EPiC Series in Computing

Volume 62, 2019, Pages 213–229

SUMO User Conference 2019

Bridging the Gap between SUMO & Kuksa: Using A Traffic Simulator for Testing Cloud-based Connected Vehicle Services

Philipp Heisig[1], Sven Erik Jeroschewski[2], Johannes Kristan[2], Robert Höttger[1], Ahmad Banijamali[3], and Sabine Sachweh[1]

[1] Dortmund University of Applied Sciences and Arts
Institute for the Digital Transformation of Application and Living Domains
Dortmund, Germany
`philipp.heisig | robert.hoettger | sabine.sachweh@fh-dortmund.de`
[2] Bosch Software Innovations GmbH
Berlin, Germany
`svenerik.jeroschewski | johannes.kristan@bosch-si.com`
[3] University of Oulu
Empirical Software Engineering in Software, Systems and Services (M3S)
Oulu, Finland
`ahmad.banijamali@oulu.fi`

Abstract

The emerging usage of connected vehicles promises new business models and a high level of innovation, but also poses new challenges for the automotive domain and in particular for the connectivity dimension, i. e. the connection between vehicles and cloud environments including the architecture of such systems. Among other challenges, IoT Cloud platforms and their services have to scale with the number of vehicles on the road to provide functionality in a reliable way, especially when dealing with safety-related functions. Testing the scalability, functionality, and availability of IoT Cloud platform architectures for connected vehicles requires data from real world scenarios instead of hypothetical data sets to ensure both the proper functionality of distinct connected vehicle services and that the architecture scales with a varying number of vehicles. However, the closed and proprietary nature of current connected vehicle solutions aggravate the availability of both vehicle data and test environments to evaluate different architectures and cloud solutions. Thus, this paper introduces an approach for connecting the Eclipse SUMO traffic simulation with the open source connected vehicle ecosystem Eclipse Kuksa. More precisely, Eclipse SUMO is used to simulate traffic scenarios including microscopic properties like the position or emission. The generated data of each vehicle is then be sent to the message gateway of the Kuksa IoT Cloud platform and delegated to an according example service that consumes the data. In this way, not only the scalability of connected vehicle IoT architectures can be tested based on real world scenarios, but also the functionality of cloud services can be ensured by providing context-specific automotive data that goes beyond rudimentary or fake data-sets.

1 Introduction

Within the automotive domain, sensors are the basis for a variety of functionality as they allow to gather detailed context information about a vehicle and its environment to detect specific events of interest like environmental conditions or unusual vehicle motion [8]. Nowadays, the increasing number of different sensor types within vehicles that emerge, for instance, from ongoing development in autonomous driving, leads to an exponential growth in data generated by vehicles. For example, according to Hitachi [27], vehicles already create up to 25GB of data per hour. At the same time, technological advances and area-wide mobile Internet have transformed vehicles into connected vehicles [8]. In this way, vehicles continually increase their functionality by interacting with their environment and sharing data among people, businesses, service providers, or OEMs within the context of the Internet of Things (IoT).

From a technical point of view, the extraction, storage, processing, and analysis of vehicle data within the IoT is the basis for data-driven business models and an important factor for future innovation. The Big Data processing capabilities of the cloud and the fusion with information obtained from other sources, e. g. smart city [20], allows to extract additional knowledge and create valuable connected vehicle services. In particular areas like road safety or smart and green transportation benefits from an additional connectivity dimension within vehicles. From a market point of view, connectivity-enabled vehicles are expected to be the next frontier in automotive revolution [19] as they promise a high degree of data-monetization and value creation based on disruptive business models and innovative, data-driven mobility services, e. g. location-dependent services like route optimization to reduce traffic congestion [29].

The rapid growth and the tremendous number of connected vehicles on the road makes the connected vehicle domain a major element of the IoT and leads to new requirements regarding vehicular networks and data processing in the cloud [8]. Thereby, designing scalable and reliable software architectures is one of the main challenges [29] which requires further domain knowledge. Providing an abstraction of those topics to developers and manufacturers through an open platform for connected driving helps to pave the way for supporting various application domains, spanning from road safety over smart, efficient and green transportation to location-dependent services [19]. Those services operate on different complexity levels and range from batch processing to real-time analysis that is fed back to the vehicle. The open-source project Eclipse Kuksa [13] aims to create a platform for establishing connected vehicle ecosystems.

As connected vehicles operate in a safety-critical and time-sensitive environment with changing conditions, they require reliable connected vehicle services. This is especially the case for safety-related applications such as cloud-based wrong-way driver warning or hazard identification. Among other challenges like privacy and security, it is thus inevitable to extensively test and evaluate the functionality of the according hardware and software components. This also includes the appropriate evaluation that the used IoT Cloud platforms and their services scale and still function with the high number of vehicles on the road, especially during peak hours. However, setting up a large number of hardware and vehicle nodes for evaluating the scalability of IoT components may not be practical due to economical and operational constraints [4]. Furthermore, using real hardware and vehicles can be challenging as it requires specific expertise and domain knowledge. The effort for setting up such an evaluation environment may then prevent the developer from focusing on the actual application and possible innovation especially for developers that come from another domain.

One way to overcome these challenges is the usage of appropriate simulators. In general, simulators allow for a proof-of-concept design and evaluation of connected vehicle services by spanning both virtual and physical domains. While simulations on a macroscopic level allow to simulate average vehicle dynamics such as traffic density, microscopic simulations are located on a more detailed level by modeling each vehicle and its dynamics individually [18]. For testing the scalability of IoT Cloud platform components, a macroscopic traffic simulation or in some cases even using dummy data would be suitable. But for validating the functionality of connected vehicle services, application and service developers require vehicle-specific data from real-word scenarios. Moreover, the creation of training data for machine learning based approaches is another use case that requires vehicle-specific data. In this way, the detection of vehicle or traffic specific scenarios can be trained and the performance of actions based on these findings can be enhanced.

The open-source traffic simulation suite Eclipse SUMO[1] is designed for microscopic simulations and supports, among other things, large road networks and the modeling of intermodal traffic systems including vehicles, public transport, and pedestrians. Eclipse SUMO is well established in the research community and provides real-world scenarios from areas such as Luxembourg [5, 6], Bologna [1], Cologne, and Monaco [7]. In this paper, we investigate how to connect Eclipse SUMO as a simulator with the connected vehicle platform Eclipse Kuksa. More precisely, we present an early approach on how to use a simulator for sending telemetry data via MQTT to an IoT gateway of an connected vehicle ecosystem. In this way, the scalability and functionality of connected vehicle services can be tested and evaluated on different levels with vehicle-specific data from real-world scenarios. We further demonstrate the approach with a small example and discuss current limitations.

The remainder of this paper is organized as follows: Section 2 reviews related work, while Section 3 gives an overview about Eclipse Kuksa. Section 4 then shows how to connect Eclipse SUMO with the Eclipse Kuksa Cloud to store vehicle-specific data from a SUMO simulation into an appropriate database. Afterward, Section 5 introduces a small example of a cloud-based connected vehicle service that will be used as basis for testing the service's functionality via the available simulation data set. Section 6 then discusses the results and current limitations, while Section 7 concludes this work.

2 Related Work

This section provides a brief review of related work on connected vehicles, academia IoT Testbeds, and existing traffic simulations with a focus on freely available solutions.

2.1 Connected Vehicles Landscape

A survey by Siegel et al. [26] on connected vehicle landscapes addresses enabling technologies, applications, and development areas. They outline that technologies such as sensors or intra-vehicle as well as inter-vehicle connectivity are the enablers of connected vehicles, which are facilitated by decreasing the power, cost, and scalability requirements [26].

[1]http://sumo.dlr.de

A similar research [19] argues that vehicle-to-infrastructure (V2I) and vehicle-to-roadside (V2R) is also necessary to accomplish connected vehicles. Other researchers [8] present that these new requirements, e. g. scalability and security, that raised from connected vehicles have changed the concept of vehicular ad-hoc networks (VANETs) to a new concept called Internet of Vehicles (IoV). IoV enables dynamic information services, intelligent vehicle control, and increased productivity due to a reduced traffic congestion [8].

2.2 IoT Testbeds

Simulation has many advantages [25] for the usage in connected vehicles environments. For example, the behavior of systems can be compared to that of the real system under study. In addition, with simulations, it is easier to comprehend and justify to management. In that regard, Chernyshev et al. [4] present a comparative analysis about existing simulators and testbeds for the Internet-of-Things (IoT). Their study covers three categories of simulators: (i) full stack simulators; (ii) big data processing simulators; and (iii) network simulators. Nevertheless, none of the considered simulators is specifically designed to simulate traffic scenarios on a microscopic level and to provide vehicle-specific data.

Datta et al. [9] present a testbed architecture for connected vehicles that integrates an end-to-end connected vehicle stack. Another study [14] proposes a multi-agent based extension of a simulator to model complex interactions between cooperative vehicles, e. g. Advanced Driver Assistance Systems (ADAS) adjusting to traffic conditions, information exchange with the infrastructure, and V2V communication. As mentioned before, a challenge in the context of connected vehicles is to provide a scalable, extensible and secure Cloud framework that hosts the connected vehicle services. Many of the simulation testbeds (e. g. [9]) are based on real data from an existing car, though it considers only a single connected vehicle due to non-availability of multiple vehicles. Thus, they are not a viable choice for testing scalability requirements.

2.3 Traffic Simulators

Due to ongoing research and development activities in the area of intelligent transportation systems (ITS), a variety of both commercial and open source simulation tools are available. Li et al. [17] compare different traffic simulators, namely GAMA and SUMO, for multi-modal scenarios in urban environments. The authors conclude that SUMO is generally more suitable for multi-modal simulations, but they also found some minor limitations such as deadlocks or that the realism of the simulation is not always up to the task.

CupCarbon[2] allows to simulate mobility based on real world maps, but focuses more on the geographic mobility flow of simulated devices [2].

For the simulation and evaluation of applications and algorithms around connected driving there is also the VSimRTI framework [22] which integrates SUMO. However, VSimRTI is so far not designed for the integration with real-world cloud instances including the usage of the actual protocols that are used for the communication with the cloud.

[2]http://www.cupcarbon.com/

3 Eclipse Kuksa

The Eclipse Kuksa project[3] wants to establish an open source ecosystem for connected vehicles [13]. Kuksa offers a platform for an open and collaborative development of solutions and services in the automotive domain. The openness of the Kuksa ecosystem enables actors from different domains and companies to work collaborative in order to foster innovation and new business models. One part of the Eclipse Kuksa project is a run-time infrastructure that is supposed to run within vehicles and enables developers to easily create and deploy applications to the vehicle. Moreover, Eclipse Kuksa provides an online-IDE to ease the development of in-vehicle applications by free an individual software developer from setting up his personal work environment. This IDE is based on the Eclipse Che [11] project. The third part of Eclipse Kuksa is the Eclipse Kuksa Cloud, which bundles services that enable the vehicle-applications to interact with a cloud environment to provide more value and information to the occupants of the vehicle. Since the focus of this paper is testing and evaluating cloud-based connected vehicle services, the following description focuses on the Kuksa Cloud.

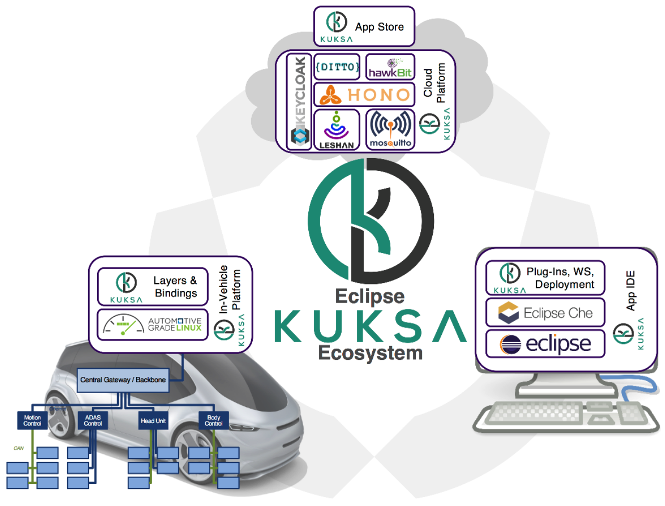

Figure 1: The components of the Eclipse Kuksa platform [13]

The Eclipse Kuksa Cloud[4] consists of multiple services that are combined together in a microservice architecture to allow the management of vehicles and the analysis of vehicle data. One goal for the Eclipse Kuksa Cloud is to provide a flexible environment for evaluation purposes that can be turned into productive use without many challenges. Therefore, one aspect is the licensing of the Eclipse Kuksa Cloud components which are mostly released under the Eclipse Public License. As shown in Figure 2, the entry point to the cloud platform is a message gateway which connects the vehicles and other devices with the cloud services in a scalable and unified way. For doing so, the gateway accepts, sends, and receives messages from and to the vehicles. Within the Eclipse Kuksa Cloud, Eclipse Hono [21] act as message gateway by providing protocol adapters for MQTT, HTTP, and CoAP. Other cloud services connect with Eclipse Hono over the AMQP 1.0 protocol. For instance, the current version of the Eclipse Kuksa

[3]https://www.eclipse.org/kuksa/
[4]https://github.com/eclipse/kuksa.cloud

Cloud consists of a connector[5] between Eclipse Hono and the time-series database InfluxDB [16] allows to store all messages received by Eclipse Hono. It is then possible for other services to fetch the information from the received messages from the database for further processing and visualization. One example for such a service is an instance of Grafana which visualizes the vehicle related information like speed, number of revolutions or the state of the engine.

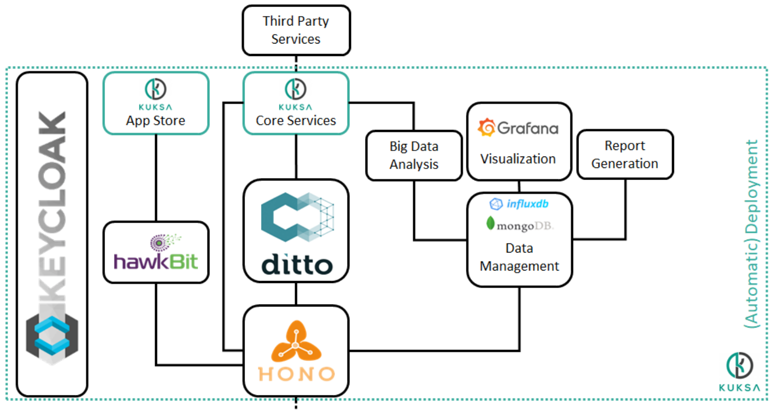

Figure 2: The Kuksa Cloud environment

Another service that is part of the Eclipse Kuksa Cloud is Eclipse hawkBit [15]. With Eclipse hawkBit one can control, plan, and distribute software updates and manage the software versioning on devices. Such updates either involve some components or the whole firmware of those. On top of Eclipse hawkBit an implementation of an app-store is provided. This app-store allows the owner of the vehicle to install applications for the vehicle which execute in the in-vehicle run-time environment. Authentication and authorization are important for the overall Eclipse Kuksa platform as well. This involves, but is not limited, to securing the access to vehicle information, the APIs of Eclipse Hono, and the app-store. Therefore, Keycloak is integrated in the Eclipse Kuksa Cloud platform and acts as authentication and authorization server. For future releases of the Eclipse Kuksa Cloud, it is planned to integrate Eclipse Ditto [10]. With Eclipse Ditto it is possible to get an abstract API for the representation of a device's state such as a vehicle. This concept is also referred to as "digital twin".

4 Connecting SUMO to Kuksa

This section shows how to connect SUMO with Eclipse Kuksa via a Python script. The reasons for choosing Eclipse SUMO as simulator are manifold. Among other advantages, Eclipse SUMO (i) provides a lot of modeling tools and APIs; (ii) fosters real-world scenarios and data sets; and (iii) supports a subset of vehicle information. Furthermore, the open-source Traffic Control Interface (TraCI) is an TCP based interface that allows to control Eclipse SUMO simulations [28]. Via the TraCI API, values of simulated objects can be retrieved and also their behavior can be altered. TraCI relies on a client/server architecture with Eclipse SUMO acting as server that serves multiple TraCI clients. For vehicles, a variety of values can be retrieved, e. g. the geolocation or the current speed.

[5]https://github.com/eclipse/kuksa.cloud/tree/master/utils/hono-influxdb-connector

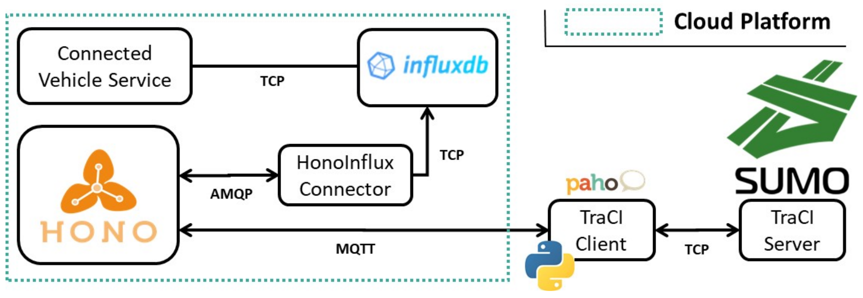

Figure 3: Data flow and technology stack for the SUMO-Kuksa bridge

Figure 3 depicts the proposed approach for bridging Eclipse SUMO with Eclipse Kuksa. To obtain data from the Eclipse SUMO simulation and send data to the cloud, a Python script based on TraCI is used. As a starting point, we used the *runner.py* script from the TraCI tutorial[6] and modified it to establish a connection with the Eclipse Kuksa Cloud. This allows us to send vehicle-specific simulation data via MQTT to a message gateway and store it in a remote database. As shown in Listing 1[7], the main function is doing two different things. At first, the connection to the Hono messaging gateway within the Eclipse Kuksa Cloud is established in the `connect_to_message_gateway()` function. For doing so, we use the Eclipse Paho MQTT client [12], which provides a client-side open-source implementations of MQTT. The client receives a CONNACK response from the server and can subscribe to the *control/+/+/req/#* topic to receive Command & Control messages from business applications in the cloud. Although Command & Control is not in the focus of this paper, it allows the integration of further use cases and thus we included it. After the MQTT connection is established, we run the simulation and obtain vehicle-specific data via TraCI. Within the `run()` function, the simulation is started and controlled via TraCI. For each simulation step, different data from each available vehicle is retrieved through the TraCI API and then transformed into a JSON object. By using the `publish()` function of the Eclipse Paho client, we can send the JSON objects via the *telemetry* topic to the message gateway (Eclipse Hono) of the Eclipse Kuksa Cloud. Eclipse Hono then delegates the incoming messages to the previously described HonoInfluxConnector (cf. Section 3), which processes the data and writes it into an InfluxDB. Figure 4 contains some log messages that show how the simulated data is actually consumed by the cloud service.

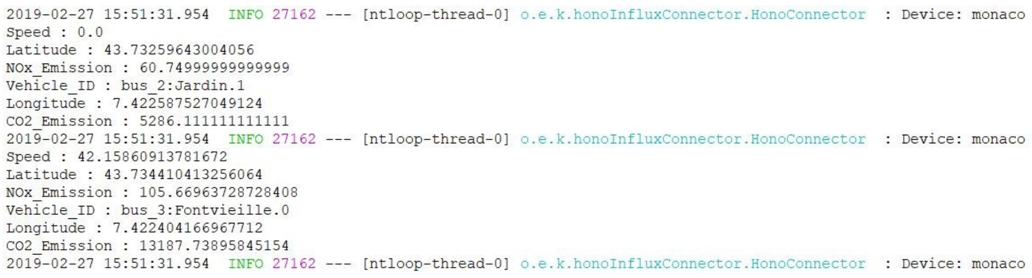
```
2019-02-27 15:51:31.954  INFO 27162 --- [ntloop-thread-0] o.e.k.honoInfluxConnector.HonoConnector  : Device: monaco
Speed : 0.0
Latitude : 43.73259643004056
NOx_Emission : 60.74999999999999
Vehicle_ID : bus_2:Jardin.1
Longitude : 7.422587527049124
CO2_Emission : 5286.111111111111
2019-02-27 15:51:31.954  INFO 27162 --- [ntloop-thread-0] o.e.k.honoInfluxConnector.HonoConnector  : Device: monaco
Speed : 42.15860913781672
Latitude : 43.734410413256064
NOx_Emission : 105.66963728728408
Vehicle_ID : bus_3:Fontvieille.0
Longitude : 7.422404166967712
CO2_Emission : 13187.73895845154
2019-02-27 15:51:31.954  INFO 27162 --- [ntloop-thread-0] o.e.k.honoInfluxConnector.HonoConnector  : Device: monaco
```

Figure 4: Incoming telemetry data is stored in a database via the HonoInfluxDBConnector

[6]https://github.com/eclipse/sumo/tree/master/tests/complex/tutorial/traci_tls
[7]Note that we just show an extract of the script to omit irrelevant details

Listing 1: Phyton script for sending simulation data from Eclipse SUMO to the cloud

```
client = mqtt.Client(client_id="c", clean_session=True, userdata=None, protocol
    =4, transport="tcp")
topic_to_publish = "telemetry"
topic_to_subscribe = "control/+/+/req/#"

if __name__ == "__main__":
    connect_to_message_gateway()
    run()

def connect_to_message_gateway():
    client.reinitialise(client_id="c",clean_session=True, userdata=None)
    client.username_pw_set(username,password)
    client.on_connect = on_connect
    client.connect(host, port, 60)
    client.loop_start()

def run():
    step = 0
    while traci.simulation.getMinExpectedNumber() > 0:
        traci.simulationStep()
        for vehicleID in traci.vehicle.getIDList():
            co2emission = traci.vehicle.getCO2Emission(vehicleID)
            noxemission = traci.vehicle.getNOxEmission(vehicleID)
            if noxemission > 0 or co2emission > 0:
                speed = traci.vehicle.getSpeed(vehicleID)*3.6
                x, y  = traci.vehicle.getPosition(vehicleID)
                longitude, latitude = traci.simulation.convertGeo(x, y)
                json_data = json.dumps({'Vehicle_ID': vehicleID, 'Speed': speed
                    , 'Latitude': latitude, 'Longitude': longitude, '
                    CO2_Emission': co2emission, 'NOx_Emission': noxemission})
                client.publish(topic_to_publish,json_data,0,False)
        step += 1
    traci.close()
    sys.stdout.flush()
```

5 Using Eclipse SUMO in the Eclipse Kuksa Application Development Process

This section shows how Eclipse SUMO can be facilitated to evaluate the functionality of a connected vehicle service. We do that by sketching the development process of a fictitious developer who develops a new application for Eclipse Kuksa and uses Eclipse SUMO to test his application. Even though the developer is purely fictional, the experience described in this section is real and collected while the authors where testing the approach described in this paper. Lets assume that the developer is about to implement a new service for the Eclipse Kuksa Cloud which uses collected vehicle data to approximate the air quality in a city. A major part of the development efforts is to test the application and check its behavior under different conditions. Collecting real life data is hard as it is barely possible to setup a controlled environment. However, by simulating the traffic and using the simulated data, it is easy to setup defined conditions and, for example, increase the traffic in certain areas of a city.

5.1 Implementing a Service

The first step for the developer is to actually implement the *Air Quality Monitor* service, which visualizes the air pollution for certain areas such as cities, districts etc. The idea of the service is to help municipalities to get an almost real-time overview of the current pollution situation. Therefore, the *Air Quality Monitor* service visualizes the air quality on a real-world heatmap. In general, a heatmap visualizes the intensity of data at geographical points. The service allows to display CO_2 and NOx pollution for a certain area in two ways: On the one hand, a *real-time view* shows the current emission for each vehicle every 5 seconds, while on the other hand an *aggregated view* visualizes the aggregated pollution for any time interval. The service is implemented as a Java-based Spring application [3]. Figure 5 shows the corresponding class diagram of the *Air Quality Monitor* service.

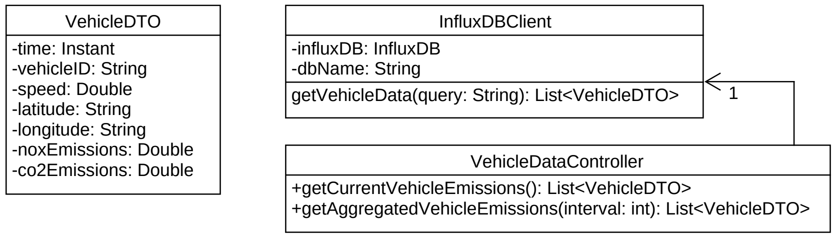

Figure 5: UML class diagram of the *Air Quality Monitor* service

To work as expected, the service requires at least information about the current geo-location and a rough estimation about the vehicle emissions, e. g. via comparative data. Accordingly, the `VehicleDTO` consists of different attributes such as longitude and latitude for the geo-location as well as the NOx and CO2 emission in *mg* and a time stamp for every measurement. The service consumes such data via the Eclipse Kuksa Cloud backend. Once deployed on a productive instance of Eclipse Kuksa, the service would receive this data through Eclipse Hono and from an application running in the Eclipse Kuksa In-vehicle platform. As the visualization is planned to be accessible on a web page, the required data points are published via a REST API implemented in `VehicleDataController`. The controller offers two RESTful interfaces, namely `getCurrentVehicleEmissions()` and `getAggregatedVehicleEmissions()`. The former one returns the current, respectively the latest, available measurement data for each vehicle from the last 15 minutes (cf. Listing 2), while the latter one returns the aggregated emission data of each vehicle for a given time interval, e. g. the last 30 minutes. Both endpoints delegate the according database query to an instance of `InfluxDBClient`, which retrieves the data from the database based on an InfluxDB Java libary[8]. Listing 3 shows the respective method.

The view of the single-page application is based on HTML, CSS, and JavaScript and runs on an embedded Tomcat web server. Via JavaScript, data from the RESTful interfaces are fetched to visualize them on a heatmap. The heatmap relies on the Google Maps JavaScript Heatmap Layer API[9] and can be built by transforming the vehicle data to according points on the heatmap. A point consists of the geolocation (latitude, longitude) and an optional weight-

[8]https://github.com/influxdata/influxdb-java
[9]https://developers.google.com/maps/documentation/javascript/heatmaplayer

ing for controlling the intensity. For both, the NOx and CO_2 emissions, an appropriate weight is added that is derived from the given values of both attributes (for NOx, we use the current value, while the CO_2 values has to be reduced by factor 10). In this way, different degrees of emissions can be taken into account.

Listing 2: RESTful interface for providing vehicle-specific emission data

```
//VehicleDataController
@RequestMapping("/getCurrentVehicleEmissions()")
public List<VehicleDTO> getCurrentVehicleEmissions() {
  return influxDBService.getVehicleData("SELECT Longitude,Latitude,CO2_Emission
      ,NOx_Emission,last(NOx_Emission) FROM cologne WHERE time > now() - 15m
      group by Vehicle_ID");
}
```

Listing 3: Querying for vehicle data via InfluxDB Java API

```
//InfluxDBClient
public List<VehicleDTO> getVehicleData(String query) {
  QueryResult result = influxDB.query(new Query(query, dbName));
  InfluxDBResultMapper resultMapper = new InfluxDBResultMapper();
  return resultMapper.toPOJO(result, VehicleDTO.class);
}
```

5.2 Setting up the Test Environment

After implementing the first version of the *Air Quality Monitor* service, the next step is to actually check if the service behaves as expected. Therefore, the developer is setting up a test environment as described in Section 4 to generate data that simulates a real world scenario. The test environment is described in the following paragraphs.

Eclipse SUMO Eclipse SUMO runs in version 1.1.0 (Build Windows-6.3.9600) on a local windows 10 machine with a Intel Core^{TM} i7-5600U CPU at 2.60GHz and 16GB RAM. The TRaCi script is executed in a Python 3.6.4 environment. The reason for running Eclipse SUMO on a local machine is the availability of a GUI to see what happen at the simulation and the *Air Quality Monitor* at the same time.

Scenarios A minor drawback for the usage of simulators is the availability of properly-working and free scenarios. However, Eclipse SUMO has already several real-world and open-source scenarios available, namely:

- The real world traffic scenario from the city of Bologna[10] covers the area around the football stadium and was set up to simulate the mobility of big events such as football matches or concerts [1].
- The Monaco SUMO Traffic (MoST) Scenario is based within the Principality of Monaco with the city covering an area of 2 km^2, while the greater area is about 22 km^2 with a total of 350 km of street lengths [7]. Due to the various tunnels and bridges, it fosters a multidimensional environment with directional traffic congestion between the inner city and the greater area.

[10] https://sourceforge.net/projects/sumo/files/traffic_data/scenarios/Bologna_small/

- The TAPASCologne scenario simulates the traffic for the city of Cologne over a period of one day. The scenario is considered as one of the largest freely available traffic simulation data sets with nearly 11455 km of lanes. However, as mentioned at the official Eclipse SUMO wiki, the scenario is only hardly usable and needs further improvements regarding the network quality to make it realistic and complete.[11]
- The Luxembourg SUMO Traffic (LuST) Scenario[12] is located at the city of Luxembourg and covers an area of almost $156 km^2$ with 930 km of roads and a total number of 288.250 vehicles over 24 hours. The traffic demand includes morning and evening rush hour peaks around 08:15 and 18:30.

As the LuST scenario is not compatible with Eclipse SUMO versions > 0.26[13] and the Bologna scenario does not provide any geolocation, the scenarios of Monaco and Cologne (6:00 and 8:00 (am)) are used for testing his service. However, due to the limitations described for the TAPASCologne scenario, the scenario is used with caution.

Cloud The Eclipse Kuksa Cloud is deployed within an Azure Kubernetes Service (AKS)[14] cluster with Eclipse Hono in version 0.8, whereas the *Air Quality Monitor* service is running on a Ubuntu 16.04 virtual machines with 4 CPUs, each 3,5GHz, and 8GB RAM. The complete deployment of the service is depicted in Figure 6.

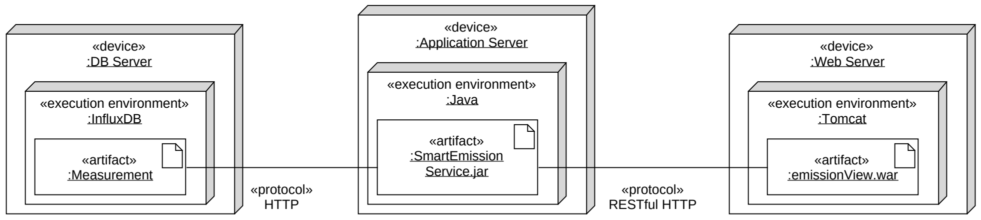

Figure 6: UML deployment diagram of the *Air Quality Monitor* service

5.3 Testing the Service

With the previously described test setup the developer is now able to run tests on his service. To have a variety of data sets, simulations on the cities of Monaco and Cologne are executed. The simulated data helps the developer already to have an early feedback in the development process as well as helpful insights into the real live behavior of his service. Running the different scenarios for a few minutes already produces a large number of data entries, e. g. for the Cologne scenario, 1.532.783 MQTT messages have been sent to the cloud between 06:00 and 06:15 in the traffic simulation. The following paragraphs depict the different problems that occurred when testing the service scalability and functionality.

[11] https://sumo.dlr.de/wiki/Data/Scenarios/TAPASCologne
[12] http://vehicularlab.uni.lu/lust-scenario/
[13] https://github.com/lcodeca/LuSTScenario/issues/8
[14] https://azure.microsoft.com/de-de/resources/videos/azure-kubernetes-service-overview/

Storage The first problem is that the initial database schema was not designed for querying large data sets in an efficient and flexible way. Thus, the developer reconsiders the schema and models `vehicleID` as a *Tag*, while all the other values remain as *fields*. In InfluxDB, *Tags* represent meta data which are indexed and thus supports performant querying. Furthermore, *Tags* are suitable for queries that include, for example, *GROUP BY()*. By this first finding, the developer was able to considerably improve the response time of the service.

Presentation Another problem with the initial service design relates to the presentation on the map. While for a smaller amount of vehicles the heatmap is clear, the map gets unclear and overcrowded with larger amounts of vehicles (cf. Figure 7). This is due to the fact that the initial weighting of the heatmap points is too coarse-grained. By changing the weighting dynamically to the amount of vehicles, a clear and scalable view is provided.

API Usage In addition to that, the *real-time view* does not update the heatmap points properly when a simulation is running at the same time. This behavior is caused by a wrong usage of the Heatmap Layer API (the view updated before the asynchronous call returned back), but with permanently changing vehicle data due to the running simulation, debugging the issue is much easier for the developer.

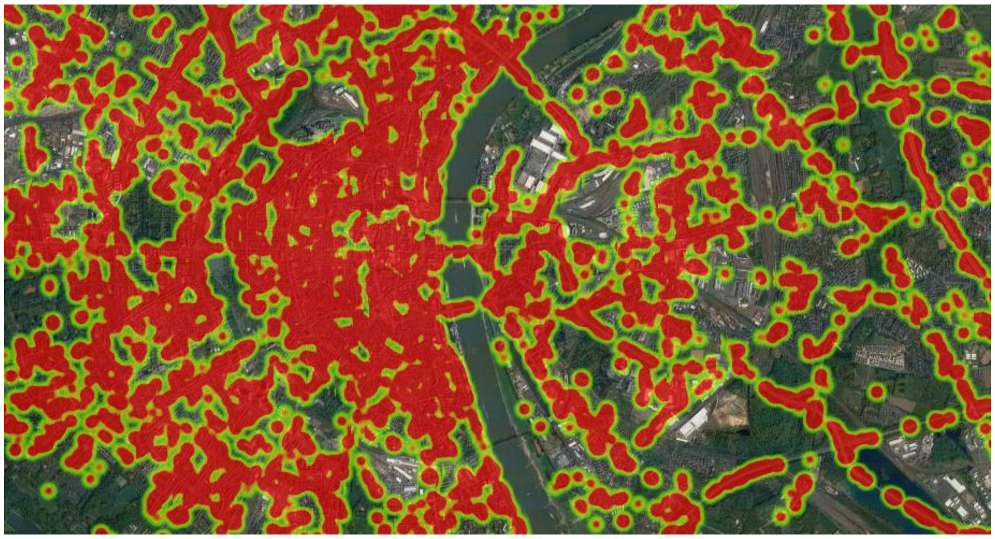

Figure 7: Unclear map at a scale of 19.244 vehicles

6 Discussion

Based on the results from the previous sections, this section discusses benefits of the proposed approach, but also current limitations.

6.1 Test Setup & Results

Installing and configuring Eclipse SUMO with TraCI to send data to the cloud was, thanks to the extensive documentation and examples, quite easy. Once set up, already available real-world scenarios can be executed without any additional configuration effort. The documentation and the available scenarios are both an result of the community around the Eclipse SUMO project which is an advantage for using Eclipse SUMO in adjacent domains like the testing of connected vehicle application. Therefore, Eclipse SUMO seems to be a really good fit to the aim of Eclipse Kuksa to break the silos between different actors and domains around the topic of connected vehicles. In this case, a developer with a background from cloud computing can focus on his applications while benefiting from the experience from the traffic simulation domain.

In general, the Eclipse Kuksa cloud by means of Eclipse Hono and the HonoInfluxConnector scaled well with the large data sets obtained from the different scenarios, but the scalability of Eclipse Hono was already proven in [23]. In contrast, testing the first version of the *Air Quality Monitor* service indicated different problems that were not considered when designing the architecture and implementing the service. One example is the improved heatmap in Figure 8, which better scales with larger amount of vehicles on the road. Another thing we improved is the querying of large and changing data sets, e. g. querying for the current emission and geolocation values of all vehicles within the Cologne scenario (06:00-06:15am) gives us 992 different vehicle data sets back within a decent time (125-1.600ms). At the same time, the finding of those issues in the developed service indicates how helpful the usage of Eclipse SUMO was in the testing process to get a fast feedback on the current implementation.

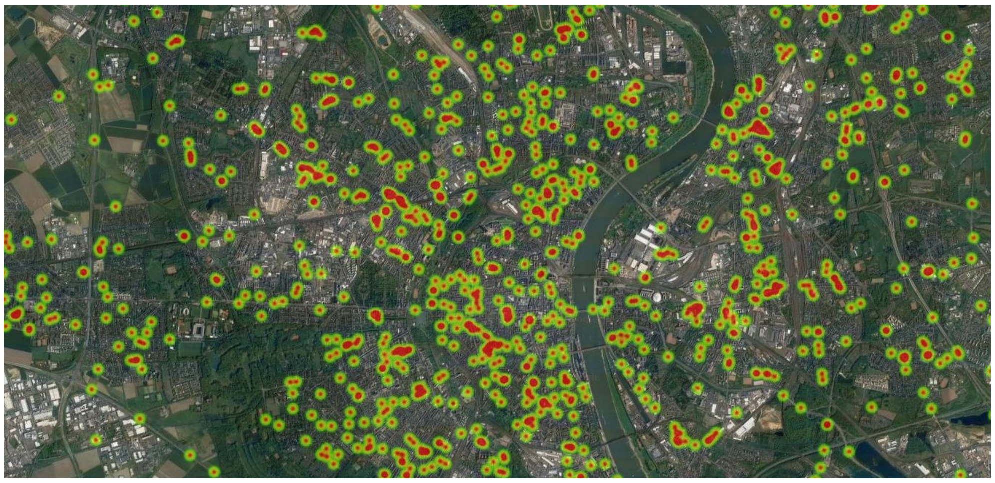

Figure 8: Air Quality Monitor for Cologne

6.2 Current Limitations

Real-Time Support Providing real-time support, for example to test safety-critical services, is always a challenge and needs thorough investigations. For most more complex scenarios, Eclipse SUMO does not provide its results in real-time in respect to the "real-world" clock of the actual cloud services that are not part of the simulated world. But for evaluating the cloud architecture and services regarding their responsiveness and reaction time, the requests need to be created with realistic timing in the real world. It should be possible to record data which is then replayed using time-stamps from the simulation. But the services in the cloud can also send control information to the vehicle. Those control commands interfere with the simulation and its results. Hence, the control information needs to be provided to Eclipse SUMO during the simulation run-time to allow a more realistic evaluation of the overall Eclipse Kuksa system.

Simulation Time Running the simulation on a local machine with limited resources is for large scenarios not suitable as the simulation gets really slow at a certain point. Because of that we might use more powerful computing resources from a cloud provider in the future to execute the simulation. However, for extensive tests of the cloud capabilities, we mostly need data from larger scenarios. For some test cases the granularity of the information processed and provided in the Eclipse SUMO simulation could already be too high for the needs of the service to test. At the same time computing that information consumes huge parts of the computational resources. Since the needed granularity is highly dependent on the service one can not make a general statement about this.

Additional Features For our approach and the testing of the Eclipse Kuksa platform, we could facilitate additional functionality around the features provided by Eclipse SUMO itself. For instance, in a real-world deployment the Eclipse Kuksa In-Vehicle platform allows to run application within the vehicle. Thus, it would be valuable to simulate the execution of applications in the vehicle. Also the integration of advanced sensor technology in vehicles to provide additional environment data like weather conditions on the road would be valuable, especially when dealing with safety-related applications such as a cloud-based hazard warning.

Creation of Complex Scenarios One drawback of using Eclipse SUMO for testing is that setting up and tuning an individual scenario can become pretty complex and time consuming. This is especially the case for developers that are not experienced with Eclipse SUMO which might be the case for many cloud developers. Even though there are a couple of tools for creating a scenario, the risk of introducing major errors and issues in the simulation is rather high. This is especially the case for larger scenarios that are at the same time more interesting for the testing of most cloud services. In exchange, the overhead of creating fitting scenarios is mitigated a bit by the number of scenarios that are already available.

7 Conclusion

With an emerging number of connected vehicles on the road, scalable and reliable service implementations are required that keep working under changing conditions. Nevertheless, testing cloud-based services for connected vehicles is still a challenge due to the non-availability of large and flexible real-world data sets. In this paper, we presented an early approach for bridging the gap between the traffic simulator Eclipse SUMO and the connected vehicle ecosystem Eclipse Kuksa by sending and storing vehicle-specific simulation data to a remote database via MQTT. Based on an exemplary *Air Quality Monitor* service, we demonstrated how to use simulation data for testing the scalability and functionality of connected vehicle services within the cloud. We were able to test different components of a connected vehicle ecosystem with a particular focus on validating the functionality of connected vehicle services. Testing the ecosystem with large and different real-world simulation data facilitates improving services regarding flexibility, efficiency, and visualization. In addition, we discussed the benefits of our approach and current limitations. As soon as all software used for this article is IP checked, the source code will be made available at the Eclipse Kuksa repository[15].

[15]https://github.com/eclipse/kuksa.apps

For the future, we are planning to extend our current approach in different aspects. Although the *Air Quality Monitor* service provided a solid evaluation basis for our proposed approach, we noticed that we need a more sophisticated connected vehicle service which considers even more the nature of connected vehicles. This includes, for example, the integration of other data sources, e. g. from the smart city, for the reason of data refinement. Also the integration of Command & Control functionalities to implement, for example, a cloud-based platooning service for trucks would be valuable. Simpla[16], a platooning plugin for the TraCI Python client, could be used to define and control the behavior of platooning vehicles. Another framework in this regard is PLEXE [24], which is an Open Source extension to Veins for the simulation of platooning systems in realistic scenarios.

As mentioned in Section 6, one drawback for our approach is that Eclipse SUMO does not deliver its results in real-time from the perspective of the cloud. For sending telemetry information from the vehicle to the cloud, we might evaluate how this gap could be closed by recording the data from the simulation and replaying it to the cloud afterward. However, we need to further investigate how to support Command & Control features where the cloud interferes with the simulation and its results by sending commands to the vehicles. In such cases, the cloud testing can not be done after the simulation because the results from the simulation depend on the input from the cloud service.

Another aspect is that the current setup neglects the effects of the communication link between the vehicles and the cloud backend, which could be considered for end-to-end scenarios. This and more functionality is provided by simulation run-times that couple Eclipse SUMO with simulators from other domains. For future work we plan to investigate ways to use such a framework like VSimRTI [22] to further improve the evaluation of the Eclipse Kuksa platform.

Moreover, the implementation and evaluation with different simulation scenarios would foster additional insights about our approach. It is furthermore interesting and necessary to investigate how our approach and Eclipse SUMO could be integrated with existing test processes, pipelines, and frameworks that are commonly used in the development of cloud applications.

References

[1] Laura Bieker, Daniel Krajzewicz, AntonioPio Morra, Carlo Michelacci, and Fabio Cartolano. Traffic simulation for all: A real world traffic scenario from the city of bologna. In Michael Behrisch and Melanie Weber, editors, *Modeling Mobility with Open Data*, pages 47–60. Springer International Publishing, 2015.

[2] A. Bounceur, L. Clavier, P. Combeau, O. Marc, R. Vauzelle, A. Masserann, J. Soler, R. Euler, T. Alwajeeh, V. Devendra, U. Noreen, E. Soret, and M. Lounis. Cupcarbon: A new platform for the design, simulation and 2d/3d visualization of radio propagation and interferences in iot networks. In *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 1–4, Jan 2018.

[3] John Carnell. *Spring Microservices in Action.* Manning Publications Co., Greenwich, CT, USA, 1st edition, 2017.

[4] M. Chernyshev, Z. Baig, O. Bello, and S. Zeadally. Internet of things (iot): Research, simulators, and testbeds. *IEEE Internet of Things Journal*, 5(3):1637–1647, June 2018.

[16]https://sumo.dlr.de/wiki/Simpla

[5] L. Codeca, R. Frank, and T. Engel. Luxembourg sumo traffic (lust) scenario: 24 hours of mobility for vehicular networking research. In *2015 IEEE Vehicular Networking Conference (VNC)*, pages 1–8, Dec 2015.

[6] L. Codeca, R. Frank, S. Faye, and T. Engel. Luxembourg sumo traffic (lust) scenario: Traffic demand evaluation. *IEEE Intelligent Transportation Systems Magazine*, 9(2):52–63, Summer 2017.

[7] L. Codecá and J. Härri. Towards multimodal mobility simulation of c-its: The monaco sumo traffic scenario. In *2017 IEEE Vehicular Networking Conference (VNC)*, pages 97–100, Nov 2017.

[8] J. Contreras-Castillo, S. Zeadally, and J. A. Guerrero-Ibañez. Internet of vehicles: Architecture, protocols, and security. *IEEE Internet of Things Journal*, 5(5):3701–3709, Oct 2018.

[9] Soumya Kanti Datta, Mohammad Irfan Khan, Lara Codeca, B Denis, Jérôme Härri, and Christian Bonnet. Iot and microservices based testbed for connected car services. In *2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pages 14–19. IEEE, 2018.

[10] Eclipse Ditto. Eclipse ditto, 2019. https://www.eclipse.org/ditto/, last viewed February 2019.

[11] Eclipse Foundation. Eclipse che, 2019. https://www.eclipse.org/che/, last viewed February 2019.

[12] Eclipse Foundation. Eclipse paho, 2019. https://www.eclipse.org/paho/, last viewed February 2019.

[13] The Eclipse Foundation. Eclipse kuksa, 2019. https://www.eclipse.org/kuksa/about/, last viewed February 2019.

[14] Maxime Guériau, Romain Billot, Nour-Eddin El Faouzi, Julien Monteil, Frédéric Armetta, and Salima Hassas. How to assess the benefits of connected vehicles? a simulation framework for the design of cooperative traffic management strategies. *Transportation Research Part C: Emerging Technologies*, 67:266 – 279, 2016.

[15] The Eclipse hawkBit Project. Eclipse hawkbit, 2019. https://www.eclipse.org/hawkbit/, last viewed February 2019.

[16] InfluxData. Influxdb, 2019. https://github.com/influxdata/influxdb, last viewed February 2019.

[17] J. Li, A. Doniec, J. Boonaert, and G. Lozenguez. Which traffic simulator is suitable for customized behaviors in multi-modal scenarios. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3791–3796, Nov 2018.

[18] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. WieBner. Microscopic traffic simulation using sumo. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2575–2582, Nov 2018.

[19] N. Lu, N. Cheng, N. Zhang, X. Shen, and J. W. Mark. Connected vehicles: Solutions and challenges. *IEEE Internet of Things Journal*, 1(4):289–299, Aug 2014.

[20] Riccardo Petrolo, Valeria Loscrì, and Nathalie Mitton. Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms. *Transactions on Emerging Telecommunications Technologies*, 28(1), 2017.

[21] The Eclipse Hono Project. Eclipse hono, 2019. https://www.eclipse.org/hono/, last viewed February 2019.

[22] Robert Protzmann, Björn Schünemann, and Ilja Radusch. Simulation of convergent networks for intelligent transport systems with vsimrti. *Networking Simulation for Intelligent Transportation Systems: High Mobile Wireless Nodes*, pages 1–28, 2017.

[23] Jens Reimann. We scaled iot – eclipse hono in the lab, 2018. https://dentrassi.de/2018/07/25/scaling-iot-eclipse-hono/, last viewed February 2019.

[24] Michele Segata, Stefan Joerer, Bastian Bloessl, Christoph Sommer, Falko Dressler, and Renato Lo Cigno. PLEXE: A Platooning Extension for Veins. In *6th IEEE Vehicular Networking Con-*

ference (VNC 2014), pages 53–60, Paderborn, Germany, December 2014. IEEE.

[25] R. E. Shannon. Introduction to the art and science of simulation. In *1998 Winter Simulation Conference. Proceedings (Cat. No.98CH36274)*, volume 1, pages 7–14 vol.1, Dec 1998.

[26] J. E. Siegel, D. C. Erb, and S. E. Sarma. A survey of the connected vehicle landscape—architectures, enabling technologies, applications, and development areas. *IEEE Transactions on Intelligent Transportation Systems*, 19(8):2391–2406, Aug 2018.

[27] Hitachi Data Systems. The internet on wheels and hitachi, ltd., 2015. https://theinternetofthings.report/view-resource.aspx?id=450, last viewed January 2019.

[28] Axel Wegener, MichałPiórkowski, Maxim Raya, Horst Hellbrück, Stefan Fischer, and Jean-Pierre Hubaux. Traci: An interface for coupling road traffic and network simulators. In *Proceedings of the 11th Communications and Networking Simulation Symposium*, CNS '08, pages 155–163, New York, NY, USA, 2008. ACM.

[29] I. Yaqoob, E. Ahmed, I. A. T. Hashem, A. I. A. Ahmed, A. Gani, M. Imran, and M. Guizani. Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges. *IEEE Wireless Communications*, 24(3):10–16, June 2017.

Acknowledgments This work has been partially funded by the German Federal Ministry of Education and Research (BMBF) under funding no. 01—S16047D as part of the APPSTACLE-Public project.