INTEGRATION AND VISUALIZATION OF HETEROGENEOUS SENSOR DATA AND GEOSPATIAL INFORMATION

Thunyathep Santhanavanich, Sven Schneider, Preston Rodrigues, Volker Coors

University of Applied Sciences Stuttgart
Schellingstraße 24
70174 Stuttgart, Germany
(thunyathep.santhanavanich, sven.schneider, preston.rodrigues, volker.coors)@hft-stuttgart.de

KEY WORDS: Sensors, Web Service, SOS, SensorThings, 3D City Models, CityGML, E-Mobility

ABSTRACT:

According to the advances in Information & Communication Technology (ICT) , nowadays, the use of Internet of Things (IoT) has become a normal part of daily life. It allows interconnections among a wide variety of devices and sensors such as smartphones, smartwatches, automobiles, or any object with a built-in sensor. However, these devices and sensors are developed by numerous different manufacturers which leads to technology lock-in in terms of data formats and protocols. In order of address this heterogeneity, an interoperable sensor protocol is the need of the hour. To address this, we propose a sensor data management system for monitoring *pedelec* usage and user fitness level. Using a proof-of-concept prototype the study is carried out in downtown of Stuttgart city. The result of the integrated analyzed data is visualized in 3D digital globe CESIUM.

1. INTRODUCTION

The digital transformation does not only concern companies and organizations, but indeed, entire cities. This has resulted in urban data becoming a *resource*. Hence it used as a corner stone in different applications targeting unique test case scenarios. Recent examples include urban mobility for public transport, car sharing-concepts, autonomous driving vehicles, waste management, energy efficient usage of resources as well as the provision of services of metropolitan administrations (Albakour et al., 2014). The fundamental idea behind smart cities is that ICT can be used for a sustainable, social and ecological design of the urban space. To realize this idea, a large collection of urban data needs to be integrated and analyzed.

Figure 1: *Pedelec* with IoT sensors

In many city administrations, there exists a large pool of urban data. This data is generally offered (at least partly) via geo data infrastructure initiatives. Additionally, a plethora of sensors data is provided under the overarching term "Internet Of Things" (IoT) which provide a perpetual stream of urban data. The study "Smart Cities Measure in Europe" has identified the crucial success factor for smart cities to be ❶ management of knowledge and ❷ access to data: "...access to the relevant data..." is as important as the guarantee of data privacy and data protection (Manville et al., 2014). Furthermore, a recent study (Kukka et al., 2013) has identified emerging information needs citizens often have in their public urban spaces. These information needs might be complex and are often not served by existing systems such as traditional web search engines. For example, these needs may include obtaining real-time information about events in the city or finding free parking spaces or the best bike rout for ones fitness level (Albakour et al., 2014).

An exceptional example to this is the city of Glasgow which has been awarded the "Excellent Award for Application of GeoSpatial Technology" for their project "Future City Glasgow" (GlasgowCityCouncil, 2018) as well as for their strategy for geo data acquisition, usage and processing. Another notable example is the Smart City Santander (Sanchez et al., 2014) which is a testbed with around 3000 WiFi capable devices on stationary and mobile platforms (e.g. on buses or a bike as in Fig. 1). They perform measurements of the environment using over 2000 IoT devices for measuring e.g. light, sound and CO_2 concentration. Furthermore, a parking management and traffic monitoring system informs about current traffic and availability of parking spaces (due to build in sensors under the parking lots). These are just few of the many examples how smart geo data can be used to enhance and evolve our live in (smart) cities.

Although data is available, access and integration is completely a different story. Different protocol, data formats and irregular time-series data streams are just some of the issues facing data integration. In order to address this heterogeneity, an unified urban information model is necessary. One such concept is the Open Urban Platform (OUP) which is a project lead by the Open Geospatial Consortium (OGC) named "SystEmic Standardisation apPRoach to Empower Smart citieS and cOmmunities (ESPRESSO-Consortium, 2018). The main goal of this project is to sustain the interoperability of smart cities solutions in order to integrate new ICT in an adequate way. In the first instance, this is happening by the development of a Smart City Information framework based on open standards such as CityGML and Sensor Things.

Since the IoT technology has improved over the decades, its advancement allows sensors to be controlled through a remote network, leading to the realization of a "Smart Cities concept (Con-

Table 1: Sensor protocol features comparison.

Features \Protocols	OGC SensorThings API	OGC SOS	ASDDS
Encoding	JSON	XML + (JSON in 52° North SOS 4.4)	JSON
Architectural Style	Resource Oriented Architecture	Service Oriented Architecture	Resource Oriented Architecture
Binding	REST	SOAP	REST
Insert new Sensors and Observation results	HTTP POST	SOS specific interface: InsertSensor() / InsertResultTemplate() / InsertResult() /	HTTP POST
3D Location of a Moving Sensor	Supported directly through [Location and Historical Location] Entities	Supported by SOS specific interface: Spatial observation()	Not supported directly. [Input as 3 Observation Values : Lat/Long/Height]
Deleting existing sensors	HTTP DELETE	OGC SOS Specific interface:	Not supported. DeleterSensor()
Deleting existing Observation	HTTP DELETE	Not supported	Not supported
Pagination	$top, $skip, $nextLink	Not Supported	Not Supported
Pub/Sub Support	MQTT	Not Supported	Not Supported
Sensor Metadata	Supported. (OGC O&M specification)	Supported. (OGC O&M specification)	Supported. (Name and Unit)
Updating properties of existing sensors or observations	HTTP PATCH and JSON PATCH	Supported	Not Supported
Linked data support	JSON-LD	Not Supported	Not Supported
Request multiple O&M Entities $*$	Using $expand.	Not Supported	Not Supported
Aggregation Function over the request	Not Supported	Not Supported	Supported. (Only mean value possible in this version)

The $*$ referres to, for example, `FeatureOfInterest` and `Observations` in one request/response.

sortium, 2017). To apply this concept to monitoring a *pedelec* in near real-time, integration of other dynamic sensors such as wearable sensor devices, smartphones, etc. becomes necessary. This integration allows researchers and developers to access various types of data. A recent study (Kohn, 2017) on the emotion analysis during *pedelec* usage found that, data from different sources provided by the *pedelec* sensors and smart wearable devices were difficult to integrate and aggregate due to a lack of a sensor network. In fact, different types of sensors provide different data formats, Application Programming Interface (API), and data model structures. This controversy leads to the interoperability problem (Jazayeri et al., 2015). In order to solve this issue, proper ways of communicating sensor locations, sensor and data parameters, and sensor instruction sets need to be addressed (Jo and Jang, 2016).

In this paper, we aim to study and compare existing sensor protocols. We target sensor protocols capability to integrate heterogeneous sensor systems together and provide their data in an efficient way. The challenge of this work is to maintain interoperability among different development layers, for example, the communication protocols and data models. Additionally, in order to solve issues related to irregular time-series data streams from numerous IoT devices, sensor networks must provide the capability to aggregate or interpolate the data over time. The rest of the paper is organized as follow: Section 2. discusses related works. Section 3. presents the proposed architecture. Section 4. details our prototype implementation and discusses sensor protocol evalation. And finally, Section 5. concludes this paper with pointers for future works.

2. RELATED WORK

Managing sensor data is a tedious task. This task becomes even more complicated when we need to integrate heterogeneous sensor systems. To address this issue, industry and academia have proposed several standards using web services. Advanced Sensor Data Delivery Service (ASDDS) is part of a student project (Storz and Dastageeri, 2017) that support aggregated results of sensor data. The main advantage of the proposed solution is to enable user to query for mean or average value of a sensed data over a specific period of time. Sensor Observation Service (SOS) (Na and Priest, 2007) is part of OGC Sensor Web Enablement (SWE) standard. It provides a standardised interface for accessing sensor data. SOS supports data access operations for retrieving both observation data `GetObservation` and sensor metadata `DescribeSensor`. Besides these data access operations, the SOS also offers a `transactional` interface which allows the insertion of sensors `InsertSensor` and their observed data `InsertObservation`. SensorThings (Liang et al., 2016) is another standard proposed by OGC targeting resource-constrained devices. It provides an open and unified framework to interconnect IoT devices. The framework provides an efficient API that enable resource-constrained IoT devices to share data among dif-

ferent applications. The Standard provides the following benefits, ❶ it permits the proliferation of new high-value services with lower overhead of development and wider reach. ❷ It lowers the risks, time and cost across a full IoT product cycle. ❸ it simplifies the connections between device-to-device and device-to-applications. Furthermore, it has a comprehensive support for location information. Moreover, it supports publish and subscribe mechanism using MQTT (Banks and Gupta, 2014) and linked data using JSON-LD (Sporny et al., 2014). Table 1 shows detail comparison of the three protocols. Due to these inherent advantages we choose to use SensorThings.

3. SYSTEM ARCHITECTURE

This section explains the overall system architecture. As depicted in Figure 2 it consists of four main steps, namely; **Data Source**, **Data preprocessing and Cleaning**, **Data Integration** and **User Application**. The functionality of each step is explained below.

❶ **Data Source:** Data is acquired from two sources. The first source, *Sensor or IoT Data Source* Ⓐ shows the three main data sources, namely; *pedelec*, smart-watch, and weather data. The *pedelec* and smart-watch data is collected from E-bike driven by volunteers from the Business Psychology department, HFT Stuttgart (Kohn, 2017). The study used a group of twelve female students aged 20 to 25 years. All participants were asked to take the same 10-kilometer bike route around Stuttgart city. To account for weather changes, open-source weather data for the Stuttgart city from OpenWeatherData was used.

The second source, *3D City Model Data Source* Ⓑ shows the 3D city model data of the Stuttgart city in CityGML format.

❷ **Data preprocessing and Cleaning:** The *Sensor/IoT Data Preparation* Ⓒ part is the intermediate processing for the *pedelec*, smart-watch data, and Open Weather Data. For the *pedelec* Transmission Control Unit (TCU) server, to set up the sensor network to receive data in a local server from the existing architecture, the first step is to deploy the STASH sensor network which will subscribe the *pedelec* data from a Vehicle Status Service (VSS) server directly into the local server. It is an implementation of the *pedelec* architecture from our *pedelec* provider which was fixed. With this solution, the *pedelec* data are sent as *Datastreams* to the server whenever there is an update data log on the TCU. However, due to the network security issue and adherence to data privacy laws, this could not be tested. A second solution, a workaround, was implemented to make communication from our server to the *pedelec* provider via File Transfer Protocol (FTP). With this, users can retrieve data every 24 hours and the *pedelec* TCU datastreams are collected into the VSS servers database. The drawback of this solution is that users cannot get the data in real-time which would otherwise be possible. For the Garmin Smart Watch data, the *Datastreams* are sent to the Garmin Connect server (Garmin, 2018) whenever the device is connected to the internet. Then, the data can be extracted into TCX or GPX format later on. For the Open Weather Data, the data source is already provided in JSON format and can be then parsed and imported directly into the SensorThings API server.

In the part of *3D City Model Data Preparation* Ⓓ, in order to process the web-based visualization of the 3D city models on the Cesiums web application with the input CityGML file, the data conversion is needed and can be done with the following solutions including 1) Using 3D CityDB to convert the data into glTF format, 2) Using FME Workbench to convert the data into Cesium 3D Tile format, and 3)Using Georocket GeoToolbox tool to convert the data into Cesium 3D Tile format.

❸ **Data Integration:** In the part of *Sensor/IoT Data Integration* Ⓔ, the sensor/IoT data in a different format are parsed and modeled in according to the OGC Observations and Measurements specification then imported into the OGC SensorThings API server. For the next part *3D City Model Data Integration* Ⓕ, the 3D city model in Cesium 3D Tile and glTF are prepared in a project folder to be used later on in the 3D Web-based application.

❹ **User Application:** This part is to build an application by utilizing the sensor data from the prepared sensor network which is done on step ❶ and ❷ together with the prepared 3D city model data from the third step ❸. The application will be built in 3D using the Cesium JavaScript library.

4. IMPLEMENTATION

4.1 The sensor network protocols evaluation

Firstly, the best suitable sensor network protocol from the three candidate protocols including 1) OGC SensorThings API, 2) OGC Sensor Observation Service (SOS), and 3) Advanced Sensor Data Delivery Service (ASDDS) is selected for this research. To do that, the evaluation of the technical performance on each sensor network protocol is made in the following metrics are used including 1) Request size of an operation, 2) Response time of an operation, 3) Response length of an operation, and 4) Support of dynamic 3D location of a moving sensor. For the request size of an operation, the SOS protocol needs a larger size of code and transaction on each request while the SensorThings API and the ASDDS use the simple HTTP GET method. For the response time of an operation, the response time of a single request has been evaluated and compared. A simple Node-JS application on the server-side is written to perform a request on one observation result and print the response time to the console window. The test result shows that the SOS takes more time until the response comes back. For the response length of an operation, the response length of the SOS provides much longer response length as an XML format if comparing to the other JSON-based standard. But, with the 52North SOS 4 implementation version, the response set of the value and time are available in JSON format and are embedded in the single response file. While the OGC SensorThings API and ASDDS provide the response as a repetitive attribute key-value pairs. However, to compensate for this repetitive drawback in SensorThings API, firstly, the users are capable to request only the needed attributed value, for example, only result value with the query capability over the HTTP request. Secondly, the SensorThings API is supported by service-driven `pagination` which the number of response body can be limited on each request. If the response size is larger than the limited number on the server-side, then the `nextLink` annotation is included in a response which represents the link to the next part of the whole response body. For the support of dynamic 3D location of a moving sensor, with the OGC SensorThings API, the 3D locations are supported in GeoJSON format. The location of the `Thing` entity can be updated through the HTTP PATCH method. After the geospatial location of the `Thing` entities have been updated, the existing location of the `Locations` entities will be collected in the `HistoricalLocations` entity. For the OGC SOS, the observation with the geospatial location is supported by specifying the sampling geometry of the observation in the GML

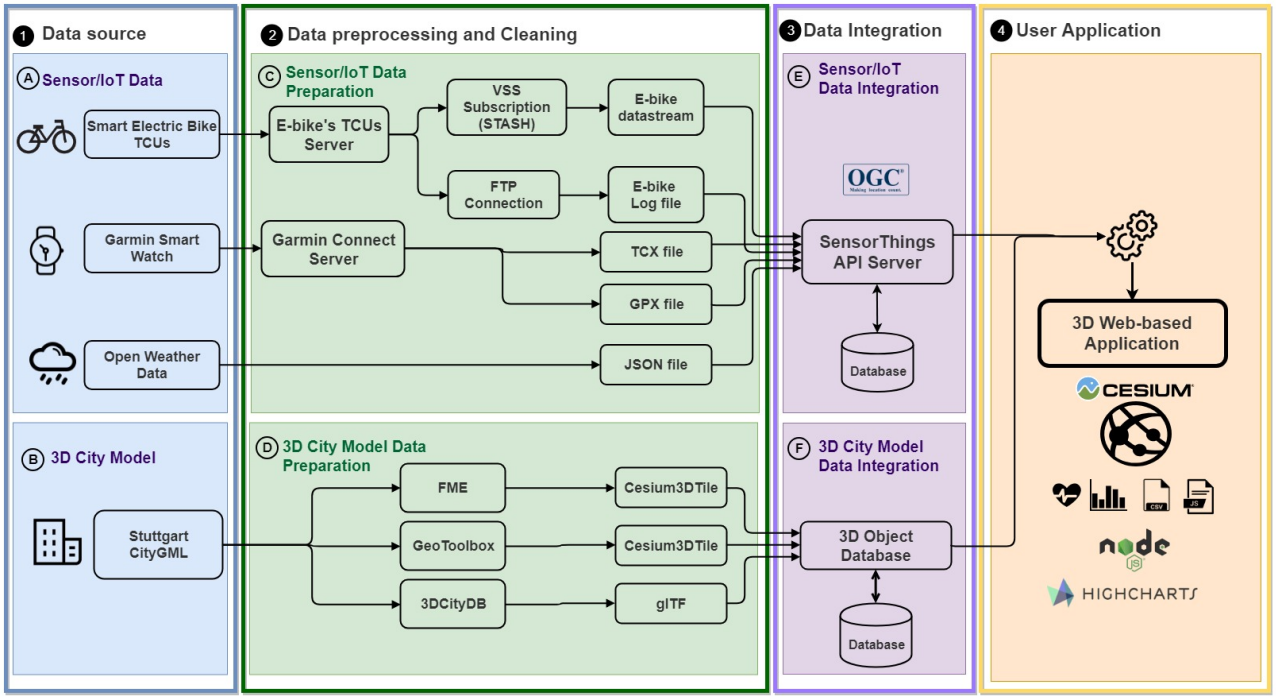

Figure 2: System Architecture

based which is much more complex request body comparing to the OGC SensorThings API. While in ASDDS, there are no features or entities to support the 2D or 3D locations of the sensor systems directly.

To summarize the finding of the sensor network protocols evaluation, the OGC SensorThings API is the most competent IoT framework among the three candidate protocols. Also, the communication of OGC SensorThings API among IoT devices, servers, and clients is in a standard-based REST style and JSON-based encoding providing its advantage for developers to understand, implement, and develop an application based on this standard in a compact way. Additionally, it is the only protocol among the three candidate protocol that supports the MQTT communication which is very useful for adding a light-weight IoT device in the future. Even though the lack of the aggregation function is its downside, there are several methods to add this feature to this protocol including 1) adding an aggregation function on a client-side, 2) adding an aggregation function a server-side, and 3) pre-computation of the aggregated results on the server-side. In conclusion, the OGC SensorThings API protocol is selected for an implementation for integrating the sensor and IoT data in this work.

4.2 The sensor network protocols implementation

To implement the OGC SensorThings API in this research, the server implementation of the OGC SensorThings API from Fraunhofer Institut IOSB is used. The first step to use this server is to use the Apache Tomcat Maven to create a Web application Archive or WAR file from the root folder of this source code. Then, the Tomcat server must be installed on the machine that is used for receiving the sensor data. In this study, the default port 8080 of the Tomcat server is used. Later on, the prepared SensorThings API WAR file is used to deploy on this server with the application name of SensorThingsService.

4.3 Data modelling for the SensorThings API entities

For the data modeling, the sensor protocol conceptual model is based on the OGC Observations and Measurements (O&M) specification. Firstly, all `Sensing` entities have to be specified starting with modeling the incoming *pedelec* data, the `Things` entity set is referred to the *pedelec* while the `Sensors` entity set is referred to sensors placed on each pedelec. The `ObservedProperties` entity set refers to the data type of the observed value from the *pedelec* TCU. The `FeaturesOfInterest` entity set is referred to users or riders. Currently, there is no system to automatically identify who uses the *pedelec* at each moment, therefore in the current sensor network, the FeaturesOfInterest entity is set to the *pedelec* user ID manually. For the location of *pedelecs*, the latest updated locations will be collected and updated in the `Location` entity and the historical data will be automatically moved to the `HistoricalLocation` entity whenever there is a new update data of the location. This enables the developer to access all *pedelec* location data in both real-time and historical data. following table.

Then, the `Datastreams` entity set is referred to the unique `ObservedProperty`, `Sensor`, `Thing` and `FeatureOfInterest` entity set. To confirm the idea about `Datastreams`, the example figure explains each entity in Observations and Measurements concept (Fig. 3, (FreeVectors.net, 2018)) After all the entities of Observations and Measurements specification are identified, the relations of entities are constructed based on the O&M standard which is described in (Section 3.1.1: OGC SensorThings API, (Liang, 2017)).

To use SensorThings API to manage the incoming data as an Observation entity, all possible `Datastream` and `FeatureOfInterest` must be identified, created and imported to the sensor network. As the `Datastream` entity is a set of combination of `Sensor`, `Thing`, `ObservedProperty` and `Location` entities, these entities must be identified first. To do this, JSON arrays of these entities are created following the SensorThings API standard. For example, Listings 1 and 2 show some part of

the JSON array of the `Things` entity of the `Sensors` entity. The figure 4 is showing the UML diagram of SensorThings API of the example incoming data indicating the charge level of the *pedelec* with the vehical identification : E-bike20131126003c.

Listing 1: JSON array: *Things* entity

```
[{
        "name": "ebike1",
        "description": eBike #1,
        "properties":{
                "vin": "eBike12345c"
},....
]
```

Listing 2: JSON array: *Sensors* entity

```
[{
"name": "ebike1–TCU",
"description": Telemetric & Connectivity,
"encodingType": "application/pdf",
"metadta": http://censored.com/EbData.pdf"
},....
]
```

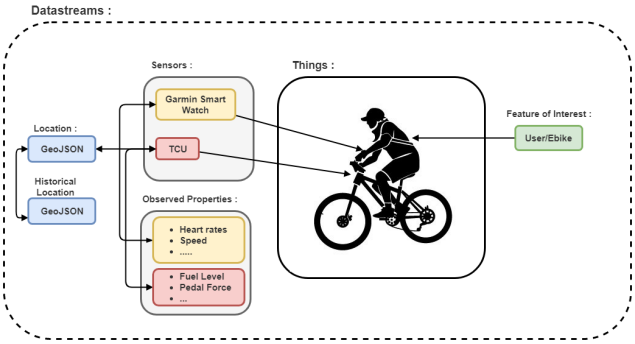

Figure 3: *pedelec*: Observations and Measurements entities (FreeVectors.net, 2018)

4.4 SensorThings API: E-bike usage datastream

After the SensorThings API is successfully deployed on the machine, this application allows users to make a communication with RESTful HTTP method to make requests and get responses in JSON format via a SensorThings base resource path as the following URL pattern.

```
http://[IP name/address]:[port]/
[application_name]/[version]/[entity_name)]
```

Then, the next part is to make a connection link between the sensor `Datastreams` and the SensorThings API database. To do this, the NodeJS program is written to execute POST command on the server side when there is an update of the `Datastreams` from the sensor. For example, the `Datastreams` that collects the charge level of the pedelec based on SensorThings API Standard are shown in fig4

4.5 A 3D Web-based Application using Cesium

The 3D Web-based application is built to illustrate utilizing many sensor data types from the implemented sensor network and the prepared 3D city model data. For users, this application is designed as a tool for researchers who interested in the relation among the different parameters from heterogeneous sensor systems together with the realistic environment simulation as a terrain height or a 3D city environment. The data that are used to build this web application including (1) Data from Ebike-TCU sensors, (2) Data from Garmin Smart Watch, (3) Historical Stuttgart Weather data from OpenWeatherData API and the prepared city models in the area of Stuttgart city.

For requesting the sensor data from the database, the JQuery library is used to make a request for the specified observation result from the SensorThings API server. With the SensorThings API standard, users can request the observation result with the specified period of time by using the filter query filter and also limit the response body to return only result and resultTime key-pair-values by using the selected query select. Then, to visualize the data on Cesium application, the data is conversed into CZML or Cesium language which is a JSON describing a time-dynamic primarily for display in a Cesium application.

The developed web application for this research is named as i_city ebike sharing and has the user interface shown in figure 5. The main user's menu is on the left window with the main functionalities as the following:

❶ **Map pin** With this function, users can simulate the important point of interest related to the E-bike usage, for example, the starting point of a journey or the charging station.

❷ **3D city model** Users can simulate the 3D city model in Stuttgart are in prepared glTF or Cesium 3D Tile. Also, they are able to adjust the transparency level.

❸ **Real-time E-bike status** Users can request the latest information on each E-bike, for example, charging level, speed, pedal force, etc.

❹ **Historical E-bike route** Users can request the historical E-bike session simulation on the selected session date. The E-bike usage information window will pop-up showing the current status matching the E-bike status on the specific time and the application clock. (figure 5)

❺ **Compute the Statistical information** Users can request the statistic of the historical E-bike usage. For this functionality, there is a menu panel that users can select date and time for the beginning and the end of for the request usage time and also the first and second Sensor or IoT data parameters. Then, after the selection, the statistical chart and relevant statistical parameters are shown to users (figure 6).

5. CONCLUSIONS AND FUTURE WORK

A 3D web-based application has been developed in the final part of this research. The Cesium JavaScript is used to build the main part of the application. For the 3D city model, the current cesium application version is not supported to show the city model in CityGML file format directly. Some additional steps for file conversion to Cesium 3D-Tile or glTF are needed. Accordingly, the conversion of CityGML to Cesium 3D-Tile and glTF is done in this work using different tools. At this stage, three different tools are deliberately examined to find out the best solution. By using the open-source 3DCityDB, the tool has proven capabilities to import the CityGML data in PostgreSQL database and export to the glTF file format with a defined styling option such as roof and wall color of the building. A drawback of this tool is that it still needs some manual steps for users to import the data into the database first before exporting the data to a preferred format.

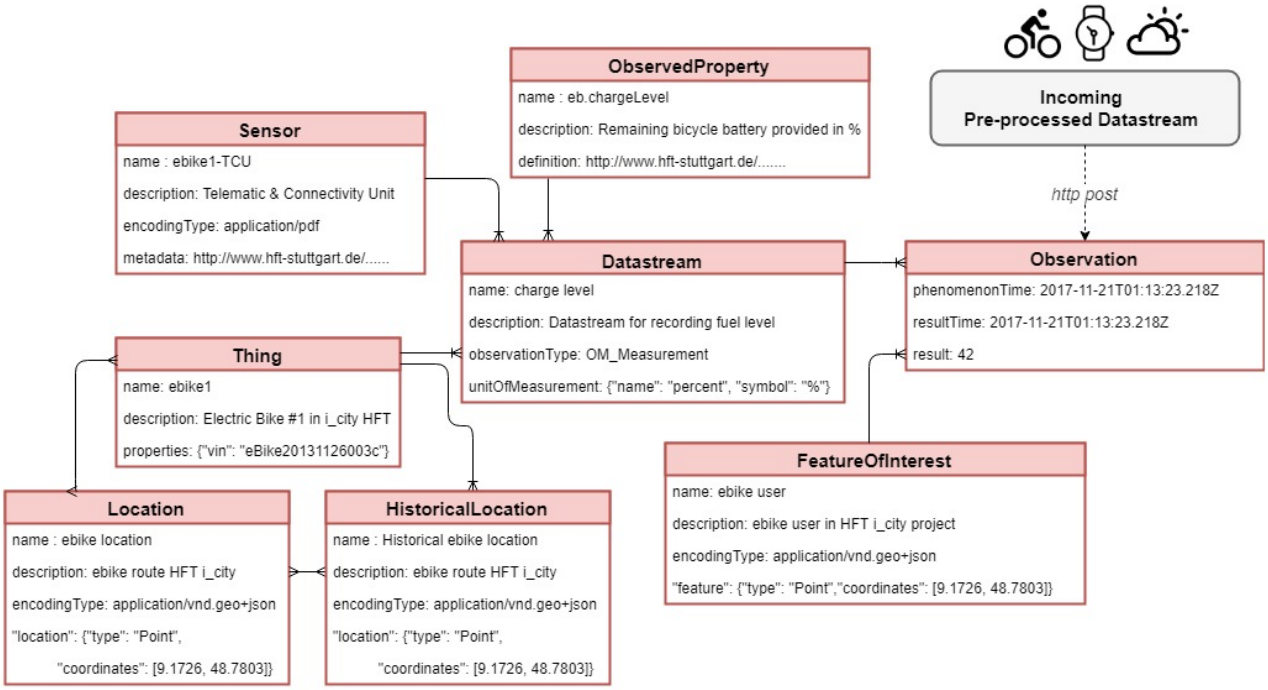

Figure 4: E-bike information structure based on SensorThings API standard

Figure 5: 3D application with *pedelec* usage information displayed for a single participant using an E-bike. 3D building model © Stadtmessungsamt Stuttgart.

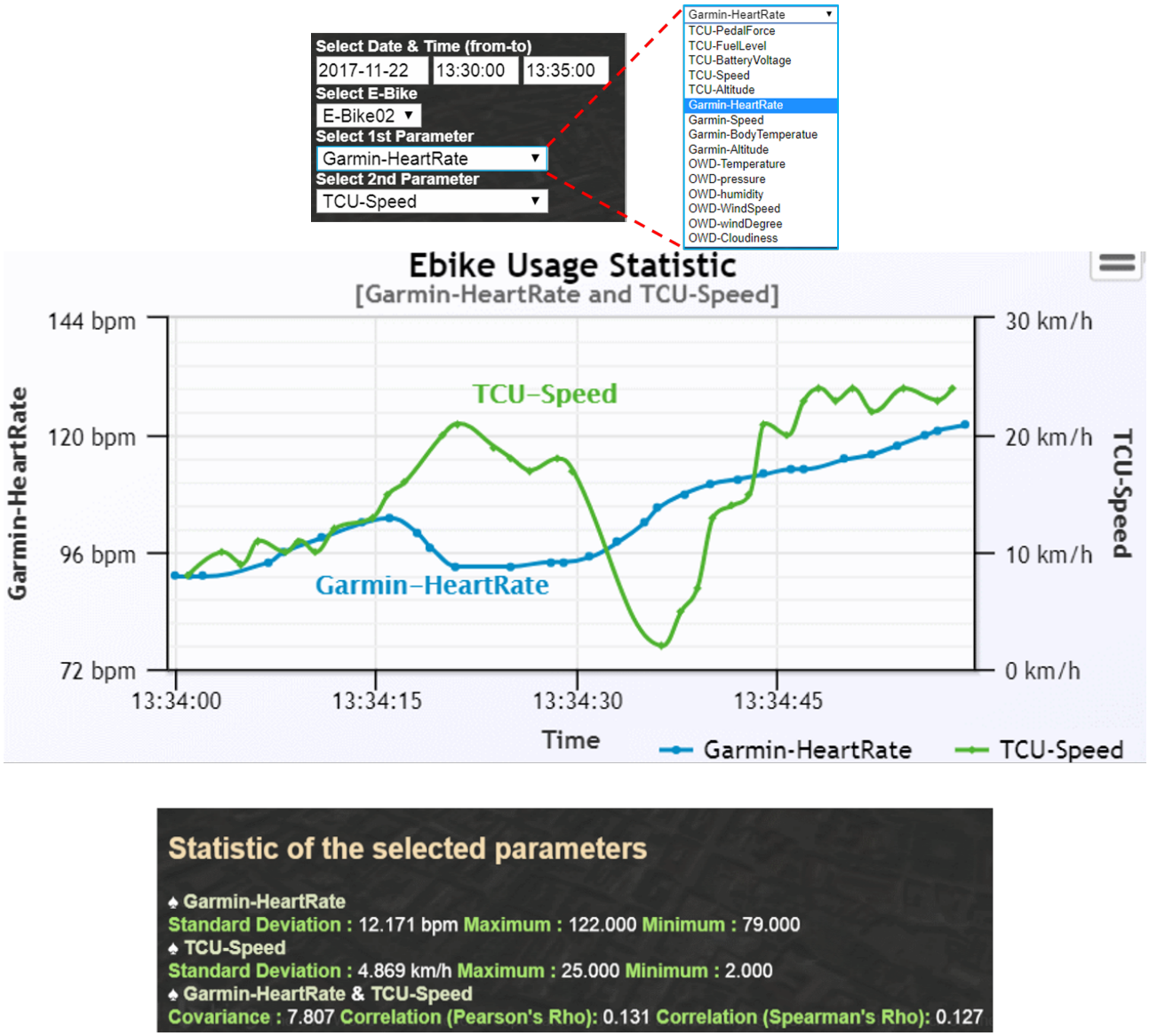

Figure 6: Participant statistics : Garmin-HeartRate v/s TCU-Speed

There is no single step to converse the CityGML data to glTF file format directly. On the other hand, the GeoRocket GeoToolbox 1.0.2, which is a lite source program written in Java for converting the input CityGML file into Cesium 3D-Tile, can run the conversion through the command window in a single step without exchanging file in the database. The Cesium 3D-Tile output from the Stuttgart CityGML dataset can be shown on the Cesium 3D application without any problem. Similarly, the FME 2017 can convert a CityGML into Cesium 3D-Tile with one step and show the result on the Cesium application as well. In addition, this tool can view and edit the model of CityGML in both 2D and 3D perspectives with a simple user-interface. However, the desktop program of this tool must be installed first.

For the web-based visualization of the 3D city model in the Cesium application, the result in Cesium 3D-Tile format has proven that it takes less time and resource on the client side to load compared to the file in glTF format. Still, at the moment, there is no open source tool to convert data from CityGML to Cesium 3D-Tile directly. Currently, Cesium 3D-Tile is under the stage of consideration where it will be certified as one of the OGC community standard under the 3D Portrayal Service. This will allow Cesium 3D-Tile to become interoperable among other 3D geospatial formats such as X3D, glTF, I3S, and CityGML.

For the application development process using Cesium JavaScript library, it is simple to learn how to use the Cesium framework as there are many examples and tutorials on the website and also a big developer community. To simulate the historical E-bike route, the CZML file format is built on the client-side by requesting the sensor data from the sensor network to show 3D E-bike path which shows the good results on the application. All the sensor data are collected into the sensor network without any constraints. For further research, some rules or constraints can be applied to validate the accuracy of its source, for example, the location feasibility if the movement speed of the sensor is realistic for E-bike. Hence, any observation data of the sensor position from one location to another with excessed defined speed will be rejected by the server. In addition, some observed property data are provided by both the E-bike sensor and the Garmin Smart Watch in parallel. Nevertheless, these duplicated data are not being deployed to improve the total accuracy or to observe which data source provides more precise data. The future work of this part could be to find a method to improve the data quality by analysis of data from many sources. Lastly, the sensor system in this project is still open for more IoT devices in daily-life such as smartphones,

wearable health sensor devices, etc. These devices could measure some missing important parameter that the Garmin Smart Watch and the E-bike sensor do not collect during the usage time. Furthermore, any supports of sensor network for various IoT products such as Apple iPhone, Samsung Galaxy, Apple Watch, etc., would give a good impression for users after the E-bike sharing project is deployed in a city level. From a researcher point of view, more sensors equipped on each E-bike means more usage information. In other words, greater opportunity for researchers and developers to analyze and utilize those data in a Smart Cities concept. For instance, an IMU sensor equipped on E-bike would give a parameter to calculate the real terrain slope over the whole usage time; or a temperature sensor to measure the outside temperature and then the relation of this value and the body temperature from a wearable device can be computed. Moreover, some type of sensors or devices can be integrated to let the users identify their simple information like age and gender which are important for the future research.

6. ACKNOWLEDGMENTS

This work has been jointly developed in the project *Simstadt 2.0* (Funding number: 03ET1459A) and *i_city* (Funding number: 03FH9I01IA). The project *i_city* is supported by the German Federal Ministry of Education and Research (BMBF), while project *Simstadt 2.0* is supported by the German Ministry of Economics (BMWi). The authors are responsible for the content of this publication. The authors would also like to thank the anonymous reviewers who have helped to increase the quality of the paper.

REFERENCES

Albakour, M.-D., Macdonald, C., Ounis, I., Clarke, C. L. A. and Bicer, V., 2014. Information Access in Smart Cities (i-ASC). In: M. de Rijke, T. Kenter, A. P. de Vries, C. Zhai, F. de Jong, K. Radinsky and K. Hofmann (eds), *Advances in Information Retrieval*, Springer International Publishing, pp. 810–814.

Banks, A. and Gupta, R., 2014. Mqtt version 3.1. 1. *OASIS standard*.

Consortium, O. G., 2017. OGC SensorThings API Part 1: Sensing; 15-078r6. http://docs.opengeospatial.org/is/15-078r6/15-078r6.html.

ESPRESSO-Consortium, 2018. The ESPRESSO Project. http://espresso.espresso-project.eu/project-2/.

FreeVectors.net, 2018. Biker Vector Image. http://www.freevectors.net/details/Biker+Vector+Image.

Garmin, 2018. Garmin Connect. https://connect.garmin.com/en-US/status.

GlasgowCityCouncil, 2018. Future City Glasgow. http://futurecity.glasgow.gov.uk/.

Jazayeri, A. M., Liang, H. S. and Huang, C.-Y., 2015. Implementation and Evaluation of Four Interoperable Open Standards for the Internet of Things. *Sensors*.

Jo, J. and Jang, I., 2016. Applying sensor web enablement to retrieve and visualize sensor observations across the Web. In: *2016 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 852–854.

Kohn, L., 2017. Erfassung emotionalen Erlebens von E-Bike Fahrenden mittels physiologischer Marker: eine Validierungsstudie im Forschungsprojekt icity der Hochschule fuer Technik Stuttgart in Kooperation mit Daimler TSS. Bachlorthesis, University of Applied Sciences Stuttgart.

Kukka, H., Kostakos, V., Ojala, T., Ylipulli, J., Suopajärvi, T., Jurmu, M. and Hosio, S., 2013. This is not classified: everyday information seeking and encountering in smart urban spaces. *Personal and Ubiquitous Computing* 17(1), pp. 15–27.

Liang, S., 2017. Amazon IoT and the candidate OGC SensorThings API Standard OGC. http://www.opengeospatial.org/blog/2315.

Liang, S., Huang, C. and Khalafbeigi, T., 2016. Ogc® sensorthings api. *Open Geospatial Consortium: Wayland, MA, USA*.

Manville, C., Cochrane, G., Cave, J., Millard, J., Pederson, J., Thaarup, R., Liebe, A., Wissner, M., Massink, R. and Kotterink, B., 2014. Mapping smart cities in the EU. https://goo.gl/f2xv6J.

Na, A. and Priest, M., 2007. Sensor observation service. *Implementation Standard OGC* p. 21.

Sanchez, L., Muñoz, L., Galache, J. A., Sotres, P., Santana, J. R., Gutierrez, V., Ramdhany, R., Gluhak, A., Krco, S., Theodoridis, E. and Pfisterer, D., 2014. SmartSantander: IoT Experimentation over a Smart City Testbed. *Comput. Netw.* 61, pp. 217–238.

Sporny, M., Longley, D., Kellogg, G., Lanthaler, M. and Lindström, N., 2014. Json-ld 1.0. *W3C Recommendation*.

Storz, M. and Dastageeri, H., 2017. Usage of advanced sensor data delivery service(asdds). Technical report.