DBL SmartCity: An Open-Source IoT Platform for Managing Large BIM and 3D Geo-Referenced Datasets

Siniša Kolarić
Digital Building Laboratory
College of Design
Georgia Institute of Technology
sinisa.kolaric@design.gatech.edu

Dennis Shelden
Digital Building Laboratory
College of Design
Georgia Institute of Technology
dennis.shelden@design.gatech.edu

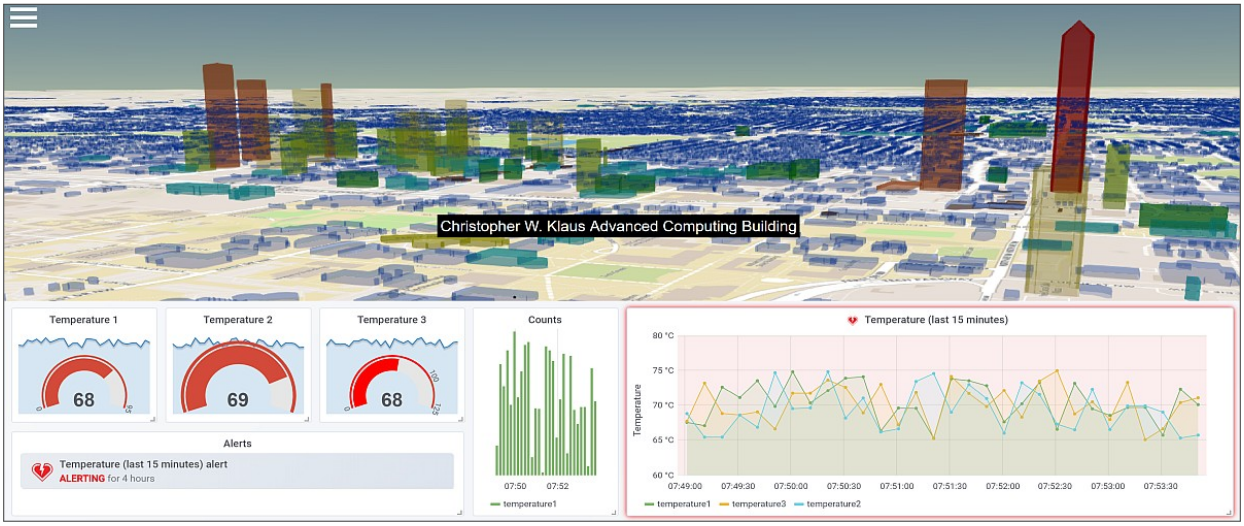

Figure 1. A partial screen capture of the *DBL SmartCity* user interface.

Abstract

The 'smart city' approach has been promoted as an effective way to manage urban environments. Information and communication technology in general, as well as 'Internet of Things' systems in particular, constitute an essential component of all smart city initiatives. However, many past and current smart city implementations place only an insufficient emphasis on the geo-spatial and 3D nature of data. In order to fill this gap, we present DBL SmartCity*, an open-source smart city IoT platform that is based on open standards and designed from the ground-up to effectively store, manage, and present large sets of BIM and 3D geo-referenced data.*

1. Introduction

Contemporary urban environments at different scales (neighborhoods, university campuses, airports, and towns, as well as cities, megacities, and metropolises) nowadays face a number of management challenges, such as energy consumption, air quality, waste management, noise levels, building structure monitoring, and traffic congestion [1]. Furthermore, cities and urban environments should be able to maintain their competitiveness, develop human capital, encourage civic participation, and improve on quality of life [2].

To meet these challenges, various *smart city* initiatives have been proposed both in industry and academia [3, 4, 5]. As one distinguishing characteristic of such initiatives, the information and communication technology (ICT) as well as 'Internet of Things'(IoT) networks in particular have been deemed to be one of their essential components [6, 2, 3].

However, as new IoT smart city systems and platforms continue to be proposed both in academia and industry, there remains a dearth of IoT implementations that place a strong emphasis on the 3D and spatio-temporal aspects characteristic for built environments, and that are open-sourced as well as firmly based on open data standards. To that end, we introduce *DBL SmartCity*, an IoT platform

URI: https://hdl.handle.net/10125/59636
ISBN: 978-0-9981331-2-6

for managing IoT data that is *open*, *customizable*, *scalable*, and strongly geared towards *3D geospatial and georeferenced data*.

In the remainder of this article, we discuss the high-level design principles that guided us during the conceptualization phase. We also discuss some of the decisions we faced when defining the platform's architecture, as well as detail the overall data model, and some of the main data transforms.

2. Motivation and Related Work

Our motivation for implementing the *DBL SmartCity* platform is to provide an *open-source smart city system/environment for managing and interacting with large, heterogeneous, and temporal 3D geospatial and IoT datasets*. Such datasets may include but are not limited to temporal 3D representations of buildings, infrastructure, vegetation, as well as point clouds, photogrammetry, and general 3D vector and scalar data related to various georeferenced IoT sources and sinks of data.

2.1. Smart City

In the first place, we are inspired by the 'smart city' vision, and how technology can help fulfill the same. To that end, what constitutes a 'smart' city?

In an early analysis, Giffinger et al [6, 2] consider that 'smart' cities are those that rank highly along the six dimensions of: (i) economy, (ii) people, (iii) governance, (iv) mobility and ICT, (v) natural environment, and (vi) quality of life. Nam and Pardo [3] similarly suggest that smart cities excel along the three dimensions of (i) technology, (ii) people, and (iii) institutions. Chourabi et al [7] identify eight factors underlying smart city initiatives, that of (i) management and organization, (ii) technology, (iii) governance, (iv) policy context, (v) people and communities, (vi) economy, (vii) built infrastructure, and (viii) natural environment. Schaffers et al [8] regard smart cities as *'living labs'* that encourage user-driven innovation. An exhaustive overview of prior smart city initiatives is beyond the scope of this article, however we point the reader to an excellent introductory survey by Cocchia [4].

2.2. Internet of Things (IoT)

The expression 'Internet of Things' refers to a novel technological paradigm where billions of objects connected through the Internet have the ability to sense, actuate, communicate, and process information. IoT represents one of the cornerstones of any smart city initiative [1, 9]. The very concept of IoT itself was initially suggested in the context of an early supply chain management application [10]. However, many other applications of IoT are possible, including of course smart cities [9] and smart buildings [11]. Survey papers such as [12, 11] provide lists of additional potential future applications, such as infrastructure monitoring, energy management, and vehicle fleet management.

One aspect of IoT systems and platforms that we are particularly interested in are the *architectures* of IoT systems and platforms. For instance, Khan et al [12] present a generic IoT architecture that consists of five layers (perceptual, network, middleware, application, and business layer). Li et al [13] propose a generic service-oriented architecture (SoA) which consists of four layers (sensing, network, service, and interface layer). In a survey paper, Al-Fuqaha et al [14] describe four different ways to conceptualize IoT system architectures: (a) three-layer (perception, network, and application layers), (b) middleware based (edge, access, backbone network, coordination, middleware, and application layers), (c) service-oriented architecture (SOA) based (objects, object abstraction, service management, service composition, and applications layers), and (d) five-layer architecture (objects, object abstraction, service management, application, and business layer). Krylovskiy et al [15] on the other hand report an IoT architecture based on microservices, and list its relative advantages when compared to more generic SOA approaches (such as simplified design and implementation of individual services, the ability to work independently on different features, and reduced amount of coordination), as well as disadvantages (increased system complexity, and delayed 'eventual consistency' of data). Rathore et al [9] present a 'big data' IoT approach to urban planning and design of smart cities, suitable for storing and processing large amounts of data inherent to any IoT platform. In industry, influential reference IoT platform architectures have been proposed by IBM [16], Cisco [17], Microsoft [18], Amazon [19], and Google [20].

Although these projects come a long way in realizing the vision of IoT, no single architecture exists that is suitable across all vertical and horizontal applications [21]. In addition, there is a dearth of IoT implementations that support 3D data generated in the context of urban environments from the outset. The recently announced 'Cesium ion' cloud architecture [22] supports management and streaming of large 3D geo-referenced datasets, however with no native support for IoT. In addition, the Cesium ion is a commercial, closed-source platform, while our *DBL SmartCity* implementation is non-commercial and based

on open source.

2.3. Open Standards for Geospatial Data

In order to promote interoperability and free data exchange, we are also interested in creating an IoT platform that adopts open data formats and open data exchange protocols. In the realm of geospatial, building information modeling (BIM), and the architecture, engineering and construction (AEC) disciplines, some of the most relevant organizations that maintain open standards include:

- Open Geospatial Consortium (OGC) [23]
- Joint W3C/OGC Committee (JWOC) [24]
- Open Source Geospatial Foundation (OSGeo) [25]
- Cesium Consortium [22]
- Khronos Group [26] and
- buildingSMART [27]

For instance, OGC maintains CityGML, an open data model and data exchange format for encoding different aspects of urban environments and landscapes, such as semantics, geometry, topology, and appearance [28, 29]. As one serious shortcoming however, CityGML does not support interaction with large datasets. The 3D City Database, based on CityGML and running atop spatially-enabled SQL databases (Oracle and Postgres), is an open source project that maintains a relational database schema for storing 3D city models and other geospatial data [30] but which likewise lacks large-dataset capabilities.

Yet another open format, that of Keyhole Markup Language (KML), was initially developed by Google for its Google Earth product, and has in the meantime been adopted as an OGC standard for expressing 2D/3D annotation and visualization of geospatial data [31]. While KML can be considered to be complementary to CityGML/GML, the vector data (*.vctr) format in the Cesium 3D Tiles specification [32] was specifically devised as a replacement for KML, and is suitable for deployment in large-datasets scenarios.

In architecture, engineering and construction (AEC) industry, the Industry Foundation Classes (IFC) data model [33] is a commonly used collaboration format in Building Information Modeling (BIM) which promotes interoperability by providing a platform-neutral, open format for building and construction data. While semantically rich, IFC format is likewise ill-suited for encoding and interacting with large datasets of 3D building and infrastructure models.

2.4. Interacting with Large Streaming 3D Datasets in Browser-Based Applications

Although existing standards fulfill many important requirements such as openness and interoperability, they are unsuitable for the purposes of efficiently interacting with voluminous datasets typically generated in smart city, IoT scenarios. In a major recent development, Analytical Graphics Inc. (AGI) present 3D Tiles [32], an open specification and reference JavaScript implementation for streaming massive heterogeneous 3D geospatial datasets. Considering its overall aims, we adopted Cesium 3D Tiles as one of the cornerstones of our platform, since it provides an efficient way to load, visualize, expose to interaction, and unload only subsets of any large set of geo-referenced 3D geometric data.

Related work confirms the effectiveness of the 3D Tiles specification. For instance, Chaturvedi et al [34] successfully utilized Cesium in order to visualize, explore, and interact with large 3D city models integrated with time series of sensor data. Schilling et al [35] likewise discussed how to convert CityGML models into 3D Tiles format in order to efficiently stream large sets of city models. In another application, Gan et al [36] integrated digital surface models (DSM) of oblique photogrammetry with 3D Tiles models, in order to enhance 3D presentations of buildings.

3. Platform Design

In concordance with the above, our high-level design requirements include: the use of open source licensing, the use of open data formats and exchange protocols, powerful 3D capabilities, and scalability.

I. Open-sourced. The platform's source code should be released under a permissive license so that it can be easily inspected, corrected, and extended by any interested party. This allows for unrestricted extensibility and customizability, limited only by a programmer's skillfulness and time. Additionally, the platform itself should be implemented using only open-sourced libraries, data stores, and processing frameworks.

II. Open data standards. Interoperability and data exchange in IoT is difficult, due to the existence of proprietary data formats and protocols [37]. This requirement, accordingly, stipulates that the platform should be based on open data standards, including:

- *Open data formats.* The system should use open, non-proprietary formats to encode data, in order to prevent interoperability issues and vendor lock-in. As an example, the platform should be able to support

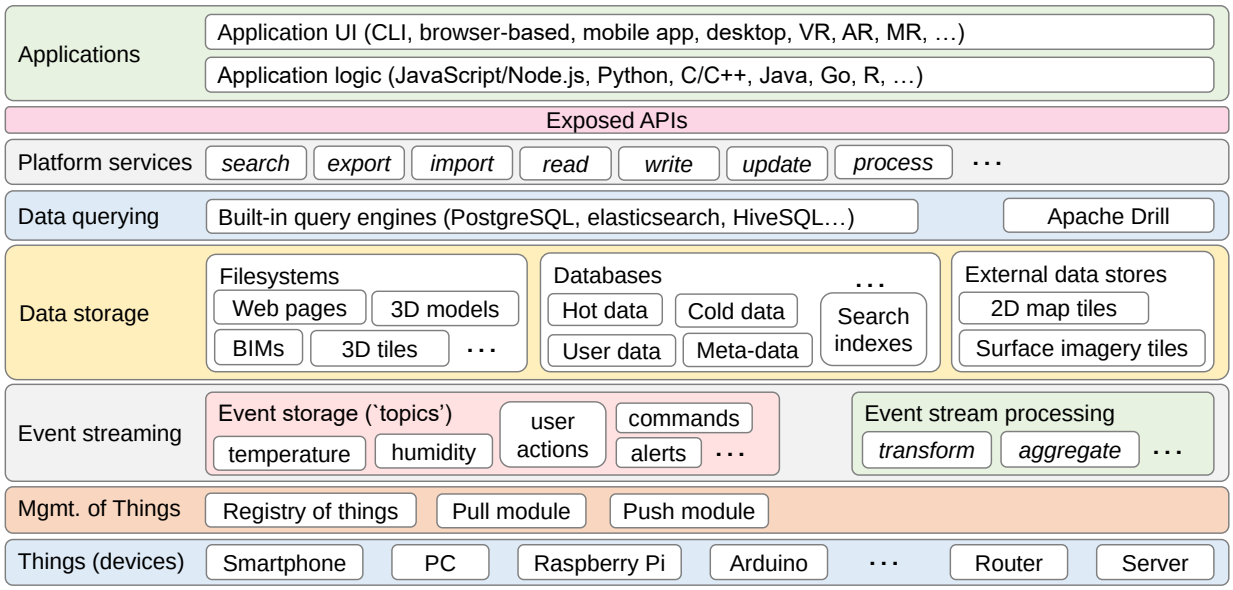

Figure 2. Summary architecture of the *DBL SmartCity* platform.

both the XML markup language and the JSON format [38] to encode IFC BIM data.

- *Open data exchange protocols*. The system should use open data exchange protocols in order to enable free, unencumbered communication.

III. Powerful 3D visualization abilities. The environment should provide for effective 3D visualization capabilities, including (a) representations of objects, entities, and processes commonly encountered in urban environments, as well as (b) means to annotate and style such representations based on derived data such as statistical measures, indicators, and other types of secondary data.

IV. Scalability. "Big data" and "big analytics" are essential capabilities of effective IoT systems [39]. The *DBL SmartCity* platform should thus provide for the ability to scale [40] effortlessly with a growing amount of data or processing load. This includes:

1. *Data and processing scalability.* From (a) data scalability from low to high data volumes, as well as (b) backend processing load scalability, from low to high processing (compute) demands.
2. *Interaction scalability.* The user interface should be able to efficiently handle interaction with even very large geospatial temporal datasets.

To cover the data and processing scalability requirement, the *DBL SmartCity* platform is architected so that it can run on a range of computing devices, from a personal computer to a large cluster of inexpensive computers running 'big data' computing frameworks. For the interaction scalability, we achieve that in the first place by adopting the Cesium library for streaming of large 3D sets of data.

4. Platform Architecture

The platform follows a multi-tier architecture (Figure 2) through which data is collected, processed, and finally presented to the users for interaction. The tiers include: (i) 'things', (ii) management of 'things', (iii) event streaming, (iv) data storage, (v) data querying, (vi) platform services, (vii) APIs, and (viii) applications.

4.1. 'Things' (IoT Devices)

This layer includes various 'things' in the Internet of Things, which in turn control a variety of nodes that can act either as sources of data (such as sensors), or sinks of data (such as actuators), or both (such as a memory location describing the intensity of a light source). Things incorporate devices which are in many cases under-powered and possess limited networking capabilities. (In further text, we use the term 'IoT device', or simply 'device' when clear from the context, to denote a 'thing' in the Internet of Things.) The schema (ontology) of 'things' or IoT devices has been heavily influenced by the Sensor Model Language (SensorML), which is maintained by OGC within its Sensor Web Enablement (SWE) stream, and which was itself later drafted as the ISO 19156:2011 standard. SensorML defines components and processes

associated with the measurement and post-measurement transformation of observations [41].

In terms of implementation, various types of hardware device can be registered with the platform, by first creating a unique device id and then using a unique, automatically-generated 'connection string' within a software client running on the hardware.

4.2. Management of Things

Within the *DBL SmartCity* platform, this layer includes the following main functions:

- Discovery, registration, and de-registration of devices to be used with the platform.
- Receiving data from devices, for instance by polling a published sensor data feed, or public transit feed.
- Sending data to devices, for instance sending a command to an actuator.

In the platform, all the registered (deregistered) IoT devices are stored into (and, respectively, removed from) a corresponding database schema. A server-side process written in Node.js iterates over this schema, and either (a) receives 'pushed' message from registered devices, or (b) periodically polls registered public data feeds (such as those provided by public transit agencies).

4.3. Event Streaming

Both the devices as well as users generate various types of events, such as for instance the event of generating a sensor value (in the case of devices), or the event of clicking on a GUI element such as a hyperlink or button (in the case of users). This layer is in charge of asynchronously collecting and processing data (or 'messages') generated by such events.

For serialization of queue messages, due to its low memory footprint and exception performance we used NATS.io message broker, and the associated NATS Streams library for stream processing. For clustered instances, we used Apache Kafka in conjunction with Kafka Streams, a combination which provides for scalable, high-throughput, fault-tolerant stream processing of live data streams.

4.4. Data Storage

Once the streaming data has been ingested through the Event streaming layer, it is serialized into the data layer.

Filesystems. For storing files such as web pages, client-side scripts, style sheets, BIM data (both proprietary and open), and 3D model data. Converters are used in order to convert 3D and BIM data, in both open (IFC) as well as proprietary (Autodesk Revit) formats, into an open format such as 3D tiles format.

Databases. Various databases (SQL and NoSQL) store both the real-time (RT) as well as historical time-series data collected from, and sent to, 'things' registered with the platform.

External data stores. We use external data sources (providers) foremostly for 2D data such as maps, satellite imagery, and terrain datasets. Such datasets are very large, and are thus partitioned into hierarchical tiles. The Cesium virtual globe library provides the means to easily include surface imagery from a number of third-party providers, such as MapBox and Microsoft Bing.

4.5. Data Querying

In the data querying layer utilizes both (a) built-in engines (such as those provided by databases natively), (b) as well as more specialized search engines, such as elasticsearch and Apache Drill.

Built-in search engines work well for low to moderate volumes of data. For large volumes of data, Apache Drill offers the ability to connect to all sources of data, including sources of structured (SQL), partially structured (NoSQL, JSON, CSV, XML), and unstructured data (filesystems containing various files such as documents, images, and photos). It also scales well over many computers, with fast-executing queries.

4.6. Platform Services

This layer contains various computational services that connect to data stores through the data querying layer, retrieve data, perform processing on data, and potentially write processed data back into the stores. Services themselves can be written in any language (JavaScript, Python, Scala, Java, Go, shell scripting, ...), and then executed within environments and frameworks such as Node.js, MapReduce, and Spark.

4.7. Exposed APIs

A selection of services implemented at the lower level are exposed through their own public REST API. In general, user-level applications interface with this layer, implemented using Node.js, in order to implement required functionality. For instance, an API allows the user to both read (download) as well as write (upload) 3D models and datasets.

4.8. Applications

Finally, using the exposed APIs, developers can write applications, or 'apps', which allow their users to view data and issue task commands, manage 3D/IoT/geospatial data, register and register IoT devices, navigate regions of Earth, conduct real-time monitoring and analytics on geo-referenced and 3D-object-linked sensor data, as well as visualize energy simulation data, again in the context of 3D/georeferenced/building models. Within this layer, the *application logic* sub-layer contains logic for the applications built on top of the system. For networking and web serving capabilities on the backend, we again used Node.js, an event-driven, asynchronous environment for building scalable network applications, such as web servers. Node.js is particularly well-suited for stateful applications, where network communication between the application and the server must persist during an interaction session. Finally, the *application UI* (or presentation) sub-layer offers the means to implement user interfaces in various interaction paradigms, such as command-line, menu-driven, graphical, web-based, 3D user interfaces, Virtual Reality (VR), Augmented Reality (AR), and Mixed Reality (MR) interfaces.

To demonstrate the platform's viability, we prototyped several 'apps' that showcase interactive features, such as (a) choosing the time interval, (b) choosing the source of data, and (c) choosing the visualization type (chart, histogram), as follows:

1. *Real-time monitoring and alerting.* This bundled app, already shown on the first page (Figure 1), allows the user to monitor and manage event data as they stream into the real-time data pipeline.
2. *Analytics.* This app (Figure 3) enables the user to analyze and study cold (historic, consolidated) data in order to obtain insights, perceive trends, and thus acquire a better understanding beyond the 'merely' operational sense, as provided by the real-time event monitoring app.

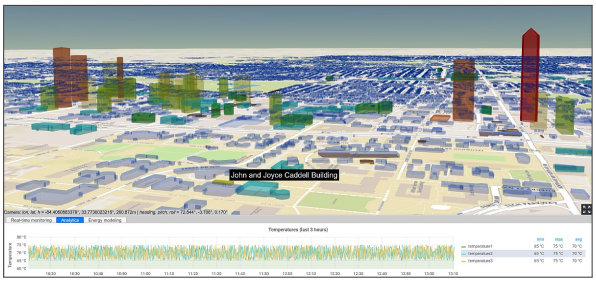

Figure 3. Visual analytics app.

3. *Energy modeling app.* This app allows the user to visualize the proportions and amounts of both as-measured as well as simulated energy expenditure data, using pie-charts as well as superposed line charts (Figure 4). The energy simulation data is currently being imported from EnergyPlus [42] and OpenStudio applications [43], with a backend simulation planned for a future version of the platform.

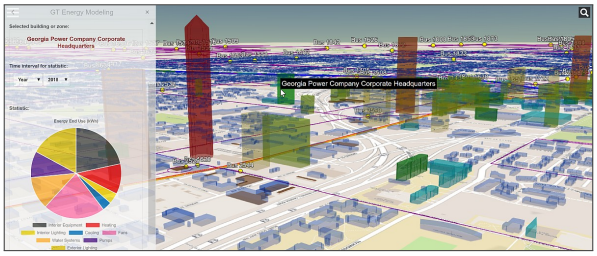

Figure 4. Energy modeling app.

4. *Public transit app.* This app polls a public feed issued by a transit agency serving a large metropolitan area in south-eastern United States, and allows the user to visualize real-time locations of active vehicles.

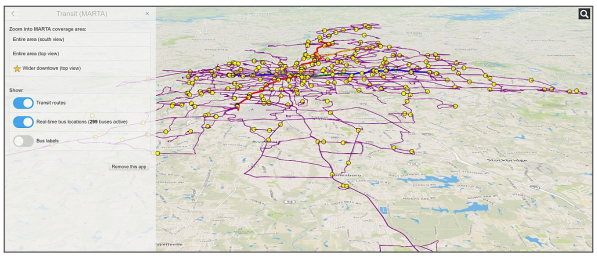

Figure 5. Public transit app.

Figure 5 depicts a total of 299 buses being active at the time of writing, while showing most of the agency's coverage area as seen from a high attitude.

5. Data Model

In addition to the overview of physical stores of data given in the previous section, this section provides an abstracted, logical view of the data managed in the *DBL SmartCity* platform.

5.1. Cesium 3D Tiles

All the data managed in the *DBL SmartCity* platform is anchored around the concepts of nested, hierarchical *tiles*. Tiling, as a way to progressively decompose (and then load) parts of the Earth surface, has been a known concept for years in digital mapping. Building

upon the glTF format by Kronos, the '3D Tiles' format by the Cesium Consortium supports streaming of massive heterogeneous 3D and geospatial data. As per specification, *features* are packed into *3D tiles*. 3D Tiles are packed, in turn, into hierarchical structures called *3D tilesets*, which follow the same precepts at those for 2D mapping tiles.

3D "Features." A feature is the smallest addressable component of any 3D tile. For example, a feature may represent a 3D model of a building, a 3D instance of a tree model, a point in a 3D point cloud, or a vector (polyline, rectangle, sphere, ...) embedded in 3D space.

3D Tiles. A number of features may be packed into a Cesium 3D tile which is, in essence, a container that encloses a region of 3D space in the WGS84 terrestrial reference system. It is defined by the six parameters of west, south, east, north, minimum height, and maximum height:

```
{
  "boundingVolume": {
    "region": [
      -1.3197351054587685, // (in radians)
      0.698829173006603,
      -1.3196243108779218,
      0.6989177498576029,
      0.0, // (in meters)
      271.4
    ]
  }, ...
}
```

Other parameters defining a Cesium 3D tile include the 'refine' property which stipulates whether the tile's 3D geometry, as the user zooms into it, is refined by either (a) replacing it fully with children tiles' 3D geometry (value 'REPLACE'), or (b) by adding children tiles' 3D geometry to it ('ADD').

A 3D tile can contain any of the following: a batch of 3D models (*.b3dm) normally used for buildings, a set of 3D instances (*.i3dm), 3D point cloud (*.pnts), and 3D vector data (*.vctr). For each feature, its attributes (such as latitude, longitude, height, ...) must be encoded as a data structure called *batch table*, and that is embedded into any *.b3dm tile, for example:

```
...
{
  "id" : "47",
  "name" : "Architecture Building (West)",
  "longitude": "-84.396109",
  "latitude": "33.776099",
  "zip" : "30318",
  ... (other attributes)
},
{
  "id" : "48",
  "name" : "John and Joyce Caddell Building",
  "longitude": "-84.397024",
  "latitude": "33.777548",
  "zip" : "30318",
  ... (other attributes)
},
...
```

The partial JSON snippet above defines which attributes will be associated with what features (i.e., buildings) in the resulting 3D tile.

3D Tileset. A 3D 'tileset' is a set of tiles that are organized into a hierarchical, spatially coherent data structure (such as octrees and k-d trees). A 3D tileset's bounding volume thus encloses all the constituent tiles' bounding volumes. A 3D tileset is basically defined by its associated tileset.json file with the following structure:

```
{
  "root": {
    "refine": "add",
    "content": {
      "url": "0/0/0.b3dm"
    },
    "children": [{
      "content": {
        "url": "1/0/0.b3dm",
      },
      "children": [...]
      }, {
        "content": {
          "url": "1/1/0.b3dm",
        },
        "children": [...]
      }, {
        "content": {
          "url": "1/1/1.b3dm",
        },
        "children": [...]
      }]}
}
```

As seen above, the tileset root contains a number of children tiles, which in turn may contain their own children tiles, and so on. (For brevity, some attributes in tileset.json, such as bounding volumes, have been omitted.)

5.2. Database Schemas

A number of different database schemas are used in the *DBL SmartCity* platform. All of these schemas can co-exist in a single database such as Postgres, with PostGIS and TimescaleDB extensions installed. Alternatively, each database schema may be installed in its own separate cluster of computers, if there is a need to scale out a particular deployment of the *DBL SmartCity* platform.

Hot data. This database schema stores recent, high-velocity data, for the purposes of real-time monitoring, alert management and response, as well as real-time analytics. The incoming, high-velocity data can either be of low volume, or high-volume (in this case, a separate cluster for hot data intake may be required). In both cases, however, incoming data is a time-series data, or data that is timestamped according to the exact date and time of production, or ingestion, or both. In *DBL SmartCity*, we implemented the ingestion

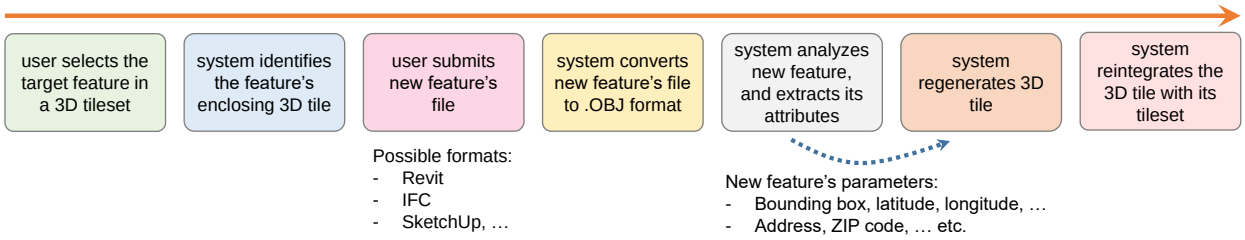

Figure 6. A data transform example: the sequence of steps for re-generating a 3D tileset.

of time series data through TimescaleDB, a time-series database (TSDB) extension for Postgres. For large-scale installations, we implemented an Apache Cassandra cluster.

Cold data. This schema is concerned with storing, processing, and accessing historical data, which includes both the real-time as well as past data. For small installations, the cold data database schema is equivalent to the hot data database schema, and can be merged with the hot data schema. In big data scenarios, however, the cold data schema may be migrated to a specialized cluster running a 'big data' storage and processing framework. In some cases, cold data may also be further de-normalized for effective storage, retrieval, and analytics over long periods of time.

Meta-data. For efficient and speedy intake, storage, indexing, and retrieval, both the hot data and cold data schemas are normalized. In other words, as the data is queried, additional relational tables, or columns and documents in the case of NoSQL databases, must be consulted in order to fully present data.

Search indexes. For modest use cases, search indexes can be stored in the single database (Postgres). Alternatively, external search indexes can be used, as is the case with Cesium's *geocoding* service, which in turn uses the Bing Maps Locations API offered by Microsoft Inc. In case of large installations, *DBL SmartCity* utilizes elasticsearch, an open-source, distributed search engine built on Apache Lucene.

6. Data Transforms

This section reviews some of the data transforms that take place within the *DBL SmartCity* platform. By 'transforms' we denote computational processes (implemented as 'services') that change or alter the form or appearance of data.

6.1. Generating Cesium 3D Tiles

We have implemented a set of routines for generating 3D tilesets that contain buildings at level-of-detail (LOD) level 100, and which mirror the series of steps shown in Figure 6. Higher LOD levels (200, 300, 400 and 500) are planned for future revisions of the platform. A 3D tileset is best understood as a depository of data (or a spatial database) that *facilitates interaction* with 3D geo-referenced data. For maintaining the 'ground truth' (including the associated meta-data) we utilize semantically richer types of data such as BIM data (IFC, Revit) as well as 3D model data, including the OpenStreetMap (OSM) export files [44]. 3D tiles are regenerated, modified, and updated whenever the user updates any of the 'ground truth' sources of data (Figure 6), such as for example an OSM extract:

1. *Obtain a regional OSM extract (tile).* Since our intent is to first visualize buildings at highest LOD level in our area, we first obtained a regional OSM extract, containing all entities present in such a format (buildings, roads, infrastructure, and other entities).
2. *Obtain initial OSM 3D buildings tile.* From the initial OSM extract, we derive a filtered OSM extract containing just the 3D building data.
3. *Convert OSM tile to OBJ tile.* We convert the filtered OSM extract tile into .OBJ Wavefront format.
4. *Convert OBJ tile to a set of 3D objects.* We loop over all objects in the OBJ tile, and then calculate/generate parameters for each object, for instance its name, geographic coordinates, bounding box, its height, followed by exporting those parameters into a JSON batch table, listing parameters per each object.
5. *Convert each OBJ tile into a 3D tile.* We convert each OBJ tile into a Cesium .B3DM file as well as the associated tileset.json file, using the batch tables generated in the previous step.
6. *Combine 3D tiles.* Finally, we merge separate tiles into a combined tileset, thus producing a generalized tileset.json file that reference each single tiles' tileset.json files as 'external' tilesets.

Re-generating 3D tiles. Any time a 'ground truth' 3D building model (such as a geo-referenced Autodesk Revit model) has been modified, the corresponding 3D tile must be regenerated too, following the same series of steps shown in Figure 6. Within the platform, a Revit model is first converted into an IFC file, and then into an .OBJ Wavefront file.

6.2. Styling of 3D Tiles

In order to support rich 3D visualizations of streaming geometrical and other data, Cesium provides means through which features can be dynamically (i.e., at runtime) *styled* by changing their color and transparency, as well as setting their visibility. A style is defined using JSON and (optionally) a small subset of JavaScript. Moreover, styling can be linked to attributes that have been previously encoded with features into a 3D tile (see §5.1).

```
{
  "show" : "${zip} === '94301'",
  "color" : {
    "conditions" : [
      ["(${height} > 100.0)", "color('red')"],
      ["(${height} > 50.0)", "color('blue')"],
      ["(${height} > 10.0)", "color('green')"]
    ]}
}
```

As an example, the snippet above will hide all buildings except those with ZIP code equal to 94301. Moreover, shown buildings will be rendered in red color if they are taller than 100.0 m, in blue if taller than 50.0 m, and in green if taller than 10.0 meters.

The final Cesium scene, as displayed to the user in the *DBL SmartCity* platform, consists of a number of superposed 2D tilesets generated from various surface imagery datasets, as well as a global, styled 3D tileset containing the following types of data:

1. *Building exteriors / envelopes.* These show features (shells or envelopes of buildings) at different LOD levels, and with a combination of ADD and REFINE modes determining how the current point of navigation and children tiles affect the subset rendered.
2. *Building interiors.* Building interiors, especially for large buildings, can be substantial in terms of data.
3. *Thermal zones.* Thermal zones are spaces within building that represent a unit of spatial decomposition for the purposes of energy simulations. However, thermal zones are not always in 1:1 mapping with existing spaces and rooms in a building. A separate layer of information represents spatial decompositions of buildings customized for this particular implementation.

In addition to the types of 3D data above, and as a final step in the generation of the complete Cesium scene, 3D tiles are frequently annotated using Cesium non-tiles formats and data types (Cesium 'Entities'), as required by a specific application module.

7. Conclusions and Future Work

We have presented the *DBL SmartCity* platform, an open and scalable IoT implementation that was designed from ground-up for efficient managing, streaming, and scalable interaction with large 3D, geo-referenced, and IoT datasets. In future work, we plan to conduct performance evaluations of the platform, as well as conduct formative usability evaluations of presented applications.

References

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things journal*, vol. 1, no. 1, pp. 22–32, 2014.

[2] R. Giffinger and H. Gudrun, "Smart cities ranking: an effective instrument for the positioning of the cities?," *ACE: Architecture, City and Environment*, vol. 4, no. 12, pp. 7–26, 2010.

[3] T. Nam and T. A. Pardo, "Conceptualizing smart city with dimensions of technology, people, and institutions," in *Proceedings of the 12th annual international digital government research conference: digital government innovation in challenging times*, pp. 282–291, ACM, 2011.

[4] A. Cocchia, "Smart and digital city: A systematic literature review," in *Smart city*, pp. 13–43, Springer, 2014.

[5] L. G. Anthopoulos, "The rise of the smart city," in *Understanding Smart Cities: A Tool for Smart Government Or an Industrial Trick?*, Springer, 2017.

[6] R. Giffinger, C. Fertner, H. Kramar, R. Kalasek, N. Pichler-Milanović, and E. Meijers, "Smart cities: Ranking of european medium-sized cities," 2007. http://www.smart-cities.eu; accessed on May 20, 2018.

[7] H. Chourabi, T. Nam, S. Walker, J. R. Gil-Garcia, S. Mellouli, K. Nahon, T. A. Pardo, and H. J. Scholl, "Understanding smart cities: An integrative framework," in *System Science (HICSS), 2012 45th Hawaii International Conference on*, pp. 2289–2297, IEEE, 2012.

[8] H. Schaffers, N. Komninos, M. Pallot, B. Trousse, M. Nilsson, and A. Oliveira, "Smart cities and the future internet: Towards cooperation frameworks for open innovation," in *The future internet assembly*, pp. 431–446, Springer, 2011.

[9] M. M. Rathore, A. Ahmad, A. Paul, and S. Rho, "Urban planning and building smart cities based on the internet of things using big data analytics," *Computer Networks*, vol. 101, pp. 63–80, 2016.

[10] K. Ashton, "That 'Internet of Things' Thing." `www.rfidjournal.com/articles/view?4986`, 2009. Online; accessed on May 20, 2018.

[11] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad hoc networks*, vol. 10, no. 7, pp. 1497–1516, 2012.

[12] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future internet: the internet of things architecture, possible applications and key challenges," in *Frontiers of Information Technology (FIT), 2012 10th International Conference on*, pp. 257–260, IEEE, 2012.

[13] S. Li, L. Da Xu, and S. Zhao, "The internet of things: a survey," *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[14] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376.

[15] A. Krylovskiy, M. Jahn, and E. Patti, "Designing a smart city internet of things platform with microservice architecture," in *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, pp. 25–30, IEEE, 2015.

[16] IBM, "IoT Reference Architecture." `www.ibm.com/cloud/garage/architectures/iotArchitecture/reference-architecture`, 2018. Online; accessed on June 4, 2018.

[17] Cisco, "Internet of Things (IoT)." `www.cisco.com/c/en/us/solutions/internet-of-things/overview.html`, 2018. Online; accessed on June 4, 2018.

[18] Microsoft, "Internet of Things." `www.microsoft.com/en-us/internet-of-things`, 2018. Online; accessed on June 4, 2018.

[19] Amazon, "AWS IoT Services Overview." `aws.amazon.com/iot`, 2018. Online; accessed on June 4, 2018.

[20] Google, "Google Cloud IoT." `cloud.google.com/solutions/iot`, 2018. Online; accessed on June 4, 2018.

[21] S. Krco, B. Pokric, and F. Carrez, "Designing iot architecture (s): A european perspective," in *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pp. 79–84, IEEE, 2014.

[22] Cesium.com, "Cesium web site." `cesium.com`, 2018. Online; accessed on August 31, 2018.

[23] OGC, "OGC Home Page." `www.opengeospatial.org`, 2018. Online; accessed on August 31, 2018.

[24] JWOC, "Joint W3C/OGC Organizing Committee (JWOC)." `www.w3.org/2017/sdwig`, 2018. Online; accessed on August 31, 2018.

[25] OSGeo, "Open Source Geospatial Foundation (OSGeo)." `www.osgeo.org`, 2018. Online; accessed on August 31, 2018.

[26] Khronos.com, "Khronos Group." `www.kronos.org`, 2018. Online; accessed on August 31, 2018.

[27] buildingSMART, "buildingSMART." `www.buildingsmart.org`, 2018. Online; accessed on August 31, 2018.

[28] OGC, "CityGML." `www.citygml.org`, 2018. Online; accessed on June 4, 2018.

[29] T. H. Kolbe, "Representing and exchanging 3d city models with citygml," in *3D geo-information sciences*, pp. 15–31, Springer, 2009.

[30] "3D City DB." `www.3dcitydb.org`. Online; accessed on June 4, 2018.

[31] OGC, "KML Standard." `www.opengeospatial.org/standards/kml`, 2018. Online; accessed on August 31, 2018.

[32] I. AGI, "Cesium 3D Tiles: A specification for streaming massive heterogeneous 3D geospatial datasets." `github.com/AnalyticalGraphicsInc/3d-tiles`, 2018. Online; accessed on June 2, 2018.

[33] buildingSMART, "Ifc overview." `www.buildingsmart-tech.org/specifications/ifc-overview`, 2018. Online; accessed on August 31, 2018.

[34] K. Chaturvedi, Z. Yao, and T. H. Kolbe, "Web-based exploration of and interaction with large and deeply structured semantic 3d city models using html5 and webgl," in *Bridging Scales-Skalenübergreifende Nah-und Fernerkundungsmethoden, 35. Wissenschaftlich-Technische Jahrestagung der DGPF*, 2015.

[35] A. Schilling, J. Bolling, and C. Nagel, "Using gltf for streaming citygml 3d city models," in *Proceedings of the 21st International Conference on Web3D Technology*, pp. 109–116, ACM, 2016.

[36] L. Gan, J. Li, and N. Jing, "Hybrid organization and visualization of the dsm combined with 3d building model," in *Image, Vision and Computing (ICIVC), 2017 2nd International Conference on*, pp. 566–571, IEEE, 2017.

[37] B. Ahlgren, M. Hidell, and E. C.-H. Ngai, "Internet of things for smart cities: Interoperability and open data," *IEEE Internet Computing*, vol. 20, no. 6, pp. 52–56, 2016.

[38] K. Afsari, C. M. Eastman, and D. Castro-Lacouture, "JavaScript Object Notation (JSON) data serialization for IFC schema in web-based BIM data exchange," *Automation in Construction*, vol. 77, pp. 24–51, 2017.

[39] S. Mitchell, N. Villa, M. Stewart-Weeks, and A. Lange, "The internet of everything for cities: Connecting people, process, data, and things to improve the 'livability' of cities and communities." `www.cisco.com/c/dam/en_us/solutions/industries/docs/gov/everything-for-cities.pdf`, 2015. Online; accessed on June 4, 2018.

[40] S. Lehrig, H. Eikerling, and S. Becker, "Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics," in *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*, pp. 83–92, ACM, 2015.

[41] OGC, "Sensor Model Language (SensorML)." `www.opengeospatial.org/standards/sensorml`, 2018. Online; accessed on June 4, 2018.

[42] EnergyPlus.net, "EnergyPlus web site." `energyplus.net`, 2018. Online; accessed on August 31, 2018.

[43] OpenStudio.net, "OpenStudio web site." `openstudio.net`, 2018. Online; accessed on August 31, 2018.

[44] openstreetmap.org, "OpenStreetMap web site." `www.openstreetmap.org`, 2018. Online; accessed on June 15, 2018.