GEOGRAPHY DOCTORAL SCHOOL
Faculty of Science

From consistency to flexibility: shifting the structure

Towards a new generation of geographical information systems

Gilles-Antoine Nys

Supervisor:
Prof. R. Billen - ULiège

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Science (PhD): Geography

June 2023

From consistency to flexibility: shifting the structure

Towards a new generation of geographical information systems

Gilles-Antoine NYS

Examination committee:
Dr. J.-P. Kasprzyk - ULiège, chair
Prof. R. Billen - ULiège, supervisor
Prof. P. Hallot - ULiège
Prof. B. Kuijpers - UHasselt
Prof. H. Ledoux - TU Delft
Prof. C. Debruyne - ULiège

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Science (PhD): Geography

June 2023

Acknowledgement

My first thoughts can only go to Prof. Roland Billen. For giving me not only the management but also for the opportunity to prove myself and the number, too great to count, of opportunities. The cover of this thesis bears my name but is the result of the work of a duo.

Members of my Thesis Committee (Prof. P. Hallot and Prof. B. Kuijpers) for their support and the influence they have had on my developments and findings. I greatly benefited from their experience in writing this document.

Members of my Thesis Jury (Prof. H. Ledoux, Prof. C. Debruyne and Dr. J.-P. Kasprzyk, as a chair) for taking the time to be interested in this work and their participation during the examination.

All the members of the "Geomatics Unit" past and present, for the more than beneficial atmosphere of the daily work.

Among them, the doctoral students, half of which have gone before me and the other half will go after, for the questions and answers that we have stated together.

Thanks that are more general will go to all people that I have met during my years at the University of Liège and in various research projects. Each of them have indirectly contributed to the experience that a thesis project represents.

Finally, Charlotte, nothing that needs to be said. This document is just one step in the endless amount of things we will do together.

Abstract

Smart Cities, *Enhanced 3D city models*, *Digital Twins*, *Small scale city models*, *Urban Shadows*, all combinations of these words, etc. are terms we hear more and more. No matter what they are called, they can be summarised as enhanced 3D city models. Behind these names are hidden concepts, algorithms and systems that focus on opening information and providing smarter tools. Implemented mechanisms should help people to reach an improved level of support in decision-making (crosschecking of information, reasoning, machine learning, etc.). The urban built environment, because of its social, political and economic components, has so many specificities, that it is necessary to propose dedicated management tools in addition to a purely geometric management of the data. The above-mentioned terms translate the increasing need to provide new solutions for an integrated data management in a digital replica of a city.

Three pillars are mandatory in order to achieve such an integrated basis: (1) usability, (2) interoperability and (3) maintenance. Things should be understandable in order to open the use by the greatest number and thus provide the full potential of shared information. Some elements could still be considered as advanced and provide additional information and complexity but the base should be kept simple. The second imposes elements to communicate in a standardized way: storages and exchanges should follow an unified schema in order to ease communication between them. Thus, information conveys data understandable by all, and everyone knows where to find the desired information in the model. Constituting stacks should be documented so that the standard can evolve or be modified but still operates in a manner that allows other application to communicate. External connections should also be made in a manner that encourages cross-compatibility. Finally, the third focuses on providing tools that handle data during all its lifecycle and ease its management. From its creation passing by modifications to its destruction, every single step should be handled in order to alleviate problems due to complex processing chains.

This thesis proposes to renew the model storage from a rigid and consistent tabular structure, which are relational databases, to a flattened solution that should improve flexibility (i.e., a document-oriented NoSQL solution). This ambitious goal is structured in the form of a "three-paper" thesis with considerations that place them in the main objective. Chapters are thus divided in three corresponding researches that concern respectively: the data/clients, the database tier and the server tier. The first contribution proposes a simple generation process for the creation of CityJSON buildings models. It develops the benefit of providing a CityJSON model directly and its new supported features (metadata, etc.). The second chapter discusses the creation of a document-oriented schema that improves the city objects and features access thanks to the flattening of the model. A web architecture is proposed enjoying its flexibility. The last chapter focuses on providing a consistent way of writing and reading information using the above-mentioned schema. 3D city models are afterwards seen as a unique integration basis for information coming from heterogeneous sources. Interactions with this single data storage are locked through the provision of a middleware, a software that lies between applications and rules their exchanges.

All of these contributions are packed and staged in a coherent whole that answers a crucial question: Could a flexible web architecture gather the consistency and interoperability aspect of the CityGML data model with better flexibility to cope with a high demand for sharing by applications from different domains?

Finally, the conclusion gives an overview on the results of the thesis. Since this thesis should be seen as a part of whole that goes toward a new generation of geographic information systems, research extensions are proposed in order to make steps towards the accessibility of a digital replica of cities. These propositions are made based on current knowledge and technologies; they are part of the present time and cannot assume what will happen next.

Résumé

Smart Cities, *Digital Twins*, *Town Maquettes*, *Urban Shadows*, toutes les combinaisons de ces mots, etc. Peu importe comment nous les appelons, derrière ses termes se cachent en réalité des modèles 3D améliorés de villes. Y sont dissimulés des concepts, algorithmes et systèmes qui se focalisent sur l'ouverture de l'information et la provision d'outils intelligents. Les nouveaux mécanismes implémentés devraient aider les gens à atteindre un niveau d'aide à la décision amélioré (croisement d'information, raisonnement, machine learning, etc.). L'environnement bâti urbain, de par ses aspects sociaux, politiques et économiques, a tellement de spécificités qu'il est nécessaire de proposer des outils de gestion dédiés en plus de la gestion purement géométrique de l'information. Les termes susmentionnés traduisent le besoin croissant de proposer de nouvelles solutions pour une gestion intégrée de la donnée dans une réplique numérique de la ville.

Trois piliers sont nécessaires pour atteindre une telle base d'intégration: facilité d'utilisation, interopérabilité et maintenance. Le premier point concerne la facilité d'utilisation par le plus grand nombre qu'ils aient des connaissances ou de l'expérience. Les choses doivent être compréhensibles de sorte à répandre l'utilisation et de fournir ainsi tout le potentiel des informations partagées. Certains éléments peuvent tout de même être considéré comme « avancé » et fournir de l'information supplémentaire et de la complexité mais la base se doit de rester simple. Le second point impose les éléments à communiquer d'une façon standardisée. Les stacks constitutives doivent interopérer de manière documentée. Les connections externes aussi doivent être faites de façon à promouvoir l'inter-compatibilité. Enfin, le troisième point s'attache à fournir des outils qui permettent de manipuler la donnée durant tout son cycle de vie et d'en faciliter la gestion. Depuis sa création en passant par ses modifications jusqu'à sa destruction, chaque étape devrait être gérée de façon à réduite les problèmes dus à de trop complexes chaines de traitement.

Cette thèse décompose une architecture web trois-tiers dans le but d'illustrer le

changement de vision pour le stockage des modèles urbains à trois dimensions. Ce changement renouvelle le mode d'enregistrement en passant d'une structure tabulaire rigide et consistante, que sont les bases de données relationnelles, vers une solution aplanie qui devrait améliorer la flexibilité (c.à.d. une solution NoSQL orientée document). Cet ambitieux objectif est structuré sous forme d'une thèse « sur articles » agrémentés de considérations qui les replacent dans le cadre de l'objectif principal. Les différents chapitres sont divisés de façon à correspondre aux recherches qui concernent respectivement: la donnée et les clients, la base de donnée et le server. La première contribution propose un processus allégé de génération pour la création des modèles CityJSON des bâtiments. Il développe les avantages d'une production directe de modèle CityJSON et le support de nouvelles fonctionnalités comme les métadonnées. Le second chapitre discute de la création du schéma de base de données orientée-document qui améliore l'accès au modèle urbain et à ses entités grâce à l'aplatissement du modèle. Une architecture web complète est proposée pour profiter de ce gain de flexibilité. Le dernier chapitre s'occupe de fournir une méthode consistante d'écriture et de lecture de l'information en utilisant le schéma susmentionné. Les modèles urbains à trois dimensions peuvent dès lors être considérés comme une base d'intégration unique pour l'information venant de sources variées. Les interactions avec cet enregistrement unique de la donnée sont cadenassées grâce à un middleware, un logiciel qui se place entre les applications et régit les échanges entre eux.

Toutes ces contributions sont empaquetées et mises en musique dans un tout cohérent qui répond à la question cruciale : Est-ce qu'une architecture web peut rassembler les aspect de consistance et d'interopérabilité du modèle de données CityGML avec une meilleure flexibilité de façon à faire face à une demande toujours plus importante en termes d'échange entre les applications dans des domaines variés?

Enfin, les conclusions donnent une synthèse des résultats de la thèse. Etant donné que la thèse doit être vue comme une partie d'un tout qui va vers une nouvelle génération de systèmes d'information géographique, des prolongements de recherche sont proposés de sorte à faire un pas de plus vers l'utopique « Digital Twin ». Ces propositions sont faites sur base de l'état actuel de nos connaissances et technologies ; ils font partie du temps présent et ne peuvent pas présumer de ce qui se passera ensuite.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Preamble

Any good story begins with the presentation of the context, its actors and must give the reader the main keys to reading. The purpose of this chapter is not to present an exhaustive state of the art on which the remainder of this thesis will be built on. It is rather a contextualisation for the reader who does not have any prior experience or knowledge in 3D city modelling and its standards. Without going deep into details, it sets out generic principles and best practices giving the reader insights into what follows. The tone is deliberately more direct and lighter what will be changed for the rest of the document, and this from the introduction.

City modelling is the set of processes, models and data that allow representing a city, its features and the potential relationships between them. Representing and proposing a digital replica of the urban built environment is nothing new: for a decade, a large American company with a strong presence in terms of mapping on the web provides 3D city model as a collection of detailed and textured triangle networks. Such models allow people to navigate into them and offer the possibility to travel in photos tunnels to find their bearings: their home, their favourite bakery, etc. In other terms, such photo-realistic rendering provides citizens with a pleasant but limited contextualization from the perspective of operators and stakeholders.

Regarding most analyses, simulations and modifications, such a monolithic composition of triangles is useless. It is not possible to manage the elements independently: a house can neither be extracted by itself nor have discrete

information included in it. By way of example, one can easily imagine the usefulness of recording the living areas of houses, for the assessment of a property tax for example, the maximum speeds of road sections, the species of trees, etc. This hulking copy of real-world assets strongly limits the possibilities offered by its use in advanced processes. Figure 1.1 depicts the two alternatives: a huge network of triangles that could be easily created using state-of-the-art photogrammetry processes and a discrete management of city features that allows storing information on each of them but comes with several added challenges.

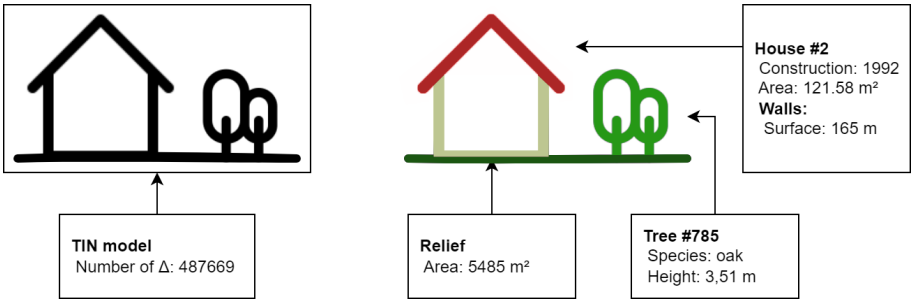

Figure 1.1: Discrete information management

Discrete management of the city elements should provide opportunities in many domains and applications given that users have a finer management of objects and their information. One can, for example, find all the buildings which have a gable roof, etc. This knowledge can take the form of a geometry (the physical representation of an entity) and/or a set of attributes (a characteristic or inherent part of an entity).

Going a little further, these data, and mainly for the geometric part but not only, concern spatial information on which it could be interesting to build a reflection. For instance, in such a management regarding spatial queries, some might be interested in having all the houses that will be affected by a new nightclub and its noise pollution, the area of fields that should be expropriated if a road crosses two municipalities, etc. Such examples are limited to two-dimensional questions but if the data allows it, each response could provide information about height or elevation: in the event of a flood, where are the building that will be flooded? Giving that a skyscraper will be built, will not the urban canyon be too uncomfortable for the city people?

All this seems interesting and the added value of such control is indisputable. Still, it comes with benefits and drawbacks. By way of technical explanation, questions on creating these independent elements, to store them in a persistent

manner and to access them and their related information separately arise. In other words, there is now a need for tools to take advantage of such individual and specific administration throughout the entity life cycle.

Thereby, such specific and public utility questions on floods, building projects, etc. require virtual replicas and an integrated approach that involves many actors and data sources. The start of a blank page will lead to creating as many models and querying as many solutions as applications: a standard, something set up by authority or by general consent as a rule or as a model, should be assessed. And of course, compromises must be made in this sense. This is the purpose behind the creation of the CityGML data model (Gröger and Plümer 2012; Kutzner et al. 2020).

Quoting the Open Geospatial Consortium (OGC) website in charge of providing spatial standards directly:

[...] The CityGML 3.0 Conceptual Model Standard describes a common semantic information model for the representation of 3D urban objects. The primary function of the model is to define the human interpretation of modelled data objects as well as their geometric representation and relationships. [...]

In other words, this common base provides definitions of abstract city objects and their potential relations and associations: what is a building exactly? How do we define a road and its characteristics? Is it possible for a tree to have an address? All these answers can be found in the CityGML data model. For instance, an important notion related to CityGML are the levels-of-detail. These levels define the degree of representation, the amount of elements that will be represented, generalized, etc. In short, the first level is limited to the building footprint. The second extrudes this footprint from an arbitrary height. Then, the roof shapes are represented and finally the openings (doors, windows, etc.).

This data model is a set of conceptual specifications that remain abstract and not exploitable in this form by "computer processes". Keeping this as is is not workable beside a support for discussion between designers: it is necessary to translate this model into something more machine-friendly. Thus, several uses are made of the OGC urban modelling standard. Two encoding formats and a database logic schema have been developed for exchanging and making this information available:

- **CityGML:** GML encoding of the CityGML data model (they have the same name). This encoding has been developed at the same time as the data model. It is the most direct use of the data model. However, as the reader will understand throughout this document, it ages badly.
- **3DCityDB:** a geo-relational database to store, represent, and manage

virtual 3D city models. More than a single database schema, a whole platform for the management of CityGML models is proposed. It is closely linked to the XML-encoding and suffers of the same drawbacks while enjoying its benefits (Yao et al. 2018).

- **CityJSON:** The newcomer: a lightweight alternative to the XML-encoding which focuses on providing easy-to-use tools and provides a solution to spread the use of the model. In short, it is a JSON-encoding of the CityGML data model, or at least a large part of it. It eases development of applications and provide web-oriented solutions. The web evolves, it was with this in mind that it was created (Ledoux, K. A. Ohori, et al. 2019).

These different uses of the CityGML data model will be referenced and discussed more in details through the chapters of this document. However, it is good to highlight here the work of the 3D Geoinformation Research Group at the Delft University of Technology in the Netherlands, especially for CityJSON. Their work and the resulting tools have marked each step of this thesis. I will mention in particular: CJIO, a Python Command Line Interface (CLI) to process and manipulate CityJSON files, CJVAL, a validator for CityJSON files, and their FME extension for the support of CityJSON.

Many questions have certainly arisen on reading this preamble. The reader will find partial answers linked to CityJSON, to modelling principles and this mainly with a view to presenting a new database storage method (see Chapter 3). It is not possible to cover all these subjects. However, asking questions would at least have the merit of having properly contextualized the subject and aroused interest.

1.2 Context

The digital transformation and related new technologies increase the use of 3D models of the built environment. In addition to an approach that would emphasize the establishment of modes of governance, new opportunities open possibilities for the usage of these models as capabilities are created. The multiplicity of actors, research applications and uses cases leads to various models to face built environment modelling challenges. Indeed, the absence of a generic framework for the creation and management of these models prevents it to move towards a utopian, shared, interoperable and global modelling. One trend, however, has attempted to put a name on the process that would make a city smarter without providing a consistent definition or standardized

requirements: a Smart City. It should be noted, however, that these terms,as well as variants as "Smart Region" or "Smart Territory", serve more as a sociological or governance definition than as a real technical solution. The difference is slight for the uninformed reader but still essential.

Smart cities were the trending topic of the past decade. The idea of having a so-called smarter municipality has appealed to many actors and stakeholders. Many trades, such as modellers, policy-makers, etc. have brought their insight in the development of what these cities should look like. It was first describing the idea that a city could be enhanced by a myriad of sensors producing consequent amount of information. The Internet of Things, a system of interrelated measuring devices, was to embrace the city and allow all aspects of it to be measured. Still, differences in the language used and the advertising were observed. In the end, since everyone works in isolation, each research or development project could bring its own model.

Over time, a shift has taken place in the very expression of the definition of what Smart Cities are. The definition has finally nothing to do with any technical aspect. Indeed, while geomaticians, or at least those who come close, were interested in modeling, generating, storing, etc. city features and their relationships, a policy-making approach has taken over. In addition, de facto, it is now in a more "governmental" approach. The illustration of this fact is simple and direct looking at the European Commission of what a “Smart City”[1] is:

A **Smart City** is a **place** where traditional networks and services are made more efficient with the use of digital solutions for the benefit of its inhabitants and business.

This definition is not a technical definition. And it has not be intended to be one. Nevertheless, in the scope of this thesis, we will take a look at it by considering it as such: it demonstrates that "digital solutions" are good umbrella terms. The common good is obviously at the heart of the definition: responsive city administration, safer public spaces, needs of an ageing population, etc. It does not refer to any financial aspect but a managerial and administrative aspect. Performance improvements are also part of the trend, but these points are classic in all technological advances and are obviously not related to the notion of computer performance as computer scientists are used to it. However, two points should be noted in the scope of any scientific progress: the use of digital solutions and the shift from traditional solutions to new structures and technologies. For instance, the management of a cadastral repository in a

[1] https://ec.europa.eu/info/eu-regional-and-urban-development/topics/cities-and-urban-development/city-initiatives/smart-cities

spreadsheet, using proprietary or open source solutions, is a digital solution as well as taking pictures of a street with a smartphone. It lacks, in our opinion, of a more reinforced technical point of view. On the other hand, the structure shift from traditional solutions to more efficient capabilities is large also. What is efficient? Is it related to performances, capabilities, features, etc.?

Nowadays, "Digital Twins" bring a more technical definition of what a digital model should be and some starting insight on its requirements. Putting aside Digital Twins that are not related to the urban environment and since there is currently no shared definition, this approach can be defined in a first step as:

A **City Digital Twin** is a **virtual representation** of a city and its assets that spans their life cycle, is updated and uses simulation, machine learning and reasoning to help decision-making.

Just like the definition of Smart Cities given above, the statement of this definition is nothing more than a support of discussion that emerges from our participation in different projects related to modelling cities and improving these models capabilities. Particular attention has been paid workshops organized by the EuroSDR in early 2022 (Ellul et al. 2022). These meetings gather national agencies, research centres, universities, etc. around the idea to work all together on a shared framework for the development of geospatial solutions: business models, modelling, information usage, education, etc. A common will aims at proposing a definition that makes consensus on Digital Twins. Nevertheless, this remains without result after a year of discussions. This kind of issue takes time and it is imperative to take this time in order not to repeat the mistakes made in Smart Cities. Moreover, this interpretation has not been validated or discussed in a research at the time of writing. It is therefore still in the thinking stage. A trend is emerging and this thesis attempts to provide a vision in line with it.

Back to the definition, avoiding complex technical aspects, three points should be noted:

- Sustainability of the representation: modelling the urban environment should address current needs, from its creation to its deletion and all events that may occur in between.
- Follow-up of updates: the core model should be modular in order to address future needs, allow new applications, improvements, manage new users, etc.
- Enriched with all kind of data and algorithms: people must be able to appropriate the model, modify it, improve it, etc. Moreover, break it into

its constituting parts in order to propose new elements. External bindings and features enhance the model and add value to the decision making. This must however be made keeping in mind backwards compatibility as we will see later.

The integrating basis of all these considerations should be a Spatial Data Infrastructure (SDI) or a Geographical Information System (GIS) no matter what it is called. As stated by Stoter (Stoter et al. 2020), the upcoming challenges for offering a wide and open use of 3D city models are their usability, interoperability and maintenance. Such an architecture, web-oriented in this case, must be based on a conceptual data model that allows for openness but also ensures that the exchanges between the different components are consistent. The purpose of a shared database and its ease of use are undoubtedly linked to its intrinsic structure. It would not be logical to propose an ultra-complex structure and think that most people will want to use it. Several intricate use cases are necessary but the vast majority of users can be satisfied with a relative use.

Naturally, in the scope of the urban built environment, the choice fell on CityGML. CityGML is an unavoidable standard in the scope of 3D city modelling. It allows modelling city objects, their relationships and allows the addition of features. Several ports of this data model are already in use.

1.3 Research questions

Among all the modelling perspectives, this data model and its associated relational database model guarantee the consistency and interoperability of built environment models. However, CityGML is not flexible due to its rigid tabular storage and XML encoding. The aspect of data maintenance that is exhausted by Digital Twin application is not encountered well by CityGML. In a process focused on providing developer-friendly and lightweight solutions, CityJSON offers an easy-to-use alternative to the GML encoding of CityGML for the storage and exchange of 3D city models. This OGC community standard is the starting point of this thesis.

In the continuity of this process that promotes the JSON encoding, the shift from rigid structure and the openness of models, all of these considerations led to the definition of a specific research question that structures this thesis:

Could a flexible web architecture gather the consistency and interoperability aspect of the CityGML data model

with better flexibility to cope with a high demand for sharing by applications from different domains?

This research question, as it represents an ambitious question, has been divided in multiple questions on specific web-architecture tiers (respectively the client, the database and the server):

1. By focusing on the geometric generation process, is it not possible to improve the accessibility, open the use and refine the design avoiding excessive model complexity? What are the clients' advantages of generated CityJSON models for sharing?
2. Does moving from relational databases and their rigidity to document-oriented NoSQL databases and their flexibility open up data for modification and sharing? Is the consistency of the data at risk?
3. How to guarantee the consistency of the architecture when the database is now flexible? What happens if many users access the data from an access control point of view (versioning, hierarchy, security, etc.)?

This thesis postulates that NoSQL document-oriented and web-related technologies can lead to a structure shift in the data life cycle management and its flexibility. The resulting architecture should reconcile the flexibility, interoperability and consistency of built environment 3D models management. It therefore proposes answers to these research questions in three separate chapters: model generation, database schema and consistency guarantee (see Figure 1.2).

Given that these three chapters are very different in their subjects and thus also in their writing, the research methodology is presented in each of them independently. If, however, it should be necessary to define a general methodology, it would be an "experimental methodology": starting with a research question, we made hypothesis (for instance, a reviewed objects structure should improve the model accessibility and ease its use). We then developed a solution that emphasise this hypothesis and confront it to other well-known solution in order to state on their respective benefits and drawbacks.

1.4 Thesis outline

These chapters are based on three articles that were reviewed in international journals throughout the doctoral research project. Since we have had the

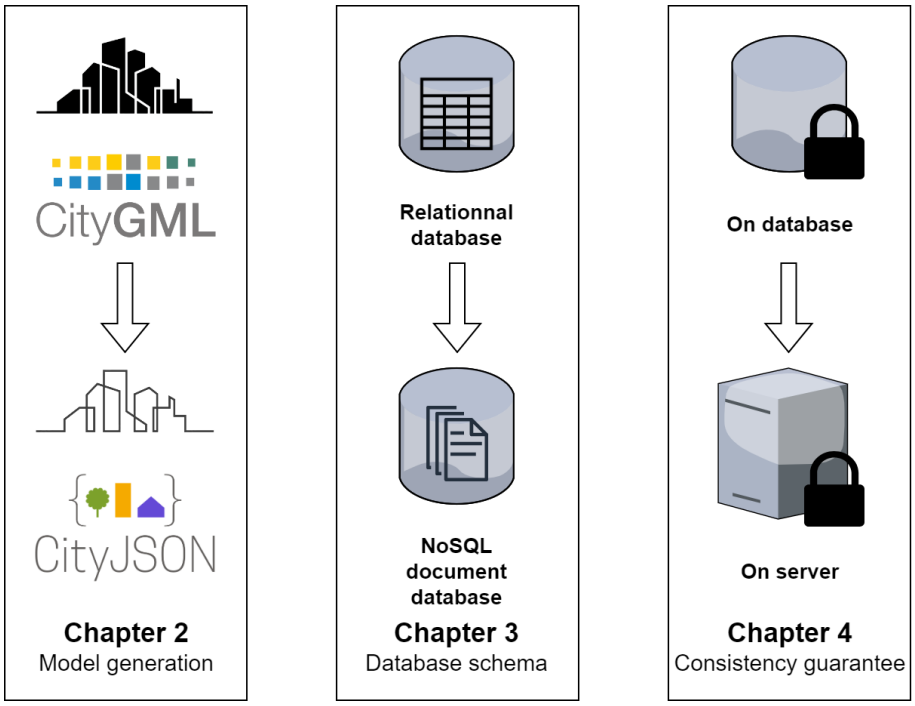

Figure 1.2: Articulation of architecture tiers associated to the thesis chapters

opportunity to modify them within the scope of this document, we propose a brief review of them in order to facilitate their understanding. They are therefore not exactly the same as those published. Before these published contributions, specific preambles introduce each chapter in order to refine the context of the writing. Subsequent works based on main articles (other papers and developments as conference papers and related projects in which we were involved) are also described after each chapter as they strengthen the whole process and illustrate the new capabilities of the structure shift.

The **Chapter 2** develops the methodology that has been implemented to generate lightweight and consistent CityJSON building models from point clouds. The answer proposed through this research highlighted the accessibility and sharing capabilities of a simple but efficient process avoiding common mistakes made in 3D city modelling such as levels of detail that are not very expressive or even undefined geometric validation thresholds. It shows new advantages for the sharing of models thanks to metadata and shared information.

The **Chapter 3** describes a simplified schema for CityJSON that allows storing 3D city models in a document-oriented NoSQL database. Drawbacks and benefits regarding differences with relational solutions are evaluated. All the hypothesis and concerns made around the schema development are discussed and illustrated thanks to two practical applications. In support of all these points, a web three-tier architecture is created and documented.

The **Chapter 4** focuses on the development of a middleware, a software that lies between applications, to guarantee the consistency of a city digital model management and its integrity. It illustrates new capabilities offered by the coupling of the above-mentioned NoSQL database with CityJSON clients and applications. Its use is justified due to the needs of various levels of accessibility, versions, etc. The common thread is the emphasis on flexibility by shifting the guarantee of consistency to another element of the architecture than the database (i.e., the consistency is no more guaranteed by the database so it was mandatory to report it on another stack). Besides this limitless querying method, a more standardized one is provided to enable interoperability and allow reaching information: OGC API - Features.

The **Chapter 5** summarizes the previous three and provides illustrative examples of what can be achieved through the structure shift and the supplied tools. A quick benchmark is also proposed since it has one of the major advantages of this contribution: ease of access and development. Several example of queries in both solutions (relational and the current) are proposed. It highlights the benefits of the contribution without neglecting its drawbacks.

After these three main chapters, which represent the bulk of the contribution, the **Chapter 6** closes on the logical articulation of the three previous chapters. The connection made between the previous chapters highlights the shift from consistency to flexibility and try to propose an alternative to what already exist. This section reviews the answers to the questions stated in the introduction or at least ways to think about it. It places this research in the current context, several years after the one presented in the introduction.

Finally, insights into the remaining challenges conclude this thesis. This last section is a loose closure as the research is still ongoing and will never end. This thesis is intended to be a punctual element in a coherent evolving trend but does not represent an end in itself.

Important note: from now on, all words in italics are elements of the CityGML data model. We will not redefine these terms and refer the reader to the definition of the terms in the data model.

Chapter 2

Model generation

Preface

This preamble puts the writing of the second chapter (i.e., the first published paper) in its context in mid-2019. The basic idea at the start of the thesis project was to state on modelling cities in three dimensions, the standards and their related storage modes. Within the framework of a parallel project, our mission was to make a relevant state of the art in standards related to sensor data management.

The main objective is to integrate dynamic data (i.e., data that are updated frequently such as sensor data) in a more static urban model, which update frequency is much slower. The conclusions are straightforward: the new Open Geospatial Consortium (OGC) standards focus on a shift from the traditional XML-encoding to JSON-encoding. It encompasses its lightness, its easier way to parse and its developer-friendly usage. For instance, the Internet of Things standards shifted from the Sensor Observation Service (Na and Priest 2007), which was verbose and not easy to handle, to the smarter SensorThings API (Liang, C.-Y. Huang, et al. 2016; Liang and Khalafbeigi 2019). The new Application Programming Interface (API) puts forward its flexibility and its readability simplifying the conceptual model but also supporting alternate encoding formats. It demonstrates that the change started towards easier exchanges and mutualisation of services on the web. Indeed, JSON is a great exchange format and it felt that it would influence many further developments in various domains.

At the same time, in the 3D City Modelling scope, CityJSON has just been made open and its definition article has been published by Prof. Ledoux and its team (Ledoux, K. A. Ohori, et al. 2019). It renews the CityGML model usage making a step forward in this dynamic focused on compact and open-source standards. Not all city elements and CityGML features are yet supported (several coordinates systems in the same dataset, raster files, etc.). However, it supports new capabilities such as adding information such as metadata and refined levels of detail.

Regarding the availibility of CityJSON models at that time, the official site provided examples of modelled neighbourhoods for big cities (Rotterdam, Vienna, New York, etc.). In these model extracts, *Buildings* are available with more or less details at least. The dummy railway example from CityGML is also on hand.

A negative point was worth pointing at the time: the available CityJSON models are translated from original CityGML models conceding a part of information loss. This point is not really damaging, but in fact, in our opinion, it missed part of the advantages of CityJSON. A single relevant source talked about generating

CityJSON models that were not translated from CityGML models but created from "raw" data (Kumar, Ledoux, et al. 2018). It relies on a well-known process called 3dfier (Ledoux, Biljecki, et al. 2021). This process extrudes polygons thanks to approximated heights from an airborne LiDAR point cloud (Light Detection and Ranging). After determining elements heights, it interpolates vertical planes thanks to predefined rules on object classes. It is still a very powerful rule-based tool that allows creating many features (not only *Buildings* but also nearly all city features). However, no other model generation process or even a process by which the roof shapes were obtained was yet available. Roofs were flat surfaces, with more or less details that correspond to LoD 1.x: a single plan or several parallel plans for the same building. Moreover, semantic surfaces were not always accessible which could be easily solved in a clever process.

As the list of our publications shows (see List of publications), point clouds offer new possibilities and a much higher level of detail than was possible at the time thanks to an upstream segmentation. We thus decided to capitalise on this foundation and study these capabilities in order to generate roof-shaped CityJSON buildings models. It has been done with varying degrees of success using smart point clouds and connected-components segmentation.

Based on article (Nys, F. Poux, et al. 2020)

CityJSON buildings generation from airborne LiDAR 3D point clouds

Nys, G.-A., Poux, F., & Billen, R. (2020). *ISPRS International Journal of Geo-Information*, 9(9), 521.

Abstract: The relevant insights provided by 3D city models greatly improve Smart Cities and their management policies. In the urban built environment, buildings frequently represent the most studied and modelled features. CityJSON format proposes a lightweight and developer-friendly alternative to CityGML. This chapter proposes an improvement to the usability of 3D models providing an automatic generation method in CityJSON, in order to ensure compactness, expressivity, and interoperability. In addition to a compliance rate in excess of 92% for geometry and topology, the generated model allows the handling of contextual information, such as metadata and refined levels-of-details, in a built-in manner. By breaking down the building generation process, it creates consistent building objects from the unique source of LiDAR point clouds.

Keywords: LiDAR; 3D City Models; CityJSON; Smart Cities; Point Cloud; Segmentation; 3D Modeling.

2.1 Introduction

The relevant insights gained from the use of digital models makes it possible to manage cities in a smarter way. Due to the significant dynamism of *Smart Cities*, considering many factors is essential for environmental diagnostics, urbanism, etc. Amongst these factors, the 3D representation of the urban fabric has a particular role to play; it serves as an integration layer for the other factors and related data (R. Billen, Cutting-Decelle, et al. 2014). In that context, standard models, such as CityGML, have emerged with a wide variety of linked 3D City object classes and features (Gröger and Plümer 2012). The usability of a 3D city model depends on its quality, on the reliability of its 3D city objects (e.g. semantic, geometrical, topological, and temporal accuracies), on its availability, and on its degree of versatility: compactness, expressivity, and interoperability.

Generating city models from geospatial data in CityGML is a standard procedure (Wang et al. 2018). However, as it is currently used, CityGML has several limitations concerning level-of-detail definitions (Biljecki, Ledoux, and Stoter 2016), a lack of standardized and normalized comparison tools (Biljecki, Ledoux, Du, et al. 2016), and contextual documentation (i.e., metadata) (Labetski et al. 2018).

This chapter is intended to address the automatic generation of compact and consistent 3D city models using LiDAR point clouds and CityJSON (Ledoux, K. A. Ohori, et al. 2019), a lightweight alternative to CityGML. The advantages of using CityJSON have already been demonstrated. Despite this, it is currently rarely used in practice; one of the reasons for this is the lack of publicly available CityJSON models (https://www.cityjson.org/datasets/), which is a consequence of limited CityJSON generation methods. In this research, we propose a new CityJSON generation method, which relies on state-of-the-art building creation components from LiDAR data (Tarsha Kurdi and Awrangjeb 2020; Wang et al. 2018).

In addition to the benefit of the compactness involved in using CityJSON, CityJSON's direct generation method enables native handling of metadata and refined levels-of-detail. These two points should improve the usability of 3D city models and offer the possibility to create them in a consistent and direct manner. This article is structured as follows. First, we present CityGML and CityJSON data formats, as well as the main building generation methods from LiDAR data (Section 2.2). Then, we outline every step of our methodology for producing CityJSON buildings from 3D point clouds (Section 2.3); namely, PC segmentation, step-by-step geometric modeling, and CityJSON model formation. In Section 2.4, we discuss the advantages of directly generated CityJSON models and assess the quality of the results, using several normalized and formal tools.

Finally, we outline our conclusions in Section 2.5.

2.2 Related Works

2.2.1 Building generation methods

Among 3D City objects, buildings are the backbone of many Smart Cities applications. Undoubtedly, providing a coherent geometric reconstruction remains a challenge. LiDAR (Light Detection and Ranging) is a valuable method of data acquisition for 3D modeling. Given that point clouds segmentation and classification are also becoming increasingly efficient, the use of information derived from airborne LiDAR data (i.e., Airborne Laser Scanning (ALS)) is overwhelming and prolific. Its use helps in many domains, such as the reconstruction of 3D city models, hazard assessment, forestry, geological mapping, watersheds, and river surveys. The large contribution of classified and semantic information to decision-making is indisputable (F. Poux, Hallot, et al. 2016).

A classification for urban model reconstruction has been proposed from LiDAR data (Wang et al. 2018). It classifies methods into three families: (1) data-driven, (2) model-driven, and (3) hybrid-driven. This allows people to compare city generation methods. Data-driven methods (prismatic and polyhedral) produce very accurate representations of urban scenes, especially free-form rooftop modelling (Lafarge and Mallet 2012). These representations are very promising for rendering real world scenes, as they consider many classes of urban objects (buildings, vegetation, and roads). However, when they are exchanged and used online or via cell phone, their size is often a significant drawback.

Coupled with other sources of information, such as building footprints and airborne imagery, LiDAR point clouds make it possible to recreate a representation of reality. However, airborne imagery is not suited to the generation of accurate and compact models. Hence, LiDAR sensors are more favorable to the generation of rooftops than imagery sensors. Indeed, many issues arise from the use of optical imagery: shadows, occlusions, texture problems, variations in brightness and contrast (disparity and entropy) (Jung and Sohn 2019). LiDAR is less influenced by these external factors (Tarsha Kurdi and Awrangjeb 2020). Moreover, as terrain relief causes relief displacement and occlusions in airborne images in oblique imagery, several LiDAR points mean the height of the same image pixel, depending of its ground-sampling distance. This consideration leads to complex dense image matching and invariably results in an additional source of error. Still, imagery is a good source of information

to detect changes and update changed areas (Tarsha Kurdi and Awrangjeb 2020). Recent works, however, are providing enhancements for the accurate use of LiDAR and imagery fusion (Zhou et al. 2020).

In terms of the data source itself, several efforts are advised to confront the sparse nature of LiDAR data: in particular, tracing the plane boundaries from the point clouds directly (Cao et al. 2017). On the other hand, the two most common existing methods for shape detection are RANdom SAmple Consensus (RANSAC) (Schnabel et al. 2007) and the Hough transform (Ballard 1987). However, even if the Hough transform could be improved by providing a dedicated data structure (Borrmann et al. 2010), RANSAC currently runs quicker (computationally more efficient) and is better tailored for detecting the shape of roof planes (X. Liu et al. 2019; Tarsha-Kurdi et al. 2007). Furthermore, RANSAC presents a higher robustness to outliers, as shown in (L. Li et al. 2017).

A major drawback to the use of non-parametric shape-detection algorithms is the definition of the initial parameters, which restricts the generalization potential. Due to the sparse point density, the non-deterministic nature of RANSAC might detect inconsistent shapes. Depending on the starting points, results may differ. This problem is less common in high density point clouds, given that cluster junctions are traced more efficiently. Plane detection could thus lead to false positives and/or false negatives or spurious planes (Xu et al. 2015). To avoid false detections, the tuning of parameters is often entrusted to an expert, especially in the case of LoD2.x Buildings (Pârvu et al. 2018). For example, note that a minimum area for the detection of roof planes can be problematic, since areas under 50 square meters have already showed poorer quality results (Awrangjeb et al. 2018; Rottensteiner et al. 2014).

Polygon generalization and shape regularization are also a source of errors, a source which is avoided in the proposed methodology. After discussing the pipeline, the next section contrasts our methodology and the state-of-the-art methodology, breaking down every step of the method.

2.2.2 CityGML and CityJSON

The use of a collaborative data format, such as CityGML (Gröger and Plümer 2012), has been widely chosen to model cities. Recently, CityJSON (Ledoux, K. A. Ohori, et al. 2019), a compact, easy-to-use, and developer-friendly format, has also offered the possibility to structure city models and to exchange them over web-based and mobile devices. It provides an interoperable and documented alternative to CityGML. Indeed, CityJSON is part of the web-oriented evolution of the CityGML standard. CityJSON respects the same conceptual scheme as

CityGML - which means it does not require the reworking of applications from a conceptual viewpoint - but only in terms of the exchange format. Interfacing between the two models, therefore, is simple to set up. Furthermore, JSON is less verbose and eases data exchanges. From the developers' perspective, handling information with JSON makes it much easier to read and its structure easier to understand. From the users' perspective, no difference should be encountered, except for applications loading faster and reduced bandwidth usages.

Due of its lack of versatility, XML is not the best solution for generating compact information. CityGML files are often large because of their intrinsic XML format. Therefore, JSON intends to reduce exchanges in easier-to-use ways. Currently, because it is new, there are not many CityJSON applications, but scientists intend to study their potential (Kumar, Ledoux, et al. 2018; Kumar, Labetski, et al. 2019; Vitalis, K. Ohori, et al. 2019). Two open source methods are already providing solutions for generating CityJSON models: (a) 3dfier makes it possible to generate city buildings with extruding topographical data sets (https://github.com/tudelft3d/3dfier). Nevertheless, generated buildings are limited to LoD 1.x (flat roof planes). (b) The second solution is a command-line tool that transforms a CityGML model into a CityJSON model and vice-versa (https://github.com/citygml4j/citygml-tools).

Contextual documentation (i.e., metadata) regarding geoinformation has greatly improved their interoperability and usability. This general statement also applies to 3D City models. Unfortunately, CityGML does not offer native handling for this contextual intelligence. To get contextual metadata, it is necessary to use additional tools, such as the "metadata Application Domain Extension" (ADE) (Labetski et al. 2018). This ADE can be incorporated into the core schema of CityGML, but it imposes ADE support for all applications that use this model. It also makes it possible to translate the extension into its portable relational database.

As shown in previous studies (Rottensteiner et al. 2014), most building generation methods simplify their output by differentiating qualitative statements, such as the shape of roofs (gabled, sheds, lean-to, etc.), and quantitative features (metric deviation from the reality on the ground, accuracy rate, etc.) (H. Huang, Brenner, et al. 2013). Beyond the fidelity indices on planes interpolation and shape factors, few metrics guide this building reconstruction (Biljecki, Ledoux, Du, et al. 2016). By limiting the possible deviations during each stage of the creation process, it is possible to obtain a result that will most appropriately respond to normative issues. New improvements in CityGML, especially through the SIG 3D Quality Working Group (OGC 2016), provide a set of values that can improve the reconstruction quality. Furthermore, the improved level-of-details proposes sixteen levels to describe and assess the

details of a building (Biljecki, Ledoux, and Stoter 2016). These sublevels are not supported in the official CityGML specifications and are underused (Biljecki, Ledoux, Du, et al. 2016).

2.3 Methodology

The purpose of the method developed in this chapter is to break down the generation of a 3D city model based on the XYZ components of the LiDAR point cloud, in order to create a normalized city model in CityJSON. The point cloud ALS classification is assumed to be correct. It makes it possible to extract the building points independently. We decided not to use other sources of information, in order to avoid mixing data quality issues and focus on the LiDAR automatic data extraction workflow. Starting with this assumption, the following sections break down the generation model systematically after the introductory comments. Figure 2.1 illustrates the successive steps in the method: from the raw point cloud to the generated city model. First, the raw point cloud is segmented into coherent subparts, using a structure-based region growing algorithm in Section 2.3.2. Once this is done, the cloud subparts (i.e. points clusters) are then processed individually, in order to generate buildings in a step-by-step geometric modelling process. In the end, all of the reconstructed building objects are concatenated into the CityJSON city model in Section 2.3.4.

2.3.1 Introductory comments

This step-by-step method proposes a hybrid form of generation that combines data-driven and model-driven solutions, such as grammar-guided reconstruction (Wichmann 2018). The author proposes an approach that is characterized by a strong integration of building knowledge. This knowledge is modelled on a separate, multi-scale knowledge graph during this process. The proposed reconstruction pipeline is also analogous to what is offered by TopoLAP (X. Liu et al. 2019). Significant differences are noted, however: the reconstructed geometries try to generate a more detailed representation of buildings, but does not fulfil any of the CityGML specifications. In addition, airborne LiDAR data is not the only data source. Both these choices result in the multiplication of sources errors, which is avoided here.

Where it was necessary to determine certain heuristic values, we used the recommended geometric and state-of-the-art specifications of revised CityGML Levels-of-Detail (LoD) (Biljecki, Ledoux, and Stoter 2016). This point strongly

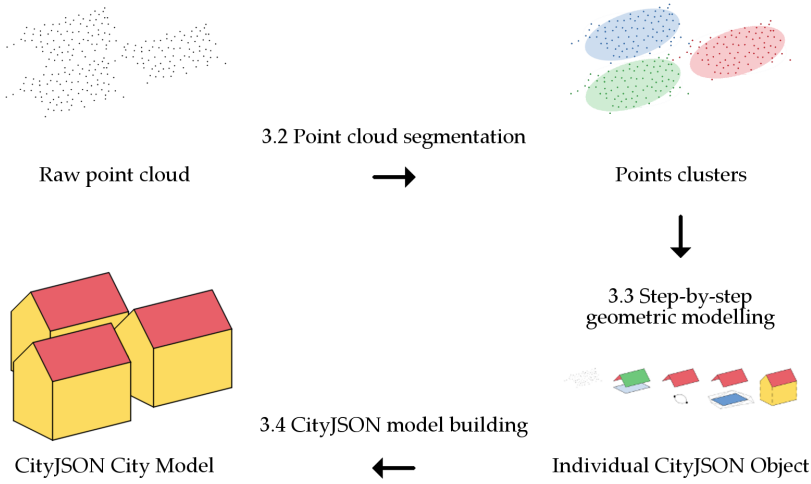

Figure 2.1: General data workflow.

distinguishes this work from others and limits the amount of inconsistency. The redefined levels of CityGML propose splitting the original LoDs into four different sublevels. The granularity of the details that are gradually added into these sub-levels provides a discrete categorization. A set of rules for simplifying 3D buildings has been set up, in accordance with these refined models (Fan and Meng 2012). Illustrated examples of these rules are present throughout the process. This makes it possible for the method to avoid inconsistencies in geometric and semantic validation.

We tested the methodology on data provided by an administrative body: the "Service Public de Wallonie" (Walloon Public Services – WPS). The airborne data were acquired in the winter of 2012. The referenced point cloud density is 0.78 points per square meter. The restricted test area is in the northern part of the city of Theux, the chief town in a rural municipality. The buildings are single-family houses, garages, and warehouses, but also shops and gathering places (restaurant, sports clubs, supermarkets, etc.) with various roof shapes. This area counts 464 elements that have been tagged as "Buildings" by the WPS. Preprocessing was limited to the application of a Statistical Outlier Removal filter (SOR filter). The filter computed the average distance of each point to its eight neighbors for each building; it then rejects points that are farther away

than the distance of one standard deviation from the average (Balta et al. 2018).

It is worth mentioning that all vector geometries are built on geometric primitives that are defined within the ISO19107 standard. Only airborne LiDAR data are used in their (X, Y, Z) shape. Python has been the preferred choice, because of its large and robust support for many libraries and its object-oriented paradigm (i.e., JSON is easier to handle using common Python scripts). It does not rely on any commercial solutions, which improves its adaptability and openness. Since it works on a different developing level (i.e., Python is interpreted, not compiled), Python is less effective than C. The points are manipulated using Python bindings from the Geospatial Data Abstraction Library (GDAL) and OpenGIS Simple Features Reference Implementation (OGR) libraries for geospatial vector data. It should also be noted that the manipulation of geometries is independent of their coordinate reference system.

2.3.2 Point-cloud segmentation

Point clouds are a simple, yet efficient, way of representing spatial data. However, despite the ease of capturing point clouds, processing them is a challenging task. Problems such as incorrectly adjusted density, clutter, occlusion, random and systematic errors, surface properties, or incorrect alignment are the main data-driven obstacles to wider adoption, and are often related to their data-structure or capture-related environmental specificities, as highlighted by F. Poux (F. Poux 2019). Secondly, structure-related problems usually emerge due to a lack of connectivity within point ensembles, which can render the surface information ambiguous (Berger et al. 2017). In order to cope with the aforementioned problems, and obtain reliable plane estimates, the plane extraction method should prioritize noise robustness. Indeed, ALS data sets often present high noise levels, which can become problematic for data-driven approaches (see figure 2.2). In this example of a projected point cloud on a vertical plane, two kinds of noise are represented: while the red points over the roof can be explained by occluded elements (chimneys, vegetation, cables, etc.), the red points over the roof are the result of walls taken with a low angle shot. The valid/invalid classification represents the goal of the segmentation. In order to avoid serialization and add fragility to the proposed approach, we favor robustness over adding a preprocessing step, such as using Statistical Outlier Removal filters (SOR filters) to filter noise beforehand.

We aim to provide an unsupervised procedure without injecting prior knowledge (whether it is from trained data or tweaking parameters). Thus, it initially focuses on extracting strict planes. In order to assess the impact of the shape detection approach on our results, we implemented another unsupervised

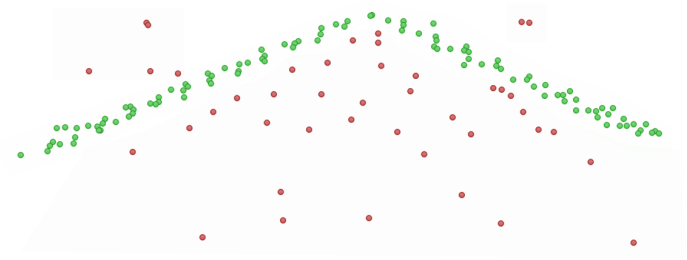

Figure 2.2: ALS noise in roof formation. Green points are points considered as belonging to the plane. Reds are not part of it under a the tolerance threshold.

approach from (Poux and Billen 2019), which was applied in an archaeological context in (F. Poux, Neuville, Van Wersch, et al. 2017) and for indoor buildings in (F. Poux 2019). Its most recent overhaul allows for the fully unsupervised segmentation of point clouds on low-end devices. The method scales up to billions of 3D points and targets a low-level grouping, in order to be independent from any application. It uses a hierarchical, multi-level segment definition to cope with potential variations in high-level object definitions. The fully unsupervised segmentation leverages planar predominance in scenes through a normal-based aggregation. For usage and simplicity, the authors designed an automatic heuristic definition for the determination of three RANSAC-inspired parameters, namely the distance threshold for the region growing, the threshold for the minimum number of points needed to form a valid planar region, and the decisive criterion for adding points to a region. The robustness of this method in various scenarios was tested for complex indoor buildings with different ground sensor platforms (depth sensors, terrestrial laser scanners, hand-held laser scanners, mobile mapping systems), but not on ALS data sets. It is robust to noise, incorrectly adjusted density, and provides a clear hierarchical point grouping, in which fully unsupervised parameter estimations give better results than "user-defined" parameters.

This process is carried out in order to obtain the n^{th} point clusters, which will become the nth support for plane estimates. In this context, it is worth remembering that, besides the use of an innovative method for point clouds segmentation, the improvement of each reconstruction step makes it possible to obtain a geometrically and topologically consistent model. Every step in the method is governed by normalized values and thresholds (Biljecki, Ledoux, Du, et al. 2016; Biljecki, Ledoux, and Stoter 2016). All these improvements make it a convincing automatic generator.

2.3.3 Step-by-step geometric modelling

CityJSON makes it possible to structure a *Building* geometry as a *Solid*, a *CompositeSolid*, or a *MultiSurface* (compliance with ISO19107 standard). Since a *Building* delimits a volume, the *Solid* primitive is entirely intended to fulfil this role in this method: a structured set of polygons that contain the closed volume of a rigid body. Note that a *CompositeSolid* might be an improvement for buildings annexes. On the other hand, *MultiSurface* represent non-volumetric parts (e.g., the overhang of a roof). The goal of the building reconstruction process is to determine the bounding polygons and the relationship between them, in order to produce a consistent model. The reconstruction method starts from the points, which is the information directly provided by the LiDAR point cloud. Based on the detected roof planes (in red), the footprint (in blue) is computed as the projection of roof planes. Finally, the walls (in yellow) are determined by linking every edge of the roof shape to its corresponding edge from the footprint. The Figure 2.3 breaks down the successive steps illustrated, by paralleling the sections and their input data.

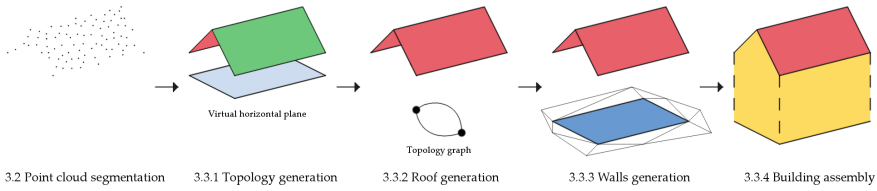

Figure 2.3: Succession of the parts generation.

In comparison with other hybrid approaches (Wichmann 2018), the planes are formed via a data-driven method, but the buildings are constructed based on a connectivity graph between these planes. However, some differences should be noted: (a) the buildings are generated following the CityGML format, but the refined CityGML levels-of-detail were not taken into account in the reconstruction process (Biljecki, Ledoux, and Stoter 2016); (b) reconstruction steps do not follow the same phases: the primitive components (small volumes) are generated based on the roof topology graph, then connected in order to generate the building model. In terms of the modelling of primitives, model-driven methods fit models to point clouds, in order to reduce metrics such as Root Mean Square Error (RMSE), Hausdorff distance (deformed shapes), tuning function distance (entire shape similarity), angle-based index, or a composition (Dorninger and Pfeifer 2008; H. Huang and Mayer 2017; Jung, Jwa, et al. 2017; Kada and McKinley 2009). A particular method implements the automatic

production of the CityGML model (Henn et al. 2013). All the surfaces are semantically consistent and independent of the process (see Figure 2.4).

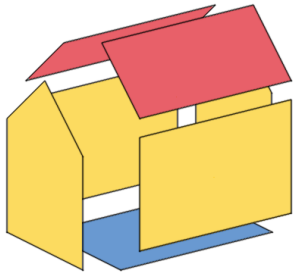

Figure 2.4: Exploded reconstruction geometry following the modules.

Note that, in CityJSON (because it is semi-structured), users are free to add to and modify the model, in order to increase its usability in dedicated applications. Further developments may pertain to the generation of other elements of the urban built environment (roads, bridges, tunnels, etc.) and the addition of specialized metadata.

Topology generation

From the formed subsets of points, and this, the topology of the roof shape is constructed for each building. The roof shape is translated based on connections between the combinations of subsets. Some generalization - and therefore a loss of accuracy to the real world - must be accepted. Again, the focus is on compliance with CityGML specifications (Biljecki, Ledoux, Du, et al. 2016). Therefore, some thresholds need to be established, as they are provided in the standard (e.g., minimum slope of 5°, in order to differentiate planes, and a minimum area of six square meters for *Buildings*).

The different subsets of points are projected onto a virtual XY horizontal plane. The minimum oriented bounding box is computed based on the coordinates of the points that comprise the subset. This polygon gives us the oriented sub-footprint determined by the plane (i.e., the oriented envelope that forms the minimum extent of a two-dimensional set of points). The connections between the different sub-footprints are considered between pairs: if the intersection line occurs within the footprint polygon, and if the intersection is consistent with the connectivity graph, the new connection is added to the topology graph.

The nature of the relationship will influence the connectivity and its physical representation (valleys, hips, and ridges). The plane relationship (O, S and N) can be classified into three constrained families and one default one (Verma et al. 2006):

- O^+ planes have normal vectors that, when projected, are orthogonal and point away from each other.
- O^- planes have normal vectors that, when projected, are orthogonal and point towards each other.
- S^+ planes have normal vectors that, when projected, are parallel and point away from each other.
- N have no constraint.

Note that the notion of the connectivity graph is part of hybrid-driven modelling (Wang et al. 2018): the Roof Attribute Graph (RAG) (Hu et al. 2018), the Region Adjacency Graph (Milde et al. 2008), or the Roof Topology Graph (Xiong et al. 2015). The main difference with these graphs lies in the handling of parallelism: The S^+ class, which is not present in other methods, provides additional information. To determine orthogonality and parallelism, the normal vectors are projected onto the XY plane. Based on a threshold of 5°, the angle between them determines the relationships (Biljecki, Ledoux, Du, et al. 2016). We assume that the method provides a good balance between the flexibility of the reconstruction methods and the quality of the reconstructed building models. The different families are illustrated in Figure 2.5.

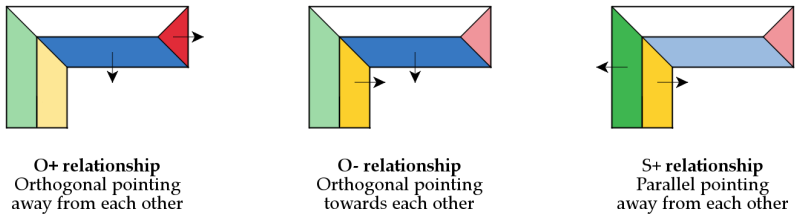

Figure 2.5: Relationships between slopes of adjacent planar roof segments, based on Verma et al. 2006.

Since the N family, planes that do not have constraint, does not provide any information, it is not stored. The assumption is made based on the absence

of information. From here, the relationship can be used to determine what this relationship consists of, materially speaking. For instance, from a S^+ relationship, one can state that the roof skeleton comprises at least one line segment and two points. Conversely, O^+ relationships provide a point for the roof skeleton. This information will be further used in the roof reconstruction process. Note that an empty graph will lead to a composition of flat or lean-to roofs. Breaking down the graph can also be easily used to simplify shapes or extract a subpart of a building. The connectivity information is computed on a couple of planes. As a result, this information can easily be stored in two matrices:

- An adjacency matrix *(i, j)*, which contains the ID of j planes connected to the i plane.
- A relationship matrix, which contains the nature of the connectivity between the i and j planes.

From the connectivity graph, if two planes are connected, the footprint is, at least, the union of both. The process loops iteratively above the upper diagonal of the adjacency matrix and constructs the footprint. Once the graph has been travelled, the polygon is generalized by computing the concave hull of the various elements to get a coherent polygon. The concave hull is significant, given that L, U, etc. shaped buildings should not be closed. Thresholds that comply with CityGML specifications are applied, in order to ensure the consistency of the footprint and clean potential errors (Fan and Meng 2012).

The result provides a flying footprint (i.e., the projected envelope of the building, without elevation). This envelope is part of the 3D bounding box of the building. The lowest Z limit can be determined, thanks to the Digital Elevation Model (DEM) generated from the airborne point cloud. It is provided with the point cloud and the results of the triangulation of the "ground" points from the LAS classification. This elevation model has an altimeter accuracy of 0.12 m and a spatial resolution of 1 meter. The last coordinate of the box, i.e., its elevation, is calculated as part of the roof generation.

Roof generation

Unless planes are perfectly parallel, two planes always intersect somewhere. This connection needs to be within the footprint, in order to be part of the skeleton. Otherwise, it could cause inconsistencies: irrelevant rupture lines will degrade the representation of the roof shape. Afterwards, all combinations of lines are computed travelling along the topology graph. For S+ relationships

(and only S+ relationships), the intersection points for these lines with the footprint are also determined. For instance, this is the case for the ridge of gable roofs, as shown by the yellow plane in Figure 2.6. According to the CityGML specifications (Biljecki, Ledoux, Du, et al. 2016), a threshold of two meters snaps the rupture points to those that have previously extruded from the footprint. This perfect snapping tolerance increases confidence in the model topology. Hence, it prevents holes (see the introduction for an explanation). The rupture points form the outline of the roof. The lines and rupture points are stored in a rupture matrix, similar to the adjacency and relationship matrices.

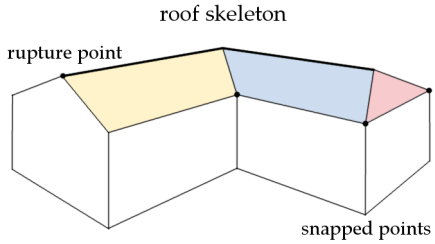

Figure 2.6: Representation of the roof skeleton in bold black. The rupture points form the ridge and the other points are snapped to each other at the corners.

Finally, the points of intersection between the rupture lines are determined and added to the rupture points. Their heights are interpolated afterwards. Note that, in cases where there is no intersection between roof planes in the footprint polygon, this means that the roof is flat or lean-to shaped. In this case, the building model cannot be refined further. Only the LoD 1.x will be generated for these buildings. Even if this kind of shape has already been detected, a second filter is applied in order to ensure they are detected. Roof planes are built while travelling segment-by-segment through the footprint. For each point, the orthogonal distances to the roof planes are computed through the normal vectors. Since a point from the outline of a roof is part of two roof planes, the two lesser distances determine the plane to which they belong. After this, the non-intersecting polygon is computed for each roof plane, by considering these points and the results stored in the rupture matrix. This polygon results from sorting two subsets of points and drawing the line that links the left-most and right-most point. The non-intersecting polygon is the concatenation of the points that are higher than the line and those that are lower, sorted from left to right and vice versa (see Figure 2.7 for an illustration). It should be mentioned that the concave hull of a set of points tends to determine the

smallest polygon that links all the points. This smallest polygon is not always the correct representation of the wall in cases with very sharp angles.

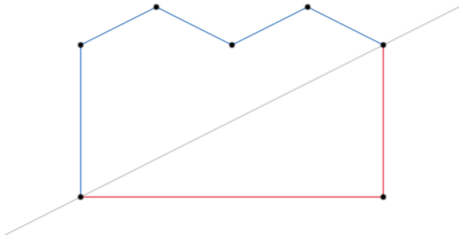

Figure 2.7: Non-intersecting polygon construction.

To conclude with roof planes, the height of the points is calculated according to the plane to which they belong. Since each vertex belongs to several planes, an arithmetic formula gives its height. Note that this operation corrupts the planarity of the different planes. An issue with non-planarity could lead to problems when calculating surface areas, for instance. However, this operation distributes the errors evenly. The denser and less noisy the point cloud, the more accurate the reconstruction. Finally, the highest point of the LoD 2.x building model gives the height of its 3D bounding box.

Wall generation

This module is simpler than the two previous modules, as it only binds results to results. As well as roof planes, walls are generated while travelling segment-by-segment through the footprint. The intersections of segments with projected roof planes are computed. The vertices of the intersection, and those that are extracted from the footprint segment, give the points that delimit the wall. Once more, the non-intersecting polygon is determined and provides the sorted vertices that draw the wall parts.

A general remark concerns all three modules. When rendering the rigid body via back-face culling, a single rule is applied: the normal vector should point towards the outside of the volume. In this way, the exterior face is the one pointing outside of the box. This translates into the order of vertices for each surface. This needs to be counter-clockwise and ordered from an exterior viewpoint. All faces are controlled under these conditions. Otherwise, the volumes are closed, but their display, among other things, may appear problematic. This depends on the viewer and the application (see Figure 2.8 for an illustration).

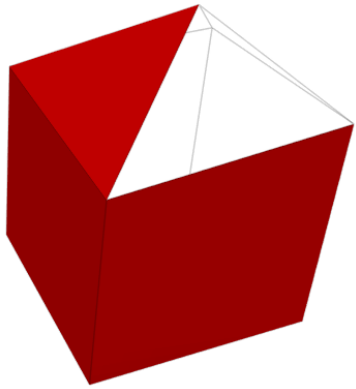

Figure 2.8: Example of a geometry in which planes are not correctly oriented. One face is inverted and therefore is represented as transparent to allow viewing the outside from the inside (back-face culling).

Building assembly

A building comprises a *Solid* bounding a closed volume. Again, the nature of the surfaces is not inferred from their slope, orientation, or position, but by the process: the footprint is always the first element generated, then roof planes, and walls close the 3D geometry (Refer to Figure 2.9 for the construction order: red, then blue, then yellow). The three possibilities (*GroundSurface*, *WallSurface*, or *RoofSurface*) are listed and mapped into an array, which corresponds to the order of the different planes. This information is stored within the JSON objects as the *semantics* JSON key within each building object. The polygons that define the different planes are stored as sorted arrays of the vertices' identifiers. They constitute the *boundaries* (the subobject giving the frontiers of the *Solid*) of the JSON object.

CityJSON allows objects to be represented in concurrent levels-of-detail. Since not all applications need a detailed level of design, the three levels-of-detail are produced. They represent stages from the footprint to the roof-shaped geometries (see Figure 2.9 for an illustration). The sub-levels (LoD 0.x and LoD 1.x) do not require additional processing: the footprint and the bounding box are already determined beforehand. Those are stored as the 0.1 and 1.1 level of the building (Biljecki, Ledoux, Stoter, and J. Zhao 2014), respectively. Note that there are multiple ways to determine the elevation of the extruded solid: minimum, maximum, mean, and median. In this case, however, using the maximum elevation seems to be the most conservative approach for defining a

volume.

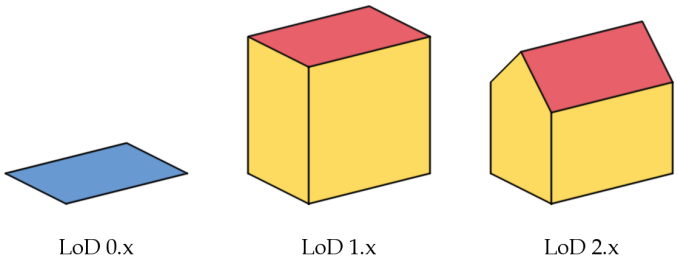

Figure 2.9: Schema representation of the different levels-of-detail for the same building.

2.3.4 CityJSON model building

The CityJSON model is created on an iterative basis through this process. The specifications require some JSON keys to be present in every file: *type*, *version*, *CityObjects*, and *vertices*. While *type* and *version* are easily created, *CityObjects* and vertices require an explanation. CityObjects represents the concatenation of all *Building* objects created in the previous section, with their own boundaries, semantics, etc. Next to this, the coordinates for the *vertices* are stored in bulk at the end of the *CityModel* file. They are stored in the order of their identifiers (i.e., their successive creation). This method makes it possible to reduce the size of the model and avoid redundant information, even if this redundancy is still allowed in the CityJSON specifications. Nonetheless, it is an additional, necessary condition for building a compact city model. Moreover, the optional *transform* JSON key allows the coordinates of the *vertices* to be decompressed, in order to improve the compactness of the model, as is already done in TopoJSON.

Another important element pertains to metadata. Due to their importance and the impact of their support, this element will be extensively discussed in the next section.

An example of a CityJSON-generated model is shown in Figure 2.10. The point cloud and the reconstruction are displayed one after another in the CloudCompare viewer, in the open-source viewer developed at the 3D GeoInformation group from TUDelft (https://github.com/tudelft3d/CityJSON-viewer/) and an open-source extension of Cesium (https://github.com/limyyj/cesium-cityjson).

A quick glance shows that pyramidal and more complex shapes are represented within the data, even if gable roof shapes represent the majority.

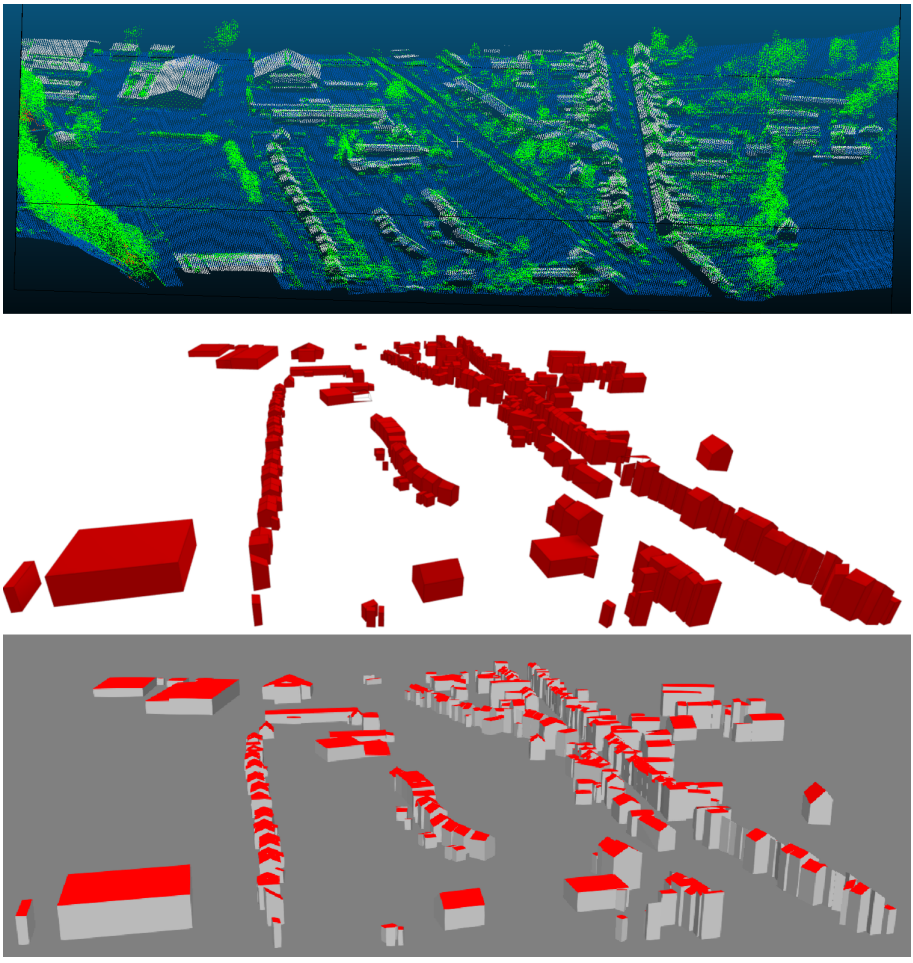

Figure 2.10: Results from the city of Theux in CloudCompare and CityJSON viewers.

2.4 Discussion

This section discusses the advantages of directly generating CityJSON models and assesses the quality of the results using several normalized and formal tools. While the first part of this section develops the research contribution of this chapter, the second section makes it possible to rule on the validity of the geometrical reconstruction in comparison with the state-of-the-art approach. Firstly, compliance with CityGML/CityJSON specifications is evaluated; the geometric validity of every building is also studied.

2.4.1 CityJSON improvements

On a global level, the construction of a model could be done in parallel for both formats. A geometry comprising edges and vertices can be formatted in either CityGML or CityJSON without any loss or inaccuracy. Consequently, the translation between them both is not a problem, but it is not as easy as a one-to-one mapping. Even if they share the same conceptual schema, several notable differences exist in their descriptive components. For instance, CityJSON makes it possible to manage two additional elements in particular: (a) the ability to handle metadata natively (Labetski et al. 2018) and (b) the refined levels-of-detail (Biljecki, Ledoux, and Stoter 2016). By providing a method that breaks down geometric reconstruction, and working on improvements to every constituent part, we manage both (a) and (b) points during the process.

CityJSON offers the ability to handle metadata natively. Conversely, CityGML requires the use of an extension for managing metadata (Labetski et al. 2018). For that reason, translation from CityGML to CityJSON, and vice versa, cannot occur without a loss of information. Taking the example of the command-line translator from CityGML to CityJSON, the information is simply neglected if it is not part of the destination format. The proposal of a generation method that directly offers a CityJSON model responds to this problem. It makes it possible to avoid such a loss.

On one hand, the refined level-of-detail definitions, and their related metrics, are managed. Throughout the generation process, thresholds are provided and we guarantee compliance with specifications. In this case, we managed to create 0.1, 1.1, and 2.1 levels-of-detail in parallel (Biljecki, Ledoux, and Stoter 2016). A related benefit is that we are able to store all of the levels at the same time. Moreover, these levels are interdependent, since the footprint - the actual 0.1 LoD - is the foundation of the 1.1 and 2.1 LoDs. These improved definitions provide important information about the form factor of the building, which is not allowed in common LoDs. The refined LoDs do not only show

individual and large building parts, but also small building parts, recesses, and extensions. Figure 2.11 depicts a comparison between common LoDs, which provide minimal specs, and refined LoDs, which provide improved conception of details. Therefore, it allows applications to make the choice of the more relevant level for their scope of interest. From a user perspective, i.e., fluid-flow simulations, this level of detail in building has an important impact (Kumar, Labetski, et al. 2019).

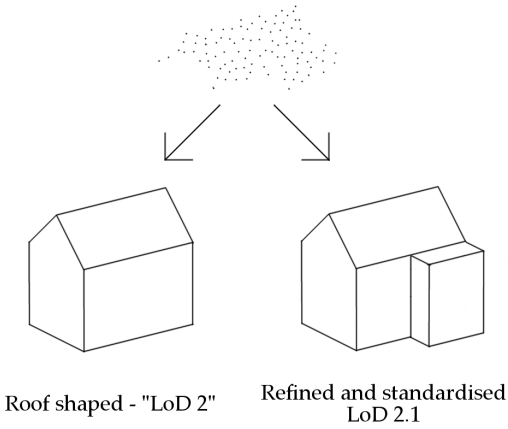

Figure 2.11: From point clouds to building models. Depending on the LoD, different outputs can be generated from the same data source.

In terms of contextual information, it is possible to compute and handle metadata during the process. Among other things, CityJSON makes it possible to manage the geographical extent of the city. In addition to the geometry of a building, its geographical extent is computed simultaneously (i.e., the minimum oriented and bounding cuboid). The choice has been made to consider the maximum height of the building, in order to be on the safe side and not underestimate the influence of elements on their built environment. Finally, the extent of buildings can be aggregated to determine the geographic extent of the whole city model. This extent is then stored in the model metadata, which is not available in a built-in manner in CityGML (see Figure 2.12). Moreover, the lack of CityGML ISO19115 support is detrimental when it comes to exchanging information, comparing concurrent models for the same element, or keeping a record of the model versioning. As a reminder, the starting hypothesis is to provide a compact model that allows easy online exchanges. Thanks to CityJSON and this method, ambiguity is no longer possible.

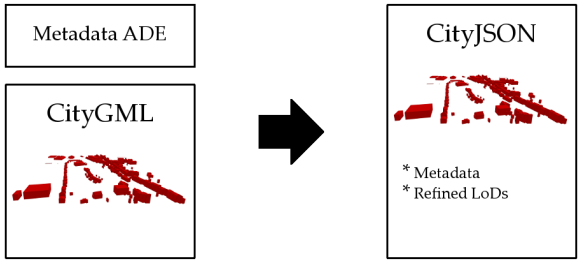

Figure 2.12: Translation from CityGML to CityJSON.

2.4.2 Format compliance

CityJSON specifications provide a subset of the CityGML data model (Ledoux, K. A. Ohori, et al. 2019). JSON, as opposed to XML, has the advantage of not using end tags, which reduce format redundancies; it does not require the repetition of information for each element when handling arrays; etc. However, both share many similarities, which make them primary formats for exchanging data online: (i) they are both self-describing, (ii) hierarchical, and (iii) fetched within HTTP requests. The JSON hierarchy is structured as nested and embedded key-value pairs (i.e., a map that allows horizontal and cross-levels links), while CityGML is structured as a tree. This is the main reason why trees can be tedious and time-consuming to parse. In short, XML is better for storing information - thanks to namespaces - and JSON is better for delivering data, thanks to its compactness.

From a format point of view, it is therefore important for the generated model to be compliant with the official CityJSON schemas (compliance is the state of being in accordance with established guidelines or specifications). To ensure this compliance, a Command-Line Interface (CLI) provided by TUDelft makes it possible to validate and transform CityJSON files (https://github.com/cityjson/cjio). No error has been detected through our various tests with the CLI (vertex indices coherent, specifics for CityGroups, semantic arrays coherent with geometry, root properties, empty geometries, duplicate vertices, orphan vertices, CityGML attributes). This is easily explained as follows: every individual inconsistency is handled upstream within our software.

2.4.3 Quality control

Every individual vertex could lead to local singularities in the topological consistency during the reconstruction process. Two sub-processes prevent this from happening and ensure that buildings are closed: vertices generalization and perfect snapping tolerance (Biljecki, Ledoux, Du, et al. 2016). Both methods are informed by the geometric specifications of the refined LoDs in CityGML (Biljecki, Ledoux, and Stoter 2016), which state that building parts that are smaller than two square meters wide should be generalized and that two vertices that are less than two meters apart are considered to be the same vertex on the XY plane. Note that the generalization happens during the footprint establishment, when vertices do not yet exist, but the snapping happens later, during the creation of the wall, roof, and footprint, when vertices do exist.

Spatio-semantic evaluation

Semantic validity is crucial: little room is left for semantic uncertainty. Again, the nature of the surfaces is not inferred from the normal orientation or relative positions, but during the process: the footprint is always the first element generated; roof planes are generated from their connectivity graph and, finally, wall planes are created. Spatio-semantic validity is therefore easy to obtain.

Given that CityGML has become the *de facto* standard for spatio-semantic city modeling, the Interoperability Experiments Joint Activity between OGC, SIG3D, and EuroSDR defined some requirements in order to normalize its use. The OGC IEJA group conducts research and provides tools to carry out quality assurance on the data, including data used in this research. Their discussions about issues relating to CityGML data quality has led to usage guidelines (OGC 2016) (i) a definition of data quality, (ii) data quality requirements and their specification, (iii) a quality checking process for CityGML data, and (iv) a description of validation results.

Geometric evaluation

The main idea is to simplify validation by cascading it: we must validate the previous steps before proceeding to the next step. Therefore, if a critical geometric inconsistency has been detected, no semantic validation should be carried out (Wagner et al. 2013). Recently, new tools are proposing the management of more discrete levels-of-detail: previous tools were limited at levels close to LoD1.x (flat roofs and vertical walls). A standard error taxonomy is necessary to help structure the evaluation. From the intrinsic features of

the models, errors can be detected using a supervised classifier (e.g., random forest) (Ennafii, Bris, et al. 2018; Ennafii, Le Bris, et al. 2019). However, this does not support CityJSON. Our methodology was validated with the versatile *val3dity* tool (Ledoux 2018). Complex geometries are systematically broken down into their constituent parts: the integrity of buildings is processed, then every 3D primitive is validated. Here, it is worth mentioning that overlapping - as it is implemented in val3dity - considers relations between *BuildingParts* and not between *Buildings* and each other. This is an important point with regard to computation time, as overlapping greatly slows down the process: the Minkowski sum is used to assess overlaps. It has $O(n^3m^3)$ runtime complexity, where n is the number of objects and m the number of constituent parts therein (Hachenberger et al. 2007). As is the case for CityGML, only planar and linear primitives are allowed: cylinders, spheres, or other curved, parametrically modelled primitives are not supported. The validation parameters in the *val3dity* tool are as follows (Ledoux 2018):

- Snap tolerance: 0.001 m – if two points are closer than this value, then they are assumed to be the same.
- Planarity tolerance: 0.05 m – the maximum distance between a point and a fitted plane.
- Overlap tolerance: 0.01 m – the tolerance used to validate adjacency between different solids.

Figure 2.13 provides a graphical overview of the results for the city of Theux in Belgium. Note that the definition of *Building* itself might differ between standards and applications, especially for LiDAR point clouds, since they are not initially intended to suit CityGML definitions. Hence, some elements do not belong to the CityGML definition. For the remainder, the footprint of a *Building* element needs to be greater than six square meters. The red buildings in Figure 2.13 represents these elements scaled onto one hundred buildings. This proportion represents forty-five buildings from the complete set (464 elements). On the other hand, 65 buildings are classified as flat or lean-to roofs (green buildings on the right of Figure 2.13). They are thus not included in the LoD2.x reconstruction benchmark. Finally, the buildings that correspond to complex roof shapes are validated. In the end, thirty-one buildings are not validated from the 355 remaining buildings (yellow buildings in Figure 2.13). This corresponds to 91.26% validity for the LoD2.x objects, i.e. the vast majority of the data set is valid. The *Buildings* that could not be reconstructed in LoD2.x are instead generated as LoD0.x and LoD1.x (footprint and extruded volumes).

The results provided by our methodology have been compared to open data sets (see Table 2.1) (ibid.). Overall, the proposed methodology provides a ratio

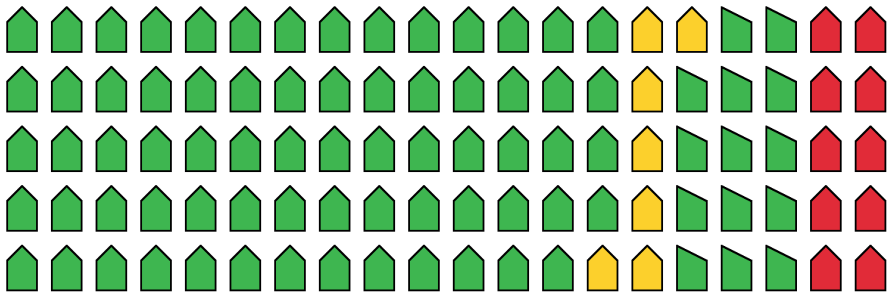

Figure 2.13: Proportions of buildings reconstruction from Theux, Belgium.

of valid/invalid geometries of over 90%. Note that the other benchmarked data concern CityGML files. No other information about the source and/or creation of the data was given. The increasing validity from *Buildings* to *Primitives* is explained by the fact that lower LoDs are easier to generate.

Table 2.1: Summarized comparison of results.

City (Method)	Size	Buildings	Valid	Primitives	Valid
Berlin	933MB	22.771	74%	89.736	90%
Den Haag	22MB	844	61%	1.990	85%
Montréal	125MB	581	76%	1.744	88%
NRW	16MB	797	83%	928	77%
Theux (UR)	689KB	420	92%	1.198	97%
Theux (RG)	656KB	400	93%	1053	96%

The difference in building numbers between the unsupervised RANSAC (UR) and the region-growing (RG) stems from preprocessing (removal of statistical outliers, clustering, etc.). Indeed, the point cluster does not always satisfy the requirements to be considered as *Buildings*, as per CityGML specifications. Here, the assumption that segmentation is effective shows its limitations: if planes are not detected, or they overlap too much, the connectivity graph will not be correct. The proposed methodology partially relies on the restrictive hypothesis that the segmentation properly detects the planes. When analyzing the results, the main source of errors currently stems from the segmentation methods and the low point density. The *val3dity* tool provides a taxonomy of errors that could be encountered during a geometric evaluation (Ledoux 2018). In the successive development phases, only one recurrent subset of these errors was detected - non-planar polygon distance - since the algorithm already prevents many of them.

Nevertheless, the distribution of these errors only deviates slightly from the theoretical threshold of 5 cm. Only a few of them stand out. It is also essential to mention that the plane-interpolation algorithms are different between *val3dity* (least mean squares) and the methodology (RANSAC) (L. Li et al. 2017). In order to assess the quality of the plane interpolation, the Root Mean Square Error (RMSE) of each roof plane for the Z-axis is computed (see Figure 2.14 for error classes). As can be seen, the distribution of errors tends to be close to 3 cm (median: 0.028 m, 95th percentile: 0.039 m). Only eight of the 644 planes have an RMSE greater than 5 cm (these are actually outliers greater than 2 meters).

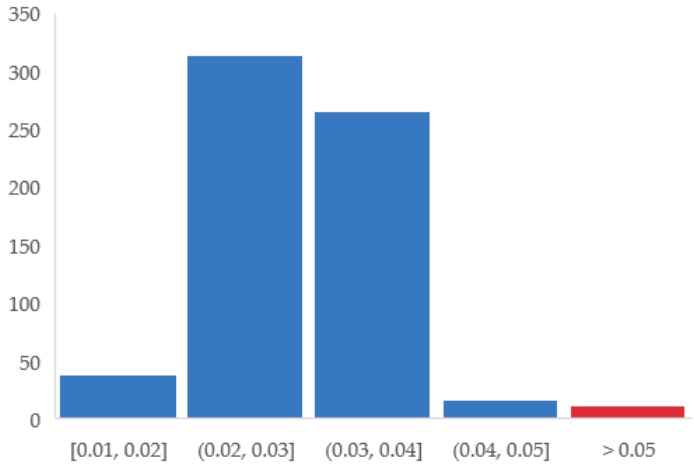

Figure 2.14: Number of planes per RMSE class.

However, one negative point should be noted: even if the number of buildings affects the processing time on a linear basis, RANSAC is still the primary source of time complexity. Taking this consideration into account, two features influence the processing time: the ratio of points within the subset and the total number of points (Raguram et al. 2013). Therefore, filtering the point cloud is mandatory. We tested filtering versus not filtering the data provided. The use of the Statistical Outlier Removal (SOR) filter corresponds to an improvement of 30% to the computation time. Another improvement of our method is the use of a k-d tree index, rather than an octree, for identifying point neighbors. This was done in order to reduce the number of spurious planes during segmentation (L. Li et al. 2017).

Buildings are not always made up of a single volume: some have annexes and outbuildings. Considering these additional parts in the reconstruction

could greatly enhance the applicability of the models. In particular, this could improve land administration and cadastre applications. In light of this, a further interesting development could use a *CompositeSolid* construction (i.e. the aggregation of one or more *Solids* - a non-empty set of *Solids*). Special attention will need to be paid to the topology of *Solids*. Independent management of stories could also be considered in line with this new management of parts.

Currently, if an LoD2.x geometry is deemed inconsistent, the generation is limited to lower levels. It could be interesting to fill the potentially detected holes, rather than treating the geometry as corrupted. For example, one can use a top-down, shrink-wrapping process to re-mesh the polygonal surfaces (Z. Zhao et al. 2013), or reconstruct these missing planes directly within the connectivity graph (Hu et al. 2018). As this makes it possible to repair common errors and fill holes (windows and doors), the integrity of LoD 2.x objects would be respected, although this envelope might smooth geometric details.

2.5 Conclusion

This chapter aims to respond to the lack of availability and versatility of 3D city models, by providing a method for generating city buildings. Besides this, thanks to the use of CityJSON and the break-down of reconstruction methods, the integration of metadata and refined levels-of-detail are now supported within the generation process. The presented methodology does not rely on commercial or proprietary solutions. We believe that the opening up of our straight-generation method will help to disseminate the use of CityJSON. It will also facilitate further related works in the context of the urban built environment. The evaluation and comparison, which is based on the validity rate of the geometries generated (format, semantic, and topologic validity), showed that it is possible to automatically reconstruct LoD2.x buildings based on LiDAR data and CityGML/CityJSON specifications.

In terms of format and semantic validity, no error has been encountered. The provided city models are consistent, in every respect, with the standard specifications. Two methods were tested for plane detection: an unsupervised RANSAC and an unsupervised, normal-vector-based region-growing algorithm. A significant ancillary result of this work is the use of indoor point-cloud segmentation in an outdoor context. As it pertains to geometries, the quality is assessed on a normalized basis, thanks to formal tools such as *CJIO* and *val3dity*. The ratio of valid/invalid buildings varies between 92% and 97%, depending on the segmentation method. Several improvements are considered, including the addition of elements of urban landscapes (roads, etc.).

Subsequent works

The opening and availability of the generation process have been used in support of other researches some of which we were invested in. The generation process was built as to provide its constituting elements as independent to one another. Those have thus have been used to make the most of the benefit of each of them (model writing, roofs shapes statistics, etc.).

Green roofs, "contained" green space on top of a human-made structure, are often discussed in the context of smart and sustainable cities as they present a multi-functional and solution-oriented approach (Joshi et al. 2020). They are an effective heating moderation tool. They reduce temperature of roof surfaces and their surrounding air due to the vegetation cover but also provide shade. Such kind of infrastructure requires a specific threshold of plane horizontality to be installed on the load-bearings structures. This is mainly due to the water flows.

Computation of potential power generated and the detection of empty roofs on which Urban Green Infrastructure (UGI) could be installed are the results of the paper. The subsequent result, which has been made possible because of our work, is the detection of what are considered to be flat roofs. Computed metrics were the number, the position, the general orientation and the specific areas of classified roof shapes. Coupled with cadastral and material information, it allowed determining the potential of roofs for greening at the whole city scale (see Figure 2.15). Outputs of the conference paper have been used to illustrate the integration capabilities of the web architecture proposed in the third chapter of this document.

The study of roof shapes thanks to the added details brought by point clouds also gave information on the buildings potential covering. Indeed, the geometry of the roof is one of the parameters that are taken into consideration when determining the roof covering. The modelled roof-shaped buildings were used in order to classify elements and determine their covering in a medium-sized European city (Wyard et al. 2022). Besides the benefit brought by the number, the area and orientation of roofs for each building, modelled elements were used as a source of validation data in support of the machine learning process.

It is here worth mentioning that this generation process, even if it was a state-of-the-art process, is no more relevant in 2023. An improved modelling method has been published and made available open source. Geoflow3D is a tool for reconstructing 3D building models from point clouds with high-detail (Peters et al. 2022). The output results of Geoflow are unbeatable in terms of quality. Therefore, we consider Geoflow to be the best source of information and have replaced its use in every stage of our projects. Only negative point in this shift:

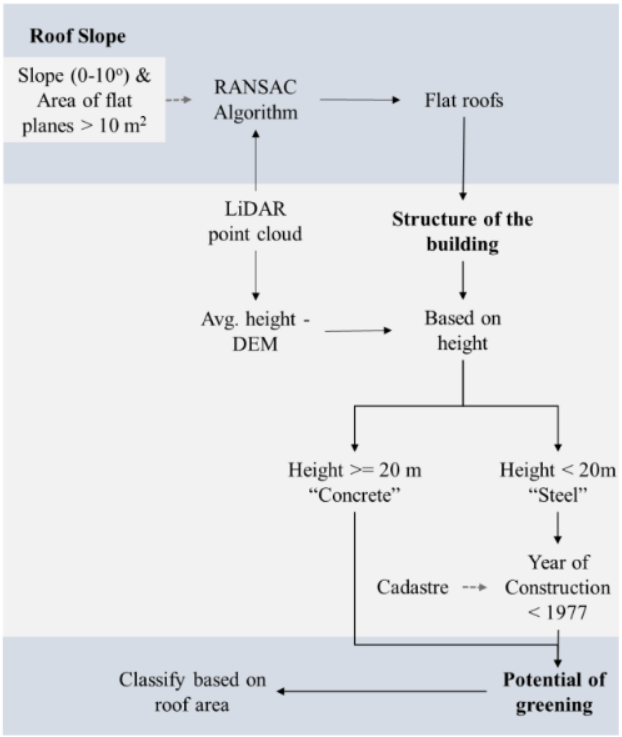

Figure 2.15: Methodology for identifying the potential of green roofs (Joshi et al. 2020)

actual Belgian data sources do not fulfil the basic requirements for generating consistent LoD 2.2 *Buildings* due to its sparse point density.

Chapter 3

Database schema

Preface

In the life-cycle of the modelling process that we follow out this document, once that the first step on model generation has been tackled, we got our hands on a lightweight and semantic 3D city model that follows the JSON encoding portage of the CityGML data model. Respecting the same conceptual data definitions, the addition of a simplistic roads network, a digital elevation model and some sparse vegetation was not a problem. This could be achieved by using an Extract Transform Load process (ETL) from the same dataset (i.e. the sparse point cloud) and some additional layers provided by the regional cartographic agency (i.e. vegetation location, elevation profiles, etc.). We had a starting and contextualised digital replica of the urban built environment constituted of various object classes.

The first question has been answered: a generation solution has been proposed to obtain simple models in CityJSON. The next logical step is the exploitation of these generated models. Without going towards complex computations and use cases, many possibilities of using city models could be handy. Still, it is important to propose a generic solution that covers basics but allows improvements and extensions following the trend. For the reminder, the objective of the project is to study the decomposition of a GIS web-architecture in its constituting tiers in order to tackle new capabilities: flexibility, openness, availability, etc. While keeping in mind the current needs and the state of our knowledge, the point of view must be ahead of time. Then comes the time to take care of the storage of the CityJSON models, the second tier.

Once more, this question had too few answers at the time. A single database distribution proposed a solution for the storage of these standardized models: 3DCityDB (Yao et al. 2018). The storage of CityJSON models in 3DCityDB, the relational model of CityGML imposes to translate models in the so-called CityGML XML encoding before storage. During the translation, information, among others, such as metadata and refined levels of detail (whose advantages have been discussed in the previous chapter) are lost. Moreover, besides its complexity and the great difficulty that people have to understand its UML scheme, storing city models in a relational database do not allow enjoying the fundamentals of CityJSON: its lightweight and developer-friendly format.

Nevertheless, the worst reason of all is the loss of flexibility that CityJSON initially provided: its flattened schema. In CityJSON, city objects are stored as a list of elements in which cross-references are allowed (children/parents relations, groups, etc.). In CityGML, the city objects respect a tree-hierarchy and all the subsequent constraints. Associations and tables of relational databases suit the translation of the tree-hierarchy but does not offer a convenient solution for the

CityJSON models because of the flattening of the schema (see Figure 3.1).

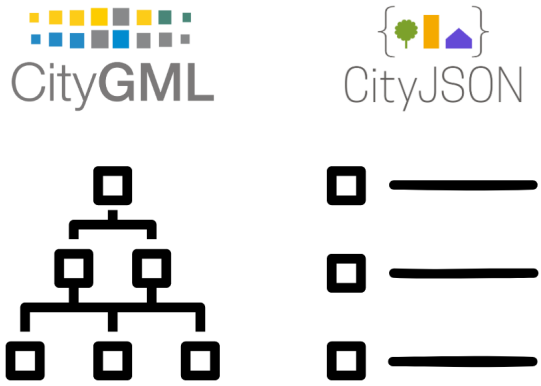

Figure 3.1: CityJSON flattens the CityGML data model compared to its XML-encoding.

Undoubtedly, relational databases have deep roots in many places. However, among other projects related to the Semantic Web and the study of the NoSQL storage, we had the opportunity to handle information using different storage solutions other than the relational mode. NoSQL databases, although this term is used to describe a large number of solutions, offer in particular a storage method called "documents". One of these distributions in particular caught our attention: MongoDB. Documents can be defined as nested or embedded objects in which other documents can be referenced without having to respect a vertical structure. An important additional point is worth specifying looking at the storage design: MongoDB encodes elements in binary JSON format (BSON). BSON is a binary-encoded serialization of JSON documents that improves scan-speed and storage space. One of the ideas developed in this chapter is the isomorphism: avoiding format translation, and thus keeping everything as we use (i.e., in JSON), allows consistency guarantee as elements should not be manipulated. This point concerns the data format in addition to their design.

This chapter studies the schema definition for the storage of CityJSON models in a document-oriented database. Besides, the XML encoding, the JSON encoding and its relational model in 3DCityDB, it is the fourth usage of the CityGML data model from a technical perspective. All developments are encapsulated in a complete web architecture that encompasses its flexibility and provided insights on future possibilities. Performance tests have been conducted in order to better highlighting these new storage capabilities compared to relational databases.

Based on article (Nys and R. Billen 2021)

From consistency to flexibility: a simplified database schema for the management of CityJSON 3D city models

Nys, G.-A., & Billen, R. (2021). *Transactions in GIS*, 25, 3048–3066.

Abstract: The use of 3D city models is now common practice; many large cities have their own digital model. Resilient and sustainable management of these models is necessary in many cases, where an application could evolve over its life cycle. The complexity of generic modelling standardization is often a limitation for a light and user-friendly usage and further developments. This paper aims to propose an alternative providing a simplified database schema implemented in a document-oriented store. Thanks to the use of the NoSQL store, the focus is on flexibility of the data schemas. In order to aim attention at the compactness in web development, CityJSON has been chosen for the encoding of the 3D city models. Finally, a full-stack application (persistent storage, consistent edition and visualization of 3D city models) has been developed to handle the simplified schema and illustrates its capabilities in two practical use cases.

Keywords: CityJSON, NoSQL, 3D City Models, Data Schema, Data Architecture.

3.1 Introduction

Nowadays, many large cities have usage of their own 3D digital model (Biljecki, Stoter, et al. 2015). These 3D city models are the integrating base for urban management tools such as fluid flows simulations, cadastral operations, urbanism, etc. In the context of urban built environment, the use of CityGML as the data model and encoding standard is now a common practice (Gröger and Plümer 2012). CityGML provides a data exchange format for the structuring of urban and landscape objects. It stores objects in multi levels-of-detail and structures their attributes, their relationships and their features on a normalized basis. Its support of an increasing number of extensions allows dealing with more and more issues: energy, noise, land administration, etc. (Biljecki, Kumar, et al. 2018; Floros and Dimopoulou 2016). From a conceptual viewpoint, these Application Domain Extensions (ADE) extend the supported features and properties of the CityGML core module. These added elements are necessary to perform computations or to store their results in simulations and analysis.

Recently, 3DCityDB, an open-source 3D geodatabase solution, has been proposed to handle city models (Yao et al. 2018). The tool proposes a system for the management, analysis, and visualization of large 3D city models according to the CityGML standard. It relies on a relational database and provides well-known tools such as Web Feature Service (WFS), the support of 3D scenes (Keyhole Markup Language (KML), COLLADA, etc.), the streaming of these formats thanks to the WFS capabilities, etc. The major drawback highlighted by the author states that the lack of flexibility of the 3DCityDB relational solution could limit its usability; even if ADEs are supported, maintaining them natively could be troublesome. Besides, the intrinsic management of a oversimplified object-oriented solution might impose to make a large number of recursive joins to represent the aggregation and inheritance hierarchies of the object-oriented data model. Moreover, to support new features, it might be necessary to add tables, which always results in an additional demand for resources and complexity of use.

This chapter aims to provide an alternative to the relational database management of 3D city model and traditional tools (SQL, CityGML, etc.). It relies on a simplified data schema for the storage of city model in a document-oriented NoSQL store. A web three-tier architecture (client, server and database), in which JavaScript articulates all the operations, illustrates the use of the derived CityJSON schema, the JSON encoding of the CityGML data model (Ledoux, K. A. Ohori, et al. 2019).

NoSQL databases offer the possibility to improve the storage flexibility by reforming the tabular structure. Besides their reorganization of their intrinsic

structure, this stores family puts forward the plasticity of the schema model (Weglarz 2004). On the other hand, CityJSON proposes a lightweight and compact alternative to the CityGML XML-encoding. Following the same conceptual model as the XML-encoding, the JSON-encoding offers the possibility to ease development of web applications. The conceptual similarities between CityJSON and document-oriented management, which stores information as document in BSON-encoding, could provide an answer to the lack of flexibility.

This chapter is divided as follows: the Section 3.2 contextualizes this research in related works on Web Geographic Information Systems architecture (Web GIS) and the trend towards an increasing use of the web (Mobasheri et al. 2020). It highlights the major drawbacks of the current relational management and put it in parallel with the current state of alternate developments. Then, Section 3.3 describes the simplified data schema and its implementation in a document-oriented store. The illustrating application architecture is decomposed in its three constituting parts: client, server and database. Section 3.5 develops the new data management concerning the modifications provided by the NoSQL database storage and several improvements on other tiers. A response is proposed and documented in order to shed light on its new capabilities. From a network load viewpoint, performances tests compare architecture capabilities in order to ensure exchanges compactness. A benchmark with a relational solution is presented. Finally, two examples of use cases illustrate these capabilities in practical situations in the section 3.6. Before considering future works, we conclude on the principal benefits of the new generation application and its advances.

3.2 Related work

A geographic information system (GIS) gathers and manages geospatial data (Tomlinson 1968). In the urban built environment, besides the management of 3D models and geometries, the specific attributes and semantic information impose their own definition: Urban GIS (Blaschke et al. 2011). From a technical viewpoint, a web-based GIS application is divided into three interdependent constituting parts at least: (1) a client, which is a consumer of spatial information; (2) a server, which is a GIS processing system; and (3) a database, which is a storage solution that deals with spatial formats, spatial indexing and/or data processing functions. In short, a Web GIS is a type of distributed information system in which components manage spatial information on the web.

Nowadays, leveraging client capabilities and thus using its resources, the browser

is no longer simply a static window on a set of data: it can also perform a set of processes (Toschi et al. 2017). The browser-based applications should outstrip standalone software thanks to their multi-user characteristics and dynamic elements. It will result in cost savings from the server without negative impact on the user experience (Kulawiak et al. 2019). Indeed, the number of clients can also increase without limiting the server performances, as it is used as a simple gateway and no longer as a computation centre.

Due to their mature support of spatial functions, indexes and storage capabilities, relational databases often represent the core base of web applications (Mobasheri et al. 2020; Zlatanova and Stoter 2006). Besides the data-modelling functions, the transactional databases can handle data processing in an efficient way (Obe and Hsu 2015). Several integrated solutions have been proposed for the management of digital city models. The majority of these solutions are based on a relational database: (a) DB4GeO is a web service-based geo-database architecture for geo-objects (Breunig et al. 2016). It relies on an object-oriented database. Nevertheless, its development is no longer maintained. (b) 3DCityDB provides a spatial relational database schema for semantic 3D city models (Yao et al. 2018). It proposes an important number of key features and functionalities for CityGML models management (Pispidikis and Dimopoulou 2016). It is interesting to note that, among other functionalities, 3DCityDB allows the streaming of CityJSON features thanks to the OGC WFS 2.0. (c) A NoSQL solution relies on a document-oriented storage and provides a 3D web-rendering tool (Doboš and Steed 2012). However, these tools used in this architecture were not as efficient as nowadays: many current libraries were unavailable (HTML5, ThreeJS, etc.), the browsers capabilities were not as efficient as today; the focus was made on the dataset and did not consider the architecture as a whole; etc. Moreover, the solution developers criticized the lack of validation on elements import in the document-oriented solutions. (d) Another NoSQL-solution development states that the document-oriented stores lacks on consistency (Višnjevac et al. 2019). The problem here is that the database cannot itself provide a sufficient guarantee of consistency. (e) The storage and manipulation of heterogeneous data sources arises problems due to the differences in data structure: sensors data, 3D city models, BIM models, etc. have a different update rate, a different representation scale, etc. Even then, in GIS applications where sensors data, 3D city models and BIM models coexist, the relational databases are preferred (Aleksandrov et al. 2019).

It is here worth mentioning that the dichotomy in which relational databases do not support JSON insertion and document does is no longer true (Chasseur et al. 2013). Relational databases have been refactored to handle JSON (Z. H. Liu et al. 2014). However, it still imposes the use of an additional mapping layer and thus does not provide a solution to the lack of flexibility. For instance, it is

the case for 3DCityDB, which translates the CityJSON in CityGML encoding before storing it into the relational database thanks to the citygml4j software.

Developments on features visualisation have recently made progress on the client side (Lim et al. 2020). They provide a comparison on web-based viewers and their specific capabilities at the building scale. However, the conclusions still draw the disadvantages of ADE modelling and the complexity raised by relational database management. Working on the storage tier, a composition of SQL/NoSQL allows enjoying advantages of both solution (Holemans et al. 2018; F. Poux, R. Billen, et al. 2020). While the relational database is still mandatory for its data-processing capabilities, the document-oriented database is useful thanks to its storage flexibility. It can be done without replication or complex mapping between the two stores since the metadata and geo-registration are handled on server side. The geospatial capabilities of the document-oriented stores bring more and more solutions to spatial-related problematics (Costa Rainho and Bernardino 2018; Lopez et al. 2016; Zhang et al. 2014). However, it shows that even if performances are overall improved with document-oriented store, it is not yet always true (Makris et al. 2019). Sometimes, relational database ranks ahead of document-oriented stores (Bartoszewski et al. 2019), sometimes it is the inverse in terms of loading (Laksono 2018) or heterogeneous sources handling (Sveen 2019).

From a technical viewpoint and in a more precisely way, MongoDB, a cross-platform document-oriented database, has already been used in several "geo" architectures. Constituting part of what is called a MERN stack (MongoDB - Express - React - NodeJS), MongoDB is acknowledged for powerful way to store and retrieve data that allows developers to move fast: MongoDB's horizontal, scale-out architecture can support huge volumes of both data and traffic. Thanks to the flexibility of its schema, this distribution has proved its usefulness in spatial 2D (Đuric 2018; Voutos et al. 2017) and 3D visualization applications (Trubka et al. 2016). The management of multiple representation structure can be visualized using such a storage in the backend (Mao and Harrie 2016). However, its limited capabilities to strict visualization could not set apart the document-oriented storages and its features.

About the stored data and the city modelling, CityJSON proposes to renew the CityGML schema and provides a lightweight alternative to the XML encoding (Ledoux, K. A. Ohori, et al. 2019). Its improved support of levels-of-detail and metadata make it a good substitute to CityGML (Nys, F. Poux, et al. 2020). However, its usage is still limited to specific applications and data encoding (Kumar, Ledoux, et al. 2018; Nys, R. Billen, and F. Poux 2020; Virtanen et al. 2021). Besides it, the new support of 3D models in QGIS should improve its usability thanks to the development of a CityJSON plugin (Vitalis, Arroyo Ohori, et al. 2020). Extensions of the core module are also promising

way to improve the CityJSON usability and its update to the 3.0 CityGML version (Nys, Kharroubi, et al. 2021). In summary, nowadays, the storage of the CityJSON models are limited to files. There is currently no solution for storing and making models available in a collaborative and open manner.It is here worth mentioning that the dichotomy in which relational databases do not support JSON insertion and document does is no longer true (Chasseur et al. 2013). Relational databases have been refactored to handle JSON (Z. H. Liu et al. 2014). However, it still imposes the use of an additional mapping layer and thus does not provide a solution to the lack of flexibility. For instance, it is the case for 3DCityDB, which translates the CityJSON in CityGML encoding before storing it into the relational database thanks to the citygml4j software.

3.3 Solution description

This section is divided in two subsections: a description of the simplified data schema for a document-oriented store and a description of the proposed architecture to demonstrate the usefulness of the proposed schema. While the first justify our choices on an efficient data accessibility and document nesting, the second is a short technical description of all the improvements made by an up-to-date WebGIS architecture.

3.3.1 Schema model

In a document-oriented database, the records are stored as documents that follow non-mandatory and semi-structured implicit schemas (Olivera et al. 2015). All the documents respecting the same pre-established and semi-opened schema are gathered in a collection. These sets of documents allow the access and the indexing on the records or on a group of them. It is the primitive of the database query engine: everything revolves around this notion of collection. We note that, some efforts have been put to handle geospatial functions already but remain limited (Boaventura Filho et al. 2016). This section develops the various steps that led to enhance and modify the CityJSON encoding into a simplified database schema.

The bulk storage of a CityJSON city model in a single document without decomposing it in different collections is possible but limits the possibilities afterwards. A single collection storing all city models should therefore be queried as the document store works around this notion. Queries and indexing need to be complex to travel the embedded objects structure (an attribute is part of an object, which is itself part of the model). Even if compound indexing is possible

(i.e., successive levels of indexing on several attributes), this is not recommended for efficient queries (Reis et al. 2018). Moreover, updating a sub-object in the model without mobilizing the whole database become complex as it imposes to go deep in the non-dependent objects embedding, get the object and then insert the modified version in the model.

Next to secondary elements such as metadata and appearances, a city model is made of *CityObjects*. Those objects are natively embedded in the city model in a CityJSON file as JSON objects. However, this data structure is not efficient enough for a dynamic use. According to the benchmark (Olivera et al. 2015), the referred models are more efficient but impose developers to build dedicated queries. Consequently, once elements are created and stored in collections, the link to referenced city objects need to be accessible from the city model in a smart way.

We propose to create different collections in order to handle elements and ease their access. Hence, we decompose the city model in five independent parts: *CityModel, Texture, Material, AbstractCityObject and Geometry*. All imported records inherit their characteristics from these five collections as their models are derived from these five top-schemas from the CityJSON specifications (e.g. of a *Building* which is a specific *AbstractCityObject* with an *address*, a *measuredHeight*, a *roofType*, a specific set of allowed *geometries*, etc.). These alternate schemas are the second-order schemas (according to CityJSON) or discriminated schemas (according to this contribution). In the core application, the five first-order collections are defined dynamically by the database and the server at start (see Figure 3.2 for inheritance relationships with second-order objects). Note that the *CityModel* collection represents the models metadata only. A CityJSON model, as a file, is thus made of the gathering of its sub-collections. Different models can be concurrently stored in the same database and the same collections. Thanks to the database smart allocation of space, if a collection is empty, no record is stored (i.e. collection does not exist at all, which implies that space is used). If a modification is made afterwards, a new collection is created on the fly if necessary.

While importing the city model in the database, the city objects are stored as independent objects in the *AbstractCityObjects* collection with a permanent link to their relative *CityModel* document. Looping iteratively on the *CityObjects* array from the CityJSON file, we create a new document for each new element and validate it depending on the city object type (i.e., the validators are built on discriminated schemas independently according to the CityJSON specifications and thus the CityGML data model). All elements are then stored in the *CityObjects* collection whether it is a Building, one of its constituting *BuildingParts*, a *SolitaryVegetationObject*, etc. In short, the schema imposes the necessary basis for files to be correctly managed by the database and to follow

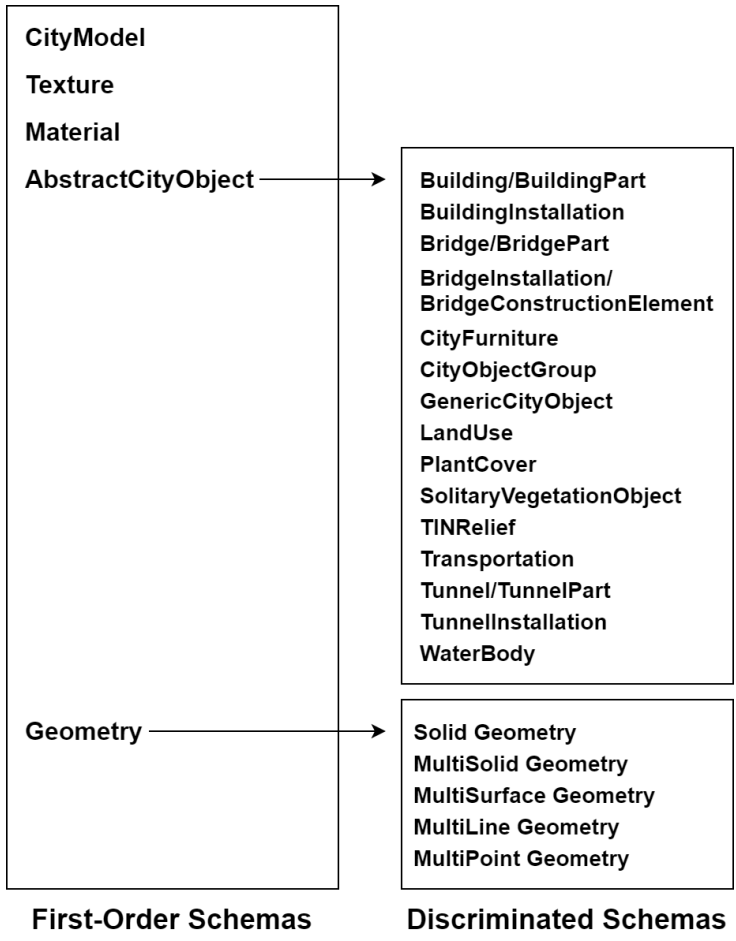

Figure 3.2: CityJSON objects schemas and inheritance. Discriminated objects are the "second-order" objects according to CityJSON.

the CityJSON core specification. However, the management of this schema in a NoSQL solution does not limit the insertion of extended attributes. We note that these extended attributes must still be coherent from a format perspective: no special characters, no insertion functions, etc. Once a document is saved, its corresponding document is afterwards referenced in the *CityModel* as a simple object stating on the type and the unique ID of the document in the *AbstractCityObject* collection (see Figure 3.3).

As stated above, every object is referred with a unique identifier specific to its life-cycle in the database (thanks to the special data type *ObjectID*). It is automatically generated and indexed by the database. This integrated management allows concurrent users to create objects at the same time but without any inconsistency insertion (i.e., users need to be aware that two modifications can be made concurrently without any guarantee of consistency in a NoSQL store). Note that the differences between the CityJSON discriminated schemas are sometimes very subtle but this substructure allow further development in a convenient manner: modification to the schema are easily made so that everything is decomposed, normalized and structured. The addition of extensions takes direct advantage of this flexibility as it might concern only a sub-schema or a part of it.

Concerning the insertion validation, during the model life-cycle, the *CityObjects* field can therefore either be an entire object as in a file, either a reference or unique identifier to the specific *CityObject* document. In order to prevent users to alter the consistency of the database, it is thus important to provide a pivot element which can take one or the other value without allowing too much deficiency (Diogo et al. 2019). It imposes the use of the *Mixed* datatype to validate the imported models. This pivot type is reused one more time for the CityObjects to geometries relation (1-N relation). The Figure 3.3 illustrates the referenced structure of the first-order schemas in the production phase; once documents have been created and referenced (i.e., value is fixed to *ObjectID* and a string specifying the type of the object). In order to handle spatial indexing and thus filtering queries responses spatially, a *geographicalExtent* attribute in computed based on the geometry of every document. It corresponds to the smallest rectangular bounding box enveloping the object geometries. This affects performances on model import.

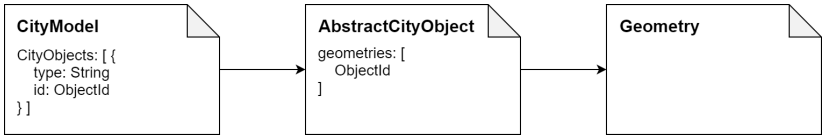

Figure 3.3: Referred documents structure in production.

All geometries, and thus the fine and complex representation of the objects, are stored in the same collection regardless of their type as has been the case with the city objects. Here, it is not about a spatial management of elements (i.e., spatial functions and indexes are not being used in the geometries collections) but about a management of elements of a spatial nature (i.e., documents are actually real 3D objects following the standardized geometry types). The geometries are complied with the ISO191607 standard according to

the CityJSON specifications. One more time, several discriminated schemas derive from the first-order Geometry schema: *Solid, MultiSolid, MultiSurface, MultiLine* and *MultiPoint* (see Figure 3.2). Note that the "composite" geometries being structurally similar to the "multi" ones, no new schema is created. They are managed as their "multi" equivalent with the difference that their type is composite and not multiple. As a reminder, the difference between the two is whether the constituent elements are contiguous or not.

As in the CityJSON files (i.e. the Wavefront .obj file structure), the object boundaries are stored as a list of vertices and arrays of pointers to vertices coordinate triplets in this list. However, the referenced vertices triplets for every object are stored in bulk within the *CityObject* document not in the whole *CityModel* one. This point set apart the database schema with the common CityJSON files since the vertices should be stored in the *CityModel* according to the specifications. In the direction of a wider support of spatial functions within the application and the streaming of features, this storage method improves an independent objects management: the spatial indexes and the consecutive references are suited for an optimized spatial function support. Note that this discrete handling of vertices affect the *CityModel* upload performances also. The support of spatial functions and tools represent an important future work. Without tackling the database, it would also be interesting to consider both server-side and client-side for spatial analysis.

Concerning the support of schema extensions, an important benefit of the application relates to the semi-openness of CityJSON specifications. While our motivation is to increase flexibility, we would not limit the possibilities offered by the semi-open schemas. Hence, the schema structure is not locked. It allows the addition of attributes and/or properties and new *CityObjects* type. We believe that CityJSON approach allow people to think about many solutions in this way and ease their development. This point on total openness goes against the 1.0.1 CityJSON specifications in which additional properties are not allowed in some *CityObjects* definitions. Hence, some drawbacks might be encountered: an exported model from the application might not be compliant with other tools in which specifications limit the model to the strict conditions of the specifications. Efforts from the developers need to be made in order to guarantee this interoperability.

3.4 WebGIS architecture

In the context of web development, when compactness and lightness are concerns, the creation of a full-stack MERN (MongoDB - Express - React - NodeJS)

application facilitates a smart deployment. MERN web applications ensure convenience for web applications that have a large amount of interactivity built into the front-end (i.e., the JavaScript clients). The following paragraphs describe the constituting components of a MERN application and decomposes its architecture in order to develop its benefits. Those benefits are mainly discussed concerning their answer to the lack of flexibility of previous architecture and the availability of a database support for CityJSON models.

Such kind of application is made up of a minimum of four technological stacks (ReactJS, NodeJS, ExpressJS and MongoDB) as shown in Figure 3.4. The increase of flexibility and resilience is demonstrated and put in parallel with the architecture components.

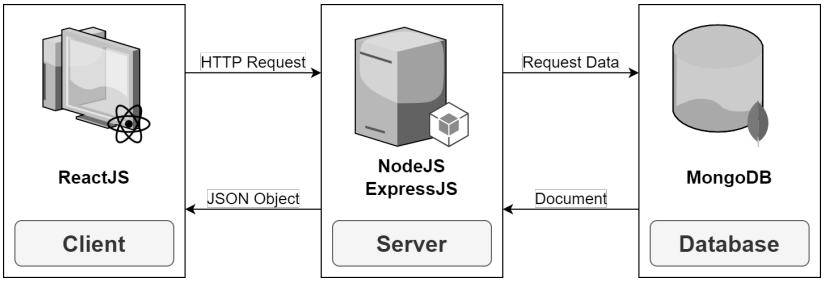

Figure 3.4: Architecture schema of a full stack MERN application.

The four open-source constituting stacks of the core application are the following:

- MongoDB – the document-oriented NoSQL database.
- ExpressJS – a minimalist web framework for NodeJS.
- ReactJS – the Facebook MVC library (Model–View–Controller).
- NodeJS – a JavaScript runtime environment.

The client tier is built based on the ReactJS library (see Figure 3.5 for illustration). ReactJS gave us the modularity necessary for the development of a new research tool as it does not dictate a pattern. We thus focused on the data architecture and the application consistency. It allows the construction of specific components and their reusability on a normalized basis. Note that the rendering scene is an extension of the NINJA viewer (Vitalis, Arroyo Ohori, et al. 2020). It is itself based on the ThreeJS library (the WebGL cross-browser JavaScript library for 3D manipulation and display). Nevertheless, the inserted

value during updates and objects modifications are tested in conformance with the *CityObject* schema and common insertion rules (i.e. no special characters, no injections, etc.). The client tier allows all the common CRUD operations (Create, Read, Update and Delete) on both *CityModels* and *CityObjects*.

The components communication is built on an event-driven paradigm: the components subscribe to particular messages on an events bus. They then react to their subscription whenever an update is published. The messages could carry information and/or simple messages. It allows decoupling components in order to increase performance, reliability and scalability (Allah Bukhsh et al. 2015). Following this, all components can be dismounted just as new components can be added modularly to open the application possibilities. Hence, two panels are left open to integrate new modules for dedicated functions: secondary view, tables, embedded objects, etc. Use cases of these panels are presented in the end of this chapter according to schema modifications during the production phase.

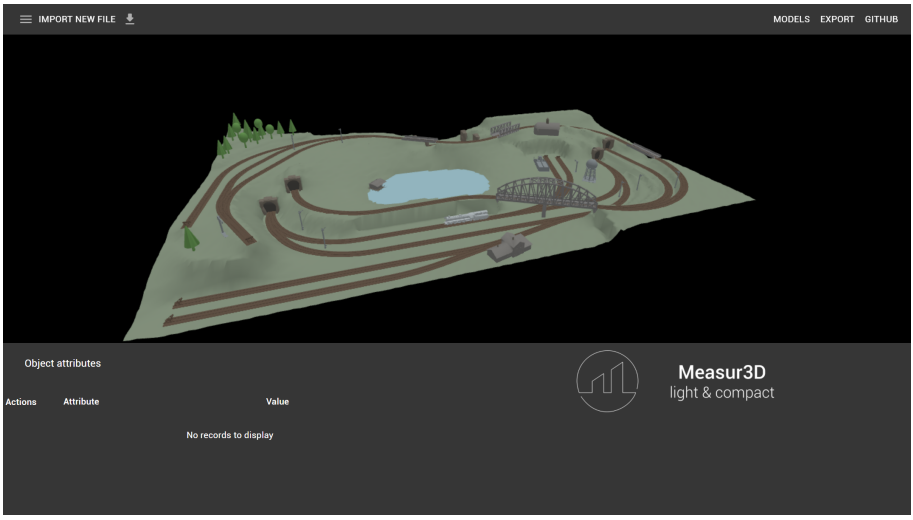

Figure 3.5: Client view of the application – the rendered model is the dummy Railway.json file provided by the 3D GeoInformation research group from TUDelft.

The server is a NodeJS JavaScript runtime environment that allows performing JavaScript code on server side (following the ECMAScript2015 specifications (International 2015)). It follows an asynchronous, event-driven, non-blocking input/output (I/O) model. These two last properties make it a very fast and resilient web server (Westerholt and Resch 2015).

Along with that, ExpressJS is a JavaScript library that simplify the task of writing web server code for NodeJS. Relying on HTTP requests (i.e., a RESTful application), it allows people to set up middleware function calls on a server: Cross-Origin Resource Sharing, rate limiter, cache, compression, authentication, etc. Currently, the REST API performs basic functions for CityJSON models and its features management such as CRUD functions. The communication layer follows the HTTP/1.1 requests specifications. We point out that the non-successful responses are possible but non-response are avoided in conformity with the BASE properties of the database. This property has been generalized to the server application. Moreover, the server tier and thus the API ensure the application consistency as the database itself does not provide any guarantee of it (Diogo et al. 2019).

The database tier is a document-oriented NoSQL store: MongoDB. Overall, the document-oriented solutions tend to improve the performances and the storage volume for dynamic data management. Despite many advantages, it is good remembering that the responsibility to maintain the data sanity is no role of the NoSQL database (ibid.). The indexing method takes advantage of the metadata of each record. The choice of a document-oriented solution has been made because of the schema flexibility and its native JSON support (database object are BSON document of Binary-JSON object).

Unlike the English-like SQL, the dedicated MongoDB query language performs CRUD functions but also aggregation, text search and a small number of geospatial queries. The functions take JSON objects as parameters. Besides referenced relationships, the collections are independent from one another. To make the comparison with relational databases, "joins" are not allowed between collections. This point will be discussed in Section 3.5.3.

3.5 Discussion on paradigm shift

Apart from the schema model and the proposed architecture, which have been discussed on a technical aspect, several conceptual points need an explanation: the use of NoSQL was not done without reason and some modifications to the CityGML/CityJSON conceptual schema had to be made. The decomposition of the CityJSON files in documents and collections schemas make up the structure of the database to perform normalized API calls. This section comments the contribution of the simplified schema in order to open its reuse in future works.

3.5.1 Structured and unstructured data

In this chapter, we propose to shift the database archetype from relational solutions to a NoSQL document-oriented store. This conversion should make it possible to open up possibilities and ease schema modifications. While structured data (i.e. relational solutions) promote a consistent data storage, unstructured data stores (i.e. NoSQL stores) intend to enhance flexibility and availability (Weglarz 2004). The relational databases represent the more rigid storage structure. It imposes a static tabular representation of the data (i.e. the data are imposed to follow a structure formatted as rows and columns). The consistency of relational databases is especially ensured by the respect of the ACID properties: Atomicity, Consistency, Isolation and Durability. The regard of these properties results in the guarantee of avoiding insertion of inconsistencies in the database. Conversely, the principal drawback of the relational family comes from the same reason: the data querying and thus its availability can be inflexible because of all the conditions imposed by ACID properties. Moreover, the table joins imposed by most queries can make them cumbersome and result in complicated processes.

For instance, in the context of urban modelling, DB4GeO provides a solution relying on an Object-Oriented Database (OODB) (Breunig et al. 2016). Focusing on the data integrity, an OODB follows the ACID properties. Even if the data structure established on objects is similar to NoSQL stores, we find here the disadvantages of the relational model mentioned above. In addition, it is difficult to make changes to an application that has been in production for some time. It imposes to rework the database structure upstream, before any use. Section 3.5 illustrates examples of how relational solutions need to be updated in order to handle new attributes and/or new features using new associations.

Oppositely, in contrast with the rigid tabular models of relational databases, a document-oriented store proposes to modify the data structure and open it. The NoSQL solutions do not follow the ACID properties but the BASE properties (Basically Available, Soft state and Eventual consistency). It results in a system in which denormalization is encouraged. The horizontal scalability is improved (i.e., the replication of the system across n-databases):

- **Basically Available:** the data are guaranteed as always available in terms of CAP theorem (Brewer 2000). For the reminder, this theorem states that any distributed data store can provide only two of the following three guarantees: consistency, availability and partition tolerance. Whether it is successful or not, there is always a response to any request: "non-response" are not possible from the store.

- **Soft state:** the state of the system could change over time. This can be possible even without input. This is due of the eventually consistent property.
- **Eventual consistency:** the system will eventually become consistent once it stops receiving input.

The document-oriented stores are composed of key-value pairs in which values can be records such as XML, JSON objects or even other documents. For instance, sets of semi-structured data might be deeply embedded and even recursive (i.e., chain references are possible). Nevertheless, the management of records and lack of standardized schemas improve their flexibility. It assumes a loss of records consistency to improve the database flexibility because of the BASE properties. The consistency insurance is thus carried over to server and client tiers and above all by the simplified schema. Here, the purpose is not the database consistency. A document-oriented store supports hierarchical documentation of data, which is akin to CityJSON models and objects management. Every single records is described by its own metadata. It uses agile and dynamic schemas without previously defined structure.

In summary, the alternative provided by the simplified database schema and its implementation in document-oriented stores allow users to ensure data availability and the flexibility of their application in a simplified manner. It is not a solution that would go beyond relational solutions but offers an opportunity to develop new functionalities. OGC API – Features should indeed be an important improvement. It would take advantage of the *CityObjects* collection, which corresponds to the notion of the standard: a set of features from a dataset. Besides, the *CityObjects* are themselves abstractions of real world phenomena and thus can be served as feature following the standard [ISO 19101-1:2014]. A discussion should take place around these considerations and state on how CityJSON and the proposed application can demonstrate it.

It is here worth mentioning that the dichotomy in which relational databases do not support JSON insertion and document does is no longer true (Chasseur et al. 2013). Relational databases have been refactored to handle JSON (Z. H. Liu et al. 2014). However, it still imposes the use of an additional mapping layer and thus does not provide a solution to the lack of flexibility. For instance, it is the case for 3DCityDB, which translates the CityJSON in CityGML encoding before storing it into the relational database thanks to the citygml4j software.

3.5.2 Stacks communication

During the development of the application, while the client was hosted on a remote machine, the application server and the database were hosted on the same machine. This design allowed us to test server load, response time and response mode from a client/server perspective. In order to assess on the best communication mode, we conducted tests on a city model loading. The web GIS client capabilities becoming greater and greater (Agrawal and Gupta 2017), we wanted to provide a benchmark of current objects managements possibilities for a unique client (i.e., Chrome's V8 JavaScript engine in both server and client sides). Tests in which n-clients query the same API has also been made (see section 3.5.4). Downloading the objects from the backend layer can be made in several ways:

- **(a) Discrete requests:** the server get all objects one by one from the database and send them to the client as soon as something is loaded. The city model reconstruction is carried by the client. It is characterised by a "flickering" apparition of elements in the rendering scene. It is a common asynchronous loading method.
- **(b) Bulk requests:** get all objects from the database then send them to the client in one aggregated object as a city model. The city model reconstruction is carried by the server. The model appears at once, in its entirety. It may take some time before seeing a result as all queries need to be resolved in order to response to the client.

We note that all exchanges are simplified thanks to the isomorphism of the application: all data are formatted as JSON objects in both back-end and front-end stacks. There is no need of translation or restructuration for the exchanges and the object management given that *CityObjects* are stored as they stand. In short, "what you store is what you access". The Figure 3.6 and Figure 3.7 represent the sequence diagrams for both solutions: continuous and bulk requests. They depict the succession of queries between the three-tier (client, server and database) and their responses.

The clients open a connection whenever they initialise themselves. The server and the database keep the connection open for future calls thanks to a NodeJS middleware. Hence, the client/server connection is made only once. Even if a client closes its connection, the database and the server keep a connection open for a limited amount of time in order to facilitate new connections. It is done given that opening a new connection takes a bit of time.

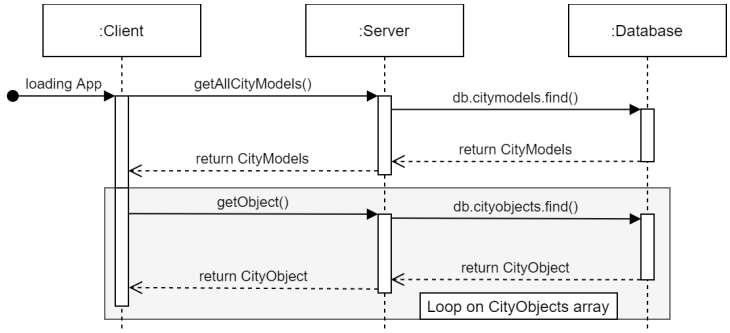

Figure 3.6: (a) Discrete loading (sequence diagram) - client-side reconstruction.

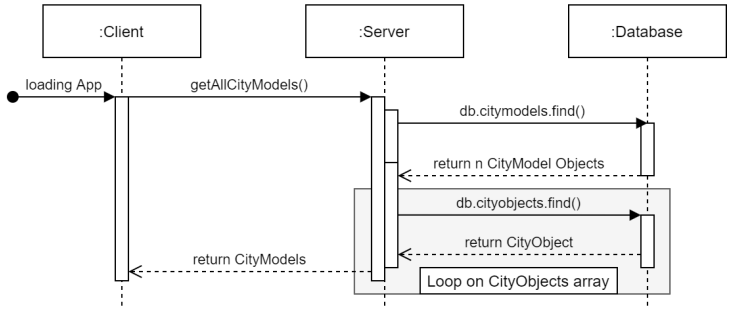

Figure 3.7: (b) Bulk loading (sequence diagram) - server-side reconstruction.

While the continuous loading allows diminishing the size of the bandwidth, the bulk loading allows making a single request on the network and reducing the global data transfer (i.e. fewer queries also means less redundancy in the formalization of query headers.). Moreover, caching the response of the bulk loading will improve performances as the model reconstruction is only made once. The tests were conducted on a small dataset, which numbers 120 *Building* objects and a *TINRelief* object. Note that, thanks to asynchrony from the NodeJS stack, the requests in the continuous loading were not stalled (i.e., no time were spent waiting because of proxy or ports negotiation before responses could be sent - the Time To First Byte (TTFB) represents very little). On the other hand, TTFB represented 99,6% of the bulk request time. It corresponds to the time for the server to process the database requests and reconstruct the whole city model before sending it. It is also important to note that time has been saved as *CityModels* are stored as they stand and thus the database does not need to formalize its responses. The whole process took twice as long

for the continuous loading for a total amount of data exchanged four times greater (each request have a header and thus multiply the size). Note that this consideration is only valid as long as the database structure does not change.

3.5.3 No joins

Within a relational database, the objects are often split in several tables. Many associations, which may be 1-1 but also 1-N and N-N cardinalities, link these tables together, making it difficult to access the data. Modifying the stored objects, the number of relations results in the modification of a potentially important number of tables. Moreover, this should be done cascading in a specific order: first tables referred by foreign keys are modified, and then tables linked with these specific keys. Hence, it is important to have a strong knowledge of the database structure and provide guidelines and documentation to simplify developers work.

On the other side, MongoDB retains the JSON objects structure and does not limit insertions. For the reminder, this is not possible with a relational database that imposed the use of conversion tools for native JSON file management. These tools often imply the creation of many tables, many joins and thus the formalisation of complex queries. Such queries and updates increase the time-consummation of processes due to the important number of joins needed. Hence, if the conceptual model is complicated, it ends up with a lot of complexity. A version attribute is modified on-the-fly allowing users to track elements. The CityGML encoding is a perfect example of a high complexity structure (Yao et al. 2018). For instance, in the 3DCityDB schema, sixty six tables are used to handle CityGML models in a relational database (against three collections in our simplified mapping and the use of the Mixed datatype). The addition of modules increases this complexity but also might imply to rework the database structure upstream. For instance, 3DCityDB and its import/export tools allow creating new tables and associations in a convenient manner during the database setup. Besides the addition of tables, it is worth specifying that these tables might be empty or not use in practice: given that ADE are generic, all information might not exist or not be relevant for the users' needs. This might be an additional source of bad resources consummation. This is not the case in document-oriented solutions: empty fields simply does not exist and documents structure evolves in accordance with the database lifecycle. In summary, the repetitive joins, which are the main drawbacks of relational databases, are avoided. This occurs in a more effective way to query, insert and store information whose structure is assumed to change frequently. To compute results on several collections at the same time, all collections need to be queried independently. The results are then gathered by the client (e.g. of MapReduce

processing techniques). As a reminder, the denormalization is encouraged so reference and links can be done cleverly depending on the use of the product.

3.5.4 Comparison reference with relational solution

To illustrate the disadvantage of the relational joint, we conducted a benchmark on several queries to 3DCityDB and our schema model. In order to perform these tests, we simulated two remote JavaScript clients conducting queries on one side on a PostgreSQL with the 3DCityDB model and on the other side on a MongoDB structured following our schema. Both databases included the same three city datasets that counts 3471 objects in total (3353 among them are *Buildings*). The query intends to get a random Building object with its attributes (*roofType*, *function*, etc.), its unique ID and one of its *Solid* geometries.

Some elements need to be discussed before any statement. Before the instantiation, both databases have a far different usage of memory. While 3DCityDB imposes the storage of 66 tables in 23Mb, our schema and its basic structure only takes 12Kb to create the three empty collections. The collection schemas and the validation of an insertion are handled by the server and not the database itself. It allows storage to be reduced and thus improves performances. Once instanced, the relational solution is 149 Mb wide against 87Mb for our schema (58%).

We have tested different interrogation methods by varying independently both the number of requests and the number of requested items. Note that the connection pool size of the database have an important impact on performances (a hundred was used). It is important to prepare it and to provide the same number of potential connections on both databases (by default, MongoDB allows only five concurrent connections. PostgreSQL allows hundred connections by default). It allows also to measure load under many clients querying asynchronously the databases. About the architecture scalability, there is still room for improvement by multiplying the number of replicated databases (W. Schultz et al. 2019). The balance should be determinate between the number of replications (n-databases), performance and the required consistency (Haughian et al. 2016). Nevertheless, MongoDB offers already the possibility to create replications in a native way, which should facilitate future work.

As stated before, the relational schema imposes to inner join three tables. Our schema simply queries an object from the *CityObjects* collection specifying that the type of the queried object is "Building". Then it queries the related unique ID of the geometry in the *Geometries* collection. Since a document-oriented store is built and indexed on such relations and nested elements, this two steps

retrieval seems to be more efficient. This hypothesis is directly reflected in the Table 3.1, which shows the databases response time.

Table 3.1: Response time for the *Buildings* queries – repetition x objects (in milliseconds)

	1x1	1x10	10x1	1x100	100x1	1x3353 (1xall)
Simplified schema	48	53	76	125	297	6678
3DCityDB	83	86	191	163	379	38089

These tests were conducted independently of the MERN application developments. In the application, a server cache avoids processing every query as some might be retained in the cache memory. In summary, this section offers an illustration of what is possible in the matter of response time thanks to the new schema, the document-oriented storage and the resilience of the MERN components. For the reminder, its contribution is a first answer to the lack of flexibility of relational databases used in traditional architecture and the support of CityJSON in a database. Hence, a convenient management of CityJSON models is thus facilitated by the simplified schema, its three collections and the "what you store is what you access". A common base is given without limiting the usefulness of the schema to a particular domain or specific end. These overall improvements of the schema and its dedicated architecture can be summarized in three points (see Figure 3.8):

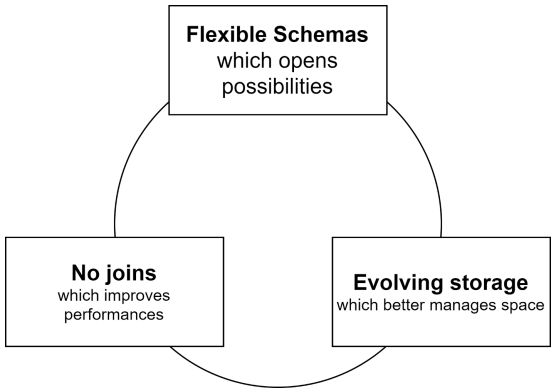

Figure 3.8: Summary of new capabilities.

3.6 Usage scenarios

Now that the schema has been presented and the database solution has been compared with a relational solution on a comparison between them, we will state on the schema flexibility through qualitative use cases. We have developed two simple extended schemas and two modules to demonstrate the usefulness and the flexibility of the schema. It is illustrated in situation of dynamic changes in the storage model during the production phase. The first one is interested in the visualization of flat roofs and their potential for the installation of green roofs. The second module concerns the management of the energy performance of buildings certification and the updating of its calculation method. As a reminder, the structure of the database is not modulated as the city objects are themselves not modified (collections are not altered). However, the objects schemas allow the addition, the deletion and modification of attributes in the stored records in a consistent way (see section 3.5.1).

3.6.1 Urban green infrastructure

Urban green infrastructures (UGIs) are part of the nature-based solutions for sustainable urban development. In a previous research, we took part in the development of a simple method for identifying the potential of green roofs along with identification of priority regions in city centers (Joshi et al. 2020). In order to estimate the potential roof surfaces of buildings, we interpolate planes based on a LiDAR point cloud and create building geometries (Nys, F. Poux, et al. 2020). Once planes have been interpolated, we extract their metrics such as the average heights of planes, their slope, their area, the number of planes per buildings, etc.

During the method development, some limitations were noticed in a 2D framework (Joshi et al. 2020): for instance, the obstructions are not considered (chimneys, elevator shafts, etc.). Taking into account a greater level of detail for the roof representation should therefore improve the conclusion and catch the user's eye. As preparatory work for this new study, we proposed to integrate the urban model into the application and add information as it goes.

Therefore, we developed an extension that handles the relevant information for UGIs installations. All information is attached to buildings geometries and integrated into the CityJSON city model as object attributes. Besides, a modified version of the simplified schema is hosted on the database. It validates the new attributes and guarantee the consistency of the application through its different usages.

It was possible to add information relating to these levels of detail, whether purely geometric or semantic, without modifying the work already done: the levels-of-detail refinement were added to the model, even if it was already used by project partners. There was no need to create an additional collection. The visual report gives users a quick glance on the zone and future development solutions (see Figure 3.9). As stated in (Joshi et al. 2020), the method can still be improved considering more socio-economic factors. Hence, the application will allow handling the modifications easily and provides a convenient integrator basis for further developments.

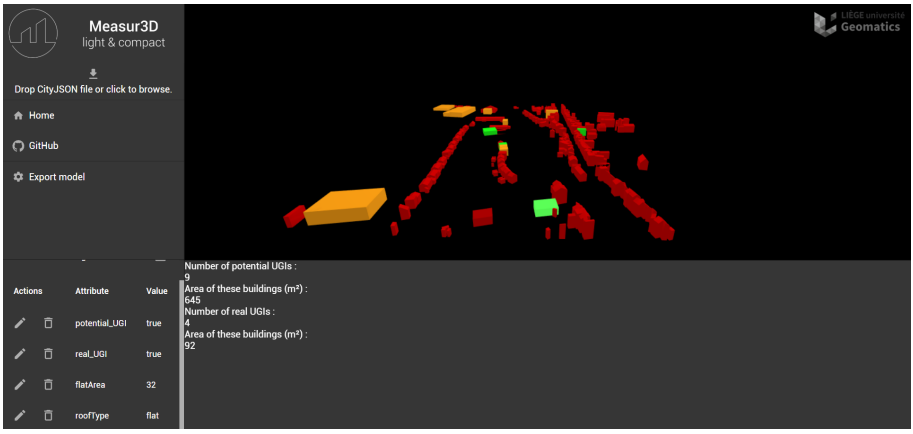

Figure 3.9: UGI module for the visualization and computation of green roofs.

For comparison purpose, the Table 3.2 has been updated to present response time of the *Building* query on the relational enhanced solution. In order to store the new information related to UGI, we added a table associated with the building one. Queries therefore impose the use of an additional join and thus affect performances, what we expected. Changes for the simplified queries in the NoSQL store are about the millisecond sometimes more, sometimes less. It has thus been not added to the table.

Table 3.2: Response time for the *Buildings* queries – repetition x objects (in milliseconds)

	1x1	1x10	10x1	1x100	100x1	1x3353 (1xall)
Simplified schema	48	53	76	125	297	6678
3DCityDB	83	86	191	163	379	38089
3DCityDB + UGI	88	91	252	172	412	41374

3.6.2 Energy performance of buildings

The European Directive 2010/31/EU of 19 May 2010 on the Energy Performance of Buildings (EPB) requires Member States to set up a system of certification. In addition to setting EPB requirements related to construction, it also imposes renovation work. The energy performance certification of buildings consists of an overall assessment of the energy performance of a building according to a defined calculation method.

In Belgium, this directive has been translated in an order of the regional government. This order reviews the calculation method on occasion and makes changes at both the semantic and conceptual levels. Depending on the modifications, the calculation of the energy potential of buildings can change: new parameters can be included, some can be deleted, new statistics and intermediate values can be useful or neglected, etc. In an EPB dedicated application based on a storage solution, all these statements result either in a structure modification for new features either storing redundant, unnecessary or incomplete information. As stated in the previous section, the usage of a NoSQL document-oriented solution allows adapting the object attributes without any condition and storing them within the same documents. This can be made without altering the database structure and frees unused space as it goes.

The use of an architecture presented in this chapter offers a flexible tool that can be easily improved through different changes in methods and legislation. Without going into details of the EPB calculation, we developed a module allowing calculating its value based on buildings attributes and metrics. It is computed on the fly and changes buildings colour following the normalised EPB scale (on the bottom left of Figure 3.10 - version updated on January 1, 2019). The Figure 3.10 illustrates a simulation on 2369 buildings in the centre of Liège, Belgium. The EPB module computes and stores the performance value based on attributes such as the type of heating, the coefficient of thermal transmission of a wall, etc. We simulated a modification in the EPB computation by taking into account the over-ventilation by manual opening of doors and windows (in accordance with the decree of 11 April 2019). It was thus sufficient to save the value but without modifying the database query mode using the REST API. The database has thus added key/value pairs to the schema and the required documents in the Buildings documents of the *AbstracCityObjets* collections.

The use of the tool proposes to handle both energy consumption data and 3D city models. Rather than manage the certification on an individual basis, we offer the possibility to build an energy cadastre at the neighbourhood scale but also of the city. The tool can be used by communities for managing their energy consumption and perhaps optimizing them: highlighting heat islands,

heat plant installation, real estate renovation campaign, etc.

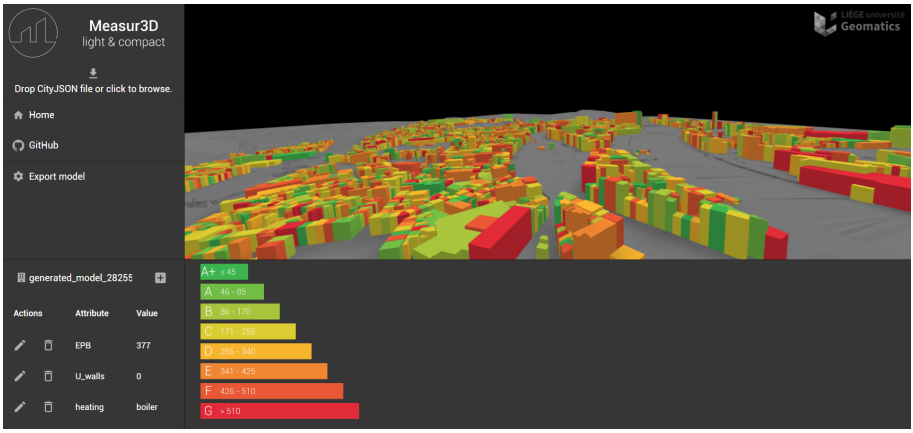

Figure 3.10: Illustration of the EPB module.

3.7 Conclusion

This chapter presents a simplified schema for the storage of 3D city model in a document-oriented store. It illustrates new capabilities in a dedicated application that allows the storage, management and visualization of CityJSON models. The JSON-encoding provided by the CityJSON specifications has been opened and partially reworked in order to extend possibilities of management. The different collections bring together the three main elements of city models (*CityModel, CityObjects* and *Geometries*) and ensure data access. The simplifications brought by this new model ease the accessibility and storage volume.

Besides, in order to demonstrate the capabilities of this simplified schema, we developed an application based on JavaScript technological stacks and a NoSQL database. This database shift proposes to go from a solution that ensure consistency (i.e. the ACID properties of the relational databases) to a solution that improves the application flexibility (i.e. the semi-openness of NoSQL schemas). The benchmark of this solution with the state of the art is convincing in terms of response time and storage weight. We believe that this application will improve the usage of CityJSON and web-based tools in urban built environment modelling. The usability of the application has been illustrated in two use cases of common practice: the visualization and the storage of urban green infrastructures and the energy performance of buildings

certification. The application allows users managing the diverse data sources and structural changes during the production phase in a convenient manner.

Future works will study the implementation of spatial functions support for the application. An important discussion will take place on the choice between the three possibilities of spatial support: database, client-side or server-side. While the former could not be done without a deep rework of the database management, the proposed architecture may have a place in the demonstration of spatial client/server capabilities enhancements. Nevertheless, such improvements should keep an eye on the implementation of the OGC API - Features standard in order to allow features fetching. A major improvement of this kind will improve the user-friendliness and the dissemination of CityJSON models.

Subsequent developments

In addition to the scientific publication of the results and the conceptual idea, the whole architecture and its code have been published on a public Git repository. It is available at https://ganys.github.io/Measur3D/. This repository is updated according to the progress of its different components and related research results. For instance, the use of the *MixedType* presented in the published paper, which allows storing mixed documents in the same attribute, has been modified to an embedded reference system. It allows gathering of constituting features among all database collections before sending it to clients as a consistent CityJSON model or features in an optimized way. The schema has also been updated to comply with the recently published CityJSON 1.1 version (i.e. the compliance with the 3.0 CityGML data model). Once more, the modularity and the interoperability of the architecture components allowed improving them independently.

This chapter represents the core of the thesis project: it is its major contribution as will be demonstrated in chapter 6. It links all developments and represents a sandbox for future researches and teaching. The architecture modularity and its independent components provide a great tool to build up proof of concepts for CityJSON extensions (as for the extension relating to point clouds (Nys, Kharroubi, et al. 2021)), to test 3D city modelling processes, to propose new solutions for future needs, etc. Significant optimization and documentation work should be done to get the most out of it. However, the simple fact that this alternative is proposed (and it is still only the only alternative at this stage), justifies its creation and study.

The development of our own architecture allowed us to perform many tests, tasks and experiences thanks to the flattening of the data schema. In the document database, we enjoy the loss of tree-hierarchy and constraints. Decomposing the web-architecture in its three constituting parts supports us to manage each aspect of the data lifecycle. First, we dealt with the generation of models to allow their storage in a second step. The server still has a role to play in the story we wanted to tell. Thanks to the provided tools, Master theses are in progress and should be successfully completed: the first proposes the complete workflow from a raw point cloud to CityJSON *Transportation* features. Point clouds are acquired by mobile mapping. The focus is made on the automation of the process. Models are stored and visualised in the architecture. It is mainly used in the end of the processing chain to make the models shareable.

The second thesis develops all the capacities of the architecture: it provides a first approach for the extension of CityJSON for the support of *Dynamizers* (Chaturvedi and Kolbe 2015). It extends the feature representations of city

models to support variations of individual feature properties and associations over time. Extensions are easily added to the core schema and stored in the database. The client viewer is then upgraded to support such dynamic information (smarter objects design, charts, etc.). Usability, flexibility and maintenance are one more highlighted.

Having such a sandbox also offers the possibility to handle "mutated" models. For example, it allowed managing point clouds in CityJSON models. The principle was to open a bit further the definition of the *CityObjects* in order to represent them as raw sets of points (Nys, Kharroubi, et al. 2021). Interesting results of this idea provide *CityModels* that are fully represented as point clouds. It is a direct mapping from the segmented point cloud file to the semantic *CityModel* and its features. Point clouds could also be used bringing added levels of detail for vegetation, interior of buildings, etc.

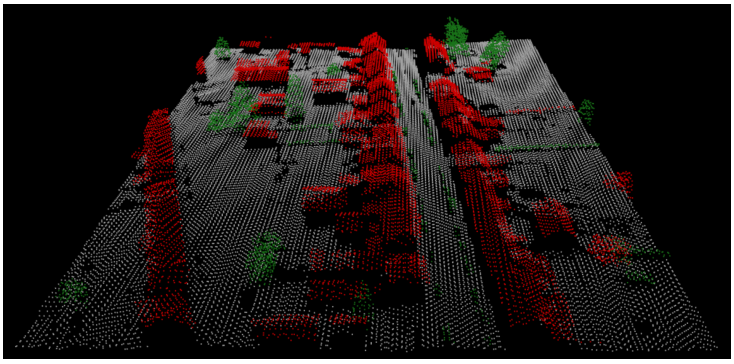

Figure 3.11: Visualization of an urban model CityJSON in point cloud only.

Chapter 4

Consistency guarantee

Preface

The flattening of the city objects structure within a model has been proposed by CityJSON. Storing the city elements in a bulk in a dedicated key *CityObjects* of the model object, the weight of the files is reduced but it is not the only benefit. Indexing this collection based on the object value (i.e., its unique identifier), it allows retrieving elements fast in the document-oriented schema without traversing a tree hierarchy structure as in CityGML. These new models capabilities have been exploited by storing them, respecting their form, in a document-oriented database rather than transcript them in a relational model. This transcription is still the only method for storing CityJSON models in a database nowadays.

The database schema has been improved in order to optimize models accessibility in the process. The shift from a rigid tabular structure to linked collections of documents (*CityObjects*, *Geometries*, etc.) enjoys an important gain of flexibility conceding a loss of consistency due to the BASE characteristics of NoSQL solutions. The features access is improved and the storage is simplified in the same way. However, the consistency of the architecture should not suffer of a too lax management. The data and its quality remain at the centre of all the proposed evolutions through these chapters; this must not change with the next contribution.

The MERN Measur3D architecture has been published and its code it publicly available on GitHub [1]. People have started using it and developing new features: *Dynamizers*, enhanced *Transportation* module, point clouds support, educational material, etc. In the meantime, it has been updated to support CityJSON 1.1.2. Among the improvements proposed by this new version, the typos in version 1.1. were already corrected in the previous chapter outputs. The majority of improvements from CityJSON 1.1 in specifications definitions are not the schemas. Therefore, we must not modify our contributions and it was indeed not complicated to switch to this version. The OGC API - Features capabilities have also been augmented in order to follow a part of recent developments.

Still, at this moment, the database itself does not provide any guarantee on data consistency; the focus is made on the flexibility and accessibility of the stored data. Since documents are not supposed to respect any predefined schema, the stored elements can take any shape, be consistent or not, refer to any other document or collection without any limitation on hierarchy and filiation, etc. We have proposed a structure but there is no technical architecture component that governs this scheme. Moreover, besides any additional security layer, people can access whatever they want on the database without worrying about access

[1] https://ganys.github.io/Measur3D/

rights, etc. In a shared and semi-open management of a unique city model, such loss must be compensated somewhere. Otherwise, some might as well not use a standard and thus not promote interoperability of the data that would be shared. One can, for example, imagine a filter that explains the errors, allows several coexisting versions of it, etc. Providing a schema in the last chapter, we have a basis to build such filter. This filter would act like Cerberus was guarding the underworld to keep access to the database. Each application must then be served by one of the heads of the dog depending on the version, the access rights, etc.

Putting the database completely aside, the postponement of the validation must be done on one of the two remaining tiers: the clients or the server. In our vision of the *web servicification* and its related-applications, it imposes that clients must stay "passive". They are consumers of data and thus must not assess on their quality, format, etc. Therefore, the only remaining possibility is the server. We wanted to provide a solution that works in both ways: writing and reading information. The creation of a middleware seemed coherent. This chapter develops the establishment of a new architecture layer that will envelop the server in order to shift the data consistency from the database to the server. Since the consistency provided by the relational database has been replaced using a NoSQL solution, the consistency guarantee should be postponed elsewhere: on the server. This filter should not be an added tier but rather a layer, as would a permeable membrane. Some Cerberus heads will be created in order to provide a proof of concept.

Based on article (Nys and R. Billen 2022)

From consistency to flexibility: Handling spatial information schema thanks to a middleware in a 3D city modelling context

Nys, G.-A., & Billen, R. (2022). *Transactions in GIS*, 26.

Abstract: Twinning elements of reality gains a growing interest in support of decision-making, learning and simulations: a single and shared model should provide a unique integrative basis for managing assets of any replica of the real world. From a technical viewpoint, sharing and opening information requires both an exchange format and a high degree of freedom and flexibility. It should allow an important number of users to manage this information, to modify it, etc. Storing and manipulating spatial information concerning the urban built context currently focuses on ensuring consistency thanks to relational databases and predefined schemas. Following a solution shift from a relational database to a NoSQL database, a schema validation middleware is proposed to improve the flexibility storage by conceding a share of its consistency. The flexibility improvements thus provide users a common basis that is able to evolve all along the lifecycle of their models and applications as required for twinning things. It allows users and their applications to take advantage of new storage features such as common: versioning, partitioning, prioritization, applications profiles, etc. The middleware and their new capabilities are illustrated thanks to the CityJSON encoding and its simplified schema for a document-oriented database.

Keywords: Schemaless database, NoSQL, middleware, 3D city model, Digital Twin, CityJSON

4.1 Introduction

The digitization of real world elements improves activities and applications in many domains. This could be achieved, for example, through the creation of a digital model providing a single integration basis for all these activities. However, in practice, even if a conceptual schema allows structuring elements around a common base, different competing models might exist and provide a different representation of the same reality: not all users are interested in the same aspect or the same details.

The design of a digital model is a long and difficult process that requires compromises. As the needs of each application are not the same, one often prefers to use a specific model, close to the needs of the application, which itself is also specific. It is often the reason that does not allow the implementation of a shared digital model: what is the vision of reality that is necessary but also sufficient regarding all the users' activities around the model? Should applications that are considered more complex make concessions by using a generic model or should we impose complexity on applications that can be limited to something simpler? This would lead to a potential disconnection with reality on the one hand due to a loss of information in a generalization. On the other hand, situations where interactions would be cumbersome and too expensive in terms of resources without reason can appear.

Therefore, several questions remain unanswered and requires a response whether it can be illustrated in a storage shift in the management of the data, in the technique and its applications or by using the new capabilities offered by the recent technological advances in hardware:

1. Is this choice still relevant nowadays?
2. Given that there is only one ground truth but an infinite number of potential digital models, why should there be any compromise?
3. Could we not propose a solution that would allow storing a unique digital reality while making the representation we make of it dynamic according to our needs?

In addition to the substantial investment involved in designing a digital model, saving the model, often in relational mode, seems to explain some of the compromises made. In a web application, the server, which allows exchanges and a part of the processing, and the client, which is the data consumer, do not impose any concrete limitation. Structured beforehand and thus guarantying the application consistency, the rigidity and inertia of the relational model

make it a change-resistant solution (complicated addition of heterogeneous data, modifications of the basic schema are always at the expense of some performance, difficult maintenance, complex horizontal scalability, etc.).

Would it not be possible to propose a storage solution that allows various independent applications to store and search relevant information in a single place? The guarantee on consistency would then be carried over to the server and the interoperability would be ensured using exchange standards. In this type of architecture, a document-oriented NoSQL database would allow completely free-shared storage of information without prior structuring. Besides the definition in collections (set of documents), the form of the documents is left completely free: even the attributes can change types without prior condition. The server would then be responsible for the information structure by filtering it during exchanges with the various clients (both for storage but client requests).

The contribution of this paper is twofold. On the one hand, from a purely technical point of view, it provides a middleware, which acts as a bi-directional filter on server queries and would then filtrate information following CityJSON semi-structured schemas. Added to the simplified database schema for the storage of CityJSON models in a document-oriented database, it provides the core basis of an accessible storage solution. Provided in a convenient and well document framework, it should allow people developing their own use keeping in mind standardization. This flexible but still consistent data management helps developers to make bridges between the constituting parts of much greater city models management platforms.

On the other hand, in a dynamic that is always moving towards greater openness and information sharing, a new solution is proposed as an alternative to traditional solutions. Digital Twinning, a unique and digital 3D replica of a city, is now possible by using this first assumption. It is illustrated relying on a storage still too little used in our opinion: NoSQL databases. NoSQL databases and web-related technologies gained interest in the scope of 3D city modelling. However, most of the time, the new propositions are framed in a succession of improvements of a recognized tools limited to the purpose they had when they were set up. The new solutions are still too often neglected in favor of traditional solutions without addressing the problem from the start: the design of the tool.

All the principles and ideas developed in this paper are illustrated in the context of three-dimensional urban modelling and city digital twinning. The contribution is therefore not about the concept of middleware itself, but also in the answers to the recent questions formulated above.

The chapter is structured as follows: the main topics studied are the exchanges

standards and the role of the database in a GIS architecture. First, the various standardized way of querying and accessing geographical information, the city modelling standards, their semantic data model (CityGML and CityJSON) and the usage of these standards in shared or unshared web architectures are presented. The state of the art is assessed to frame this research in storing 3D city models in databases and deliver them on the web. Then, after a quick presentation on what NoSQL databases are and their differences with relational databases, the solution shift to a NoSQL database and its basic specifications are evaluated. The principle and benefits of a schemaless database are discussed afterwards. Insights are also given concerning the usage of middleware in geospatial data management. Different methods of accessing information through features query services are presented in parallel with the major contribution of this paper: a bi-directional filter that simplifies the recording of information but guarantees the consistency of exchanges. Finally, future developments are considered as improvements and new possibilities that can be developed thanks to this new storage solution and the middleware.

4.2 Related works

Related works are divided in two different but interconnected parts: "Exchanges and standardization" and "Role of the database". While the first presents standards for structuring information in the urban built environment, the second part is a focus on the role of the database and its various shapes. The logical articulation of this part goes from a more general section to a more specific section that places our contribution in its context.

4.2.1 Exchanges and standardization

A "Digital Twin" is defined as "a virtual representation of a physical asset enabled through data and simulators for real-time prediction, monitoring, control and optimization of the asset for improved decision making throughout the life cycle of the asset and beyond" (Rasheed et al. 2019). On a conceptual level, especially in city modelling, the prospective potential of reality twinning is large (Shahat et al. 2021). Even if a wholly mirrored city is yet not available, improvements are relatively fast. In particular, improved data processing would make it easier to use the models and find information, but also to share it. Above all, the pooling of information from all kinds of sources is the main advantage of twinning. At this stage, all these considerations are anticipated to accurately reflect and affect the city and model's functions: data management, visualization, situational awareness, planning and prediction, integration and collaboration.

Consequently, the search and processing of data must be simple and attractive (Schrotter and Hürzeler 2020). Indeed, supporting decision processes should be made in a comprehensible way all along the lifecycle management. The focus is made on the contrast between the static of the relational databases and the continuous evolution of users' needs. Behind the idea that the digitalization enhances the communication, The World Avatar (TWA) is a project led by the CARES center of the University of Cambridge in Singapore (Mei Qi et al. 2021). The TWA intends to capture the idea of representing every aspect of the real world in a digital model. It is thus a large-scale project gathering various researchers in a wide range of research areas. In concrete terms, it takes the form of a dynamic Knowledge Graph (dKG) that should improve the interoperability between heterogeneous data formats, software and applications (Chadzynski et al. 2021).

In GIS architecture, many efforts have been made on the database tier (Zlatanova and Stoter 2006). However, there is still much room for improvement. For instance, relational databases do not support co-existing schema versions natively. It is thus complex to develop tools without imposing them to be created prior of any production launch. Smart solutions need to be found in order to allow concurrent versioning. Among these solutions, a bidirectional database evolution language provides a solution for the co-existence of schema versions using delta-code (Herrmann, Voigt, Behrend, et al. 2017). This language allows increasing the freedom to easily change the physical table schema but at the expense of some performance. Once the schema of a relational database has evolved, the stored data should also comply with the new structure. It imposes to guarantee the usability of the newly ordered database but also its completeness. A formal basis, which helps developers with the expensive and error-prone task of manual co-evolution (of both schema and data) is compulsory (Herrmann, Voigt, Rausch, et al. 2018).

The consumption of performance is highlighted in comparison between the features of the relational versus the NoSQL databases. An empirical comparison of their average execution times gives insight on their specific advantages (Baralis et al. 2017). The number of concurrent users and data set cardinalities have been also considered as they represent the great advantages of NoSQL. Among all the NoSQL database variations that exist, the document-oriented databases allow a great flexibility regarding the information structuration and their modifications. It allows storing documents in many convenient ways without imposing any predefined and strict schema, as would a relational database. Research is being carried out on the automatic creation of structures based on UML diagrams. However, it ensures the storage flexibility as it is the main asset of these NoSQL stores. A validation scenario presents the creation, its complexity metrics and states on the NoSQL assets (Gómez et al. 2021).

Indeed, modelling a relational database might become a tremendous process: all requirements must be assessed beforehand in order to build an application that meets all user needs. In the NoSQL environment, there is no equivalent to the Unified Modelling Language (UML) used by relational databases. Some could use new notation based on UML or Entity Relationship (ER), eXtensible Markup Language (XML), etc. (Vera-Olivera et al. 2021). A systematic review on NoSQL databases explores the current state of research regarding their design methods (Roy-Hubara and Sturm 2020). One of its findings states that database design should meet non-functional requirements. It means that database design should not state on what to do or must do but how to do things: in other words, the absence of predefined schema is an opportunity and must be taken to its advantage.

A middleware is a piece of software that implements communication solutions for an operating system. It is commonly used in distributed architecture to support input/output between stacks. In the scope of GIS architecture, it allows merging multiple and heterogeneous data sources (Cha et al. 1999) and multi-storage architectures (D. Li et al. 2018; Wong et al. 2002). Handling inputs and outputs also favors data integration without impacting on performance (Haas et al. 1999). For example, some propose to facilitate the merging of city modelling and building information modelling standards through a dedicated middleware (C. Schultz and Bhatt 2013).

Looking at the large family of NoSQL databases, the validation of exchanges using schemas is nothing new. For example for knowledge graphs such as these based on the RDF model uses the Shapes Constraint Language (SHACL) (Knublauch and Kontokostas 2017). In the context of spatial validation, it emphasizes recursive filtering and validation (Corman et al. 2018) and the reusability of validation schemes (Debruyne and McGlinn 2021). Such a technical solution is one of the basic pillars of the work towards a global European infrastructure (W. Huang et al. 2019). These principles are commonly called "application profiles".

In the same way but at a different level of the web architecture and a more global ingestion process, GraphQL is an API layer that allows people querying and mutating already existing data. It is the closest thing to a universal method of questioning. The request defines itself the desired structure of the answer. Recent improvements on GraphQL demonstrate their usage in network bandwidth optimization (Brito et al. 2019). However, like any new technology, it comes with drawbacks (Hartig and Pérez 2018; Wittern et al. 2019). There is however no official spatial features nor capabilities.

Geospatial data is data about objects, events, or phenomena that have a location on the surface of the earth. It combines location information, which

can be static or dynamic (usually coordinates or combinations and complex arrangements of them) and attribute information (characteristics and knowledge of the object). Given all these considerations, the exchange and the storage of such information imposes the usage of dedicated tools: spatial standards and spatial databases. The Open Geospatial Consortium (OGC) has a mission to improve geodata accessibility providing standards and normative exchange formats. These standards are global resources that are publicly available and free to use. Among others, the Web Features Service (WFS) Interface Standard provides an interface allowing requests for 2D geographical features. A new version has recently been published in a legacy review (Clemens et al. 2019). It has been done as to allow platform-independent calls across the web. This review is part of a new bigger family: "OGC APIs". These APIs are developed in order to make it easy for anyone to provide geospatial data on the web but in a standardized way. The different APIs are meant to provide building blocks that can be used to build APIs that are novel and more complex. Along with the maps, coverage and processing services, the features are part of the improvements brought in this new standards family. The "OGC API - Features - Part 1: Core" is restricted to read-access and describes the mandatory capabilities to implement a data access interface (ibid.). Future capabilities such as creation and modification of existing features but also additional coordinate references should be developed in future parts. Alongside, 3D Tiles is designed for streaming and rendering of massive 3D content (Patrick et al. 2019). It should not be confused with the OGC API - Features as the second concerns a way to serve information on a specific element and all its semantic information: attributes, versioning, etc.

In addition to the exchange protocols, the OGC standards also provide standards for the exchanges and representation of knowledge. CityGML is the most widely used standard for 3D city modelling (Gröger and Plümer 2012). Recent developments are related to extending the standard features: linking with other common standards (Biljecki, Lim, et al. 2021), wind simulations (Deininger et al. 2020), heating demand prediction (Rossknecht and Airaksinen 2020), etc. Among other solutions, 3DCityDB is a software package that consists of a database schema for spatially enhanced relational databases. It improves the database with a set of procedures and software tools allowing to import, manage, analyze, visualize, and export CityGML models (Yao et al. 2018). Another CityGML data model usage consists of a compact and developers-friendly encoding alternative of this data model: CityJSON (Ledoux, K. A. Ohori, et al. 2019). Besides its simplicity and easiness to handle city models, many advantages derive from the JSON encoding and its semi-opened structure: native support of metadata and refined levels-of-detail (Nys, F. Poux, et al. 2020), easier integration in common GIS tools (Vitalis, Arroyo Ohori, et al. 2020), lightweight and scalable base to support complex web applications (Virtanen et al. 2021),

usage of combinatorial maps in topology structure (Vitalis, K. Ohori, et al. 2019), etc. This new encoding solution opens possibilities by reducing the cost of modifying data but also facilitates its exchange. It is part of a dynamic that is increasingly focused on the web and the pooling of knowledge: servicification. This dynamic is the process to migrate code and applications to a modular and service-oriented architecture. This results in the production of reusable and decoupled components while also reducing duplication. It finally results in a better usage of resources and the sharing of capabilities and information. Servicification in geographical systems is well illustrated in SOA architecture (Service-Oriented Architecture) (Allah Bukhsh et al. 2015; Nys and R. Billen 2021). A flexible architecture allows the composition and sequencing of data processing. The geospatial intelligence provided by such services is a proper solution to most of the geospatial application problems (Fricke et al. 2018).

4.2.2 Role of the database

It is understood that 3D city models are great integrating bases for complex studies in various fields. This can be seen from the ever-increasing number of application domains extensions (ADEs) for CityGML (Biljecki, Kumar, et al. 2018): energy, noise, 3D cadaster, etc. However, even if the semantic information is well integrated in such models, their usability in simulations is not straightforward: this kind of linkage is often studied by the actors in the field of 3D modelling and not simulation experts. The method of storage is not necessarily responsible (Widl et al. 2021). One is proposing to review the way in which the information, recorded in a relational database, is accessed and thus linked to the simulation tools (Yao et al. 2018). Without modifying the base, this solution makes it possible to spread the use of city models and their linked information.

The management of versions and history within 3D city modelling, which can be generalized by allowing different views on the same information, can be done through the use of an ADE of CityGML (Chaturvedi, Smyth, et al. 2017). This independent extension considers new aspects as managing multiple temporal interpretations of a city and its features. It is now part of the CityGML 3.0 data model and should thus be implemented in its various uses (Kutzner et al. 2020). Despite the proposed solutions for versioning, several issues remain (Kutzner et al. 2020; Vitalis, Labetski, et al. 2019). Six issues were evaluated and discussed among the data providers' incentives, the database implementation, etc. but more specifically: the need to collect additional lifecycle and versioning information (Eriksson and Harrie 2021). The problem highlighted on the additional information is that it requires a substantial restructuring of the technical solution and work processes. In addition, the increasing complexity of

the database implementation increases with the number of versioning features included (Eriksson, Sun, et al. 2021).

Besides the relational databases, the vast panel of NoSQL databases offer complementary solutions. NoSQL databases propose to review the storage structure of relational database. Among others, when the links between the elements are preponderant, graph databases are the most suitable. For instance, thanks to the graph isomorphism tools, even if they are resources consuming, change detection is made between versions of CityGML models (Nguyen and Kolbe 2020). Moreover, a much precise definition of the change types is given based on the graph structure. As it has been said, the graphs are useful for modelling the relationships between the city features. More precisely, the translation of these relations in Resource Description Framework (RDF) triples structures the semantic information of the urban built environment: the only inconvenient is that the geometric information is neglected (Malinverni et al. 2020). It is worth mentioning that ontologies are preserved during data conversions and can therefore be queried afterwards. It opens up fusion possibilities for city models with various sources using a NoSQL graph database: IFC, IndoorGML, etc. Structuring information in graphs also provide solution for bi-directional transformations. It allows deriving models from real CityGML models and instrument modelling and analysis facilities for digital models (Visconti et al. 2021).

Document-oriented NoSQL databases offer interesting possibilities. Besides any processing efficiency, the whole data structure has been reformed. It is much simpler than relational databases that use joint keys for example (Bartoszewski et al. 2019). Changing the users' perspective on data can improve and even rethink the basic idea of relational databases. The database design itself gives an answer to the multi-purpose needs for WebGIS (Sutanta and Nurnawati 2019). Without providing a complete solution compared to what relational solutions offer, the NoSQL databases offer premises of spatial data management on the web (Costa Rainho and Bernardino 2018). Especially in 3D city modelling, the shift from consistency to flexibility opens many possibilities (Nys and R. Billen 2021). In this research, a combination between CityJSON and the NoSQL document-oriented database provides an alternative to the traditional geodata management. The parallel can be drawn with 3DCityDB, which proposes a data schema for storage in a relational database. The comparison between the two tools was made in terms of performance but also in terms of their capabilities. In short, it improves the modularity of information thanks to the lack of schema for the database. Gains of performances and capabilities are remarkable kiss-cool effects too. For instance, proposing new extensions, and thus improving and adding features to the schema, is easier and supported in a convenient way thanks to the schema and its translation in the semi-open

database structure (Nys and R. Billen 2021).

4.3 Schemaless database

This definition of Rasheed et al. 2019 for "Digital Twin", even if it remains vague on the "virtual representation" term, focuses on the long-term usage and lifecycle of the information. This representation should therefore be required to be modular and flexible in order to adapt to current but also future needs. Without going for a complete avatar, a digital replica whose main characteristic is its shared uniqueness is a point worth studying. Even if relational databases provide solutions and capabilities, those are not suited for development in line with modifications in usage needs and horizontal scaling. It can therefore be considered that they do not address the root of the problem: the flexibility of schemes and thus the whole architecture modularity.

Tacking a step back, a webGIS architecture is constituted of three components: (1) a client, (2) a server and (3) a database. While there is no limitation on the number for each tier, it should be at least one element for each. Thus, a wide range of combinations is possible. Moreover, the elements are not always parts of the same whole; they might be under responsibility of different organization, located in various places, etc. Most of the time, the server and the database are closely linked and why not installed on the same physical machine (the architecture thus become a "two-tier architecture"). A brief explanation of the usefulness of each tier provides a better understanding of the shift proposed started in previous research in which this contribution fits (Nys and R. Billen 2021).

The client is the consumer of the data. It can be a viewer, a GIS standalone software, a web application, etc. Since the "frontend's" capabilities are evolving, clients support more and more processing. For instance, the web browsers, thanks to the creation of the V8 JavaScript Engine (Chromium Project of Google), handle more and more capabilities (Kulawiak et al. 2019): heavy graphics computations, graphs manipulation, etc.

The server takes care of the processing part, or at least part of it, as the frontend improves as mentioned above. It manages the database connections and receive the clients' queries (Wagemann et al. 2018). It is possible for a client to query a database directly, but the presence of a server makes it possible to improve security, set up statistics, structure and guarantee the consistency of exchanges. With the database, it is part of what is called "backend".

The database saves information, it structures the data and allows its accessibility.

For example, the relational mode structures information in tables and defines the relationships between them thanks to associations and cardinalities. Therefore, a predefined schema is mandatory so that the defined boxes and their links can be filled in later. It is the main advantage of using relational databases: the guarantee of consistency. Still, one can suffer of the predefinition of such framework. The users' needs and applications capabilities might evolve and no longer fit this schema. It could then be interesting to provide an alternative that concedes a loss of consistency to improve the architecture flexibility. A partial answer to this problem is to shift the use of a traditional database and move towards a NoSQL solution (Nys and R. Billen 2021). This contribution is in line with this answer and proposes to make a step further from the consistency to the flexibility of databases in the scope of modeling urban environments.

4.3.1 NoSQL models

Before considering NoSQL solutions, attempts to improve the relational model are worth mentioning. One of these is the BiDEL language (Herrmann, Voigt, Behrend, et al. 2017). However, these solutions gets around the problem without tackling its root. The language acts like an additional layer that improve the relational database capabilities. The database itself is not adequate to handle specific features. For instance, thanks to BiDEL, the versioning is simplified but it imposes to manage a new technology that adds complexity and potential problems. Tackling the rigid structure of the relational databases is avoided but not solved. It would be more interesting to find an integrated solution.

The research topics of the TWA project study the formalization, the evaluation and the repair of ontologies based on the CityGML and many other data models (in field such as environment, weather, etc.) (Mei Qi et al. 2021). Their integrated and dynamic knowledge graph structures information from a semantic point of view at least. As a complementary layer, the 3D geometric information brings unavoidable information concerning urban management. Undoubtedly, it should find an interest in developing a geometry support, if not at the beginning at least at some point. This project nevertheless illustrates an important need: NoSQL databases not only offer new capabilities but also provide a very new storage and many advantages. Subsequent to it, it is not only the arrangement of the data that changes; it is the whole perception of it.

At this point, an explanation on the NoSQL storage should be given. NoSQL solutions (Not Only SQL) are defined as "everything that is not relational". In fact, it is much more complex than that. The NoSQL family responds to capabilities that are indeed different from the relational databases but still correspond to a set of definitions. The main difference between relational

databases and NoSQL solutions lies in the management of their schemes. NoSQL databases, without going into the details of their various families, do not limit the data to be filled in predefined boxes. In other words, the database does not impose a schema for the data to be stored. NoSQL databases are "schema less databases". Besides the ACID characteristics of traditional databases (Atomicity, Consistency, Isolation and Durability), the NoSQL databases follows the BASE principles:

- **Basically Available**: the data are always available; there is no downtime despite any network failure or temporary inconstancies. A "non-response" is impossible from the store. Whether it is a success or an error, there is always an answer to every request.
- **Soft state**: even without any input, the system state could change over time. This characteristic is required for the following "eventually consistent" property.
- **Eventual consistency**: if no further updates are made to an item for a long enough period, all users will see the same value for the updated item. In the meantime, anything can happen. The system will eventually become consistent once it stops receiving input.

The "eventual consistency" characteristic is the linchpin. The soft state characteristic is one of its requirements and the availability is a quality of life asset but does not have any link with the consistency. The third characteristic is indeed the most interesting one: the eventual consistency means that the consistency is not set by the database itself and might not be always guaranteed. The database could deliver different information to various users in some state. The compromises made on consistency and the above-mentioned responses' heterogeneity can be considered as potentially harmful. This is true if the database is considered as an isolated component. The server, and why not, the clients, might have a role to play in the consistency assessments.

As they have been defined in the previous section, clients are passive consumers and thus free regarding the data structure. Both databases and clients should be independent services but clients must be able to work with what the databases provide, as they are more flexible. Even if they can support a part of the computations, clients should not require to control and validate the server responses. They just visualize or process the data but does not restructure or modify it. Otherwise, they will become an active component and the server may have neglected some of its responsibility.

The key idea of this contribution is to take the opposite view of the "schema less" database and to take advantage of this actual flexibility. Since no schema

is mandatory by the database, the opportunity is to store data without any restrictions beside technical constraints: format, encoding, etc. Any shape of information can thus be stored in the same place. Taking the assumption that an important number of record variants can be stored in a unique database, one can consider that some records will share a common basis or correspond to a common structure. Where several pieces of information relate to the same real object, the use of a single and unequivocal identifier should allow connecting these pieces. Moreover, each element might have common attributes and/or ways of representation with one another (versions, extensions, etc.). They are actually different copies or views of the same entity. Stating on a common basis and referring to real objects uniquely, one can consequently define the foundation of a shared but limitless model.

Every city is unique. It has its own history, its specific space, its citizens and their own lifestyles, etc. Many public services and stakeholders have their own views of the city and its assets. However, they should not be allowed to harm or modify those of others. In addition, if interactions should be possible, they should be done at least under pre-established conditions.

Back to the data store framework and its infinite theoretical set of city models, a common basis should determine the constituting elements of a city and their relationships: it is the main purpose of standards such as the CityGML data model. Since CityGML is a semi-open standard, it consist not only of a shared ground for city modelling but also for extensions and future applications. It thus offers the possibility to reuse the compliant data in different fields and applications that are themselves compliant to the data model. However, the majority of recent developments in 3D city modelling accept the relational storage mode and its advantages without questioning its initial capabilities. Hence, they focus on developing extensions proposing new features, new attributes and new relationships without considering any use of a unique and common digital model. Such a model, whose core can itself evolve as improvements are made, has been little studied. Among others, the ACID characteristics are part of these limitations.

4.3.2 Architecture specifications

Before presenting the architecture capabilities and the benefits of the new component, a specific point of its features should be discussed regarding the storage shift. As defined in the previous sections, its groundwork relies on three things: the usage of a NoSQL database that improves flexibility at the expense of some consistency, the usage of a common definition basis such as the CityGML data model and finally a new component that filtrates information.

These specifications are available thanks to the simplified database schema for the management of CityJSON 3D city models in a document-oriented store (Nys and R. Billen 2021).

Thence, a first step towards ensuring consistency is done by implicitly choosing that all applications must be standard compliant. In the case of urban modelling and JSON-related technologies, CityJSON 1.1 is unavoidable. It is here worth mentioning that it remains an unspoken consensus for some tier: the database itself is not structured following any schema. No conditions are set during creation and modification on records about any cardinalities, document structure, document size limitations, etc. This is the role of the proposed component, which is mounted on the existing application server, and only it. The server can thus be used to lock users' exchanges and structure queries on the server but nowhere else.

While the current applications developed around this simplified database schema of CityJSON concerned the storage of multiple city models in a unique store, the new architecture will make an additional hypothesis: the store remains unique but the unicity is now generalized to the stored model also. Note that the number of frontend elements is still limitless. In summary, one database is shared by several server or Application Programming Interfaces (APIs), that are themselves receiving queries from an unlimited number of clients on the web. All this is done under the assumption that the hardware is not a limited resource. The figure 4.1 illustrates the architecture of the shared database.

Two requirements depicted in the Figure 4.1 need an explanation: accordingly, in the simplified schema, a document, or record, corresponds to a model of a city or to an element of the urban built environment (i.e., an *AbstractCityObject*). Hence, in order to be able to refer to the correct record, a document stored in the database must be defined by a unique way of identification. For example the "name" attribute in each level of the architecture should be formatted in a similar manner. Secondly, a *Class*, which specifies the city object family, might also be given to objects in order to simplify the various queries. These classes are used to manage the different schemes by the server. Examples of *Classes* are *CityModel*, *Buildings*, *SolitaryVegetationObject*, etc. according to the CityGML model specifications. Other parameters (Param_1, Param_2, etc.) might also be defined in the core specification but also come from extensions. For instance, since the stored documents should implicitly comply with the specifications of CityJSON, it is thus possible for an application to query a Building object knowing beforehand some of its attribute: *address*, *roofType*, etc.

These considerations are generic to any number of clients and applications. As a result, information can be derived from a theoretically important number of architecture elements except for the database, which is deliberately intended

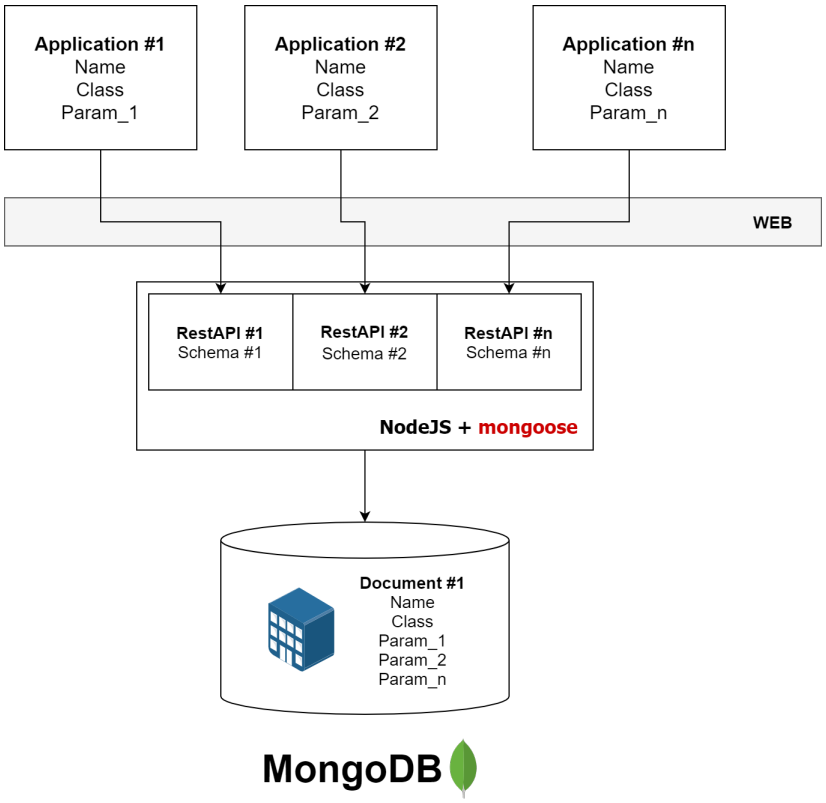

Figure 4.1: Architecture of a shared and unique database.

to be unique. Hence, the whole architecture can be abstracted by a tree, so that the database would be the trunk and the clients the leaves. The servers will then be the tree branches (see Figure 4.2). For the growth of the tree, the only limitations are network and hardware considerations since the data is constrained.

Besides the semantic formalization of the shared information, this component could be the basis of more complex mechanisms. One can imagine that the unique model is used by various users from the same city government. While each city service is working on its specific aspect of the city model, details can be brought thanks to the filters (and thus versions, extensions, etc.). Besides, the security issues of a non-strict user, several concurrent schemas might be used by the same user community limiting accessibility in respect of the grade or hierarchy like applications profiles do. This choice is left to the developers,

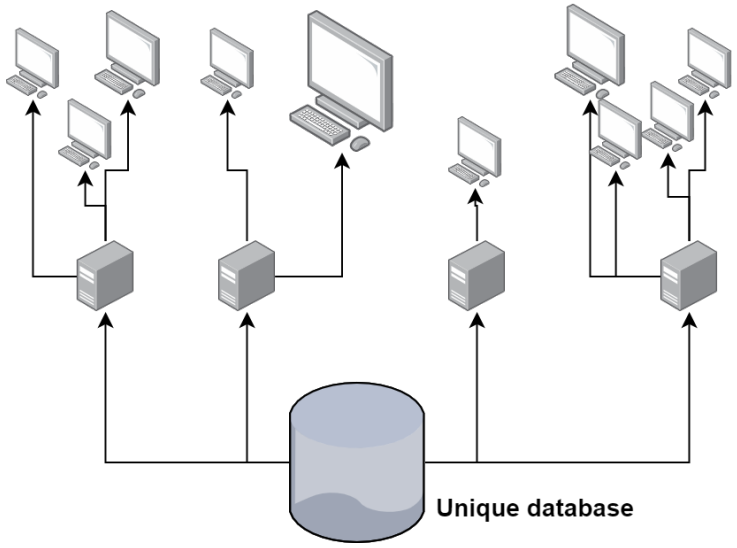

Figure 4.2: Abstraction of the architecture in tree form.

as they may be interested in developing stand-alone applications or in routing users through a larger application.

Concerning the versioning, the new architecture not only ease data versions management but also the data model versioning. It is customary that a database and thus the stored information comply with a unique data model version. Updating the data when a new version of the data model is released can lead to the database becoming obsolete if there has been no insight on backwards compatibility. As a result, in this architecture, there is no need for data update in the database, only technical maintenance is mandatory. The filter will then serve information in respect of the desired version picking relevant information from the semi-structured whole. The same information can therefore be easily shared by different versions or different applications.

Thanks to the BASE characteristics of the NoSQL databases, the data are always available. This means that the database should always be operational and able to respond at any time. As the server component is independent of the database, any maintenance on a specific application will not limit the usage of the others. It therefore improve the flexibility of the whole architecture and provides a modularly solution for further developments.

As a comparison with a current good practice, GraphQL is an API layer that

lies between a server and clients. It allows querying and mutating data in a generalized way (Wittern et al. 2019). Several notable differences are to be noted: first, GraphQL only allows a single endpoint on a database and imposes developing new features in the same way. Server functions and data manipulation are limited (Brito et al. 2019). Such a limitation would have limited us in the development of the application clients and especially with its links to the OGC API. Moreover, the document collections in the predefined simplified schema are built keeping in mind performances and most common queries. Staying with the development of advanced capabilities, GraphQL lacks of temporal and spatial features (Hartig and Pérez 2018). Such features are mandatory in the scope of city modelling. Mutations have another major concern because the tool was not originally created for this purpose: mutating functions are not conducted in parallel: each change waits until the previous one is finished. It is a major drawback when it comes to open the architecture to the many. It affects users experience in a very negative way. Finally, errors handling can become tremendous for developers since HTTP requests only serve 200 status queries(or 5xx if the server is not available at all). Avoiding these drawbacks and allowing developers to create the best endpoints they need is an imperative.

Since maintaining the data consistency is no more the responsibility of the database, it is now the role of another architecture tier: the server. In the proposed architecture, because of the storage shift, this guarantee is transferred to the server or at least to one of its components. The present improvements are made in line with the previous: it proposes a proof of concept using JavaScript libraries. The new component is hosted on a NodeJS server, a JavaScript runtime environment. The component is built on the mongoose library, an open source solution that provides built-in type casting, schema validation, query formalization and building, business logic hooks, etc. (https://www.npmjs.com/package/mongoose). Among these features, formalizing a query and the built-in type casting do not concern any aspect of storage consistency. By contrast, the schema validation is the cornerstone of the proposed improvement. In practical terms, mongoose acts like a bi-directional coat that filtrates information between the client, the server and the database. It acts both as a mediator and as a wrapper (i.e., in both directions). A predefined schema on the server maps a requested resource to a collection of documents in the database and serves relevant information to client and vice versa. It also allows verifying the format and encoding of any exchanged resource. This second feature does not play a role in the consistency of schemes besides technical considerations.

As far as semantic information is concerned, thanks to the discriminated schemas of the middleware and its inheritance capabilities (which are not possible

in JSON schemas); some variations can be added to the schema definitions without having to modify the initial requirements. For instance, a *Building* is nothing more than an *AbstractCityObject* with an *address*, a *roofType*, etc. The added information is still compliant with the *AbstractCityObject* schema. A *SolitaryVegetationObject* is an *AbstractCityObject* that might have a *specie*, a *trunkDiameter*, etc. Nevertheless, there is no requirement for each application to have the same exact definition of what a type of feature exactly is. It is a matter of agreeing on the common basis from the CityJSON specifications. The notion of hierarchy being absent from JSON schemas, this point reinforces the demonstration of the architecture flexibility as it simplify modifications without damaging the already existing schemas. Note that JSON schemas require checking several concurrent schemas rather than offering the possibility to specialize them.

By going further into the technical definition of the architecture: such a filter stands as a middleware. A middleware is a software that lies between an operating system (i.e. the server) and the applications running on it. Common middleware provide security layers (limiting the number of request, cryptography, etc.), cross-origin requests managers (accessing restricted resources from a remote domain), authentication layers (checking tokens and/or registered users, etc.), etc.

Beside these technical features, it also allows for the removal of excess information sent by clients or delivered to them but also the databases. The component filters information and maps it to the documents collection in the document-oriented database. This is done so that the semi-structured database is not polluted by incorrectly structured or unwanted information. This mapping consist of a collection of features schemas (not JSON schemas), themselves built on the CityJSON specifications. In addition to the schemas, inheritance is added between collection elements thanks to the mongoose features. In the proposed architecture, all the capabilities of a middleware are used such as it acts like a bidirectional filter:

(a) In one direction, based on the queries made by the clients (i.e., in a writing way), it filters the city objects and their attributes before any storage and/or potential updates. For the reminder, every API might be independent and built in such a way as to allow flexibility. They offer several different types of requests with different accesses, different connections, and different schemas as a result. For instance (see figure 4.3), the application #2 is not allowed to modify (and perhaps damage) the objects and attributes handled by the application #1. Be aware, however, that some objects might be shared by several applications and the same thing for the attributes of shared objects. The value of a common standardized and documented basis is again demonstrated here. It is a major prerequisite.

(b) In the other direction, for the documents queried by the clients from the database, the filter works exactly the same way. There are not only the format and the encoding that are verified but also the semantic information thanks to the schema specifications. It is retrieved from the database given that the attributes are checked and validated by the application-related schema. No feature object or attribute that has not been defined beforehand in the schema will be served as a response. It is important to note that while format consistency can easily be checked, logical consistency, i.e. the compliance of values with their semantics, cannot be verified regarding coherent meaning. This could limit the applications interaction and information retrieving but it is part of the responsibility of each data producer and its policy on whether or not to open the data and document it. In this context, the technical elements necessary for this sharing are provided without taking a position on this last aspect.

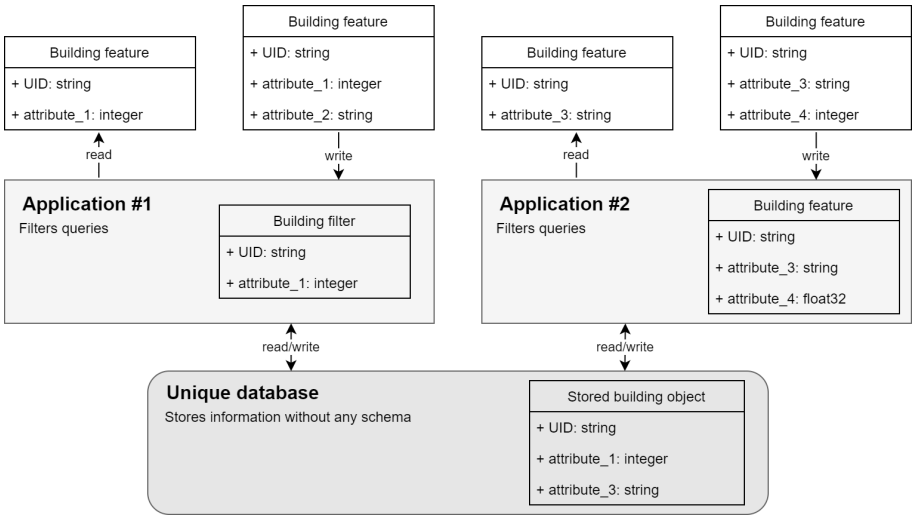

Figure 4.3: Illustrated example of the bi-directional filter principles.

In the figure 4.3, the example represents two applications that want to register and retrieve information on the same object of the same class. Considering that both the unique identifiers are the same (whether they are URIs (Uniform Resource Identifier), UUID (Universally Unique Identifier), etc.), the object is defined by four attributes: two attributes handled by each applications. Some points are noticeable:

- The attribute_2 is not stored in the database because it is not allowed in the server schema of the first application. As it does not pass the filter

in a writing way, the information is not sent to the database and thus cannot be queried afterwards.

- The attribute_4 is not stored because it is not properly formatted with respect to what is required in the second application's schema. If this attribute has been correctly formatted, it will be stored in the database and made searchable by clients.
- Both attribute_1 and attribute_3 are stored in the database but their use is limited to the separate framework of the two applications.

In the example above, the attributes are "basic and common" data types: string, integer, float32, etc. In the context of spatial information, and in the even more specific 3D city modelling, "spatial" data types require a dedicated management to handle their specificities. In addition to the complex representation of the built environment, the formatting of geographical information and features geometries imposes conditions. It should be noted that geometries must follow well-defined patterns most of which are defined by international standards. Among others, the concept of level of detail needs to be discussed and addressed (Biljecki 2013).

As defined in the simplified CityJSON schema for document-oriented databases, the geometries are managed in a dedicated mass-collection regardless of their type, the number per element and their level of detail (Nys and R. Billen 2021). For the reminder, every type of geometry has its own validation schema whether it is a *Solid*, a *MultiSurface*, etc. They all share a common basis but some specificities are brought in their specific sub-schema definitions by inheritance. It works the same way as the *AbstractCityObject* and the *Buildings* schemas. The schemas are independent of any level of detail and all types of geometries can be arranged to create various types of levels of detail. A unique identifier refers these geometry documents, one for each level of detail, to their related feature documents in the city objects collection. While storing and thus writing a geometry in the database, each element and level are checked against their scheme. On the contrary, in order to optimize and better adapt to the users' needs, querying a specific geometry can be done specifying the desired level-of-detail. All attributes of the geometry are served since it is a feature in its own right.

In practice, given that CityJSON handles the "refined levels-of-detail" (Biljecki, Ledoux, and Stoter 2016), the geometries can be queried in a compound manner. Either, the specified LoD is itself a refined one and thus can be retrieved if it exists. Either, if it does not exist or is a broader one: the most detailed one is recovered while remaining in a coherent level order. For instance, for a geometry stored in 2.1 and 2.2 levels, querying a unique geometry for the 2nd

level will respond with only the 2.2 document. Besides it, one cannot retrieve a LoD greater than the expected. It is done in way to reduce exchange weight and providing redundant information.

4.3.3 OGC API - Features

Besides the operations presented above and their request mode, it could be interesting for the users to handle features in a more standardized way. It is important to allow every user to have a view of what is stored in the database. It must be done in a reading-limited way so as not to compromise the database. Therefore, the new OGC API - Features has been implemented in order to provide a normalized and convenient way to do so (Clemens et al. 2019). It thus guarantees the consistency of the database keeping things secure and avoiding data mutation. Again, this could be done not through the database itself but thanks to the middleware. It is worth mentioning that not all information should be requested by everyone: the idea is to "see" what can be obtained. This must of course be done within the access limitation, security, hierarchy, etc.

The Figure 4.4 depicts how the OGC API service is connected to the architecture. While all the other well-separated applications are limited to their own part of the data (and to the shared parts), the OGC API – Features service has access to everything (under conditions of safety, regulations, etc.). It is important to clarify that this service is a read-only protocol and should be considered as a view on the stored model.

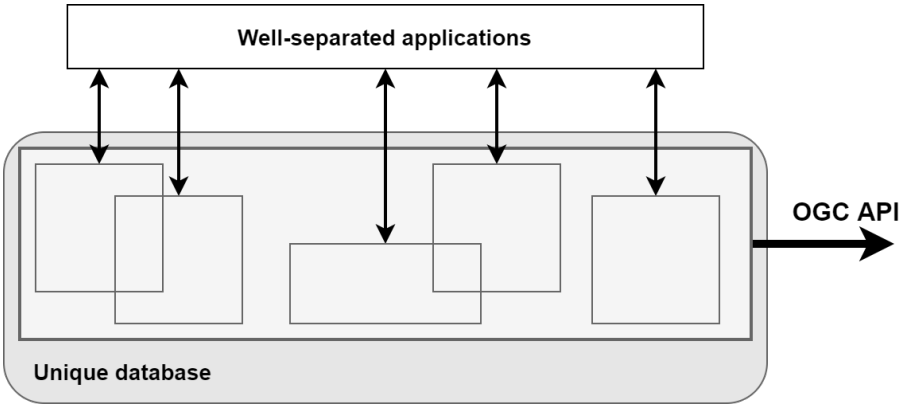

Figure 4.4: Implementation of the OGC API - Features service.

A problem arises from the fact that most of the exchange standards, protocols and OGC services (Web Features Service, Web Map Service, etc.) are suited for two-dimensional geodata. CityJSON, as a 3D modelling standard, cannot be queried in a convenient way using the standards with a few exceptions. Moreover, the document-oriented database does not support any 3D indexing methods. It was thus necessary to build workaround solutions in order to allow spatial filtering at least in two dimensions.

Initially, OGC standards serve features with a single geometry. However, a CityJSON object can have an undefined number of geometries. These geometries provide a wide range of information corresponding to various levels-of-detail. An alternative is proposed to limit misunderstandings. Besides the limit, offset and bounding boxes parameters already used in the specifications, a new attribute is added to the query parameters: the lod (level-of-detail). As a reminder, the geometries are stored independently of any feature as a bulk in a dedicated collection. The simplified schema separates them in several documents even if they concern the same object. This is justified by the fact that the level of detail plays an important and very specific role in urban information management but also because of the spatial indexing capabilities of the database. The lod is thus introduced as a parameter in its own right in requests. In any case, where this parameter is not supported by an application, it is simply neglected and it is the greatest *lod* that is served.

In the official schema of the *AbstractCityObject*, the *geographicalExtent* attribute stores the 3D boundary box of the distinct features. Projecting the 3D box, a new 2D attribute bbox is created on the fly during the storage process. This new spatial extent is then used to build the spatial indexing in the database. This extent is also created for the whole city model itself. An important condition is imposed by MongoDB or spatial indexing: the coordinate of the any spatial information should be expressed as a GeoJSON object (RFC 7946). It thus imposes the use of the World Geodetic System 1984 (WGS84 - EPSG:4326) and thus the projection of the bbox attribute. If no reference system is provided in the model, it is considered as already expressed in WGS84 by default. This is a major drawback for the management of spatial information in document-oriented databases.

Future work should include the improvements brought by the added parts of the OGC API – Features family: Coordinate Reference Systems by Reference, Filtering and the Common Query Language and the Simple Transactions. This should be handled by the middleware, as it is neither the responsibility of the database provider.

Second possible improvement coming from the semantic web, the Uniform Resource Identifier (URI) is a great candidate for the unique identifier format.

This URI identifier is a unique sequence of characters that identifies a logical or physical resource used by web technologies. While the purpose of the URI is to allow data extracted from various databases to be linked and to be identified unambiguously, it could also improve the management of the legitimacy of data. Such an identifier could be part of a certification process in which the responsibility of the city objects is part of the prerogatives of the city services. Its identity and its responsibility can thus be translated in the URI syntax in one way or another.

4.4 Conclusion

This chapter makes a step towards a shift for the storage of geographical information: it provides a technical component and insights that concedes a loss of consistency in favor of more flexibility. It is illustrated in the context of 3D city modelling thanks to the implementation of a schema validation middleware. This could be done performed, among other approaches, with the replacement of a relational database by a NoSQL document-oriented database. The main characteristic of the database considers that it does not handle any data schema. It therefore does not require filling in predefined boxes or meeting non-technical requirements as does relational. Conversely, the logical, conceptual and physical models are not prerequisites. The consistency management is then shifted to the server and more specifically to a filter layer: a schema validation middleware.

With a more focused view to this new architecture, the database can be considered as the principal foundation of a more complex whole: the database is unique but allows an undefined number of applications to retrieve information. A condition is however imposed in order to shift the consistency guarantee from the database to the server: exchanges should comply with a common standard. In the particular context of 3D city modelling, and keeping in mind the simplicity of use, applications and their exchanges should favor the CityJSON specifications.

Technically, the server filters all requests in both directions: from clients to the database and from the database to clients. This bi-directional filter allows storing and updating elements on the database by restricting them to predefined semantic information. The other way, it limits the information requested from the database depending on the users' right access, versions, etc. A view on the actual state of the database can be given thanks to the OGC API – Features exchange standard. Restricted to the read access, this view allows users to get generic information on the models elements. In summary, such type of filter

can be used in order to implement security layers, versioning and above all to enclose the users' possibilities.

The consistency counterbalanced by the middleware implementation will open many possibilities in application development and digital twining. City stakeholders should benefit from a single data store that can be shared across all their activities and responsibilities. Without any limitations or compromises made on previous storage capabilities, the city models will become real integrating bases for all the city services activities.

Back to the introduction, the answer to the first question on the relevancy of a new generation is nuanced. It is important to provide a new solution for the management of a "unique and digital 3D replica of a city" that improves the applications flexibility but the usage of the traditional solutions is not outdated. An ecosystem based on several solutions should provide a relevant answer. At the same time a document-oriented solution for its flexibility and accessibility, a knowledge graph solution for the support of contextualization and semantic both linked to a relational solution, which has already demonstrated its capabilities in handling spatial methods, should meet current requirements and tackle future needs. The product resulting from the fusion of these storage modes could ideally take advantage of the benefits of each while attempting to offset their disadvantages. Therefore, few compromises should be made considering them in the very beginning of the conception rather than providing partial solutions on a succession of choices. We believe that this contribution makes a step further towards such a hybrid architecture. Shifting the storage solution should then not be seen as a complete reverse but rather as a more global vision that would allow reaching a better management of what "digital twins" are intend to be.

Remaining challenges could also be divided in improvements specific to the middleware and improvements specific to the vision of a unique replica and its contextualization. Developments should concern the identity of a feature through its lifecycle (creation, modifications, etc.). The middleware and its various schemas could suffer from a lack of management added to the unicity of the stored information. A dedicated study thus need to be conducted on the optimized way to identify city models and their elements. While CityJSON features are commonly identified by UUID or GML_ID, the Uniform Resource Identifier is freer in its use. However, it should be considered as a very relevant solution since an URI can take whatever shape needed as long as it provides a means of locating (on the web, not spatial). One can for instance create a formatted URI translating the identity of the data provider. The data responsibility could then be established and both documentation and support could be released in a very convenient manner.

Since we considered GraphQL and SHACL as related works, specific access

methods could be developed to propose them as alternatives to our middleware and the OGC API services. Just like the latter, it would be normalized windows on the data stored by the provider no matter the users' habits. This consideration, alongside with the proposed usage of URIs, could lead to a hybrid storage solution based on document and graph oriented databases in which the identification of an object and its uniqueness would be guaranteed. We assume it will consist a good base to climb the Semantic Web Stacks: in our opinion, the principal requirement to reach the dreamed "Digital Twins". Finally, still with this objective in mind, data integration should also be one of the main future developments following this contribution. Regardless of the access method chosen, different levels of integration must be considered: Does the base model need to be enhanced? Should new attributes be created? Is this part of the data model's mission or should applications handle the integration themselves?

Chapter 5

Travelogue: usage history

This thesis is based on three articles published in international journals with a reading committee of academics. They follow rules of presentation, formalism, rule on points known to people familiar with the field, etc. This leads to a heavy and undoubtedly complicated reading. For an uninitiated reader, and even scholars, this can represent a difficulty and lead to missing the purpose of a thesis: presenting original work and ideas in an expert field and advertise it: in this case, providing an alternate viewpoint on 3D city modelling data management. Since the primary goal of a thesis is not to lose the reader but to provide a means of communication on an original reflection, this last chapter intends to give keys of understandings. The inter-texts between the different chapters (introduction, subsequent work, etc.) make it possible to contextualize their writing but do not take up their shared ideas.

However, even if the articulation between the chapters is trivial ranging from creation of a model through the accessibility of cities modelled through their storage, a lack of global narration affects its impact. This section focuses on the creation of this story. It is written in the same way as the preamble, which intends to provide people with the key to understanding: to use short, direct and simple declarations. Therefore, there will be no reference or use of tools and concepts related to other researches. We will keep it to illustrations and examples on this contribution and its benefits, drawbacks, improvements, etc.

In concrete terms, due to the advances in modelling, it is the third chapter on the database pseudo-schema which deserves an explanation. The fourth chapter is never more than the extension of this central chapter and the pooling of these principles. It is therefore more conceptual and difficult to illustrate and measure.

5.1 Building the database

As part of an illustration, it is interesting to test the hypotheses developed in the previous chapters using a real use case and propose short example of the new capabilities. even if examples have already been provided in the various chapters, repeating the explanations and providing additional information a few years later can only help the reader's understanding. The idea is not to make a marketing presentation of a solution either, but the money will still be approached quickly.

For this purpose, a generic model has been generated using real data for a neighbourhood in the city of Liège, Belgium. In this specific area, we find: the district of Hors-Chateau, the mountain of Buren as well as the surroundings of the hospital of the Citadel. There are buildings and large administrative

buildings, the Meuse (one of the main rivers in Belgium), roads, bridges and tunnels, as well as a forest and many isolated trees. This area has been chosen for its heterogeneity in city objects types.

Two different datasets were used: (1) PICC of the SPW ("Projet Informatique de Cartographie Continue", in French, "Continuous Mapping Computer Project", badly translated in English), which provides vector data on the inventory of Wallonia, Belgium; (2) the LiDAR campaign provided by the same office (see Chapter 2 for more information on point cloud, their density, etc.).

In the scope of the project, the modelled entities were limited to the buildings (for a total of 2777), the roads (total of 314), the relief (limited to a single Triangulated Irregular Network (TIN)) and isolated trees (total of 218) (see Figure 5.1). It counts a total of 3310 city features in a single CityJSON file (v. 1.1.3). This file size is 29.29 Mb and is valid under cjval 0.5.0. and val3dity 2.3.1. Speaking of model size, its equivalent stored in CityGML XML-encoding is 254.43 MB large (or 8.68 times its CityJSON version!). This model is the result of a master student project on developing the complete modelling pipe from the airborne point cloud to the final city features (the students were L. Paques, C. Petit, Q. van den Spiegel and C. Wilkin in 2022 - they gave their agreement for the use of the model built together).

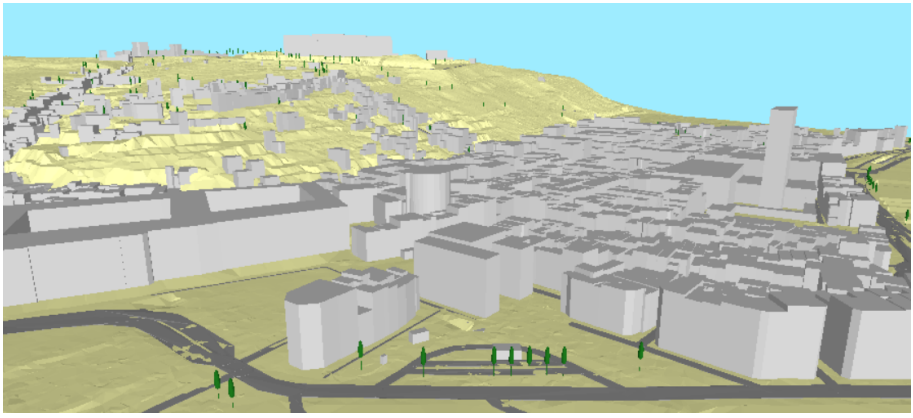

Figure 5.1: Illustration of the model generated in Liège, Belgium.

A few remarks are worth noting:

- Given that for the students, the goal was to provide the complete chain, it was necessary to limit the complexity of each link: buildings are depicted in LoD 1.2. and roads are not perfectly applied to the relief. Another

project uses Geoflow with enormous success to have the forms of roofs in addition and to rebuild larger models but we did not have the rights to share the results.

- The trees share a replicated template according to the CityJSON specifications (i.e., unlike CityGMLs ImplicitGeometries).
- The process was carried out on FME which is more ergonomic than coding the whole thing but limited to the software capabilities.

All these specifications allow to get an idea of what can be found in the illustrative file. Next step is to store it in various database solutions and compare them: both on storage size, in the current section, and then on accessibility in the next section.

First, both databases were installed empty. Using the 3DCityDB tools, the CityGML schema is imported on a PostgreSQL and tables, views, etc. were deployed. Considering the document-oriented schema installation, it is established on-the-fly during the application server start (Measur3D in this case) and might be updated by the application at every start. For the reminder, it allows updating it in a very convenient way given that the NoSQL database itself has no schema and accepts any shape of object. In line with Chapter 5, several servers (understand applications) can be installed using the same database due to this lack of imposed Caneva: everything is modular and open to change.

Once that the relational database structure has been set, importing the model can be made using the 3DCityDB tools. An importer/exporter allows people to handle their city models and provides integrated ways to manage their extensions. Afterwards, developers can query the features using classic SQL commands and PostGIS functions (the spatial extension of PostgreSQL).

Using Measur3D (the application developed in Chapter 4), the database is completely schema-less. It is the role of the server to handle the data structure and its accessibility. In other terms, in this case, the database is the slave and the server is the master. In this way, it is the server, and why not the client, who will be responsible for handling the information. The database, due to its BASE characteristics, is simply the backer of storage, and absolutely nothing else.

This point sets both solutions apart given that NoSQL oriented-document are limited in their spatial management. It is the major drawback of letting relational structure on the side: even if they are simple spatial functions in MongoDB, it is not yet ready for advanced spatial processing. For instance, spatial indexing allowed providing the OGC API Features filters on spatial

queries but advanced 3D processing is impossible within the scope of a MongoDB query. However, is that really a problem? The flexibility and modularity of applications can only be increased if the data is not constrained upstream. This is the whole point of Chapter 4. The access to the ThreeJS engine (the basis of the Measur3D viewer) already allows many analyses to be carried out (collisions, cut slices, etc.).

As it improves model usability and access, it is worth mentioning that both solutions, relational and NoSQL, should be seen as complementary. Given that the relational model is older and more standardized, it also means that it is stabler and more time has been put into developments and improvements. We note that, at the time of this writing, TUDelft is developing a relational solution for the support of CityJSON and its specificities. It will be important in the future to follow their work.

Focusing back on the illustration and what can be built with the tools presented in this document, this table summarises the necessary space for different state of the database in both solutions: relational solution using the 3DCityDB schema and a document-oriented solution using the schema provided in Chapter 4.

Given that the database and the data schema have an intrinsically linked shape (mainly the JSON-encoded city features that are stored as Binary JSON object in the database), these numbers are proportional to their file-equivalent. Most of the added necessary space is due to the indexes (alphanumeric and spatial).

Table 5.1: Average database size for various schema and database structure

PostgreSQL empty:	15.8 Mb
PostgreSQL + 3DCityDB empty:	24.27 Mb
PostgreSQL + 3DCityDB + Model:	280.68 Mb
MongoDB empty:	24.58 Kb
MongoDB + Model:	39.09 Mb
MongoDB + Model + compressed:	14.57 Mb

We remind the reader that the MongoDB + schema has no reason to be. While the relational database is focused on providing advanced functionalities and computations, the document-oriented solution provides a lighter storage and more accessible querying methods. We note that the last line (MongoDB + Model + compressed) is the actual size taken by the model as handled on the machine formatted in BSON (binary-encoded serialization of JSON-like documents).

These numbers count the size of the storage itself but also the indexes. However, even if one knows that a megabit represents nearly one million of bits, great numbers remain little perceptible and comparable. Another view, removing

the units, allows representing them relatively in a representation close to a Treemap (see Figure 5.2). A bit remains a bit in both solutions so things should be seen in perspective: it is not less than a factor of 20 which is in favour of the document-oriented solution. However, this occupied place comes with conditions.

Figure 5.2: Treemap-like of the needed storage size for PostgreSQL and MongoDB nodes.

In the previous chapters, the cost of things has not been discussed. A nevertheless important metric is the price represented by these different solutions. This point should not be addressed when comparing things from a functional and non-functional point of view as has been done in the previous chapters. However, whatever one may say, this is often how development projects in the industry are synthesized balanced with their risks. We therefore take advantage of this lighter chapter to discuss it.

In order to compare comparable elements, we decided to price both solution on a global cloud platform which saves costs compared to own hosting. By chance, our comparison was done on Microsoft's Azure because of our current professional position, but the other platforms are also quite valid: Amazon Web Services, Google Cloud, etc.

On the Azure platform, we choose Azure Cosmos DB, the fully managed NoSQL and relational database for modern app development, because both PostgreSQL and MongoDB nodes are available. Obviously, other solutions are possible in the wide range of services but Cosmos simple pre-sets also offer an easy modulable setup.

The node configuration is the following (these requirements are common for both relational and NoSQL document-oriented):

- Location: German West (Frankfurt, Germany).
- Single node without replications (no replication intra/extra region).
- No reservation, which means that we did not get savings on long subscriptions
- No specific service-level agreement (SLA) or uptime.

Prices were obtained the 1st April 2023 without any affiliation plan (and no savings on large-scale partnership). In the end, the price does not matter since both solutions have the same price no matter the storage size. But what matters here is that the ratio between the two solutions is therefore the same for the size of the storage as for the price. This point alone will be enough to convince most project managers and sponsors.

5.2 Installation and accessibility

Now that the database is installed, the question that arises is how can we best take advantage of the tools offered?

First, the entire code is openly available on GitHub:

Figure 5.3: Logo of the Measur3D application on GitHub.

https://ganys.github.io/Measur3D/

It includes the schema management application server, the client for data manipulation as well as the definition of schemas (outside the database for the reminder). People are invited to modify it, customize it, propose improvements,

etc. An installation guide is provided which allows running both backend (a basic server is provided and open doors to other developments are proposed) and frontend stacks (API calls for the management of city features, etc.). The only prerequisite is the access to a MongoDB instance, either in the cloud or local. All the techno-bricks (i.e., JavaScript libraries for the majority) are downloaded and updated via the installer.

In addition, once the backend has been installed and started, a Swagger API documentation is generated on-the-fly while parsing the different API routes and their capabilities automatically (see Figure 5.4): the OGC API - Features and Measur3D dedicated routes are thus defined in details in several formats (HTTP pages for humans, JSON objects for automated parsers, etc.). The access protocols and the pseudo-database schemas are given and open to any changes. Once again, people are invited to improve it and build their own application on top of it, at least on a technical level.

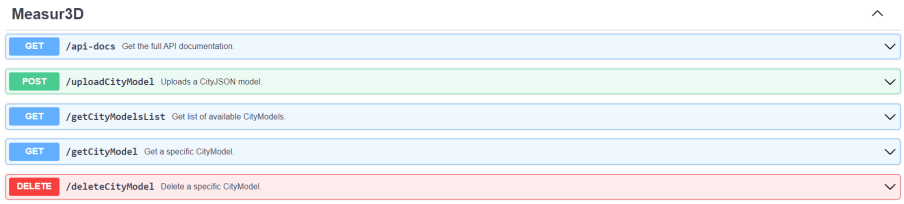

Figure 5.4: Extract of the Swagger API routes for Measur3D on GitHub.

Figure 5.5: Extract of the Swagger documentation for Measur3D on GitHub.

Parallel to this, the discussion around the data schema is not long: the data schema used is based on the CityGML data model which is itself well-known since 2012. Even if a third version has recently been published, the various developments all along this journey have rigorously followed the evolutions of CityJSON. Just as 3DCityDB follows the evolutions of CityGML, Measur3D

(or whatever will be its name in the future) should follow the improvements and modifications of CityJSON. There should not be any problem of retro-compatibility since Measur3D follows the changes of CityJSON, which itself guarantees such a compatibility. Regression tests should not be a big part of any new development or integration based on this. Be careful, however, that this is not a reason not to follow a coherent and well-thought-out testing strategy.

An improvement has been proposed though the use of mongoose and the middleware compared to the JSON-schema: objects can be defined as specialized version of one another. This represents an important difference with common JSON-schema which imposes a concurrent validation to specialise JSON objects: specialised objects then should to comply a set of schemas (allOf), any of a set of schemas (anyOf) or at least one of the objects in a set (oneOf) according to the JSON-Schema specification v2020-12. In our case, the improvement allows discriminating objects types one another. Therefore, an object can be defined as the refinement of an object but allowing updates on its previously defined attributes also. The Figure 5.6 presents the two methods of specializing objects: the JSON-schema on the left, which imposes independent conditions on an object, and on the right, our solution which allows updating the previous attributes and providing new ones. Once again, this point leads to easier maintenance of the model by taking advantage of an already widespread technical advance.

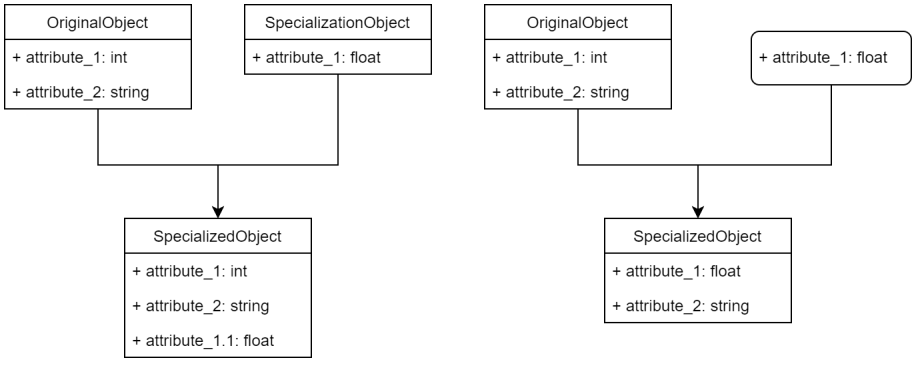

Figure 5.6: Specialization of an object in JSON-schema and middleware

In this picture, on the left for JSON-Schema, the SpecializedObject is the fusion of two objects definitions. And thus, the new provided attribute should be defined in another object. In the end, both attributes are valid but it imposed the coexistence (and therefore the renaming of the attribute) of attributes giving the same information but formatted differently. On the right, in the scope of discriminated objects, the update is simple applied on the original object.

The provided architecture should be seen as a sandbox giving people basic stacks for the building of their own application. It was used by students but also by us to illustrate basic principles and test hypotheses.

While there are many functions for deleting and getting the objects envelopes proposed in 3DCityDB (those are advanced functions that goes beyond asking the content of a table), it is surprising that there are no formalized views for common queries like getting an object, all its attributes and its geometries. There is no view, no procedure, no trigger nor materialized views after the 3DCityDB import. As a reminder, the idea here is to simplify access to information for as many developers as possible: there is no external admin table handled independently by MongoDB. At the end, for the end users, it does not change anything: all these points are DBA considerations but a simple view should improve the features accessibility without complexifying something that is already wide. This point demonstrates a certain frustration with the provision of a very complex model that does not take full advantage of the relational model and its functionality.

Here are examples of tables (set of elements in a relational SQL database) and collections (set of elements in a document-oriented database) creation for both solutions. The idea is to create a simple table that stores additional information on a *Building* (an object type that is already well defined in the data model).

Listing 5.1: SQL creation of a table

```
CREATE TABLE building_architect_information (
    id MEDIUMINT NOT NULL
        AUTO_INCREMENT,
    cityobject_id Number,
    style_type char(1),
    PRIMARY KEY (id)
)
```

Listing 5.2: SQL creation of an instance

```
INSERT INTO building_architect_information (
    cityobject_id, style_type)
VALUES ("abc123", 1);
```

We note here that the style types enumeration imposes to create another table that stores the various style types (contemporary, modern, etc.) if you want to limit the possible values to an enumeration. The style type attribute then stores the key of the desired type from this second table. Binding the *Building* information to the legacy cityobject table imposes also the creation of a foreign key to connect both tables. To then request the style information, all these

tables must be joined. We note that another solution could prefer to add a new attribute to the *cityobjects* directly using an "alter table" SQL command but it opens the possibility to add a style type to any city objects. Creating a new table allows increasing the number of additional information afterwards and keeping it away of the CityGML core model. It is also why the association goes from the *Building* information to the *Building* and not from the *cityobjects* that is defined as a *Building* to its information: limiting modification of the standard model to promote interoperability of the implementation.

For storing additional information to a *Building* in the document-oriented solution, we have nothing to do. The new attribute is simply added while creating the instance (or updating it) and the information will be added to the good document. All the other *Building* elements are not modified as this new attribute has not been set creating them.

Listing 5.3: MongoDB Query Language (MQL) updating of a document

```
citymodel.cityobjects.updateOne( { _id: "abc123", style:
    "modern" } )
```

Flexibility of this solution creates possibilities while limiting the added complexity.

Nevertheless, an illustrated example of a common query expressed in both solutions should demonstrate the specific "ease to use for developers". The scope here is not to propose a very complex query highlighting the completeness of the CityGML data model. For the sake of the demonstration and more in particular, keeping in mind the reader who is not a SQL expert nor a MongoDB aficionado, the first proposed query searches all *Buildings* and their attributes (address, function, etc). As a first step, geometries have been skipped. Here are other examples of simple queries on *"Building"* city objects from the above-mentioned city model stored on both a relational and NoSQL document-oriented solution:

Listing 5.4: Mongoose populate command example

```
SELECT
        co.id
FROM cityobject co
        JOIN building b ON b.id = co.id
        JOIN objectclass oc ON co.objectclass_id = oc.id
            AND oc.classname::text = 'Building'::text
```

This query took 54 milliseconds on average using pgadmin once the query buffer has been set. It took 300 milliseconds on first try on a "common" laptop on

which the databases are both running. The example above only returns the identifiers of city objects that are *"Buildings"*, nothing more. Its equivalent in MQL is written as follows:

Listing 5.5: Mongoose populate command example

```
citymodel.cityobjects.find( { type : 'Building' } )
```

This query took averagely 4 milliseconds on 100 repetitions on the same machine using MongoDB Compass, the equivalent of pgadmin for MongoDB. It is mainly because the language used follows the structure of the stored JSON and its embedded/nested elements. Moreover, the expressivity of the query is strengthened: it follows a cascading path and provide smart elements of filtering. In the end, getting the geometries can be achieved in a very convenient way using the populate mechanism of mongoose (the JavaScript library that allows handling MongoDB instances). Under the condition that a reference to another collection is provided within a document (which is the case in the proposed data model in Chapter 4), this mechanism nests elements inside each other using the path proposed in the request.

Listing 5.6: Mongoose populate command example

```
citymodel.cityobjects.find({type:'Building'}).populate('
    geometry')
```

In the same way a model can be reconstructed entirely using an embedded sets of objects and references. The mechanism then cascading populate the model: the model is getting the objects, the objects are getting their geometries, the geometries are getting their templates if relevant, etc. Since the elements are stored as they are accessed/formatted in the CityJSON file, it provides a very convenient query engine using the three collections from Chapter 4.

On a non-functional analysis, the NoSQL document-oriented solution, keeping in mind that the model accessibility and flexibility are the focus, is unbeatable. Afterwards, if a real comparison had to be made on a functional level, many efforts should still be invested in developing spatial capabilities. However, this is dependent of the MongoDB providers and get out the scope of this study. It was nevertheless necessary to specify it.

Chapter 6

Conclusion

This chapter reviews the different objectives and research questions that have been discussed in this document but also proposes some research extensions that could represent a logical continuation. Through this document, we tried to give answers to three research questions proposing new methods and models that better suits what we consider the future needs of users in 3D city modelling. It is proposed as a complementary solution that does not pretend to be the sole and mandatory solution. It is part of a trend without being its main spearhead. These successive explanations are given corresponding to the three tiers of a common web architecture. They are actually parts of a much bigger question that will be detailed and discussed at the end of this conclusion. The much bigger picture goes toward the implementation of a unique and faithful digital representation of the real cities and their assets. Such a Digital Twin intend to provide an integrating basis in which every city actors could find an added-value, an improved decision-making tool or simply a virtual representation of a tangible reality. The paradigm shift can be summarized in the following picture that encompasses the three improvements (see Figure 6.1). In short, the problem concerned the study of giving up a part of database consistency for the benefit of schema model flexibility and its report on another stack: the application server.

The current chapter, i.e. the conclusion, is the counterpart of the first, i.e. the introduction, and thus follows the same structure. It contextualizes the thesis in its writing context before developing the partial answers given to the four questions asked in the introduction. Some research extensions giving insights on what could come next are discussed in a chronological order.

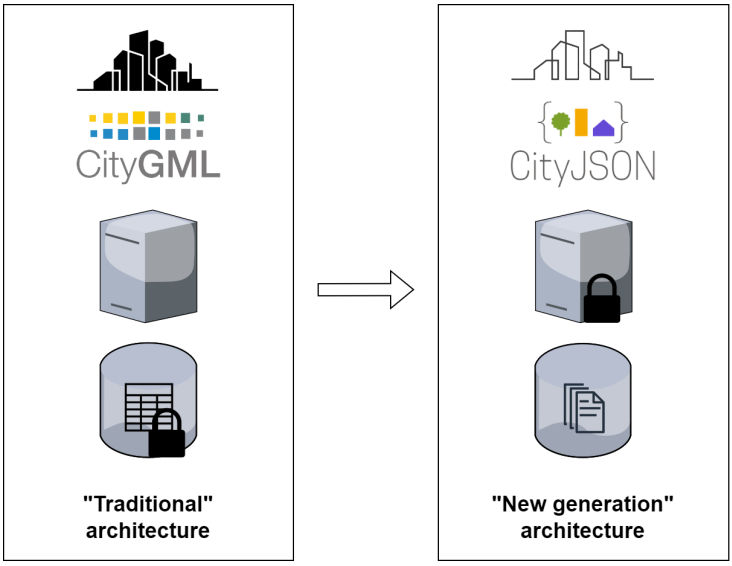

Figure 6.1: Illustration of the paradigm shift for the three-tier architecture.

6.1 Research questions

The main objective of this thesis is to propose a new generation of geographic information systems that provides increased schema flexibility for the CityJSON data it manages. However, it should not arm the data consistency that is the main advantage of using a standardized design model. The shift decomposes a traditional web architecture in its three constituting parts in order to tackle their data management in a flexible but still consistent way. The main purpose is to provide a solution that improves usability (i.e., ease of use and learning), interoperability (i.e., ease of exchanging through standardization of communications and storage processes) and maintenance (i.e., ease of managing the data all along its lifecycle).

It order to focus the developments, it has been summarized in a general research question as follows:

Could a flexible web architecture gather the consistency and interoperability aspect of the CityGML data model with better flexibility to cope with a high demand for sharing by applications from different domains?

In itself, the answer is obviously true: considering the added features and

capabilities of new technologies, it is obvious that things should be improved. Whether the architecture is based on new elements or uses old ones, any form of progress will always bring a gain of something: time, consumption of resources, money, etc. What is important to conclude is rather in the additional capabilities that are now possible.

The data consistency (i.e. the confidence that one can have in the information handled and exchanges by the architecture) is usually managed through database rules, relational associations, etc. However, it lacks of adaptability and resilience to modifications. In a dynamic that evolves quickly and aim attention at potential new needs, this mode of structuring has some flaws: we have listed them more or less exhaustively in chapter 3 on the database schema. The NoSQL databases are complementary solutions to relational databases, their tables, their associations and their hierarchies. For the reminder, as stated in the third chapter, the flattening of the CityGML schema thanks to the CityJSON encoding is its main improvement. It was thus important to propose a solution that enjoys this advance. It is indeed the main goal of the proposed architecture and the database schema. The contribution of a fourth use of the CityGML data model is itself a major step forward an increased use of document-oriented databases in 3D city modelling. We hope that this will also give a place to NoSQL in geographic information systems. This is not about performances and efficiency but rather about things made possible: metadata support, storage isomorphism, no joins, information filtering, etc.

The urban built environment offers a great showcase that gathers complex features, relationships between them, three-dimensional information, etc. The integration of heterogeneous information sources and their cross-checking illustrates the new abilities and allows understanding the differences between the two modes while making parallels with things that everyone knows (i.e. *Buildings*, *Roads*, etc.). For instance, since a *Building* is a *CityObjects* with an *address*. It illustrates objects hierarchy, their specifications and finally a discrete management by object. Moreover, the relations between the city elements and their semantic are easily understandable on a semantic level since it is a well-known environment for the most part. In a teaching and academic context, it offers a great sandbox for developments.

By taking up the elements, chapter by chapter, the thesis asks the following questions. The successive summaries reply to them and provide observations on their answers contributions.

By focusing on the geometric generation process, is it not possible to improve the accessibility, open the use and refine the design avoiding excessive model complexity? What are the clients' advantages of generated CityJSON models for sharing?

The light buildings generation process that follows from the second chapter develops new features of CityJSON: native support of metadata, refined levels of detail, etc. Providing a direct CityJSON model generation process simplify its storage in a dedicated storage solution since it does not impose its translation upstream. We take advantage of the simplicity of the model without having to complicate it before registration in a relational database. It was, at this time, a good generation process in terms of speed, accuracy and fidelity. However, in 2022, its results, speaking of performances, geometric errors and building's roof shapes complexity, is no more a good solution. The answer can thus be expressed as: it simplified and opened the usage of CityJSON models and thus improves their sharing but on a performance level, it is outdated.

Does moving from relational databases and their rigidity to document-oriented NoSQL databases and their flexibility open up data for modification and sharing? Is the consistency of the data at risk?

This chapter proposes the fourth usage of the CityGML data model: the definition of a document-oriented database schema. The modifications and the possibilities it has to be modified on the fly, thanks to its links with the CityJSON encoding, simplifies not only the users understanding but also their use. The proposed web architecture allows people developing their own applications in a convenient way constituting it with supplied stacks or modified versions of their own elements in an interoperable base. The schema flexibility benefit has been demonstrated in two specific applications: a cadaster of urban green infrastructure and a cadaster of the energy performance of buildings. The reply to the chapter query can be formulated as follows: the document-oriented databases suits the flattening of the data model proposed by CityJSON. It optimizes its storage and querying. The data consistency should then be shifted on the server stack in order to keep the confidence in the data management.

How to guarantee the consistency of the architecture when the database is now flexible? What happens if many users access the data from an access control point of view (versioning, hierarchy, security, etc.)?

The proposed solution relies on a middleware that filters queries and thus exchanges between application stacks. The core schema of the middleware is the CityJSON specifications but can be modified in such a way that handles both server and database consistency. Updates and modifications can easily be made and thus allow extending the schema in order to support new capabilities: versioning, hierarchy, etc. This chapter answers the question asked providing a solution to create a unique integrating basis for as many applications as possible. Under the condition that they all respect a set of predefined rules and filter their exchanges, they could connect to a single digital replica of real objects.

Such definition refers to a concept that have been exposed from the very first lines of this document: a *Digital Twin*. We believe that providing a flexible architecture, from the beginning of its creation, should facilitate and therefore improve the creation of such replicas. Proposals for research extensions should provide a clear vision of what we could expect after this thesis and what could be done in short or medium term.

These different explanations encompass a more ambitious answer that has been divided in the three chronological but also logical papers. Has it has been said in the beginning of this chapter: yes, flexibility can be improved without arming its consistency. However, this should be done in such a way that respect conditions. This point could be seen as contradictory with the notion of flexibility. Still, the flexibility we speak about in the thesis is the data model flexibility, not the architecture flexibility nor the possibility to create completely independent and erratic stacks. Developers must still work under requirements that respect basics.

Subsequent works and findings conclude each chapter and place the thesis in a much general context. It illustrates that this thesis is not isolated and does not result of the work of a sole researcher: it is in line with the state of the art and new users' needs. As researchers (i.e. people who are paid by public bodies), it is our duty to propose new solutions by taking the time to propose a new way of thinking. This comfort must be appreciated and used to its full potential given that the industry must always go for profit.

6.2 Research extensions

The potential of the architecture and the data model flexibility have already been showed in subsequent works of the third and fourth chapter. Extensions of the CityJSON core model are easily developed and tested, not only validating the extended schemas but also illustrating the new provided capabilities. Dedicated tools are added in the same way modularizing the viewer interface and the API. For instance, the expansion of the module support to *Dynamizers* required the creation of components that handle time series data and specific connections to sensors mock servers. This demonstrates that the interoperability and the usability of the provided architecture offers a great sandbox for the future works. We believe that its potential has however not been promoted or tested. Still, an important work on the architecture should be done and some mistakes need to be corrected (i.e. it is a good proof of concept but should remain humble on code quality). This point does not concern any great modification but rather maintenance and production.

The middleware should be improved in order to handle filters that are more detailed and complex. It has for now only be used to limit access writing and reading access of information of CityJSON features. Besides following the progress of the various OGC standards, a proposition could be made on a better integration of the SensorThings API (standard dedicated to sensors and Internet of Things) as was done for the *Dynamizers*. Moreover, all the supported standards of the *Features and Geometries JSON* working group could be integrated. It could be made exactly as we integrated the middleware (i.e. providing a different API route).

Enjoying the NodeJS server, its capabilities, its online repository for the publishing of open-source projets (i.e. npm), its important community and its improved flexibility, many improvements could be made in order to open its use. For instance, data analysis could be implemented on the server-side thanks to all these elements. Spatio-temporal reasoning and machine learning stacks could be added to the API capabilities and thus integrated in a much greater design. Once more, the architecture could be used as an integration basis to create improved decision-making tools.

The semantic web and its related standards (RDF, OWL, SPARQL, etc.) are bases of shared machine-readable but also human-readable interlinked data on the web: the Linked Open Data cloud. Every single concept defined in the LOD-cloud is identified by a unique key called Internationalized Resource Identifier. Even if city features are stored in a document-oriented database instead of a proper RDF graph-oriented one, links can be established by using these non-equivocal IRIs. These features references allow defining elements in a univocal way and reach a perfect crosschecking of information. Better than UUID, because of the semantic that these URL-like id carry, the IRI could provide information on the data owner, the version, etc. It should be, without a doubt, a constituting technological stack to implement such a LOD-proof Digital Twin. By going further, but really far this time, we can dream and imagine integrating all this in a blockchain and reach the utopian “truth”. Keeping things simple, the idea is to provide a shared solution to guarantee the storage of shared information (agreeing and sharing information is called "truth" in web semantic). Semantic of the 3D city models can thus be trusted thanks to the blockchain verifications and its sharing is guaranteed by the RDF/IRI solutions.

These last points might seem fuzzy or unrealizable but we believe they will be partial answers to what we expect as future needs. This represent as comforting point since all these technologies already exist, at least in early stages.

Bibliography

Agrawal, S. and R. D. Gupta (Dec. 2017). "Web GIS and its architecture: a review". en. In: *Arabian Journal of Geosciences* 10.23, p. 518. ISSN: 1866-7511, 1866-7538. DOI: 10.1007/s12517-017-3296-2.

Aleksandrov, M., A. Diakité, J. Yan, W. Li, and S. Zlatanova (Sept. 30, 2019). "SYSTEMS ARCHITECTURE FOR MANAGEMENT OF BIM, 3D GIS AND SENSORS DATA". en. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-4/W9, pp. 3–10. ISSN: 2194-9050. DOI: 10.5194/isprs-annals-IV-4-W9-3-2019.

Allah Bukhsh, Z., M. van Sinderen, and P. M. Singh (2015). "SOA and EDA: A Comparative Study - Similarities, Differences and Conceptual Guidelines on their Usage". In: [Online; accessed 2019-01-10]. International Conference on e-Business. Colmar, Alsace, France: SCITEPRESS - Science, pp. 213–220. ISBN: 978-989-758-113-7. DOI: 10.5220/0005539802130220.

Awrangjeb, M., S. Gilani, and F. Siddiqui (Sept. 21, 2018). "An Effective Data-Driven Method for 3D Building Roof Reconstruction and Robust Change Detection". en. In: *Remote Sensing* 10.10, p. 1512. ISSN: 2072-4292. DOI: 10.3390/rs10101512.

Ballard, D. (1987). "GENERALIZING THE HOUGH TRANSFORM TO DETECT ARBITRARY SHAPES". en. In: *Readings in Computer Vision*. DOI: 10.1016/B978-0-08-051581-6.50069-6. Elsevier, pp. 714–725. ISBN: 978-0-08-051581-6.

Balta, H., J. Velagic, W. Bosschaerts, G. De Cubber, and B. Siciliano (2018). "Fast Statistical Outlier Removal Based Method for Large 3D Point Clouds of Outdoor Environments". en. In: *IFAC-PapersOnLine* 51.22, pp. 348–353. ISSN: 24058963. DOI: 10.1016/j.ifacol.2018.11.566.

Baralis, E., A. Dalla Valle, P. Garza, C. Rossi, and F. Scullino (Dec. 2017). "SQL versus NoSQL databases for geospatial applications". In: [Online; accessed

2018-06-27]. IEEE, pp. 3388–3397. ISBN: 978-1-5386-2715-0. DOI: 10.1109/BigData.2017.8258324.

Bartoszewski, D., A. Piorkowski, and M. Lupa (2019). "The Comparison of Processing Efficiency of Spatial Data for PostGIS and MongoDB Databases". en. In: *Beyond Databases, Architectures and Structures. Paving the Road to Smart Data Processing and Analysis*. Ed. by S. Kozielski, D. Mrozek, P. Kasprowski, B. Malysiak-Mrozek, and D. Kostrzewa. Vol. 1018. Cham: Springer International Publishing, pp. 291–302. ISBN: 978-3-030-19092-7.

Berger, M., A. Tagliasacchi, L. M. Seversky, P. Alliez, G. Guennebaud, J. A. Levine, A. Sharf, and C. T. Silva (Jan. 2017). "A Survey of Surface Reconstruction from Point Clouds". In: *Computer Graphics Forum* 36.1, pp. 301–329. ISSN: 14678659. DOI: 10.1111/cgf.12802.

Biljecki, F., H. Ledoux, X. Du, J. Stoter, K. H. Soon, and V. H. S. Khoo (Oct. 5, 2016). "The most common geometric and semantic errors in CityGML datasets". en. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-2/W1, pp. 13–22. ISSN: 2194-9050. DOI: 10.5194/isprs-annals-IV-2-W1-13-2016.

Biljecki, F., K. Kumar, and C. Nagel (Dec. 2018). "CityGML Application Domain Extension (ADE): overview of developments". en. In: *Open Geospatial Data, Software and Standards* 3.1, p. 13. ISSN: 2363-7501. DOI: 10.1186/s40965-018-0055-6.

Biljecki, F., H. Ledoux, and J. Stoter (Sept. 2016). "An improved LOD specification for 3D building models". en. In: *Computers, Environment and Urban Systems* 59, pp. 25–37. ISSN: 01989715. DOI: 10.1016/j.compenvurbsys.2016.04.005.

Biljecki, F., H. Ledoux, J. Stoter, and J. Zhao (Nov. 2014). "Formalisation of the level of detail in 3D city modelling". en. In: *Computers, Environment and Urban Systems* 48, pp. 1–15. ISSN: 01989715. DOI: 10.1016/j.compenvurbsys.2014.05.004.

Biljecki, F., J. Lim, J. Crawford, D. Moraru, H. Tauscher, A. Konde, K. Adouane, S. Lawrence, P. Janssen, and R. Stouffs (Jan. 2021). "Extending CityGML for IFC-sourced 3D city models". en. In: *Automation in Construction* 121, p. 103440. ISSN: 09265805. DOI: 10.1016/j.autcon.2020.103440.

Biljecki, F., J. Stoter, H. Ledoux, S. Zlatanova, and A. Çöltekin (Dec. 18, 2015). "Applications of 3D City Models: State of the Art Review". en. In: *ISPRS International Journal of Geo-Information* 4.4, pp. 2842–2889. ISSN: 2220-9964. DOI: 10.3390/ijgi4042842.

Billen, R., A.-F. Cutting-Decelle, O. Marina, J.-P. de Almeida, C. M., G. Falquet, T. Leduc, C. Métral, G. Moreau, J. Perret, G. Rabin, R. San Jose, I. Yatskiv, and S. Zlatanova (2014). "3D City Models and urban information: Current issues and perspectives: European COST Action TU0801". In: ed. by R. Billen, A.-F. Cutting-Decelle, O. Marina, J.-P. de Almeida, C. M., G. Falquet, T. Leduc, C. Métral, G. Moreau, J. Perret, G. Rabin, R. San Jose, I. Yatskiv, and S. Zlatanova. [Online; accessed 2019-03-08]. 3D City Models, urban information: Current issues, and perspectives – European COST Action TU0801. Liège, Belgium: EDP Sciences, pp. I–118. ISBN: 978-2-7598-1153-3. DOI: 10.1051/TU0801/201400001.

Billen, R., B. Jonlet, A. L. Jancsó, R. Neuville, G.-A. Nys, F. Poux, M. V. Ruymbeke, M. Piavaux, and P. Hallot (2018). "La transition numérique dans le domaine du patrimoine bâti: un retour d'expériences". In: *Bulletin de la Commission royale des Monuments, Sites et Fouilles* 30, pp. 119–146.

Blaschke, T., G. J. Hay, Q. Weng, and B. Resch (Aug. 19, 2011). "Collective Sensing: Integrating Geospatial Technologies to Understand Urban Systems–An Overview". en. In: *Remote Sensing* 3.8, pp. 1743–1776. ISSN: 2072-4292. DOI: 10.3390/rs3081743.

Boaventura Filho, W., H. V. Olivera, M. Holanda, and A. Favacho (Oct. 2016). "Geographic Data Modelling for NoSQL Document-Oriented Databases". In: GEOProcessing 2015 : the Seventh International Conference on Advanced Geographic Information System, Applications and Services. Lisbon, Portugal.

Borrmann, D., J. Elseberg, K. Lingemann, and A. Nüchter (2010). "A Data Structure for the 3D Hough Transform for Plane Detection". en. In: *IFAC Proceedings Volumes* 43.16, pp. 49–54. ISSN: 14746670. DOI: 10.3182/20100906-3-IT-2019.00011.

Breunig, M., P. V. Kuper, E. Butwilowski, A. Thomsen, M. Jahn, A. Dittrich, M. Al-Doori, D. Golovko, and M. Menninghaus (July 2016). "The story of DB4GeO – A service-based geo-database architecture to support multi-dimensional data analysis and visualization". en. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 117, pp. 187–205. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2015.12.006.

Brewer, E. A. (2000). "Towards robust distributed systems". In: *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing - PODC '00*. the nineteenth annual ACM symposium. Portland, Oregon, United States: ACM Press, p. 7. ISBN: 978-1-58113-183-3. DOI: 10.1145/343477.343502.

Brito, G., T. Mombach, and M. T. Valente (Feb. 2019). "Migrating to GraphQL: A Practical Assessment". In: [Online; accessed 2022-05-20]. 2019 IEEE 26th

International Conference on Software Analysis, Evolution and Reengineering (SANER). Hangzhou, China: IEEE, pp. 140–150. ISBN: 978-1-72810-591-8. DOI: 10.1109/SANER.2019.8667986.

Cao, R., Y. Zhang, X. Liu, and Z. Zhao (July 3, 2017). "3D building roof reconstruction from airborne LiDAR point clouds: a framework based on a spatial database". en. In: *International Journal of Geographical Information Science* 31.7, pp. 1359–1380. ISSN: 1365-8816, 1362-3087. DOI: 10.1080/13658816.2017.1301456.

Cha, S. K., K. H. Kim, C. B. Song, J. K. Kim, and Y. S. Kwon (1999). "A Middleware Architecture for Transparent Access to Multiple Spatial Object Databases". In: *Interoperating Geographic Information Systems*. Ed. by M. Goodchild, M. Egenhofer, R. Fegeas, and C. Kottman. Boston, MA: Springer US, pp. 267–282. ISBN: 978-1-4613-7363-6.

Chadzynski, A., N. Krdzavac, F. Farazi, M. Q. Lim, S. Li, A. Grisiute, P. Herthogs, A. von Richthofen, S. Cairns, and M. Kraft (Dec. 2021). "Semantic 3D City Database - An enabler for a dynamic geospatial knowledge graph". en. In: *Energy and AI* 6, p. 100106. ISSN: 26665468. DOI: 10.1016/j.egyai.2021.100106.

Chasseur, C., Y. Li, and J. Patel (June 2013). "Enabling JSON Document Stores in Relational Systems". In: WebDB 2013. New York, NY, USA.

Chaturvedi, K. and T. H. Kolbe (2015). "Dynamizers - Modeling and Implementing Dynamic Properties for Semantic 3D City Models". In: The Eurographics Association. DOI: 10.2312/udmv.20151348.

Chaturvedi, K., C. S. Smyth, G. Gesquière, T. Kutzner, and T. H. Kolbe (2017). "Managing Versions and History Within Semantic 3D City Models for the Next Generation of CityGML". In: *Advances in 3D Geoinformation*. Ed. by A. Abdul-Rahman. Cham: Springer International Publishing, pp. 191–206. ISBN: 978-3-319-25689-4.

Clemens, P., V. Panagiotis, and H. Charles (Oct. 2019). *OGC API - Feature - Part 1: Core*.

Corman, J., J. L. Reutter, and O. Savković (2018). "Semantics and Validation of Recursive SHACL". In: *The Semantic Web – ISWC 2018*. Ed. by D. Vrandečić, K. Bontcheva, M. C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L.-A. Kaffee, and E. Simperl. Vol. 11136. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 318–336. ISBN: 978-3-030-00670-9 978-3-030-00671-6. DOI: 10.1007/978-3-030-00671-6_19.

Costa Rainho, F. da and J. Bernardino (June 2018). "Web GIS: A new system to store spatial data using GeoJSON in MongoDB". In: [Online; accessed 2019-04-18]. 2018 13th Iberian Conference on Information Systems and Technologies (CISTI). Caceres: IEEE, pp. 1–6. ISBN: 978-989-98434-8-6. DOI: 10.23919/CISTI.2018.8399279.

Debruyne, C. and K. McGlinn (June 7, 2021). "Reusable SHACL Constraint Components for Validating Geospatial Linked Data". In: *Reusable SHACL Constraint Components for Validating Geospatial Linked Data*. CEUR.

Deininger, M. E., M. von der Grün, R. Piepereit, S. Schneider, T. Santhanavanich, V. Coors, and U. Voß (Oct. 31, 2020). "A Continuous, Semi-Automated Workflow: From 3D City Models with Geometric Optimization and CFD Simulations to Visualization of Wind in an Urban Environment". en. In: *ISPRS International Journal of Geo-Information* 9.11, p. 657. ISSN: 2220-9964. DOI: 10.3390/ijgi9110657.

Diogo, M., B. Cabral, and J. Bernardino (Feb. 14, 2019). "Consistency Models of NoSQL Databases". en. In: *Future Internet* 11.2, p. 43. ISSN: 1999-5903. DOI: 10.3390/fi11020043.

Doboš, J. and A. Steed (2012). "3D revision control framework". en. In: [Online; accessed 2019-07-09]. the 17th International Conference. Los Angeles, California: ACM Press, p. 121. ISBN: 978-1-4503-1432-9. DOI: 10.1145/2338714.2338736.

Dorninger, P. and N. Pfeifer (Nov. 17, 2008). "A Comprehensive Automated 3D Approach for Building Extraction, Reconstruction, and Regularization from Airborne Laser Scanning Point Clouds". en. In: *Sensors* 8.11, pp. 7323–7343. ISSN: 1424-8220. DOI: 10.3390/s8117323.

Đuric, M. (June 7, 2018). "GEOPORTAL FOR SEARCHING AND VISUALIZATION OF CADASTRAL DATA". In: 13.1. [Online; accessed 2020-11-05]. ISSN: 2566-4484. DOI: 10.7251/STP1813687A.

El Yamani, S., R. Hajji, G.-A. Nys, M. Ettarid, and R. Billen (Mar. 5, 2021). "3D Variables Requirements for Property Valuation Modeling Based on the Integration of BIM and CIM". In: *Sustainability* 13.5, p. 2814. ISSN: 2071-1050. DOI: 10.3390/su13052814.

Ellul, C., J. Stoter, and B. Bucher (Oct. 2022). "Location-enabled Digital Twins – Understanding the Role of NMCAS in a European Context". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* X-4/W2-2022, pp. 53–60. DOI: 10.5194/isprs-annals-X-4-W2-2022-53-2022.

Ennafii, O., A. L. Bris, F. Lafarge, and C. Mallet (2018). "Semantic evaluation of 3D city models". en. In: *Unpublished*. [Online; accessed 2019-03-05]. DOI: 10.13140/rg.2.2.15922.45765.

Ennafii, O., A. Le Bris, F. Lafarge, and C. Mallet (Dec. 1, 2019). "A Learning Approach to Evaluate the Quality of 3D City Models". en. In: *Photogrammetric Engineering & Remote Sensing* 85.12, pp. 865–878. ISSN: 0099-1112. DOI: 10.14358/PERS.85.12.865.

Eriksson, H. and L. Harrie (Jan. 28, 2021). "Versioning of 3D City Models for Municipality Applications: Needs, Obstacles and Recommendations". en. In: *ISPRS International Journal of Geo-Information* 10.2, p. 55. ISSN: 2220-9964. DOI: 10.3390/ijgi10020055.

Eriksson, H., J. Sun, V. Tarandi, and L. Harrie (Feb. 2021). "Comparison of versioning methods to improve the information flow in the planning and building processes". en. In: *Transactions in GIS* 25.1, pp. 134–163. ISSN: 1361-1682, 1467-9671. DOI: 10.1111/tgis.12672.

Fan, H. and L. Meng (June 2012). "A three-step approach of simplifying 3D buildings modeled by CityGML". en. In: *International Journal of Geographical Information Science* 26.6, pp. 1091–1107. ISSN: 1365-8816, 1362-3087. DOI: 10.1080/13658816.2011.625947.

Floros, G. and E. Dimopoulou (Oct. 5, 2016). "Investigating the enrichment of a 3D city model with various CityGML modules". en. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2/W2, pp. 3–9. ISSN: 2194-9034. DOI: 10.5194/isprs-archives-XLII-2-W2-3-2016.

Fricke, A., J. Döllner, and H. Asche (2018). "Servicification – Trend or Paradigm Shift in Geospatial Data Processing?" In: *Computational Science and Its Applications – ICCSA 2018*. Ed. by O. Gervasi, B. Murgante, S. Misra, E. Stankova, C. M. Torre, A. M. A. Rocha, D. Taniar, B. O. Apduhan, E. Tarantino, and Y. Ryu. Vol. 10962. Cham: Springer International Publishing, pp. 339–350. ISBN: 978-3-319-95167-6.

Gómez, P., C. Roncancio, and R. Casallas (July 2021). "Analysis and evaluation of document-oriented structures". en. In: *Data & Knowledge Engineering* 134, p. 101893. ISSN: 0169023X. DOI: 10.1016/j.datak.2021.101893.

Gröger, G. and L. Plümer (July 2012). "CityGML – Interoperable semantic 3D city models". en. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 71, pp. 12–33. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2012.04.004.

Haas, L., R. J. Miller, B. Niswonger, M. Roth, P. Schwarz, and E. Wimmers (Mar. 1999). "Transforming heterogeneous data with database middleware: beyond integration". In: *IEEE Data Engineering Bulletin* 22, pp. 31–36.

Hachenberger, P., L. Kettner, and K. Mehlhorn (Sept. 2007). "Boolean operations on 3D selective Nef complexes: Data structure, algorithms, optimized implementation and experiments". en. In: *Computational Geometry* 38.1-2, pp. 64–99. ISSN: 09257721. DOI: 10.1016/j.comgeo.2006.11.009.

Hartig, O. and J. Pérez (2018). "Semantics and Complexity of GraphQL". en. In: [Online; accessed 2022-05-20]. the 2018 World Wide Web Conference. Lyon, France: ACM Press, pp. 1155–1164. ISBN: 978-1-4503-5639-8. DOI: 10.1145/3178876.3186014.

Haughian, G., R. Osman, and W. J. Knottenbelt (2016). "Benchmarking Replication in Cassandra and MongoDB NoSQL Datastores". In: *Database and Expert Systems Applications*. Ed. by S. Hartmann and H. Ma. Vol. 9828. Cham: Springer International Publishing, pp. 152–166. ISBN: 978-3-319-44405-5.

Henn, A., G. Gröger, V. Stroh, and L. Plümer (Feb. 2013). "Model driven reconstruction of roofs from sparse LIDAR point clouds". en. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 76, pp. 17–29. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2012.11.004.

Herrmann, K., H. Voigt, A. Behrend, J. Rausch, and W. Lehner (May 9, 2017). "Living in Parallel Realities – Co-Existing Schema Versions with a Bidirectional Database Evolution Language". In: *Proceedings of the 2017 ACM International Conference on Management of Data*. arXiv: 1608.05564, pp. 1101–1116. DOI: 10.1145/3035918.3064046.

Herrmann, K., H. Voigt, J. Rausch, A. Behrend, and W. Lehner (Feb. 2018). "Robust and simple database evolution". en. In: *Information Systems Frontiers* 20.1, pp. 45–61. ISSN: 1387-3326, 1572-9419. DOI: 10.1007/s10796-016-9730-2.

Holemans, A., J.-P. Kasprzyk, and J.-P. Donnay (2018). "Coupling an Unstructured NoSQL Database with a Geographic Information System". In: The Tenth International Conference on Advanced Geographic Information. Rome, Italy, pp. 23–28.

Hu, P., B. Yang, Z. Dong, P. Yuan, R. Huang, H. Fan, and X. Sun (July 17, 2018). "Towards Reconstructing 3D Buildings from ALS Data Based on Gestalt Laws". en. In: *Remote Sensing* 10.7, p. 1127. ISSN: 2072-4292. DOI: 10.3390/rs10071127.

Huang, H., C. Brenner, and M. Sester (May 2013). "A generative statistical approach to automatic 3D building roof reconstruction from laser scanning data". en. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 79, pp. 29–43. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2013.02.004.

Huang, H. and H. Mayer (2017). "Towards Automatic Large-Scale 3D Building Reconstruction: Primitive Decomposition and Assembly". In: *Societal Geo-innovation*. Ed. by A. Bregt, T. Sarjakoski, R. van Lammeren, and F. Rip. Cham: Springer International Publishing, pp. 205–221. ISBN: 978-3-319-56758-7.

Huang, W., S. A. Raza, O. Mirzov, and L. Harrie (July 19, 2019). "Assessment and Benchmarking of Spatially Enabled RDF Stores for the Next Generation of Spatial Data Infrastructure". In: *ISPRS International Journal of Geo-Information* 8.7, p. 310. ISSN: 2220-9964. DOI: 10.3390/ijgi8070310.

International, E. (June 2015). *ECMAScript 2015*. Tech. rep. Geneva, Switzerland.

Joshi, M. Y., W. Selmi, M. Binard, G.-A. Nys, and J. Teller (Sept. 15, 2020). "POTENTIAL FOR URBAN GREENING WITH GREEN ROOFS: A WAY TOWARDS SMART CITIES". en. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* VI-4/W2-2020, pp. 87–94. ISSN: 2194-9050. DOI: 10.5194/isprs-annals-VI-4-W2-2020-87-2020.

Jung, J., Y. Jwa, and G. Sohn (Mar. 19, 2017). "Implicit Regularization for Reconstructing 3D Building Rooftop Models Using Airborne LiDAR Data". en. In: *Sensors* 17.3, p. 621. ISSN: 1424-8220. DOI: 10.3390/s17030621.

Jung, J. and G. Sohn (Mar. 2019). "A line-based progressive refinement of 3D rooftop models using airborne LiDAR data with single view imagery". en. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 149, pp. 157–175. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2019.01.003.

Kada, M. and L. McKinley (Sept. 2009). "3D Building reconstruction from LiDAR based on a cell decomposition approach". In: vol. XXXVIII. City Models, Roads and Traffic 09. Paris, France: International Archives of Photogrammetry and Remote Sensing, pp. 47–52.

Knublauch, H. and D. Kontokostas (July 2017). *Shapes constraint language (SHACL)*. W3C.

Kulawiak, M., A. Dawidowicz, and M. E. Pacholczyk (Aug. 2019). "Analysis of server-side and client-side Web-GIS data processing methods on the example of JTS and JSTS using open data from OSM and geoportal". en. In: *Computers & Geosciences* 129, pp. 26–37. ISSN: 00983004. DOI: 10.1016/j.cageo.2019.04.011.

Kumar, K., H. Ledoux, and J. Stoter (Sept. 12, 2018). "Dynamic 3D Visualization of Floods: Case of the Netherlands". en. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-4/W10, pp. 83–87. ISSN: 2194-9034. DOI: 10.5194/isprs-archives-XLII-4-W10-83-2018.

Kumar, K., A. Labetski, K. A. Ohori, H. Ledoux, and J. Stoter (Dec. 2019). "The LandInfra standard and its role in solving the BIM-GIS quagmire". en. In: *Open Geospatial Data, Software and Standards* 4.1, p. 5. ISSN: 2363-7501. DOI: 10.1186/s40965-019-0065-z.

Kutzner, T., K. Chaturvedi, and T. H. Kolbe (Feb. 26, 2020). "CityGML 3.0: New Functions Open Up New Applications". en. In: *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*. [Online; accessed 2020-02-28]. ISSN: 2512-2789, 2512-2819. DOI: 10.1007/s41064-020-00095-z.

Labetski, A., K. Kumar, H. Ledoux, and J. Stoter (Dec. 2018). "A metadata ADE for CityGML". en. In: *Open Geospatial Data, Software and Standards* 3.1, p. 16. ISSN: 2363-7501. DOI: 10.1186/s40965-018-0057-4.

Lafarge, F. and C. Mallet (Aug. 2012). "Creating Large-Scale City Models from 3D-Point Clouds: A Robust Approach with Hybrid Representation". en. In: *International Journal of Computer Vision* 99.1, pp. 69–85. ISSN: 0920-5691, 1573-1405. DOI: 10.1007/s11263-012-0517-8.

Laksono, D. (Aug. 2018). "Testing Spatial Data Deliverance in SQL and NoSQL Database Using NodeJS Fullstack Web App". In: [Online; accessed 2020-11-05]. 2018 4th International Conference on Science and Technology (ICST). Yogyakarta: IEEE, pp. 1–5. ISBN: 978-1-5386-5813-0. DOI: 10.1109/ICSTC.2018.8528705.

Ledoux, H. (Dec. 2018). "val3dity: validation of 3D GIS primitives according to the international standards". en. In: *Open Geospatial Data, Software and Standards* 3.1, p. 1. ISSN: 2363-7501. DOI: 10.1186/s40965-018-0043-x.

Ledoux, H., F. Biljecki, B. Dukai, K. Kumar, R. Peters, J. Stoter, and T. Commandeur (Jan. 26, 2021). "3dfier: automatic reconstruction of 3D city models". In: *Journal of Open Source Software* 6.57, p. 2866. ISSN: 2475-9066. DOI: 10.21105/joss.02866.

Ledoux, H., K. A. Ohori, K. Kumar, B. Dukai, A. Labetski, and S. Vitalis (Feb. 25, 2019). "CityJSON: A compact and easy-to-use encoding of the CityGML data model". In: *arXiv:1902.09155 [cs]*. arXiv: 1902.09155.

Li, D., W. Chen, M. Pan, H. Li, H. Liu, and Y. Tang (2018). "DBHUB: A Lightweight Middleware for Accessing Heterogeneous Database Systems".

In: *Cloud Computing and Security*. Ed. by X. Sun, Z. Pan, and E. Bertino. Vol. 11063. Cham: Springer International Publishing, pp. 408–419. ISBN: 978-3-030-00005-9.

Li, L., F. Yang, H. Zhu, D. Li, Y. Li, and L. Tang (May 3, 2017). "An Improved RANSAC for 3D Point Cloud Plane Segmentation Based on Normal Distribution Transformation Cells". en. In: *Remote Sensing* 9.5, p. 433. ISSN: 2072-4292. DOI: 10.3390/rs9050433.

Liang, S., C.-Y. Huang, and T. Khalafbeigi (2016). *SensorThings API - Part 1: Sensing*.

Liang, S. and T. Khalafbeigi (2019). *SensorThings API - Part 2: Tasking Core*.

Lim, J., P. Janssen, and F. Biljecki (Sept. 3, 2020). "VISUALISING DETAILED CITYGML AND ADE AT THE BUILDING SCALE". en. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLIV-4/W1-2020, pp. 83–90. ISSN: 2194-9034. DOI: 10.5194/isprs-archives-XLIV-4-W1-2020-83-2020.

Liu, X., Y. Zhang, X. Ling, Y. Wan, L. Liu, and Q. Li (June 8, 2019). "TopoLAP: Topology Recovery for Building Reconstruction by Deducing the Relationships between Linear and Planar Primitives". en. In: *Remote Sensing* 11.11, p. 1372. ISSN: 2072-4292. DOI: 10.3390/rs11111372.

Liu, Z. H., B. Hammerschmidt, and D. McMahon (2014). "JSON data management: supporting schema-less development in RDBMS". en. In: [Online; accessed 2020-07-15]. the 2014 ACM SIGMOD international conference. Snowbird, Utah, USA: ACM Press, pp. 1247–1258. ISBN: 978-1-4503-2376-5. DOI: 10.1145/2588555.2595628.

Lopez, M., S. Couturier, and J. Lopez (Dec. 2016). "Integration of NoSQL Databases for Analyzing Spatial Information in Geographic Information System". In: [Online; accessed 2019-11-12]. 2016 8th International Conference on Computational Intelligence and Communication Networks (CICN). Tehri, India: IEEE, pp. 351–355. ISBN: 978-1-5090-1144-5. DOI: 10.1109/CICN.2016.75.

Makris, A., K. Tserpes, and D. Anagnostopoulos (2019). "Performance Evaluation of MongoDB and PostgreSQL for spatio-temporal data". In: Workshops of the EDBT/ICDT 2019 Joint Conference. Lisbon, Portugal: CEUR-WS.org, p. 8.

Malinverni, E. S., B. Naticchia, J. L. Lerma Garcia, A. Gorreja, J. Lopez Uriarte, and F. Di Stefano (Aug. 24, 2020). "A semantic graph database for the interoperability of 3D GIS data". en. In: *Applied Geomatics*. [Online; accessed

2021-08-30]. ISSN: 1866-9298, 1866-928X. DOI: 10.1007/s12518-020-00334-3.

Mao, B. and L. Harrie (Oct. 10, 2016). "Methodology for the Efficient Progressive Distribution and Visualization of 3D Building Objects". en. In: *ISPRS International Journal of Geo-Information* 5.10, p. 185. ISSN: 2220-9964. DOI: 10.3390/ijgi5100185.

Mei Qi, L., W. Xiaonan, I. Oliver R., and K. Markus (2021). *The World Avatar - a world model for facilitating interoperability.* Tech. rep.

Milde, J., Y. Zhang, C. Brenner, L. Plümer, and M. Sester (Jan. 2008). "Building reconstruction using a structural description based on a formal grammar". In.

Mobasheri, A., H. Mitasova, M. Neteler, A. Singleton, H. Ledoux, and M. A. Brovelli (Oct. 2020). "Highlighting recent trends in open source geospatial science and software". en. In: *Transactions in GIS* 24.5, pp. 1141–1146. ISSN: 1361-1682, 1467-9671. DOI: 10.1111/tgis.12703.

Na, A. and M. Priest (2007). *Sensor Observation Service.*

Nguyen, S. H. and T. H. Kolbe (Sept. 3, 2020). "A MULTI-PERSPECTIVE APPROACH TO INTERPRETING SPATIO-SEMANTIC CHANGES OF LARGE 3D CITY MODELS IN CITYGML USING A GRAPH DATABASE". en. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* VI-4/W1-2020, pp. 143–150. ISSN: 2194-9050. DOI: 10.5194/isprs-annals-VI-4-W1-2020-143-2020.

Nys, G.-A. and R. Billen (July 16, 2021). "From consistency to flexibility: A simplified database schema for the management of CityJSON 3D city models". en. In: *Transactions in GIS*, tgis.12807. ISSN: 1361-1682, 1467-9671. DOI: 10.1111/tgis.12807.

– (Dec. 28, 2022). "From consistency to flexibility: Handling spatial information schema thanks to a middleware in a 3D city modeling context". en. In: *Transactions in GIS*, tgis.12807. ISSN: 1361-1682, 1467-9671. DOI: 10.1111/tgis.13014.

Nys, G.-A., R. Billen, and F. Poux (Aug. 12, 2020). "AUTOMATIC 3D BUILDINGS COMPACT RECONSTRUCTION FROM LIDAR POINT CLOUDS". en. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLIII-B2-2020, pp. 473–478. ISSN: 2194-9034. DOI: 10.5194/isprs-archives-XLIII-B2-2020-473-2020.

Nys, G.-A., J.-P. Kasprzyk, P. Hallot, and R. Billen (Sept. 19, 2018). "Towards an ontology for the structuring of remote sensing operations shared by different

processing chains". In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-4, pp. 483–490. ISSN: 2194-9034. DOI: 10.5194/isprs-archives-XLII-4-483-2018.

Nys, G.-A., J.-P. Kasprzyk, P. Hallot, and R. Billen (June 5, 2019). "A Semantic Retrieval System in Remote Sensing Web Platforms". In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2/W13, pp. 1593–1599. ISSN: 2194-9034. DOI: 10.5194/isprs-archives-XLII-2-W13-1593-2019.

Nys, G.-A., A. Kharroubi, F. Poux, and R. Billen (July 2021). "AN EXTENSION OF CITYJSON TO SUPPORT POINT CLOUDS". en. In: vol. XLIII-B4-2021. Nice, France, pp. 301–306. DOI: 10.5194/isprs-archives-XLIII-B4-2021-301-2021.

Nys, G.-A., F. Poux, and R. Billen (Aug. 31, 2020). "CityJSON Building Generation from Airborne LiDAR 3D Point Clouds". en. In: *ISPRS International Journal of Geo-Information* 9.9, p. 521. ISSN: 2220-9964. DOI: 10.3390/ijgi9090521.

Nys, G.-A., M. Van Ruymbeke, and R. Billen (Oct. 2018). "Spatio-Temporal Reasoning in CIDOC CRM: An Hybrid Ontology with GeoSPARQL and OWL-Time". In: *Proceedings of the 2nd Workshop On Computing Techniques For Spatio-Temporal Data in Archaeology And Cultural Heritage*. COARCH2018. Vol. 2230. Melbourne, Australia: CEUR-WS.org, pp. 37–50.

Nys, G.-A., C. Dubois, C. Goffin, P. Hallot, J.-P. Kasprzyk, M. Treffer, and R. Billen (2022). "Geodata quality assessment and operationalisation of the INSPIRE directive: feedback". In: *Bulletin de la Société Géographique de Liège*. ISSN: 2507-0711, 0770-7576. DOI: 10.25518/0770-7576.6698.

Obe, R. O. and L. S. Hsu (2015). *PostGIS in action*. Second edition. OCLC: ocn872985108. Shelter Island, NY: Manning. ISBN: 978-1-61729-139-5.

OGC (2016). *CityGML quality interoperability experiment 16-064r1*. Tech. rep.

Olivera, H. V., M. Holanda, V. Guimarâes, F. Hondo, and W. Boaventura Filho (2015). "Data Modeling for NoSQL Document-Oriented Databases". In: SIMBig'15 Annual International Symposium on Information Management and Big Data. Cusco, Peru.

Pârvu, I. M., F. Remondino, and E. Ozdemir (Dec. 1, 2018). "LoD2 Building Generation Experiences and Comparisons". In: *Journal of Applied Engineering Sciences* 8.2, pp. 59–64. ISSN: 2284-7197, 2247-3769. DOI: 10.2478/jaes-2018-0019.

Patrick, C., L. Sean, and G. Gabby (Jan. 2019). *3D Tiles Specification 1.0*.

Peters, R., B. Dukai, S. Vitalis, J. van Liempt, and J. Stoter (2022). "Automated 3D Reconstruction of LoD2 and LoD1 Models for All 10 Million Buildings of the Netherlands". English. In: *Photogrammetric Engineering and Remote Sensing* 88.3, pp. 165–170. ISSN: 0099-1112. DOI: 10.14358/PERS.21-00032R2.

Pispidikis, I. and E. Dimopoulou (Oct. 5, 2016). "DEVELOPMENT OF A 3D WEBGIS SYSTEM FOR RETRIEVING AND VISUALIZING CITYGML DATA BASED ON THEIR GEOMETRIC AND SEMANTIC CHARACTERISTICS BY USING FREE AND OPEN SOURCE TECHNOLOGY". en. In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-2/W1, pp. 47–53. ISSN: 2194-9050. DOI: 10.5194/isprs-annals-IV-2-W1-47-2016.

Poux and Billen (May 7, 2019). "Voxel-Based 3D Point Cloud Semantic Segmentation: Unsupervised Geometric and Relationship Featuring vs Deep Learning Methods". en. In: *ISPRS International Journal of Geo-Information* 8.5, p. 213. ISSN: 2220-9964. DOI: 10.3390/ijgi8050213.

Poux, F., P. Hallot, R. Neuville, and R. Billen (Oct. 5, 2016). "SMART POINT CLOUD: DEFINITION AND REMAINING CHALLENGES". en. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-2/W1, pp. 119–127. ISSN: 2194-9050. DOI: 10.5194/isprs-annals-IV-2-W1-119-2016.

Poux, F. (Oct. 2019). "The Smart Point Cloud: Structuring 3D intelligent point data". en. DOI: 10.1201/9781351018869-9. PhD thesis, pp. 127–149. ISBN: 9789463754224.

Poux, F., R. Billen, J.-P. Kasprzyk, P.-H. Lefebvre, and P. Hallot (Oct. 7, 2020). "A Built Heritage Information System Based on Point Cloud Data: HIS-PC". en. In: *ISPRS International Journal of Geo-Information* 9.10, p. 588. ISSN: 2220-9964. DOI: 10.3390/ijgi9100588.

Poux, F., R. Neuville, G.-A. Nys, and R. Billen (Sept. 5, 2018). "3D Point Cloud Semantic Modelling: Integrated Framework for Indoor Spaces and Furniture". In: *Remote Sensing* 10.9, p. 1412. ISSN: 2072-4292. DOI: 10.3390/rs10091412.

Poux, F., R. Neuville, L. Van Wersch, G.-A. Nys, and R. Billen (Sept. 30, 2017). "3D Point Clouds in Archaeology: Advances in Acquisition, Processing and Knowledge Integration Applied to Quasi-Planar Objects". en. In: *Geosciences* 7.4, p. 96. ISSN: 2076-3263. DOI: 10.3390/geosciences7040096.

Raguram, R., O. Chum, M. Pollefeys, J. Matas, and J.-M. Frahm (Aug. 2013). "USAC: A Universal Framework for Random Sample Consensus". In: *IEEE*

Transactions on Pattern Analysis and Machine Intelligence 35.8, pp. 2022–2038. ISSN: 0162-8828, 2160-9292. DOI: 10.1109/TPAMI.2012.257.

Rasheed, A., O. San, and T. Kvamsdal (Oct. 3, 2019). "Digital Twin: Values, Challenges and Enablers". In: *arXiv:1910.01719 [eess]*. arXiv: 1910.01719.

Reis, D. G., F. S. Gasparoni, M. Holanda, M. Victorino, M. Ladeira, and E. O. Ribeiro (2018). "An Evaluation of Data Model for NoSQL Document-Based Databases". In: *Trends and Advances in Information Systems and Technologies*. Ed. by Á. Rocha, H. Adeli, L. P. Reis, and S. Costanzo. Vol. 745. Cham: Springer International Publishing, pp. 616–625. ISBN: 978-3-319-77702-3.

Rossknecht, M. and E. Airaksinen (Oct. 12, 2020). "Concept and Evaluation of Heating Demand Prediction Based on 3D City Models and the CityGML Energy ADE–Case Study Helsinki". en. In: *ISPRS International Journal of Geo-Information* 9.10, p. 602. ISSN: 2220-9964. DOI: 10.3390/ijgi9100602.

Rottensteiner, F., G. Sohn, M. Gerke, J. D. Wegner, U. Breitkopf, and J. Jung (July 2014). "Results of the ISPRS benchmark on urban object detection and 3D building reconstruction". en. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 93, pp. 256–271. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2013.10.004.

Roy-Hubara, N. and A. Sturm (Mar. 2020). "Design methods for the new database era: a systematic literature review". en. In: *Software and Systems Modeling* 19.2, pp. 297–312. ISSN: 1619-1366, 1619-1374. DOI: 10.1007/s10270-019-00739-8.

Ruymbeke, M. v. and G.-A. Nys (2022). "SEEING OR BEING SEEN: VISIBILITY ANALYSES FROM THE CHÈVREMONT SITE (MUNICIPALITY OF CHAUDFONTAINE, BELGIUM)". In: *Bulletin de la Société Géographique de Liège*. ISSN: 2507-0711, 0770-7576. DOI: 10.25518/0770-7576.6674.

Schnabel, R., R. Wahl, and R. Klein (June 2007). "Efficient RANSAC for Point-Cloud Shape Detection". en. In: *Computer Graphics Forum* 26.2, pp. 214–226. ISSN: 0167-7055, 1467-8659. DOI: 10.1111/j.1467-8659.2007.01016.x.

Schrotter, G. and C. Hürzeler (Feb. 2020). "The Digital Twin of the City of Zurich for Urban Planning". en. In: *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science* 88.1, pp. 99–112. ISSN: 2512-2789, 2512-2819. DOI: 10.1007/s41064-020-00092-2.

Schultz, C. and M. Bhatt (2013). "InSpace3D: A Middleware for Built Environment Data Access and Analytics". en. In: *Procedia Computer Science* 18, pp. 80–89. ISSN: 18770509. DOI: 10.1016/j.procs.2013.05.171.

Schultz, W., T. Avitabile, and A. Cabral (Aug. 2019). "Tunable consistency in MongoDB". en. In: *Proceedings of the VLDB Endowment* 12.12, pp. 2071–2081. ISSN: 2150-8097. DOI: 10.14778/3352063.3352125.

Shahat, E., C. T. Hyun, and C. Yeom (Mar. 18, 2021). "City Digital Twin Potentials: A Review and Research Agenda". en. In: *Sustainability* 13.6, p. 3386. ISSN: 2071-1050. DOI: 10.3390/su13063386.

Stoter, J., K. Arroyo Ohori, B. Dukai, A. Labetski, K. Kavisha, S. Vitalis, and H. Ledoux (Apr. 9, 2020). "State of the Art in 3D City Modelling: Six Challenges Facing 3D Data as a Platform". In: *GIM International: the worldwide magazine for geomatics* 34.

Sutanta, E. and E. K. Nurnawati (Nov. 2019). "The Design of Relational Database for Multipurpose WebGIS Applications". In: *Journal of Physics: Conference Series* 1413, p. 012029. ISSN: 1742-6588, 1742-6596. DOI: 10.1088/1742-6596/1413/1/012029.

Sveen, A. F. (Dec. 2019). "Efficient storage of heterogeneous geospatial data in spatial databases". en. In: *Journal of Big Data* 6.1, p. 102. ISSN: 2196-1115. DOI: 10.1186/s40537-019-0262-8.

Tarsha Kurdi, F. and M. Awrangjeb (June 17, 2020). "Automatic evaluation and improvement of roof segments for modelling missing details using Lidar data". en. In: *International Journal of Remote Sensing* 41.12, pp. 4702–4725. ISSN: 0143-1161, 1366-5901. DOI: 10.1080/01431161.2020.1723180.

Tarsha-Kurdi, F., T. Landes, and P. Grussenmeyer (Sept. 2007). "Hough-Transform and Extended RANSAC Algorithms for Automatic Detection of 3D Building Roof Planes from Lidar Data". In: ISPRS Workshop on Laser Scanning 2007 and SilviLaser 2007. Espoo, Finland, pp. 407–412.

Tomlinson, R. (Aug. 1968). "Geographic Information System for Regional Planning". In: CSIRO Symposium. GA Stewart.

Toschi, I., E. Nocerino, F. Remondino, A. Revolti, G. Soria, and S. Piffer (May 31, 2017). "GEOSPATIAL DATA PROCESSING FOR 3D CITY MODEL GENERATION, MANAGEMENT AND VISUALIZATION". en. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-1/W1, pp. 527–534. ISSN: 2194-9034. DOI: 10.5194/isprs-archives-XLII-1-W1-527-2017.

Trubka, R., S. Glackin, O. Lade, and C. Pettit (July 2016). "A web-based 3D visualisation and assessment system for urban precinct scenario modelling". en. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 117, pp. 175–186. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2015.12.003.

Van Ruymbeke, M., P. Hallot, G.-A. Nys, and R. Billen (Feb. 2, 2018). "Implementation of multiple interpretation data model concepts in CIDOC CRM and compatible models". In: *Virtual Archaeology Review* 9.19. ISSN: 1989-9947. DOI: 10.4995/var.2018.8884.

Vera-Olivera, H., R. Guo, R. C. Huacarpuma, A. P. B. Da Silva, A. M. Mariano, and H. Maristela (July 2021). "Data Modeling and NoSQL Databases - A Systematic Mapping Review". en. In: *ACM Computing Surveys* 54.6, pp. 1–26. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3457608.

Verma, V., R. Kumar, and S. Hsu (2006). "3D Building Detection and Modeling from Aerial LIDAR Data". In: vol. 2. [Online; accessed 2019-12-20]. 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06). New York, NY, USA: IEEE, pp. 2213–2220. ISBN: 978-0-7695-2597-6. DOI: 10.1109/CVPR.2006.12.

Virtanen, J.-P., K. Jaalama, T. Puustinen, A. Julin, J. Hyyppä, and H. Hyyppä (Mar. 4, 2021). "Near Real-Time Semantic View Analysis of 3D City Models in Web Browser". en. In: *ISPRS International Journal of Geo-Information* 10.3, p. 138. ISSN: 2220-9964. DOI: 10.3390/ijgi10030138.

Visconti, E., C. Tsigkanos, Z. Hu, and C. Ghezzi (Dec. 2021). "Model-driven engineering city spaces via bidirectional model transformations". en. In: *Software and Systems Modeling* 20.6, pp. 2003–2022. ISSN: 1619-1366, 1619-1374. DOI: 10.1007/s10270-020-00851-0.

Višnjevac, N., R. Mihajlovic, M. Šoškic, Ž. Cvijetinovic, and B. Bajat (May 10, 2019). "Prototype of the 3D Cadastral System Based on a NoSQL Database and a JavaScript Visualization Application". en. In: *ISPRS International Journal of Geo-Information* 8.5, p. 227. ISSN: 2220-9964. DOI: 10.3390/ijgi8050227.

Vitalis, S., A. Labetski, K. Arroyo Ohori, H. Ledoux, and J. Stoter (Sept. 23, 2019). "A DATA STRUCTURE TO INCORPORATE VERSIONING IN 3D CITY MODELS". en. In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-4/W8, pp. 123–130. ISSN: 2194-9050. DOI: 10.5194/isprs-annals-IV-4-W8-123-2019.

Vitalis, S., K. Arroyo Ohori, and J. Stoter (June 24, 2020). "CityJSON in QGIS: Development of an open-source plugin". en. In: *Transactions in GIS* VI-4/W1-2020, tgis.12657. ISSN: 1361-1682, 1467-9671. DOI: 10.1111/tgis.12657.

Vitalis, S., K. Ohori, and J. Stoter (Aug. 1, 2019). "Incorporating Topological Representation in 3D City Models". en. In: *ISPRS International Journal of Geo-Information* 8.8, p. 347. ISSN: 2220-9964. DOI: 10.3390/ijgi8080347.

Voutos, Y., P. Mylonas, E. Spyrou, and E. Charou (Nov. 21, 2017). "A Social Environmental Sensor Network Integrated within a Web GIS Platform". en. In: *Journal of Sensor and Actuator Networks* 6.4, p. 27. ISSN: 2224-2708. DOI: 10.3390/jsan6040027.

Wagner, D., M. Wewetzer, J. Bogdahn, N. Alam, M. Pries, and V. Coors (2013). "Geometric-Semantical Consistency Validation of CityGML Models". In: *Progress and New Trends in 3D Geoinformation Sciences*. Ed. by J. Pouliot, S. Daniel, F. Hubert, and A. Zamyadi. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 171–192. ISBN: 978-3-642-29792-2.

Wang, R., J. Peethambaran, and D. Chen (Feb. 2018). "LiDAR Point Clouds to 3D Urban Models: A Review". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 11.2, pp. 606–627. ISSN: 1939-1404, 2151-1535. DOI: 10.1109/JSTARS.2017.2781132.

Weglarz, G. (2004). "Two Worlds of Data – Unstructured and Structured". In: *DM Review*.

Westerholt, R. and B. Resch (June 2015). "Asynchronous Geospatial Processing: An Event-Driven Push-Based Architecture for the OGC Web Processing Service: Push-Based Async Geo-Processing with the OGC WPS". en. In: *Transactions in GIS* 19.3, pp. 455–479. ISSN: 13611682. DOI: 10.1111/tgis.12104.

Wichmann, A. (2018). "Grammar-guided reconstruction of semantic 3D building models from airborne LiDAR data using half-space modeling". en. In: [Online; accessed 2020-01-30]. DOI: 10.14279/DEPOSITONCE-6803.

Widl, E., G. Agugiaro, and J. Peters-Anders (Aug. 5, 2021). "Linking Semantic 3D City Models with Domain-Specific Simulation Tools for the Planning and Validation of Energy Applications at District Level". en. In: *Sustainability* 13.16, p. 8782. ISSN: 2071-1050. DOI: 10.3390/su13168782.

Wittern, E., A. Cha, J. C. Davis, G. Baudart, and L. Mandel (2019). "An Empirical Study of GraphQL Schemas". en. In: *Service-Oriented Computing*. Ed. by S. Yangui, I. Bouassida Rodriguez, K. Drira, and Z. Tari. Vol. 11895. Cham: Springer International Publishing, pp. 3–19. ISBN: 978-3-030-33701-8.

Wong, S., S. Swartz, and D. Sarkar (June 2002). "A middleware architecture for open and interoperable GISs". In: *IEEE Multimedia* 9.2, pp. 62–76. ISSN: 1070986X. DOI: 10.1109/93.998065.

Wyard, C., B. Beaumont, T. Grippa, G.-A. Nys, H. Fauvel, and E. Hallot (May 2022). "Mapping roof materials using WV3 imagery and a state-of-the-art OBIA processing chain: application over Liège, Belgium". In: *Proceedings*

of esa Living Planet Symposium 2022. esa Living Planet Symposium 2022. Bonn, Germany. DOI: 10.13140/RG.2.2.19567.51363.

Xiong, B., M. Jancosek, S. Oude Elberink, and G. Vosselman (Mar. 2015). "Flexible building primitives for 3D building modeling". en. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 101, pp. 275–290. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2015.01.002.

Xu, B., W. Jiang, J. Shan, J. Zhang, and L. Li (Dec. 23, 2015). "Investigation on the Weighted RANSAC Approaches for Building Roof Plane Segmentation from LiDAR Point Clouds". en. In: *Remote Sensing* 8.1, p. 5. ISSN: 2072-4292. DOI: 10.3390/rs8010005.

Yao, Z., C. Nagel, F. Kunde, G. Hudra, P. Willkomm, A. Donaubauer, T. Adolphi, and T. H. Kolbe (Dec. 2018). "3DCityDB - a 3D geodatabase solution for the management, analysis, and visualization of semantic 3D city models based on CityGML". en. In: *Open Geospatial Data, Software and Standards* 3.1. [Online; accessed 2019-03-13]. ISSN: 2363-7501. DOI: 10.1186/s40965-018-0046-7.

Zhang, X., W. Song, and L. Liu (June 2014). "An implementation approach to store GIS spatial data on NoSQL database". In: [Online; accessed 2019-04-24]. 2014 22nd International Conference on Geoinformatics. Kaohsiung, Taiwan: IEEE, pp. 1–5. ISBN: 978-1-4799-5714-9. DOI: 10.1109/GEOINFORMATICS.2014.6950846.

Zhao, Z., H. Ledoux, and J. Stoter (Sept. 13, 2013). "AUTOMATIC REPAIR OF CITYGML LOD2 BUILDINGS USING SHRINK-WRAPPING". en. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* II-2/W1, pp. 309–317. ISSN: 2194-9050. DOI: 10.5194/isprsannals-II-2-W1-309-2013.

Zhou, K., R. Lindenbergh, B. Gorte, and S. Zlatanova (Apr. 2020). "LiDAR-guided dense matching for detecting changes and updating of buildings in Airborne LiDAR data". en. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 162, pp. 200–213. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2020.02.005.

Zlatanova, S. and J. Stoter (2006). "The role of DBMS in the new generation GIS architecture". en. In: *Frontiers of Geographic Information Technology*. Ed. by S. Rana and J. Sharma. Berlin/Heidelberg: Springer-Verlag, pp. 155–180. ISBN: 978-3-540-25685-4.

Curriculum

Gilles-Antoine Nys (1992) was born in Saint-Ghislain, Belgium. He obtained a Bsc and Msc in Industrial Engineering, land-surveying orientation, from the Haute Ecole de la Province de Liège, Belgium. During his master, he had the chance to do an internship at the Geomatics Unit of the University of Liège. He then began working in the same research unit as a Research Engineer at the end of this internship.

During his PhD research, he thus worked on several projects for public services as well as for private companies. Projects were related to NoSQL databases, data quality assessment, Semantic Web, etc. He then moved on to a position as a Teaching Assistant in GIS and Remote Sensing courses. Logically in the trend of his PhD thesis, he supervised the WebGIS and GIS Project courses. He has also been involved in Msc thesis supervision at the University of Liège but also at the Agronomic and Veterinary Institute Hassan II, Rabat, Morocco. In parallel with this technical and teaching work, he served the University community as a representative to the Geography Department Council, the SPHERES Research Unit Council and the Geography Doctoral College. He was also elected to represent the doctoral students in Science and Technology on the Doctoral Office, a transversal university body.

He now works as a Consultant Business Analyst for major Belgian companies managing various networks: electricity, optical fibre. Due to his status as "Scientific Collaborator", he maintains a strong link with the research group and still follows several ongoing research projects related to urban 3D modeling and CityJSON.

List of publications

International Journal Papers (ISI)

F. Poux, R. Neuville, L. Van Wersch, et al. (Sept. 30, 2017). "3D Point Clouds in Archaeology: Advances in Acquisition, Processing and Knowledge Integration Applied to Quasi-Planar Objects". en. In: *Geosciences* 7.4, p. 96. ISSN: 2076-3263. DOI: 10.3390/geosciences7040096

M. Van Ruymbeke et al. (Feb. 2, 2018). "Implementation of multiple interpretation data model concepts in CIDOC CRM and compatible models". In: *Virtual Archaeology Review* 9.19. ISSN: 1989-9947. DOI: 10.4995/var.2018.8884

R. Billen, B. Jonlet, et al. (2018). "La transition numérique dans le domaine du patrimoine bâti: un retour d'expériences". In: *Bulletin de la Commission royale des Monuments, Sites et Fouilles* 30, pp. 119–146

F. Poux, R. Neuville, G.-A. Nys, et al. (Sept. 5, 2018). "3D Point Cloud Semantic Modelling: Integrated Framework for Indoor Spaces and Furniture". In: *Remote Sensing* 10.9, p. 1412. ISSN: 2072-4292. DOI: 10.3390/rs10091412

G.-A. Nys, F. Poux, et al. (Aug. 31, 2020). "CityJSON Building Generation from Airborne LiDAR 3D Point Clouds". en. In: *ISPRS International Journal of Geo-Information* 9.9, p. 521. ISSN: 2220-9964. DOI: 10.3390/ijgi9090521

S. El Yamani et al. (Mar. 5, 2021). "3D Variables Requirements for Property Valuation Modeling Based on the Integration of BIM and CIM". in: *Sustainability* 13.5, p. 2814. ISSN: 2071-1050. DOI: 10.3390/su13052814

G.-A. Nys and R. Billen (July 16, 2021). "From consistency to flexibility: A simplified database schema for the management of CityJSON 3D city models". en. In: *Transactions in GIS*, tgis.12807. ISSN: 1361-1682, 1467-9671. DOI: 10.1111/tgis.12807

M. v. Ruymbeke and G.-A. Nys (2022). "SEEING OR BEING SEEN: VISIBILITY ANALYSES FROM THE CHÈVREMONT SITE (MUNICIPALITY OF CHAUDFONTAINE, BELGIUM)". in: *Bulletin de la Société Géographique de Liège*. ISSN: 2507-0711, 0770-7576. DOI: 10.25518/0770-7576.6674

G.-A. Nys, C. Dubois, et al. (2022). "Geodata quality assessment and operationalisation of the INSPIRE directive: feedback". In: *Bulletin de la Société Géographique de Liège*. ISSN: 2507-0711, 0770-7576. DOI: 10.25518/0770-7576.6698

G.-A. Nys and R. Billen (Dec. 28, 2022). "From consistency to flexibility: Handling spatial information schema thanks to a middleware in a 3D city modeling context". en. In: *Transactions in GIS*, tgis.12807. ISSN: 1361-1682, 1467-9671. DOI: 10.1111/tgis.13014

International Refereed Conferences

G.-A. Nys, M. Van Ruymbeke, et al. (Oct. 2018). "Spatio-Temporal Reasoning in CIDOC CRM: An Hybrid Ontology with GeoSPARQL and OWL-Time". In: *Proceedings of the 2nd Workshop On Computing Techniques For Spatio-Temporal Data in Archaeology And Cultural Heritage*. COARCH2018. Vol. 2230. Melbourne, Australia: CEUR-WS.org, pp. 37–50

G.-A. Nys, J.-P. Kasprzyk, et al. (Sept. 19, 2018). "Towards an ontology for the structuring of remote sensing operations shared by different processing chains". In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-4, pp. 483–490. ISSN: 2194-9034. DOI: 10.5194/isprs-archives-XLII-4-483-2018

G.-A. Nys, J.-P. Kasprzyk, et al. (June 5, 2019). "A Semantic Retrieval System in Remote Sensing Web Platforms". In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2/W13, pp. 1593–1599. ISSN: 2194-9034. DOI: 10.5194/isprs-archives-XLII-2-W13-1593-2019

G.-A. Nys, R. Billen, and F. Poux (Aug. 12, 2020). "AUTOMATIC 3D BUILDINGS COMPACT RECONSTRUCTION FROM LIDAR POINT CLOUDS". en. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLIII-B2-2020, pp. 473–478. ISSN: 2194-9034. DOI: 10.5194/isprs-archives-XLIII-B2-2020-473-2020

G.-A. Nys, A. Kharroubi, et al. (July 2021). "AN EXTENSION OF CITYJSON TO SUPPORT POINT CLOUDS". en. In: vol. XLIII-B4-2021. Nice, France,

pp. 301–306. DOI: 10.5194/isprs-archives-XLIII-B4-2021-301-2021

M. Y. Joshi et al. (Sept. 15, 2020). "POTENTIAL FOR URBAN GREENING WITH GREEN ROOFS: A WAY TOWARDS SMART CITIES". en. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* VI-4/W2-2020, pp. 87–94. ISSN: 2194-9050. DOI: 10.5194/isprs-annals-VI-4-W2-2020-87-2020

FACULTY OF SCIENCE
DEPARTMENT OF GEOGRAPHY
GEOMATICS UNIT
Quartier Agora, B5a - Liège, Belgium
B-4000 Liège