From consistency to flexibility: a database schema for the management of CityJSON 3D City Models

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 storage. Thanks to the use of the NoSQL store, the focus is on flexibility of the data schemas and thus its clarification. 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

---

1. Introduction

Nowadays, many large cities have usage of their own 3D digital model (Biljecki 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. (Floros and Dimopoulou 2016; Biljecki, Kumar, and Nagel 2018). 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 WFS services, the support of 3D scenes (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 relational 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 paper 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 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 paper is divided as follows: the section 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, the section 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. The section 4 develops the new data management paradigm 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 5. Before considering future works, we conclude on the principal benefits of the new generation application and its advances.

2. Related works

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 definitions; Urban GIS

(Blaschke et al. 2011). From a technical viewpoint, a web-based GIS application is divided into three interdependent constituting parts at least: a client, which is a consumer of spatial information; a server, which is a GIS processing system; and 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). Given that, 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, Dawidowicz, and Pacholczyk 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, the relational databases often represent the core base of web applications (Zlatanova and Stoter 2006; Mobasheri et al. 2020). 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, Li, and Patel 2013). Relational databases have been refactored to handle JSON (Liu, Hammerschmidt, and McMahon 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, Janssen, and Biljecki 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, Kasprzyk, and Donnay 2018; Poux et al. 2020).

While the relational database is still mandatory for its data-processing capabilities, the document-oriented 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 (Zhang, Song, and Liu 2014; Lopez, Couturier, and Lopez 2016; da Costa Rainho and Bernardino 2018). However, it shows that even if performances are overall improved with document-oriented store, it is not yet always true (Makris, Tserpes, and Anagnostopoulos 2019). Sometimes, relational database ranks ahead of document-oriented stores (Bartoszewski, Piorkowski, and Lupa 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” architecture. 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 database schema, this distribution has proved its usefulness in spatial 2D (Đurić 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 et al. 2019). Its improved support of levels-of-detail and metadata make it a good substitute to CityGML (Nys, Poux, and Billen 2020). However, its usage is still limited to specific applications and data encoding (Kumar, Ledoux, and Stoter 2018; Nys, Billen, and 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 (Stelios Vitalis, Arroyo Ohori, and Stoter 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 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.

3. Solution description

This section is divided in two subsections, respectively; 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.

3.1. Schema model

In the document-oriented database, the records are stored as documents that follow non-mandatory and semi-structured 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. 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 aroung this notion of set. 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 nondependent 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 (Olivera et al. 2015). According to the benchmark (Olivera et al. 2015), the referred models are more efficient but impose 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 a n *address*, a *measuredHeight*, a *roofType*, a specific set of allowed geometries, etc.). These alternate schemas are the second-order schemas or discriminated schemas. In the core application, the five first-order collections are defined dynamically by the database and the server at startup (see Figure 1**Error! Reference source not found.** 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 none space is used). If a modification is made afterward, a new collection is created on the fly if necessary.

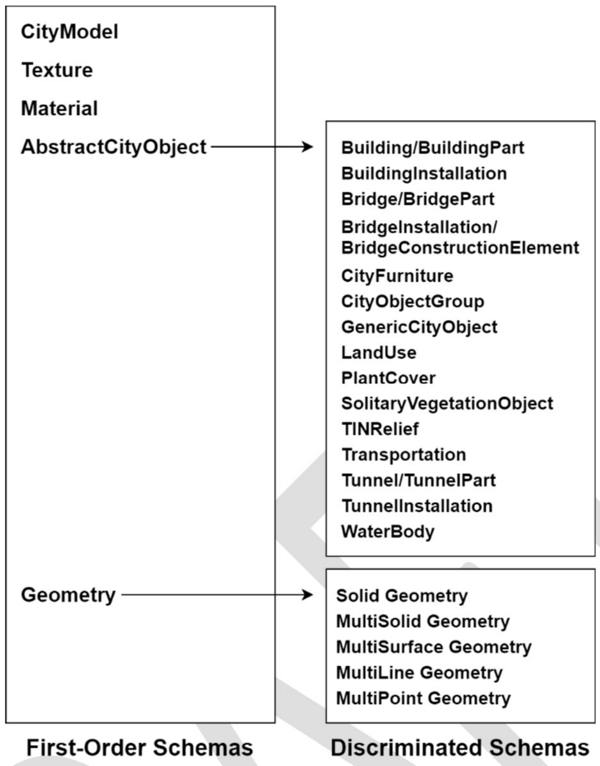

Figure 1. CityJSON objects schemas and inheritance

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 the CityJSON core specification. However, the management of this schema in a NoSQL solution does not limit the insertion of extended attributes. 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 simplest object stating on the type and the unique ID of the document in the *AbstractCityObject* collection (see Figure 2**Error! Reference source not found.**).

As stated above, every object is referred with a unique identifier specific to its lifecycle 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 subschema or a part of it.

Concerning the insertion validation, during the model lifecycle, 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, Cabral, and Bernardino 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 2**Error! Reference source not found.** 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 bouding box enveloping the object geometries. This impacts performances on model import.

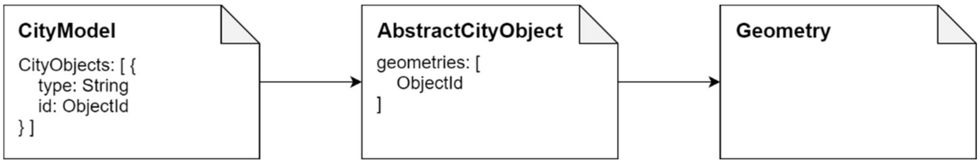

Figure 2. Referred documents structure in production

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 ISO19107 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 1**Error! Reference source not found.**). 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 schemas 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.

3.2. 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) app facilitates a smart deployment. MERN web apps 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. The increase of flexibility and resilience is demonstrated and put in parallel with the architecture components.

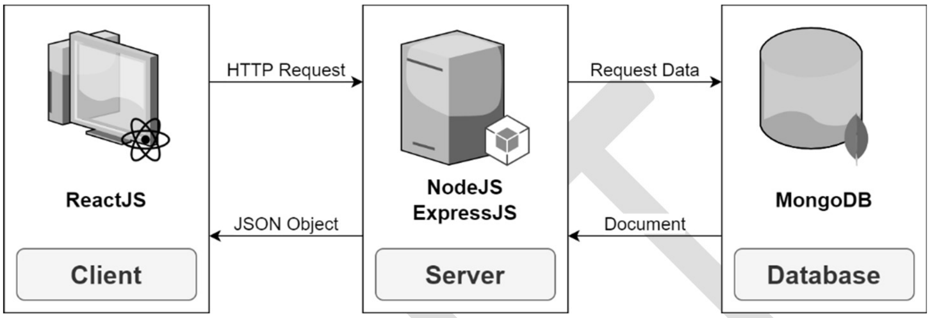

Figure 3. Architecture schema of a full stack MERN application

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 4 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 (S. Vitalis 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, van Sinderen, and Singh 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 paper according to schema modifications during the production phase.

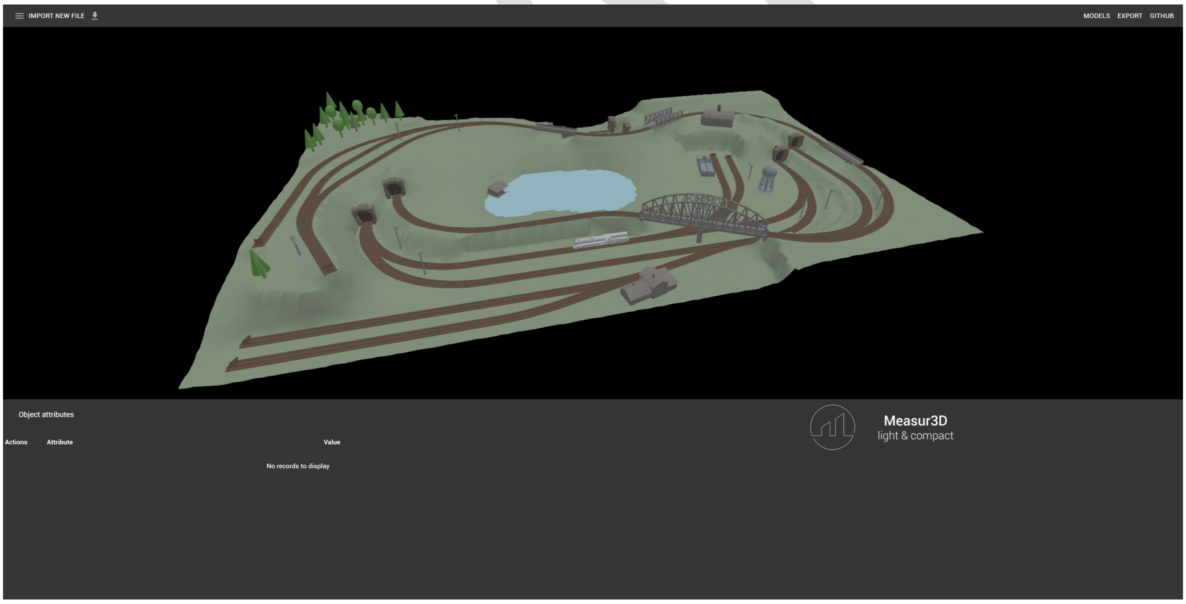

Figure 4. Client view of the application – the rendered model is the dummy Railway.json file provided by the 3D GeoInformation research group from TUDelft

The server is a NodeJS JavaScript runtime environment that allows performing JavaScript code on server side (following the ECMAScript2015 specifications (Ecma

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 server to set up middleware function calls: 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. Please 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 have 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 (Diogo et al. 2019). 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 4.3.

4. 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.

4.1. Structured and unstructured data

In this paper, 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.

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 slowed and 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). 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. The section 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-database):

- Basically Available: the data are guaranteed as always available in terms of CAP theorem. 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 on 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 store 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.

4.2. 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 queries the same API has also been made (see section 4.4). Downloading the objects from the backend layer can be made in several ways:

- (a) Continuous 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. 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.

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 5**Error! Reference source not found.** and Figure 6**Error! Reference source not found.** represent the sequence diagrams for both solution: continuous and bulk requests. They depict the succession of queries between the three-tier (client, server and database) and their responses.

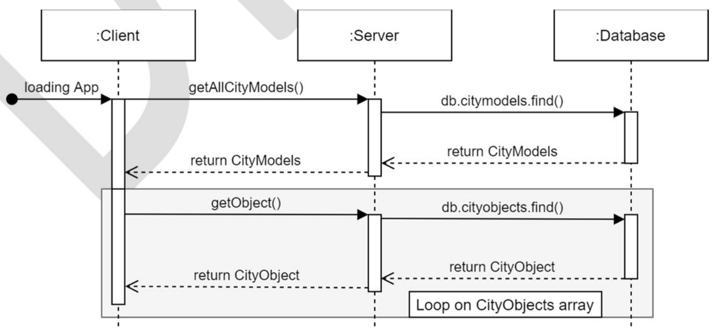

Figure 5. (a) Continuous loading (sequence diagram) - client-side reconstruction.

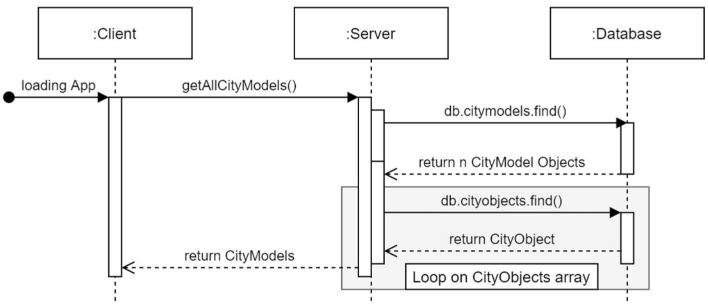

Figure 6. (b) Bulk loading (sequence diagram) - server-side reconstruction.

The clients open a connection whenever they initialise themselves. The server and the database keep the connection open for future calls thanks to the 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.

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) was much nil). 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.

4.3. 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.

4.4. Comparison reference with relational solution

To illustrate the disadvantage of the relational joints, 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. For the reminder, 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 n-clients querying asynchronously the databases. About the architecture scalability, there is still room for improvement by multiplying the number of replicated databases (Schultz, Avitabile, and Cabral 2019). The balance should be determinate between the number of replications (n-databases), performance and the required consistency (Haughian, Osman, and Knottenbelt 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 1, which shows the databases response time.

Table 1. Response time for the *Buildings* queries – repetition x objects (in milliseconds)

	1 x 1	**1 x 10**	**10 x 1**	**1 x 100**	**100 x 1**	**1 x 3353 (1 x all)**
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” paradigm. 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 7):

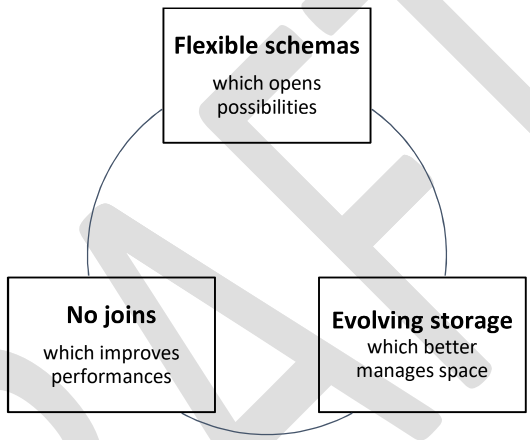

Figure 7. Summary of new capabilities

5. Usage scenarios

Now that the schema has been presented and the database solution has been compared with a relational solution on a quantitative benchmark, 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 3.1).

5.1. 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 simplistic 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, 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 8). 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.

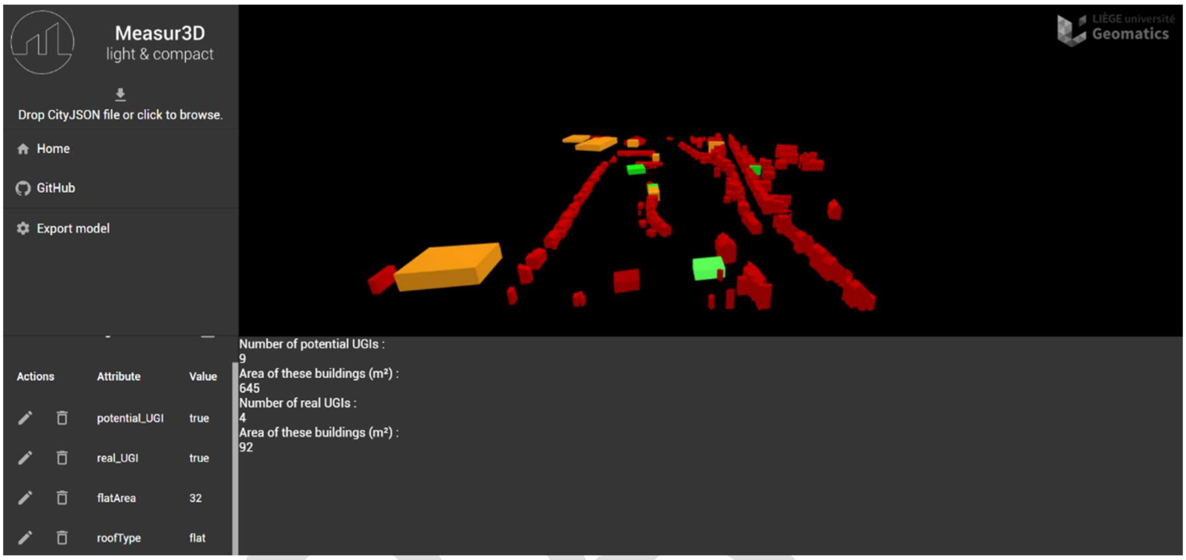

Figure 8. UGI module for the visualization and computation of green roofs

For comparison purpose, the Table 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 2. Response time for the *Buildings* queries – repetition x objects (in milliseconds)

	1 x 1	**1 x 10**	**10 x 1**	**1 x 100**	**100 x 1**	**1 x 3353 (1 x all)**
Simplified schema	48	53	76	125	297	6678

3DCityDB	83	86	191	163	379	38089
3DCityDB + UGI	88	91	252	172	412	41374

5.2.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 stats 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 paper 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 9 - version updated on January 1, 2019).

The Figure 9 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.

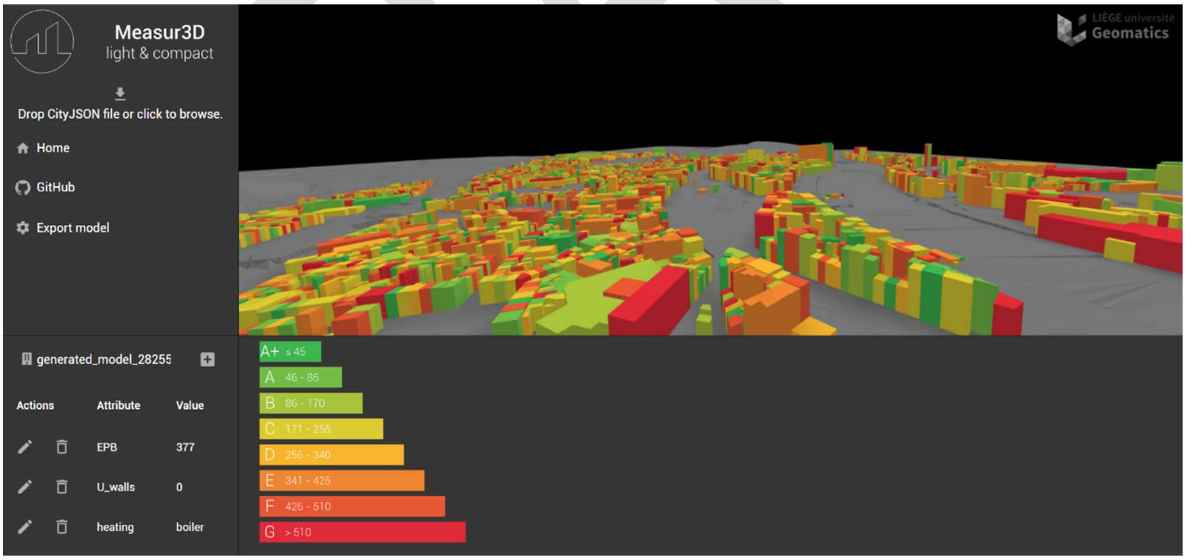

Figure 9. Illustration of the EPB module

6. Conclusion

This paper 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 paradigm 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.

References

Agrawal, Sonam, and R. D. Gupta. 2017. ‘Web GIS and Its Architecture: A Review’. *Arabian Journal of Geosciences* 10(23):518. doi: 10.1007/s12517-017-3296-2.

Aleksandrov, M., A. Diakité, J. Yan, W. Li, and S. Zlatanova. 2019. ‘SYSTEMS ARCHITECTURE FOR MANAGEMENT OF BIM, 3D GIS AND SENSORS DATA’. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-4/W9:3–10. doi: 10.5194/isprs-annals-IV-4-W9-3-2019.

Allah Bukhsh, Zaharah, Marten van Sinderen, and P. M. Singh. 2015. ‘SOA and EDA: A Comparative Study - Similarities, Differences and Conceptual Guidelines on Their Usage’: Pp. 213–20 in *Proceedings of the 12th International Conference on e-Business*. Colmar, Alsace, France: SCITEPRESS - Science and and Technology Publications.

Bartoszewski, Dominik, Adam Piorkowski, and Michal Lupa. 2019. ‘The Comparison of Processing Efficiency of Spatial Data for PostGIS and MongoDB Databases’. Pp. 291–302 in *Beyond Databases, Architectures and Structures. Paving the Road to Smart Data Processing and Analysis*. Vol. 1018, *Communications in Computer and Information Science*, edited by S. Kozielski, D. Mrozek, P. Kasprowski, B. Małysiak-Mrozek, and D. Kostrzewa. Cham: Springer International Publishing.

Biljecki, Filip, Kavisha Kumar, and Claus Nagel. 2018. ‘CityGML Application Domain Extension (ADE): Overview of Developments’. *Open Geospatial Data, Software and Standards* 3(1):13. doi: 10.1186/s40965-018-0055-6.

Biljecki, Filip, Jantien Stoter, Hugo Ledoux, Sisi Zlatanova, and Arzu Çöltekin. 2015. ‘Applications of 3D City Models: State of the Art Review’. *ISPRS International Journal of Geo-Information* 4(4):2842–89. doi: 10.3390/ijgi4042842.

Blaschke, Thomas, Geoffrey J. Hay, Qihao Weng, and Bernd Resch. 2011. ‘Collective Sensing: Integrating Geospatial Technologies to Understand Urban Systems—An Overview’. *Remote Sensing* 3(8):1743–76. doi: 10.3390/rs3081743.

Boaventura Filho, Wagner, Harley Vera Olivera, Maristela Holanda, and Aleteia Favacho. 2016. ‘Geographic Data Modelling for NoSQL Document-Oriented Databases’. Lisbon, Portugal.

Breunig, Martin, Paul V. Kuper, Edgar Butwilowski, Andreas Thomsen, Markus Jahn, André Dittrich, Mulhim Al-Doori, Darya Golovko, and Mathias Menninghaus. 2016. ‘The Story of DB4GeO – A Service-Based Geo-Database Architecture to Support Multi-Dimensional Data Analysis and Visualization’. *ISPRS Journal of Photogrammetry and Remote Sensing* 117:187–205. doi: 10.1016/j.isprsjprs.2015.12.006.

Brewer, Eric A. 2000. ‘Towards Robust Distributed Systems’. P. 7 in *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing - PODC ’00*. Portland, Oregon, United States: ACM Press.

Chasseur, Craig, Yinan Li, and Jignesh Patel. 2013. ‘Enabling JSON Document Stores in Relational Systems’. in *Proceedings of the 16th International Workshop on the Web and Databases 2013*. New York, NY, USA.

da Costa Rainho, Filipe, and Jorge Bernardino. 2018. ‘Web GIS: A New System to Store Spatial Data Using GeoJSON in MongoDB’. Pp. 1–6 in *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*. Caceres: IEEE.

Diogo, Miguel, Bruno Cabral, and Jorge Bernardino. 2019. ‘Consistency Models of NoSQL Databases’. *Future Internet* 11(2):43. doi: 10.3390/fi11020043.

Doboš, Jozef, and Anthony Steed. 2012. ‘3D Revision Control Framework’. P. 121 in *Proceedings of the 17th International Conference on 3D Web Technology - Web3D ’12*. Los Angeles, California: ACM Press.

Đurić, Mladen. 2018. ‘GEOPORTAL FOR SEARCHING AND VISUALIZATION OF CADASTRAL DATA’. *САВРЕМЕНА ТЕОРИЈА И ПРАКСА У ГРАДИТЕЉСТВУ* 13(1). doi: 10.7251/STP1813687A.

Ecma International. 2015. *ECMAScript 2015. Language Specification*. 6. Geneva, Switzerland.

Floros, G., and E. Dimopoulou. 2016. ‘Investigating the Enrichment of a 3D City Model with Various CityGML Modules’. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2/W2:3–9. doi: 10.5194/isprs-archives-XLII-2-W2-3-2016.

Gröger, Gerhard, and Lutz Plümer. 2012. ‘CityGML – Interoperable Semantic 3D City Models’. *ISPRS Journal of Photogrammetry and Remote Sensing* 71:12–33. doi: 10.1016/j.isprsjprs.2012.04.004.

Haughian, Gerard, Rasha Osman, and William J. Knottenbelt. 2016. ‘Benchmarking Replication in Cassandra and MongoDB NoSQL Datastores’. Pp. 152–66 in *Database and Expert Systems Applications*. Vol. 9828, *Lecture Notes in Computer Science*, edited by S. Hartmann and H. Ma. Cham: Springer International Publishing.

Holemans, Amandine, Jean-Paul Kasprzyk, and Jean-Paul Donnay. 2018. ‘Coupling an Unstructured NoSQL Database with a Geographic Information System’. Pp. 23–28 in *Proceedings of GEOProcessing 2018*. Rome, Italy.

Joshi, M. Y., W. Selmi, Marc Binard, Gilles-Antoine Nys, and Jacques Teller. 2020. ‘POTENTIAL FOR URBAN GREENING WITH GREEN ROOFS: A WAY

TOWARDS SMART CITIES'. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* VI-4/W2-2020:87–94. doi: 10.5194/isprs-annals-VI-4-W2-2020-87-2020.

Kulawiak, Marcin, Agnieszka Dawidowicz, and Marek Emanuel Pacholczyk. 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'. *Computers & Geosciences* 129:26–37. doi: 10.1016/j.cageo.2019.04.011.

Kumar, K., H. Ledoux, and J. Stoter. 2018. 'Dynamic 3D Visualization of Floods: Case of the Netherlands'. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-4/W10:83–87. doi: 10.5194/isprs-archives-XLII-4-W10-83-2018.

Laksono, Dany. 2018. 'Testing Spatial Data Deliverance in SQL and NoSQL Database Using NodeJS Fullstack Web App'. Pp. 1–5 in *2018 4th International Conference on Science and Technology (ICST)*. Yogyakarta: IEEE.

Ledoux, Hugo, Ken Arroyo Ohori, Kavisha Kumar, Balázs Dukai, Anna Labetski, and Stelios Vitalis. 2019. 'CityJSON: A Compact and Easy-to-Use Encoding of the CityGML Data Model'. *ArXiv:1902.09155 [Cs]*.

Lim, J., P. Janssen, and F. Biljecki. 2020. 'VISUALISING DETAILED CITYGML AND ADE AT THE BUILDING SCALE'. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLIV-4/W1-2020:83–90. doi: 10.5194/isprs-archives-XLIV-4-W1-2020-83-2020.

Liu, Zhen Hua, Beda Hammerschmidt, and Doug McMahon. 2014. 'JSON Data Management: Supporting Schema-Less Development in RDBMS'. Pp. 1247–58 in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data - SIGMOD '14*. Snowbird, Utah, USA: ACM Press.

Lopez, M., S. Couturier, and J. Lopez. 2016. 'Integration of NoSQL Databases for Analyzing Spatial Information in Geographic Information System'. Pp. 351–55 in *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)*. Tehri, India: IEEE.

Makris, Antonios, Konstantinos Tserpes, and Dimosthenis Anagnostopoulos. 2019. 'Performance Evaluation of MongoDB and PostgreSQL for Spatio-Temporal Data'. P. 8 in *Proceedings of EDBT/ICDT 2019*. Lisbon, Portugal: CEUR-WS.org.

Mao, Bo, and Lars Harrie. 2016. 'Methodology for the Efficient Progressive Distribution and Visualization of 3D Building Objects'. *ISPRS International Journal of Geo-Information* 5(10):185. doi: 10.3390/ijgi5100185.

Mobasheri, Amin, Helena Mitasova, Markus Neteler, Alexander Singleton, Hugo Ledoux, and Maria Antonia Brovelli. 2020. ‘Highlighting Recent Trends in Open Source Geospatial Science and Software’. *Transactions in GIS* 24(5):1141–46. doi: 10.1111/tgis.12703.

Nys, Gilles-Antoine, Roland Billen, and Florent Poux. 2020. ‘AUTOMATIC 3D BUILDINGS COMPACT RECONSTRUCTION FROM LIDAR POINT CLOUDS’. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLIII-B2-2020:473–78. doi: 10.5194/isprs-archives-XLIII-B2-2020-473-2020.

Nys, Gilles-Antoine, Abderrazzaq Kharroubi, Florent Poux, and Roland Billen. 2021. ‘AN EXTENSION OF CITYJSON TO SUPPORT POINT CLOUDS’. in *Proceedings of the XXIVth ISPRS Congress*. Nice, France.

Nys, Gilles-Antoine, Florent Poux, and Roland Billen. 2020. ‘CityJSON Building Generation from Airborne LiDAR 3D Point Clouds’. *ISPRS International Journal of Geo-Information* 9(9):521. doi: 10.3390/ijgi9090521.

Obe, Regina O., and Leo S. Hsu. 2015. *PostGIS in Action*. Second edition. Shelter Island, NY: Manning.

Olivera, Harley Vera, Maristela Holanda, Valeria Guimarâes, Fernanda Hondo, and Wagner Boaventura Filho. 2015. ‘Data Modeling for NoSQL Document-Oriented Databases’. Cusco, Peru.

Pispidikis, I., and E. Dimopoulou. 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’. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-2/W1:47–53. doi: 10.5194/isprs-annals-IV-2-W1-47-2016.

Poux, Florent, Roland Billen, Jean-Paul Kasprzyk, Pierre-Henri Lefebvre, and Pierre Hallot. 2020. ‘A Built Heritage Information System Based on Point Cloud Data: HIS-PC’. *ISPRS International Journal of Geo-Information* 9(10):588. doi: 10.3390/ijgi9100588.

Reis, Debora G., Fabio S. Gasparoni, Maristela Holanda, Marcio Victorino, Marcelo Ladeira, and Edward O. Ribeiro. 2018. ‘An Evaluation of Data Model for NoSQL Document-Based Databases’. Pp. 616–25 in *Trends and Advances in Information Systems and Technologies*. Vol. 745, edited by Á. Rocha, H. Adeli, L. P. Reis, and S. Costanzo. Cham: Springer International Publishing.

Schultz, William, Tess Avitabile, and Alyson Cabral. 2019. ‘Tunable Consistency in MongoDB’. *Proceedings of the VLDB Endowment* 12(12):2071–81. doi: 10.14778/3352063.3352125.

Sveen, Atle Frenvik. 2019. ‘Efficient Storage of Heterogeneous Geospatial Data in Spatial Databases’. *Journal of Big Data* 6(1):102. doi: 10.1186/s40537-019-0262-8.

Tomlinson, Roger. 1968. ‘Geographic Information System for Regional Planning’. in *Papers of a CSIRO Symposium*. GA Stewart.

Toschi, I., E. Nocerino, F. Remondino, A. Revolti, G. Soria, and S. Piffer. 2017. ‘GEOSPATIAL DATA PROCESSING FOR 3D CITY MODEL GENERATION, MANAGEMENT AND VISUALIZATION’. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-1/W1:527–34. doi: 10.5194/isprs-archives-XLII-1-W1-527-2017.

Trubka, Roman, Stephen Glackin, Oliver Lade, and Chris Pettit. 2016. ‘A Web-Based 3D Visualisation and Assessment System for Urban Precinct Scenario Modelling’. *ISPRS Journal of Photogrammetry and Remote Sensing* 117:175–86. doi: 10.1016/j.isprsjprs.2015.12.003.

Virtanen, Juho-Pekka, Kaisa Jaalama, Tuulia Puustinen, Arttu Julin, Juha Hyyppä, and Hannu Hyyppä. 2021. ‘Near Real-Time Semantic View Analysis of 3D City Models in Web Browser’. *ISPRS International Journal of Geo-Information* 10(3):138. doi: 10.3390/ijgi10030138.

Višnjevac, Nenad, Rajica Mihajlović, Mladen Šoškić, Željko Cvijetinović, and Branislav Bajat. 2019. ‘Prototype of the 3D Cadastral System Based on a NoSQL Database and a JavaScript Visualization Application’. *ISPRS International Journal of Geo-Information* 8(5):227. doi: 10.3390/ijgi8050227.

Vitalis, S., A. Labetski, F. Boersma, F. Dahle, X. Li, K. Arroyo Ohori, H. Ledoux, and J. Stoter. 2020. ‘CITYJSON + WEB = NINJA’. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* VI-4/W1-2020:167–73. doi: 10.5194/isprs-annals-VI-4-W1-2020-167-2020.

Vitalis, Stelios, Ken Arroyo Ohori, and Jantien Stoter. 2020. ‘CityJSON in QGIS: Development of an Open-source Plugin’. *Transactions in GIS* tgis.12657. doi: 10.1111/tgis.12657.

Voutos, Yorghos, Phivos Mylonas, Evaggelos Spyrou, and Eleni Charou. 2017. ‘A Social Environmental Sensor Network Integrated within a Web GIS Platform’. *Journal of Sensor and Actuator Networks* 6(4):27. doi: 10.3390/jsan6040027.

Weglarz, Geoffrey. 2004. ‘Two Worlds of Data – Unstructured and Structured’. *DM Review*.

Westerholt, Rene, and Bernd Resch. 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'. *Transactions in GIS* 19(3):455–79. doi: 10.1111/tgis.12104.

Yao, Zhihang, Claus Nagel, Felix Kunde, György Hudra, Philipp Willkomm, Andreas Donaubauer, Thomas Adolphi, and Thomas H. Kolbe. 2018. '3DCityDB - a 3D Geodatabase Solution for the Management, Analysis, and Visualization of Semantic 3D City Models Based on CityGML'. *Open Geospatial Data, Software and Standards* 3(1). doi: 10.1186/s40965-018-0046-7.

Zhang, Xiaomin, Wei Song, and Liming Liu. 2014. 'An Implementation Approach to Store GIS Spatial Data on NoSQL Database'. Pp. 1–5 in *2014 22nd International Conference on Geoinformatics*. Kaohsiung, Taiwan: IEEE.

Zlatanova, Sisi, and Jantien Stoter. 2006. 'The Role of DBMS in the New Generation GIS Architecture'. Pp. 155–80 in *Frontiers of Geographic Information Technology*, edited by S. Rana and J. Sharma. Berlin/Heidelberg: Springer-Verlag.