GML). Therefore, the concepts can also be partially or fully applied to other standards utilising model-driven transformations. Although CityGML already allows representing the indoor environment, there are also other standards such as IFC and IndoorGML. These standards are prevalent and are used worldwide for numerous indoor modelling applications such as construction, evacuation management, indoor navigation, robot navigation, Industry 4.0, and Smart Factory. It will be interesting to apply the new concepts of CityGML to the other standards like IFC and IndoorGML and enable them to be used in indoor and BIM applications.
2. **Artificial Intelligence in Smart Cities** Artificial Intelligence (AI) is considered as one of the significant components in Smart City and Digital twin applications. AI algorithms analyse extensive data feeds and make projections into the future. Several new fields such as Traffic Management, Waste Management, and Smart Parking rely on AI algorithms to make accurate predictions and hence, prepare the government, city authorities, and citizens for the future. However, for performing data analysis, AI algorithms rely majorly on real-time data streams (e.g. from sensors and IoT devices) and historical data (e.g. retrievable from a database, an external files, or an API). The Dynamizer concept (developed in this thesis) already allows associating real-time as well as historical time-series data with individual city object properties. Hence, such dynamically enriched 3D city models can also be used with AI algorithms. For example, city objects changing their properties in real-time may benefit an AI algorithm for an autonomous

vehicle enabling the vehicle to see, hear, analyse, and make a decision just like human drivers do.

3. **Geo-visualisation** It is also essential to determine the ways how 3D city modelling visualisation clients interpret such dynamic 3D city models more efficiently. There is an already well-established OGC standard called 3D Tiles<sup>131</sup> allowing visualising arbitrary geospatial datasets such as 3D models, point clouds, imagery etc. in a very efficient way. 3D Tiles is already being extended for supporting time-dynamic streaming. It will be interesting to see how time-dynamic 3D Tiles can interpret CityGML Dynamizers within the Cesium virtual globe.

---

<sup>131</sup><https://github.com/CesiumGS/3d-tiles/blob/master/3d-tiles-overview.pdf>





## BIBLIOGRAPHY

- AAA (2014). *Dokumentation zur Modellierung der Geoinformationen des amtlichen Vermessungswesens (GeoInfoDok), Version 7.0. Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland (AdV)*. URL: <http://www.adv-online.de/AAA-Modell/> (visited on 22/12/2019).
- Aalders, H. and Moellering, H. (2001). 'Spatial data infrastructure'. In: *Proceedings of the 20th international cartographic conference. Beijing, China*, pp. 2234–2244. URL: [http://www.gdmc.nl/publications/2001/Spatial\\_data\\_infrasructure.pdf](http://www.gdmc.nl/publications/2001/Spatial_data_infrasructure.pdf).
- Adams, D. (1979). *The Hitchhiker's Guide to the Galaxy*. London: Pan Books.
- Agugiaro, G. (2016). 'Energy planning tools and CityGML-based 3D virtual city models: experiences from Trento (Italy)'. In: *Applied Geomatics* 8.1, pp. 41–56. URL: <https://doi.org/10.1007/s12518-015-0163-2>.
- Agugiaro, G., Benner, J., Cipriano, P. and Nouvel, R. (2018). 'The Energy Application Domain Extension for CityGML: enhancing interoperability for urban energy simulations'. In: *Open Geospatial Data, Software and Standards* 3.1, p. 2. URL: <https://doi.org/10.1186/s40965-018-0042-y>.
- Almeida, J. E., Rossetti, R. J. F. and Coelho, A. L. (2013). 'Crowd Simulation Modeling Applied to Emergency and Evacuation Simulations using Multi-Agent Systems'. In: *CoRR* abs/1303.4692. URL: <http://arxiv.org/abs/1303.4692>.
- Amirebrahimi, S., Rajabifard, A., Mendis, P. and Ngo, T. (2016). 'A framework for a microscale flood damage assessment and visualization for a building using BIM–GIS integration'. In: *International Journal of Digital Earth* 9.4, pp. 363–386. URL: <https://doi.org/10.1080/17538947.2015.1034201>.
- Anjos, D., Carreira, P. and Francisco, A. P. (2014). 'Real-Time Integration of Building Energy Data'. In: *2014 IEEE International Congress on Big Data*, pp. 250–257. URL: <https://doi.org/10.1109/BigData.Congress.2014.44>.
- Asahara, A., Shibasaki, R., Ishimaru, N. and Burggraf, D. (2015). *OGC Moving Features Encoding Part I: XML Core, OGC Document No. 14-083r2*. <https://www.opengeospatial.org/standards/movingfeatures>. (Visited on 09/09/2019).
- Ayele, E. D., Meratnia, N. and Havinga, P. J. M. (2018). 'Towards a New Opportunistic IoT Network Architecture for Wildlife Monitoring System'. In: *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5. URL: <https://doi.org/10.1109/NTMS.2018.8328721>.
- Baghdoust, A. (2017). 'Visualizing dynamic spatial height information in a dam monitoring context'. Master's Thesis. Chair of Geoinformatics, Technische Universität München. URL: <https://mediatum.ub.tum.de/doc/1374648/>.

- Banks, A., Briggs, E., Borgendale, K. and Gupta, R. (2019). *MQTT Version 5.0 - an OASIS Standard*. URL: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html> (visited on 09/09/2019).
- Bartoli, A, Hernández-Serrano, J, Soriano, M, Dohler, M, Kountouris, A and Barthel, D (2011). 'Security and privacy in your smart city'. In: *Proceedings of the Barcelona smart cities congress*. Vol. 292. URL: <https://pdfs.semanticscholar.org/a8eb/00601cdb94ff6bbfc03118f3fcb7575ba07a.pdf>.
- Batty, M. (2018). 'Digital twins'. In: *Environment and Planning B: Urban Analytics and City Science* 45.5, pp. 817–820. URL: <https://doi.org/10.1177/2399808318796416>.
- Batty, M., Axhausen, K. W., Giannotti, F., Pozdnoukhov, A., Bazzani, A., Wachowicz, M., Ouzounis, G. and Portugali, Y. (2012). 'Smart cities of the future'. In: *The European Physical Journal Special Topics* 214.1, pp. 481–518. URL: <https://doi.org/10.1140/epjst/e2012-01703-3>.
- Beauregard, B. and Speckhard, B. (2014). *Oracle Database Workspace Manager Developer's Guide, 12c Release 1 (12.1) E17893-07*. URL: <https://docs.oracle.com/database/121/ADWSM/title.htm> (visited on 02/11/2019).
- Becker, T., Nagel, C. and Kolbe, T. H. (2009). 'A multilayered space-event model for navigation in indoor spaces'. In: *3D geo-information sciences*. Springer, pp. 61–77. URL: [https://doi.org/10.1007/978-3-540-87395-2\\_5](https://doi.org/10.1007/978-3-540-87395-2_5).
- Behrisch, M., Bieker, L., Erdmann, J. and Krajzewicz, D. (2011). 'SUMO—simulation of urban mobility: an overview'. In: *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind. URL: <https://elib.dlr.de/71460/>.
- Beil, C. and Kolbe, T. H. (2017). 'CityGML and the streets of New York- A proposal for detailed Street Space Modelling'. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences IV-4/W5*, pp. 9–16. URL: <https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-4-W5/9/2017/>.
- Benevolo, C., Dameri, R. P. and D'Auria, B. (2016). 'Smart Mobility in Smart City'. In: *Empowering Organizations*. Ed. by Torre, T., Braccini, A. M. and Spinelli, R. Cham: Springer International Publishing, pp. 13–28. URL: [https://doi.org/10.1007/978-3-319-23784-8\\_2](https://doi.org/10.1007/978-3-319-23784-8_2).
- Bertolini, L. and Le Clercq, F. (2003). 'Urban development without more mobility by car? Lessons from Amsterdam, a multimodal urban region'. In: *Environment and planning A* 35.4, pp. 575–589. URL: <https://doi.org/10.1068/a3592>.
- Biljecki, F., Heuvelink, G. B. M., Ledoux, H. and Stoter, J. (Dec. 2015a). 'Propagation of positional error in 3D GIS: Estimation of the Solar Irradiation of building roofs'. In: *International Journal of Geographical Information Science* 29.12, pp. 2269–2294. URL: <http://doi.org/10.1080/13658816.2015.1073292>.
- Biljecki, F., Kumar, K. and Nagel, C. (2018). 'CityGML application domain extension (ADE): overview of developments'. In: *Open Geospatial Data, Software and Standards* 3.1, p. 13. URL: <https://doi.org/10.1186/s40965-018-0055-6>.
- Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S. and Çöltekin, A. (2015b). 'Applications of 3D city models: State of the art review'. In: *ISPRS International Journal of Geo-Information* 4.4, pp. 2842–2889. URL: <https://doi.org/10.3390/ijgi4042842>.
- Biswas, K. and Muthukkumarasamy, V. (2016). 'Securing Smart Cities Using Blockchain Technology'. In: *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 1392–1393. URL: <https://doi.org/10.1109/HPCC-SmartCity-DSS.2016.0198>.

- Borning, A., Waddell, P. and Förster, R. (2008). 'Urbansim: Using Simulation to Inform Public Deliberation and Decision-Making'. In: *Digital Government: E-Government Research, Case Studies, and Implementation*. Ed. by Chen, H., Brandt, L., Gregg, V., Traunmüller, R., Dawes, S., Hovy, E., Macintosh, A. and Larson, C. A. Boston, MA: Springer US, pp. 439–464. URL: [https://doi.org/10.1007/978-0-387-71611-4\\_22](https://doi.org/10.1007/978-0-387-71611-4_22).
- Borrmann, A., König, M., Koch, C. and Beetz, J. (2015). 'Building Information Modeling'. In: *Technologische Grundlagen Und Industrielle Anwendungen: Vieweg+ Teubner Verlag*. URL: <https://www.springer.com/de/book/9783658056056>.
- Bosson, C. and Lauderdale, T. A. (2018). 'Simulation evaluations of an autonomous urban air mobility network management and separation service'. In: *2018 Aviation Technology, Integration, and Operations Conference*, p. 3365. URL: <https://doi.org/10.2514/6.2018-3365>.
- Botts, M. (2014). *Sensor Model Language (SensorML) | OGC Document No. 12-000*. URL: <http://www.opengeospatial.org/standards/sensorml> (visited on 22/12/2019).
- Braeckel, A., Bigagli, L. and Echterhoff, J. (2016). *OGC Publish/Subscribe Interface Standard 1.0 - Core, OGC Document No. 13-131r1*. URL: <https://www.opengeospatial.org/standards/pubsub> (visited on 09/09/2019).
- Bröring, A., Echterhoff, J., Jirka, S., Simonis, I., Everding, T., Stasch, C., Liang, S. and Lemmens, R. (2011). 'New Generation Sensor Web Enablement'. In: *Sensors* 11.3, pp. 2652–2699. URL: <https://doi.org/10.3390/s110302652>.
- Bröring, A., Schmid, S., Schindhelm, C.-K., Khelil, A., Kabisch, S., Kramer, D., Le Phuoc, D., Mitic, J., Anicic, D. and Teniente López, E. (2017). 'Enabling IoT ecosystems through platform interoperability'. In: *IEEE software* 34.1, pp. 54–61. URL: <https://doi.org/10.1109/MS.2017.2>.
- Bröring, A., Stasch, C. and Echterhoff, J. (2012). *Sensor Observation Service Interface Standard (SOS) | OGC Document No. 12-006*. URL: <http://www.opengeospatial.org/standards/sos> (visited on 22/12/2019).
- Burggraf, D. (2015). *OGC Keyhole Markup Language (KML) 2.3 | OGC Document No. 12-007r2*. <http://www.opengeospatial.org/standards/kml>. (Visited on 17/09/2019).
- Cantor, S., Kemp, J., Philpott, R. and Maler, E. (2005). *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*. URL: <https://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf> (visited on 30/04/2018).
- Chaturvedi, K. and Kolbe, T. H. (2017). *Future City Pilot 1 Engineering Report - OGC Doc. No. 16-098*. Tech. rep. Open Geospatial Consortium. URL: <http://docs.opengeospatial.org/per/16-098.html> (visited on 20/10/2017).
- Chaturvedi, K., Matheus, A., Nguyen, S. H. and Kolbe, T. H. (2019). 'Securing Spatial Data Infrastructures for Distributed Smart City applications and services'. In: *Future Generation Computer Systems* 101, pp. 723–736. URL: <http://www.sciencedirect.com/science/article/pii/S0167739X18330024>.
- Chaturvedi, K., Yao, Z. and Kolbe, T. H. (2015). 'Web-based Exploration of and Interaction with Large and Deeply Structured Semantic 3D City Models using HTML5 and WebGL'. In: *Bridging Scales - Skalenübergreifende Nah- und Fernerkundungsmethoden, 35. Wissenschaftlich-Technische Jahrestagung der DGPF*. Ed. by Kersten, T. P. Vol. 24. Deutsche Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V. Köln: Deutsche Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V. URL: [https://www.dgpf.de/src/tagung/jt2015/proceedings/papers/34\\_DGPF2015\\_Chaturvedi\\_et\\_al.pdf](https://www.dgpf.de/src/tagung/jt2015/proceedings/papers/34_DGPF2015_Chaturvedi_et_al.pdf).
- Chu, H., Yu, J., Wen, J., Yi, M. and Chen, Y. (2019). 'Emergency Evacuation Simulation and Management Optimization in Urban Residential Communities'. In: *Sustainability* 11.3. URL: <https://www.mdpi.com/2071-1050/11/3/795>.

- Cousins, S. (2017). '3D mapping Helsinki: How mega digital models can help city planners'. In: *Construction Research and Innovation* 8.4, pp. 102–106. URL: <https://doi.org/10.1080/20450249.2017.1396747>.
- Cox, S, Daisy, P, Lake, R, Portele, C and Whiteside, A (2004). *OpenGIS Geography Markup Language (GML 3.1), Implementation Specification Version 3.1.0, Recommendation Paper, OGC Doc. No. 03-105r1*. URL: <http://www.ogc.org/standards/gml> (visited on 09/09/2019).
- Cox, S. (2006). *Object identifiers in GML*. URL: <https://www.seegrid.csiro.au/wiki/AppSchemas/GmlIdentifiers> (visited on 12/12/2018).
- Cox, S. (2013). *Observations and Measurements (O&M) | OGC Document No. 10-004r3*. URL: <http://www.opengeospatial.org/standards/om> (visited on 17/09/2019).
- Cui, L., Xie, G., Qu, Y., Gao, L. and Yang, Y. (2018). 'Security and Privacy in Smart Cities: Challenges and Opportunities'. In: *IEEE Access* 6, pp. 46134–46145. URL: <https://doi.org/10.1109/ACCESS.2018.2853985>.
- Czerwinski, A., Sandmann, S., Stöcker-Meier, E. and Plümer, L. (2007). 'Sustainable SDI for EU noise mapping in NRW-best practice for INSPIRE'. In: *International Journal of Spatial Data Infrastructures Research* 2.2, pp. 90–111. URL: <https://ijsdir.sadl.kuleuven.be/index.php/ijsdir/article/view/63>.
- Dave, B., Buda, A., Nurminen, A. and Främling, K. (2018). 'A framework for integrating BIM and IoT through open standards'. In: *Automation in Construction* 95, pp. 35 –45. URL: <https://doi.org/10.1016/j.autcon.2018.07.022>.
- Davila Delgado, J. M., Butler, L. J., Gibbons, N., Brilakis, I., Elshafie, M. Z. E. B. and Middleton, C. (2017). 'Management of structural monitoring data of bridges using BIM'. In: *Proceedings of the Institution of Civil Engineers - Bridge Engineering* 170.3, pp. 204–218. URL: <https://doi.org/10.1680/jbren.16.00013>.
- De Luca, L., Busarayat, C., Stefani, C., Renaudin, N., Florenzano, M. and Véron, P. (2010). 'An Iconography-Based Modeling Approach for the Spatio-Temporal Analysis of Architectural Heritage'. In: *2010 Shape Modeling International Conference*, pp. 78–89. URL: <https://doi.org/10.1109/SMI.2010.28>.
- Deal, B. (2006). 'A Spatially Explicit Urban Simulation Model: Landuse Evolution and Impact Assessment Model (LEAM)'. In: *Smart growth and climate change*. Ed. by Ruth, M. New horizons in regional science. E. Elgar, pp. 181–203.
- Douglas, N., Voges, U. and Bigagli, L. (2014). *Catalogue Services 3.0 - General Model | OGC Document No. 12-168r6*. URL: <http://docs.opengeospatial.org/is/12-168r6/12-168r6.html> (visited on 22/12/2015).
- ESRI (2004). *Versioning White Paper*. URL: <http://support.esri.com/es/knowledgebase/whitepapers/view/productid/19/metaid/721>. (visited on 02/11/2019).
- FIWARE (2018). *Open source Platform for the Smart Digital Future*. URL: <https://www.fiware.org/> (visited on 16/05/2018).
- Frontoni, E., Loncarski, J., Pierdicca, R., Bernardini, M. and Sasso, M. (2018). 'Cyber Physical Systems for Industry 4.0: Towards Real Time Virtual Reality in Smart Manufacturing'. In: *Augmented Reality, Virtual Reality, and Computer Graphics*. Ed. by De Paolis, L. T. and Bourdot, P. Cham: Springer International Publishing, pp. 422–434. URL: [https://doi.org/10.1007/978-3-319-95282-6\\_31](https://doi.org/10.1007/978-3-319-95282-6_31).
- Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

- Ganzha, M., Paprzycki, M., Pawlowski, W., Szmeja, P. and Wasielewska, K. (2016). 'Semantic Technologies for the IoT - An Inter-IoT Perspective'. In: *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pp. 271–276. URL: <https://doi.org/10.1109/IoTDI.2015.22>.
- Gharaibeh, A., Salahuddin, M. A., Hussini, S. J., Khreishah, A., Khalil, I., Guizani, M. and Al-Fuqaha, A. (2017). 'Smart Cities: A Survey on Data Management, Security, and Enabling Technologies'. In: *IEEE Communications Surveys Tutorials* 19.4, pp. 2456–2501. URL: <https://doi.org/10.1109/COMST.2017.2736886>.
- Gojmerac, I., Reichl, P., Podnar Žarko, I. and Soursos, S. (2016). 'Bridging IoT islands: the symbIoTe project'. In: *e & i Elektrotechnik und Informationstechnik* 133.7, pp. 315–318. URL: <https://doi.org/10.1007/s00502-016-0439-1>.
- GPX (2004). *GPS Exchange Format (GPX) 1.1 Schema Documentation*. URL: [www.topografix.com/GPX/1/1/](http://www.topografix.com/GPX/1/1/) (visited on 02/11/2019).
- Grieves, M. and Vickers, J. (2017). 'Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems'. In: *Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches*. Ed. by Kahlen, F.-J., Flumerfelt, S. and Alves, A. Cham: Springer International Publishing, pp. 85–113. URL: [https://doi.org/10.1007/978-3-319-38756-7\\_4](https://doi.org/10.1007/978-3-319-38756-7_4).
- Gröger, G., Kolbe, T. H., Nagel, C. and Häfele, K.-H. (2012). *City Geography Markup Language (CityGML) v 2.0, OGC Doc. No. 12-019*. <http://www.opengeospatial.org/standards/citygml>. (Visited on 09/09/2019).
- Gunduz, M, Isikdag, U and Basaraner, M (2017). 'Integration of BIM, Web Maps and IoT for Supporting Comfort Analysis.' In: *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences* IV-4/W4. URL: <https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-4-W4/221/2017/isprs-annals-IV-4-W4-221-2017.pdf>.
- Hagemann, R., Corsmeier, U., Kottmeier, C., Rinke, R., Wieser, A. and Vogel, B. (2014). 'Spatial variability of particle number concentrations and NO<sub>x</sub> in the Karlsruhe (Germany) area obtained with the mobile laboratory 'AERO-TRAM''. In: *Atmospheric Environment* 94, pp. 341–352. URL: <http://www.sciencedirect.com/science/article/pii/S1352231014003987>.
- Hancke, G. P., Silva, B. D. C. e. and Hancke Jr., G. P. (2013). 'The Role of Advanced Sensing in Smart Cities'. In: *Sensors* 13.1, pp. 393–425. URL: <https://www.mdpi.com/1424-8220/13/1/393>.
- Hardt, D. (2012). *The OAuth 2.0 Authorization Framework*. URL: <https://www.ietf.org/rfc/rfc6749.txt> (visited on 30/04/2018).
- Hashem, I. A. T., Chang, V., Anuar, N. B., Adewole, K., Yaqoob, I., Gani, A., Ahmed, E. and Chiroma, H. (Oct. 2016). 'The Role of Big Data in Smart City'. In: *Int. J. Inf. Manag.* 36.5, 748–758. URL: <https://doi.org/10.1016/j.ijinfomgt.2016.05.002>.
- Helgoland (2018). *Sensor Web Client for Visual Exploration and Analysis of Sensor Web Data*. <https://github.com/52North/helgoland>. (Visited on 16/05/2018).
- IFC (2016). *Industry Foundation Classes Version 4 - Addendum 2*. URL: <http://www.buildingsmart-tech.org/ifc/IFC4/Add2/html/> (visited on 09/09/2019).
- INSPIRE (2013). *Generic Conceptual Model of the INSPIRE data specifications*. URL: <https://inspire.ec.europa.eu/documents/inspire-generic-conceptual-model> (visited on 22/12/2019).
- IRCELINE (2018). *Air Quality Belgium App*. URL: <https://github.com/irceline/air-quality-belgium-app> (visited on 16/05/2018).
- Isikdag, U., Zlatanova, S. and Underwood, J. (2013). 'A BIM-Oriented Model for supporting indoor navigation requirements'. In: *Computers, Environment and Urban Systems* 41, pp. 112–123. URL: <https://doi.org/10.1016/j.compenvurbsys.2013.05.001>.

- Jazayeri, M. A., Liang, S. H. L. and Huang, C.-Y. (2015). 'Implementation and Evaluation of Four Interoperable Open Standards for the Internet of Things'. In: *Sensors* 15.9, pp. 24343–24373. URL: <http://www.mdpi.com/1424-8220/15/9/24343>.
- Jennings, C., Shelby, Z., Arkko, J., Keranen, A. and Bormann, C. (2018). *Media types for Sensor Measurement Lists (SenML)*. URL: <https://tools.ietf.org/html/draft-ietf-core-senml-13> (visited on 16/05/2019).
- Jensen, C. S. and Snodgrass, R. T. (1999). 'Temporal data management'. In: *IEEE Transactions on knowledge and data engineering* 11.1, pp. 36–44.
- Jirka, S., Broering, A. and Stasch, C. (2009). 'Applying OGC sensor web enablement to risk monitoring and disaster management'. In: *GSDI 11 World Conference : Spatial data infrastructure convergence : building SDI bridges to address global challenges*, p. 13. URL: <http://www.gsdi.org/gsdiconf/gsd11/papers/pdf/96.pdf>.
- Jirka, S., Wieman, S., Brauner, J. and Jürrens, E. H. (2011). 'Linking Sensor Web Enablement and Web Processing Technology for Health-Environment Studies'. In: *Proceedings of the Integrating Sensor Web and Web-based Geoprocessing Workshop at the AGILE*. URL: <http://ceur-ws.org/Vol-712/paper9.pdf>.
- Jones, M. (2012). *The OAuth 2.0 Authorization Framework: Bearer Token Usage*. URL: <https://www.ietf.org/rfc/rfc6750.txt> (visited on 30/04/2018).
- Kaden, R. and Kolbe, T. H. (2013). 'City-Wide Total Energy Demand Estimation of Buildings using Semantic 3D City Models and Statistical Data'. In: *Proc. of the 8th International 3D GeoInfo Conference*. Vol. II-2/W1. URL: <https://mediatum.ub.tum.de/doc/1185881/1185881.pdf>.
- Kang, Y., Kim, H. and Han, S. (2015). 'Visualization of the Synthetic Environment Data Representation & Interchange Specification data for verifying large-scale synthetic environment data'. In: *The Journal of Defense Modeling and Simulation* 12.4, pp. 507–518. URL: <https://doi.org/10.1177/2F1548512914548601>.
- Kesteren, A. van (2014). *Cross-Origin Resource Sharing*. URL: <https://www.w3.org/TR/cors/> (visited on 30/04/2018).
- Khan, A. A. (2015). 'Constraints and concepts for the support of different locomotion types in indoor navigation'. PhD Thesis. Chair of Geoinformatics, Technische Universität München. URL: <https://mediatum.ub.tum.de/1233285>.
- Kim, J.-S., Yoo, S.-J. and Li, K.-J. (2014). 'Integrating IndoorGML and CityGML for Indoor Space'. In: *Web and Wireless Geographical Information Systems*. Ed. by Pfoser, D. and Li, K.-J. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 184–196. URL: [https://doi.org/10.1007/978-3-642-55334-9\\_12](https://doi.org/10.1007/978-3-642-55334-9_12).
- Kitchin, R. (2014). 'The real-time city? Big data and smart urbanism'. In: *GeoJournal* 79.1, pp. 1–14. URL: <https://doi.org/10.1007/s10708-013-9516-8>.
- Knapp, S. and Coors, V. (2007). 'The use of eParticipation systems in public participation: the VEPs example'. In: *Urban and Regional Data Management*. CRC Press, pp. 105–116. URL: <https://www.taylorfrancis.com/books/e/9780429224096/chapters/10.4324/9780203931042-10>.
- Kolbe, T. H. (2009). 'Representing and Exchanging 3D City Models with CityGML'. In: *3D Geo-Information Sciences*. Ed. by Lee, J. and Zlatanova, S. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 15–31. URL: [https://doi.org/10.1007/978-3-540-87395-2\\_2](https://doi.org/10.1007/978-3-540-87395-2_2).
- Kolbe, T. H., Gröger, G. and Plümer, L. (2008). 'CityGML–3D city models and their potential for emergency response'. In: *Geospatial information technology for emergency response*. Taylor & Francis: London, UK, pp. 257–274.

- Köninger, A. and Bartel, S. (1998). '3D-GIS for urban purposes'. In: *Geoinformatica* 2.1, pp. 79–103. URL: <https://doi.org/10.1023/A:1009797106866>.
- Kotsev, A., Schleidt, K., Liang, S., Schaaf, H. Van der, Khalafbeigi, T., Grellet, S., Lutz, M., Jirka, S. and Beaufiles, M. (2018). 'Extending INSPIRE to the Internet of Things through SensorThings API'. In: *Geosciences* 8.6. URL: <https://doi.org/10.3390/geosciences8060221>.
- Krukar, J., Schwering, A. and Anacta, V. J. (2017). 'Landmark-based navigation in cognitive systems'. In: *KI - Künstliche Intelligenz* 31, pp. 121–124. URL: <https://doi.org/10.1007/s13218-017-0487-7>.
- Kutzner, T. (2016). 'Geospatial Data Modelling and Model-driven Transformation of Geospatial Data based on UML Profiles'. PhD Thesis. Chair of Geoinformatics, Technische Universität München. URL: <https://mediatum.ub.tum.de/1341432>.
- Kutzner, T., Chaturvedi, K. and Kolbe, T. H. (2020). 'CityGML 3.0: New Functions Open Up New Applications'. In: *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, p. 19. URL: <https://doi.org/10.1007/s41064-020-00095-z>.
- Kutzner, T., Hijazi, I. and Kolbe, T. H. (2018). 'Semantic Modelling of 3D Multi-utility Networks for Urban Analyses and Simulations – The CityGML Utility Network ADE'. In: *International Journal of 3-D Information Modeling (IJ3DIM)* 7.2, pp. 1–34. URL: <https://doi.org/10.4018/IJ3DIM.2018040101>.
- Kwak, H.-J. and Park, G.-T. (2012). 'Study on the Mobility of Service Robots'. In: *International Journal of Engineering and Technology Innovation* 2.2, pp. 97–112. URL: <http://ojs.imeti.org/index.php/IJETI/article/view/84>.
- Lee, J., Li, K.-J., Zlatanova, S., Kolbe, T. H., Nagel, C. and Becker, T. (2014). *OGC Indoor GML: OGC Doc. No. 14-005r3*. URL: <http://docs.opengeospatial.org/is/14-005r3/14-005r3.html> (visited on 12/05/2016).
- Liang, S., Huang, C.-Y. and Khalafbeigi, T. (2015). *SensorThings API Part 1: Sensing | OGC Document No. 15-078r6*. URL: <http://docs.opengeospatial.org/is/15-078r6/15-078r6.html> (visited on 22/12/2019).
- Luginger, L. (2016). 'Success Factors of Integrated Multimodal Mobility Services'. Master's Thesis. Chair of Urban Structure and Transport Planning, Technische Universität München. URL: <https://mediatum.ub.tum.de/doc/1446938>.
- Mäs, S., Reinhardt, W. and Wang, F. (2006). 'Conception of a 3D geodata web service for the support of indoor navigation with GNSS'. In: *Innovations in 3D geo information systems*. Springer, pp. 307–316. URL: [https://doi.org/10.1007/978-3-540-36998-1\\_24](https://doi.org/10.1007/978-3-540-36998-1_24).
- Maureira, M. A. G., Oldenhof, D. and Teernstra, L. (2014). *ThingSpeak-an API and Web Service for the Internet of Things*. URL: [https://staas.home.xs4all.nl/t/swtr/documents/wt2014\\_thingspeak.pdf](https://staas.home.xs4all.nl/t/swtr/documents/wt2014_thingspeak.pdf) (visited on 16/08/2019).
- Mohammadi, N. and Taylor, J. E. (2017). 'Smart City Digital Twins'. In: *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–5. URL: <https://doi.org/10.1109/SSCI.2017.8285439>.
- Morel, M. and Gesquière, G. (2014). 'Managing Temporal Change of Cities with CityGML'. In: *Eurographics Workshop on Urban Data Modelling and Visualisation*. Ed. by Besuievsky, G. and Tourre, V. The Eurographics Association. URL: <http://dx.doi.org/10.2312/udmv.20141076>.
- Moshrefzadeh, M., Chaturvedi, K., Hijazi, I., Donaubaue, A. and Kolbe, T. H. (2017). 'Integrating and Managing the Information for Smart Sustainable Districts - The Smart District Data Infrastructure (SDDI)'. In: *Geoinformationssysteme 2017 – Beiträge zur 4. Münchner GI-Runde*. Ed. by Kolbe, T. H., Bill, R. and Donaubaue, A. Wichmann Verlag. URL: <https://mediatum.ub.tum.de/doc/1350813/46723.pdf>.

- Moshrefzadeh, M., Machl, T., Gackstetter, D., Donaubaue, A. and Kolbe, T. H. (2020). 'Towards a Distributed Digital Twin of the Agricultural Landscape'. In: *Journal of Digital Landscape Architecture* 5. URL: <https://mediatum.ub.tum.de/1540127>.
- Mynzhasova, A., Radojicic, C., Heinz, C., Kölsch, J., Grimm, C., Rico, J., Dickerson, K., García-Castro, R. and Oravec, V. (2017). 'Drivers, standards and platforms for the IoT: Towards a digital VICINITY'. In: *2017 Intelligent Systems Conference (IntelliSys)*, pp. 170–176. URL: <https://doi.org/10.1109/IntelliSys.2017.8324287>.
- Nedkov, S. (2012). 'Knowledge-based optimisation of three-dimensional city models for car navigation devices'. Master's Thesis. Department of GIS Technology, TU Delft. URL: <http://resolver.tudelft.nl/uuid:b429e899-9955-4a23-9ceb-66ffb6210b30>.
- Nguyen, S. H., Yao, Z. and Kolbe, T. H. (2017). 'Spatio-semantic comparison of large 3D city models in CityGML using a graph database'. In: *Proceedings of the 12th International 3D GeoInfo Conference 2017*, pp. 99–106. URL: <https://doi.org/10.5194/isprs-annals-IV-4-W5-99-2017>.
- Nobis, C. (2006). 'Carsharing as key contribution to multimodal and sustainable mobility behavior: Carsharing in Germany'. In: *Transportation Research Record* 1986.1, pp. 89–97. URL: <https://doi.org/10.1177/0361198106198600112>.
- Nour, M. and Beucke, K. (2010). 'Object versioning as a basis for design change management within a BIM context'. In: *Proceedings of the 13th international conference on computing in civil and building engineering (ICCCBE-XIII), Nottingham, UK*. URL: <http://www.engineering.nottingham.ac.uk/icccbe/proceedings/pdf/af74.pdf>.
- Okaya, M. and Takahashi, T. (2013). 'Evacuation Simulation with Guidance for Anti-disaster Planning'. In: *RoboCup 2012: Robot Soccer World Cup XVI*. Ed. by Chen, X., Stone, P., Sucar, L. E. and Zant, T. van der. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 202–212. URL: [https://doi.org/10.1007/978-3-642-39250-4\\_19](https://doi.org/10.1007/978-3-642-39250-4_19).
- Oulasvirta, A., Estlander, S. and Nurminen, A. (2009). 'Embodied interaction with a 3D versus 2D mobile map'. In: *Personal and Ubiquitous Computing* 13.4, pp. 303–320. URL: <https://doi.org/10.1007/s00779-008-0209-0>.
- Partescano, E., Brosich, A., Lipizer, M., Cardin, V. and Giorgetti, A. (2017). 'From heterogeneous marine sensors to sensor web: (near) real-time open data access adopting OGC sensor web enablement standards'. In: *Open Geospatial Data, Software and Standards* 2.1, p. 22. URL: <https://doi.org/10.1186/s40965-017-0035-2>.
- Patel, S., Uday Kumar R.Y. and Prasanna Kumar B. (2016). 'Role of smart meters in smart city development in India'. In: *2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)*, pp. 1–5. URL: <https://doi.org/10.1109/ICPEICES.2016.7853363>.
- Pfeiffer, M., Carré, C., Delfosse, V., Hallot, P. and Billen, R. (2013). 'Virtual Leodium: from an Historical 3D City Scale Model to an Archaeological Information System'. In: *ISPRS Annals—Volume II-5/W1, 2013*. URL: <https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/II-5-W1/241/2013/isprsanals-II-5-W1-241-2013.pdf>.
- Pfeil, M., Bartoschek, T. and Wirwahn, J. A. (2015). 'OPENSENSEMAP - A Citizen Science Platform For Publishing And Exploring Sensor Data as Open Data'. In: *Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings*. Vol. 15. URL: <https://doi.org/10.7275/R56971SW>.
- Quarati, A., Clematis, A., Roverelli, L., Zereik, G., D'Agostino, D., Mosca, G. and Masnata, M. (2017). 'Integrating Heterogeneous Weather-Sensors Data into a Smart-City App'. In: *2017 International Conference on High Performance Computing Simulation (HPCS)*, pp. 152–159. URL: <https://doi.org/10.1109/HPCS.2017.33>.



- Randt, B., Bildstein, F. and Kolbe, T. H. (2007). 'Use of virtual 3d landscapes for emergency driver training'. In: *Proc. of the Int. Conference on Visual Simulation IMAGE*. Vol. 2. URL: <http://www.redaktion.tu-berlin.de/fileadmin/fg227/Publications/IMAGE2007-RDE-DrivingSim-5-Letter.PDF>.
- Reitz, T. and Schubiger-Banz, S. (2014). 'The Esri 3D city information model'. In: *IOP Conference Series: Earth and Environmental Science* 18, p. 012172. URL: <https://doi.org/10.1088%2F1755-1315%2F18%2F1%2F012172>.
- Richer, J. (2015). *OAuth 2.0 Token Introspection*. URL: <https://www.ietf.org/rfc/rfc7662.txt> (visited on 30/04/2018).
- Robert, J., Kubler, S., Kolbe, N., Cerioni, A., Gastaud, E. and Främling, K. (2017). 'Open IoT Ecosystem for Enhanced Interoperability in Smart Cities—Example of Métropole De Lyon'. In: *Sensors* 17.12. URL: <http://www.mdpi.com/1424-8220/17/12/2849>.
- Roche, S. (2014). 'Geographic Information Science I: Why does a smart city need to be spatially enabled?' In: *Progress in Human Geography* 38.5, pp. 703–711. URL: <https://doi.org/10.1177/0309132513517365>.
- Ruhdorfer, R. (2017). 'Kopplung von Verkehrssimulation und semantischen 3D-Stadtmodellen in CityGML'. Master's Thesis. Chair of Geoinformatics, Technische Universität München. URL: <http://mediatum.ub.tum.de/node?id=1396796>.
- Ruohomäki, T., Airaksinen, E., Huuska, P., Kesäniemi, O., Martikka, M. and Suomisto, J. (2018). 'Smart City Platform Enabling Digital Twin'. In: *2018 International Conference on Intelligent Systems (IS)*, pp. 155–161. URL: <https://doi.org/10.1109/IS.2018.8710517>.
- Sakimura, N., Bradley, J., Jones, M., Medeiros, B. de and Mortimore, C. (2014). *OpenID Connect Core 1.0 incorporating errata set 1*. URL: <https://openid.net/specs/openid-connect-core-1\0.html> (visited on 30/04/2018).
- Salimzadeh, N., Vahdatikhaki, F. and Hammad, A. (2018). 'BIM-based surface-specific solar simulation of buildings'. In: *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*. Vol. 35. IAARC Publications, pp. 1–8. URL: <https://doi.org/10.22260/ISARC2018/0124>.
- Samuel, J., Servigne, S. and Gesquière, G. (2018). 'URBANCO2FAB: Comprehension of Concurrent Viewpoints of Urban Fabric based on GIT.' In: *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences-Volume IV-4/W6*, pp. 65–72. URL: <https://doi.org/10.5194/isprs-annals-IV-4-W6-65-2018>.
- Santhanavanich, T. and Coors, V. (2019). 'CityThings: A concept to integrate Dynamic Sensor data in a CityGML 3D City Model using OGC Sensorthings API.' In: *Proceedings of the second international conference on Urban Informatics 2019, Hong Kong*.
- Santhanavanich, T., Schneider, S., Rodrigues, P. and Coors, V. (2018). 'Integration and Visualization of Heterogeneous Sensor Data and Geospatial Information.' In: *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences-Volume IV-4/W7*, pp. 115–122. URL: <https://doi.org/10.5194/isprs-annals-IV-4-W7-115-2018>.
- Schilling, A., Coors, V. and Laakso, K. (2005). 'Dynamic 3D maps for mobile tourism applications'. In: *Map-based Mobile Services*. Springer, pp. 227–239. URL: [https://link.springer.com/chapter/10.1007/3-540-26982-7\\_15](https://link.springer.com/chapter/10.1007/3-540-26982-7_15).
- Scholl, J. (2019). 'Integration of BIM-based pedestrian simulations in the early design stages'. Bachelor's Thesis. Chair of Computational Modeling and Simulation, Technische Universität München. URL: [https://publications.cms.bgu.tum.de/theses/bachelor\\_thesis\\_janik\\_scholl\\_2019.pdf](https://publications.cms.bgu.tum.de/theses/bachelor_thesis_janik_scholl_2019.pdf).

- Schulte, C. and Coors, V. (2009). 'Development of a CityGML ADE for dynamic 3D flood information'. In: *Applied Geoinformatics for Society and Environment* 103, p. 10.
- Schwab, B. and Kolbe, T. H. (2019). 'Requirement Analysis of 3D Road Space Models for Automated Driving'. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-4/W8, pp. 99–106. URL: <https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-4-W8/99/2019/>.
- Shaffer, B. (2018). *A library for implementing an OAuth2 Server in php*. URL: <https://github.com/bshaffer/oauth2-server-php> (visited on 30/04/2018).
- Slingsby, A. and Raper, J. (2008). 'Navigable space in 3D city models for pedestrians'. In: *Advances in 3D geoinformation systems*. Springer, pp. 49–64. URL: [https://doi.org/10.1007/978-3-540-72135-2\\_3](https://doi.org/10.1007/978-3-540-72135-2_3).
- Sookhak, M., Tang, H., He, Y. and Yu, F. R. (2019). 'Security and Privacy of Smart Cities: A Survey, Research Issues and Challenges'. In: *IEEE Communications Surveys Tutorials* 21.2, pp. 1718–1743. URL: <https://doi.org/10.1109/COMST.2018.2867288>.
- Stadler, A. and Kolbe, T. H. (2007). 'Spatio-semantic coherence in the integration of 3D city models'. In: *Proceedings of the 5th International ISPRS Symposium on Spatial Data Quality ISSDQ 2007 in Enschede, The Netherlands, 13-15 June 2007*. Ed. by Stein, A. ISPRS Archives. ISPRS. URL: [http://www.isprs.org/proceedings/XXXVI/2-C43/Session1/paper\\_Stadler.pdf](http://www.isprs.org/proceedings/XXXVI/2-C43/Session1/paper_Stadler.pdf).
- Strzalka, A., Bogdahn, J., Coors, V. and Eicker, U. (2011). '3D City modeling for urban scale heating energy demand forecasting'. In: *HVAC&R Research* 17.4, pp. 526–539. URL: <https://www.tandfonline.com/doi/abs/10.1080/10789669.2011.582920>.
- Suciu, G., Vulpe, A., Halunga, S., Fratu, O., Todoran, G. and Suciu, V. (2013). 'Smart Cities Built on Resilient Cloud Computing and Secure Internet of Things'. In: *2013 19th International Conference on Control Systems and Computer Science*, pp. 513–518. URL: <https://doi.org/10.1109/CSCS.2013.58>.
- Taylor, P. (2014). *WaterML 2.0: Part 1 - Timeseries, OGC Document No. 10-126r4*. URL: <http://www.openeospatial.org/standards/waterml> (visited on 15/10/2019).
- Thingful (2018). *A Search Engine for the Internet of Things*. URL: <http://www.thingful.net/> (visited on 16/05/2018).
- TimeseriesAPI (2018). *A RESTful web binding to OGC Sensor Observation Service*. <http://sensorweb.demo.52north.org/sensorwebclient-webapp-stable/api-doc/index.html>. (Visited on 16/05/2018).
- Toma, D. M., Rio, J. del, Jirka, S., Delory, E., Pearlman, J. and Waldmann, C. (2015). 'NeXOS smart electronic interface for sensor interoperability'. In: *OCEANS 2015 - Genova*, pp. 1–5. URL: <https://doi.org/10.1109/OCEANS-Genova.2015.7271586>.
- Tomkins, J. and Lowe, D. (2016). *Timeseries Profile of Observations and Measurements, OGC Document No. 15-043r3*. URL: <http://www.openeospatial.org/standards/tsml> (visited on 09/09/2019).
- Tomko, M. and Winter, S. (2019). 'Beyond digital twins – A commentary'. In: *Environment and Planning B: Urban Analytics and City Science* 46.2, pp. 395–399. URL: <https://doi.org/10.1177/2399808318816992>.
- Vesperman, J. (2003). *Essential CVS*. O'Reilly Media, Inc. URL: <http://shop.oreilly.com/product/9780596004590.do>.
- Vretanos, P. (2010). *OpenGIS Web Feature Service 2.0 Interface Standard (WFS) | OGC Document No. 12-100*. URL: <http://www.openeospatial.org/standards/wfs> (visited on 22/12/2015).
- Wang, H., Gluhak, A., Meissner, S. and Tafazolli, R. (2013). 'Integration of BIM and live sensing information to monitor building energy performance'. In: *The CIB 30th International Conference*

- on Applications of IT in the AEC Industry*. URL: <http://architektur-informatik.scix.net/pdfs/w78-2013-paper-146.pdf>.
- Willenborg, B., Sindram, M. and Kolbe, T. H. (2017). 'Applications of 3D City Models for a better understanding of the Built Environment'. In: *Trends in Spatial Analysis and Modelling*. Ed. by Behnisch, M. and Meinel, G. Vol. 19. Geotechnologies and the Environment 19. Berlin, Heidelberg: Springer International Publishing, pp. 167–191. URL: [https://doi.org/10.1007/978-3-319-52522-8\\_9](https://doi.org/10.1007/978-3-319-52522-8_9).
- Wolff, M. and Asche, H. (2009). 'Towards Geovisual Analysis of Crime Scenes – A 3D Crime Mapping Approach'. In: *Advances in GIScience*. Ed. by Sester, M., Bernard, L. and Paelke, V. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 429–448.
- Wupperverband (2017). *TaMIS - Development of a Dam Surveillance and Information System for the Management of Natural Hazards*. URL: <https://52north.org/references/tamis-talsperrenmessinformationssystem/> (visited on 16/05/2018).
- Yang, L., Dam, K. H. van, Majumdar, A., Anvari, B., Ochieng, W. Y. and Zhang, L. (2019). 'Integrated design of transport infrastructure and public spaces considering human behavior: A review of state-of-the-art methods and tools'. In: *Frontiers of Architectural Research* 8.4, pp. 429–453. URL: <http://www.sciencedirect.com/science/article/pii/S209526351930072X>.
- Yang, L., Zhang, L., Stettler, M. E., Sukitpaneenit, M., Xiao, D. and Dam, K. H. van (2020). 'Supporting an integrated transportation infrastructure and public space design: A coupled simulation method for evaluating traffic pollution and microclimate'. In: *Sustainable Cities and Society* 52, p. 101796. URL: <http://www.sciencedirect.com/science/article/pii/S2210670719309667>.
- Yao, Z. (2020). 'Domain Extendable 3D City Models – Management, Visualization, and Interaction'. PhD Thesis. Chair of Geoinformatics, Technische Universität München. URL: <https://mediatum.ub.tum.de/1574231>.
- Yao, Z. and Kolbe, T. H. (2017). 'Dynamically Extending Spatial Databases to support CityGML Application Domain Extensions using Graph Transformations'. In: *Kulturelles Erbe erfassen und bewahren - Von der Dokumentation zum virtuellen Rundgang, 37. Wissenschaftlich-Technische Jahrestagung der DGPF*. Ed. by Kersten, T. P. Vol. 26. Publikationen der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation (DGPF) e.V. Würzburg: Deutsche Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V., pp. 316–331. URL: [http://www.dgpf.de/src/tagung/jt2017/proceedings/proceedings/papers/30\\_DGPF2017\\_Yao\\_Kolbe.pdf](http://www.dgpf.de/src/tagung/jt2017/proceedings/proceedings/papers/30_DGPF2017_Yao_Kolbe.pdf).
- Yao, Z., Nagel, C., Kunde, F., Hudra, G., Willkomm, P., Donaubauer, A., Adolphi, T. and Kolbe, T. H. (2018). '3DCityDB - a 3D geodatabase solution for the management, analysis, and visualization of semantic 3D city models based on CityGML'. In: *Open Geospatial Data, Software and Standards* 3.1, pp. 1–26. URL: <https://doi.org/10.1186/s40965-018-0046-7>.
- Yin, C., Xiong, Z., Chen, H., Wang, J., Cooper, D. and David, B. (2015). 'A Literature Survey on Smart Cities'. In: *Science China Information Sciences* 58.10, pp. 1–18. URL: <https://doi.org/10.1007/s11432-015-5397-4>.
- Zada, A. J., Tizani, W and Oti, A. (2014). 'Building Information Modelling (BIM)—versioning for collaborative design'. In: *Computing in Civil and Building Engineering (2014)*, pp. 512–519. URL: <https://ascelibrary.org/doi/abs/10.1061/9780784413616.064>.
- Zahn, W. (2015). 'Sonneneinstrahlungsanalyse auf und Informationsanreicherung von großen 3D-Stadtmodellen im CityGML-Schema'. Master's Thesis. Chair of Geoinformatics, Technische Universität München. URL: <http://mediatum.ub.tum.de/node?id=1276236>.

- Zhang, K., Ni, J., Yang, K., Liang, X., Ren, J. and Shen, X. S. (2017). 'Security and Privacy in Smart City Applications: Challenges and Solutions'. In: *IEEE Communications Magazine* 55.1, pp. 122–129. URL: <https://doi.org/10.1109/MCOM.2017.1600267CM>.
- Zhu, W., Simons, A., Wursthorn, S. and Nichersu, A. (2016). 'Integration of CityGML and Air Quality Spatio-Temporal Data Series via OGC SOS'. In: *Proceedings of the Geospatial Sensor Webs Conference (GSW), Münster, Germany*, pp. 29–34. URL: <http://ceur-ws.org/Vol-1762/Zhu.pdf>.
- Zlatanova, S. and Holweg, D. (2004). '3D Geo-information in emergency response: a framework'. In: *Proceedings of the 4th International Symposium on Mobile Mapping Technology (MMT'2004), March*, pp. 29–31. URL: [http://www.gdmc.nl/publications/2004/3D\\_Geo-information\\_Emergency\\_Response.pdf](http://www.gdmc.nl/publications/2004/3D_Geo-information_Emergency_Response.pdf).

## ORIGINAL PUBLICATIONS

1. Kutzner, T., **Chaturvedi, K.** and Kolbe, T. H. (2020). 'CityGML 3.0: New Functions Open Up New Applications'. en. In: *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, p. 19. URL: <https://doi.org/10.1007/s41064-020-00095-z>
2. **Chaturvedi, K.**, Yao, Z. and Kolbe, T. H. (2019). 'Integrated Management and Visualization of Static and Dynamic Properties of Semantic 3D City Models'. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLII-4/W17*, pp. 7–14. URL: <https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-4-W17/7/2019/>
3. **Chaturvedi, K.** and Kolbe, T. H. (2019). 'A Requirement Analysis on extending Semantic 3D City Models for supporting Time-dependent properties.' In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences IV-4/W9*, pp. 19–26. URL: <https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-4-W9/19/2019/>
4. **Chaturvedi, K.** and Kolbe, T. H. (2019). 'Towards Establishing Cross-Platform Interoperability for Sensors in Smart Cities'. In: *Sensors* 19.3. URL: <https://www.mdpi.com/1424-8220/19/3/562>
5. **Chaturvedi, K.**, Matheus, A., Nguyen, S. H. and Kolbe, T. H. (2019). 'Securing Spatial Data Infrastructures for Distributed Smart City applications and services'. In: *Future Generation Computer Systems* 101, pp. 723 –736. URL: <http://www.sciencedirect.com/science/article/pii/S0167739X18330024>
6. **Chaturvedi, K.** and Kolbe, T. H. (2018). 'InterSensor Service: Establishing Interoperability over Heterogeneous Sensor Observations and Platforms for Smart Cities'. In: *2018 IEEE International Smart Cities Conference (ISC2)*, pp. 1–8. URL: <https://doi.org/10.1109/ISC2.2018.8656984>
7. **Chaturvedi, K.**, Matheus, A., Nguyen, S. H. and Kolbe, T. H. (2018). 'Securing Spatial Data Infrastructures in the Context of Smart Cities'. In: *2018 International Conference on Cyberworlds (CW)*, pp. 403–408. URL: <https://doi.org/10.1109/CW.2018.00078>
8. **Chaturvedi, K.** and Kolbe, T. H. (2017). *Future City Pilot 1 Engineering Report - OGC Doc. No. 16-098*. Tech. rep. Open Geospatial Consortium. URL: <http://docs.opengeospatial.org/per/16-098.html> (visited on 20/10/2017).
9. **Chaturvedi, K.**, Willenborg, B., Sindram, M. and Kolbe, T. H. (2017). 'Solar Potential Analysis and Integration of the Time-dependent Simulation Results for Semantic 3D City Models using Dynamizers'. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences IV-4/W5*, pp. 25–32. URL: <https://doi.org/10.5194/isprs-annals-IV-4-W5-25-2017>
10. **Chaturvedi, K.**, Smyth, C. S., Gesquière, G., Kutzner, T. and Kolbe, T. H. (2017). 'Managing Versions and History Within Semantic 3D City Models for the Next Generation of CityGML'. In: *Advances in 3D Geoinformation*. Ed. by Abdul-Rahman, A. Cham: Springer International Publishing, pp. 191–206. URL: [https://doi.org/10.1007/978-3-319-25691-7\\_11](https://doi.org/10.1007/978-3-319-25691-7_11)

11. Moshrefzadeh, M., **Chaturvedi, K.**, Hijazi, I., Donaubaue, A. and Kolbe, T. H. (2017). 'Integrating and Managing the Information for Smart Sustainable Districts - The Smart District Data Infrastructure (SDDI)'. en. In: *Geoinformationssysteme 2017 – Beiträge zur 4. Münchner GI-Runde*. Ed. by Kolbe, T. H., Bill, R. and Donaubaue, A. Wichmann Verlag. URL: <https://mediatum.ub.tum.de/doc/1350813/46723.pdf> .
12. **Chaturvedi, K.** and Kolbe, T. H. (2016). 'Integrating Dynamic Data and Sensors with Semantic 3D City Models in the context of Smart Cities'. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences IV-2/W1*, pp. 31–38. URL: <https://doi.org/10.5194/isprs-annals-IV-2-W1-31-2016>
13. Yao, Z., **Chaturvedi, K.** and Kolbe, T. H. (2016). 'Browser-basierte Visualisierung großer 3DStadtmodelle durch Erweiterung des Cesium Web Globe'. de. In: *Geoinformationssysteme 2016 - Beiträge zur 3. Münchner GI-Runde*. Ed. by e.V., R. T. G. Runder Tisch GIS e.V. München: Wichmann Verlag, pp. 77–89. URL: <https://mediatum.ub.tum.de/node?id=1296408>
14. **Chaturvedi, K.** and Kolbe, T. H. (2015). 'Dynamizers - Modeling and Implementing Dynamic Properties for Semantic 3D City Models'. In: *Eurographics Workshop on Urban Data Modelling and Visualisation*. Ed. by Biljecki, F. and Tourre, V. The Eurographics Association. URL: <http://dx.doi.org/10.2312/udmv.20151348>
15. **Chaturvedi, K.**, Yao, Z. and Kolbe, T. H. (2015). 'Web-based Exploration of and Interaction with Large and Deeply Structured Semantic 3D City Models using HTML5 and WebGL'. en. In: *Bridging Scales - Skalenübergreifende Nah- und Fernerkundungsmethoden, 35. Wissenschaftlich-Technische Jahrestagung der DGPF*. Ed. by Kersten, T. P. Vol. 24. Deutsche Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V. Köln: Deutsche Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V. URL: <https://mediatum.ub.tum.de/node?id=1245285>