A Flow Based Architecture for Efficient Distribution of Vehicular Information in Smart Cities

Roger Young
Department of Computer & Software Engineering
Athlone Institute of Technology
Athlone, Ireland
r.young@research.ait.ie

Dr. Sheila Fallon,
Department of Computer & Software Engineering
Athlone Institute of Technology
Athlone, Ireland
sheilafallon@ait.ie

Dr. Paul Jacob
Department of Computer& Software Engineering
Athlone Institute of Technology
Athlone, Ireland
pjacob@ait.ie

Dr. Denis O Dwyer
Tech-Auto Ltd
Tullamore, Ireland
dodwyer@tech-auto.ie

***Abstract*—The Introduction of the Internet of Things has made the vision of "Smart Cities" a very reachable goal. Aggregating data from a wealth of sensors throughout a city, with the aim of improving quality of life for its dwellers has become a major focus for city developers. The automotive sector is in prime position to play a pivotal role in the success of Smart Cities. Vehicle Telematics, the process of gathering and transmitting Geo-Enriched vehicle data, is seen as a key enabler in improving mobility in urban areas. Modern vehicles generate up to 5GB an hour, with newer models generating far more. How this data is to be processed and transmitted is a hotly researched topic. This work focuses on raw vehicle data extraction and the valuable insights it holds in terms of fuel consumption and driver behaviour. Data is extracted and distributed from the vehicle using a novel architecture consisting of the FMS standard and a Flow Based Programming inspired approach. Road Side Units (RSUs) are set to play a key role in the area of vehicle communication in urban areas. This work will evaluate the level of governance the RSUs can achieve in Vehicle-to-Infrastructure (V2I) scenarios.**

Keywords—Smart Cities, Connected Vehicles, RSUs, Vehicle Telematics, Flow Based Programming, FMS, OBDII, Apache Minifi, Apache Nifi

I. INTRODUCTION

In time, connected vehicles may be seen as the success story of Internet of Things. Put simply, one vehicle is a moving cluster of sensors that can provide a wealth of information on its surroundings. If harvested correctly, thousands of vehicles moving throughout a city can provide insight into traffic flows, driving behaviour, road and weather conditions and much more. There has been much research on extracting vehicle data from the vehicle network (CAN-Bus), however, the majority has acquired the data via the OBDII interface. In this work, a comparison is provided between OBDII and another standard called FMS. OBDII is widely used due to its ease of access and cheap devices, however, many experts in the field consider it unreliable and potentially harmful to the vehicle network. FMS on the other hand, was designed specifically for fleet management.

An efficient architecture is required to seamlessly transmit vehicle data. This work aims to show how Flow Based Programming (FBP) may be well suited to ingesting and processing this large vehicle data in real time, while gaining valuable insight that may improve traffic and mobility issues in urban areas. Through FBP, it may be possible to add compute power anywhere in the network from source to destination, enabling critical decisions to be made and appropriate actions be taken with minimum delay. Further processing can be achieved at a higher location in the FBP architecture.

This work was sponsored by Irish Research Council & Tech-Auto Ltd Grant no. EBPPG/2018/206

A. CAN-Bus Network

Modern vehicles are equipped with a highly complex network of around 70 Electronic Control Units (ECUs). ECUs communicate with each other over a standard communication protocol known as Controller Area Network (CAN). First introduced by Bosch in 1991, the CAN aimed to simplify the communication between different ECUs within a vehicles using one single pair of twisted wires, known as CAN-High and CAN-Low. CAN provides a communication rate of up to 1Mbps [1], and utilizes a broadcasting system. Many CAN signals are broadcasted at a rate of every 10ms. There are two well-known interfaces and standards that connect to the CAN-Bus; OBDII and FMS. While the CAN-Bus can be regarded as the networking system, OBDII and FMS can be viewed as languages that interpret the CAN data.

OBDII (On-board Diagnostics) [2] was first introduced in the U.S in 1996, and 2001 in Europe. All OBDII compliant vehicles will have an OBDII port within two feet of the steering wheel. The OBDII standard specifies the connector and its pinout, the electrical signaling protocols available, and the messaging format. The goal of OBDII was to provide communication to the CAN-Bus, giving access to real-time data from different ECUs of the vehicle. However, the purpose was to aid in locating issues within the vehicle, not for tracking purposes as it has been used over the past decade.

There are over 200 available parameters (PIDS) available via OBDII. PIDs can be requested using an OBDII adapter, which can transmit the output to a smartphone using Bluetooth or Wi-Fi. The recommended rate of requesting PIDs is 1Hz. Due to its ease of access, a large number of telematics companies identified the OBDII port as a way of obtaining vehicle data to support the delivery of new fleet telematics services and started to exploit it in their solutions [3]. However, OBDII works on a request basis, with many experts regarding it as an unsuitable, and intrusive way of gaining data.

In 2002, the six major truck manufacturers (Volvo, Scania, Iveco, MAN, DAF, and Mercedes-Benz) created a standardized vehicle interface for GPS based tracking systems, called the

FMS (Fleet Management System) standard [4]. FMS is a subset of CAN-Bus signals that is specifically created for fleet management. We are currently on FMS standard version 4, which includes over 30 parameters. FMS can read CAN signals at high frequency, with a data refresh rate of 10 milliseconds. Although FMS was initially only available in trucks and buses, the recent introduction of FMS gateways allow for the standard to be used in passenger cars and vans. Unlike its OBDII counterpart, FMS gateways have access to all CAN-bus parameters, which, in some vehicles is greater than 2500.

B. Flow Based Programming

Although Flow Based Programming has been around since the late 1960s, it has seen a recent surge in popularity. Projects such as NoFlo, NodeRed, Apache Nifi, and Cisco's Kinetic have noticed and taken advantage of the strengths of FBP and the processing of data flows, which is a major requirement of the modern data-driven applications, thus making it a viable programming model for this oncoming paradigm shift to IoT.

FBP [5] can be viewed as a technology where an application is constructed as a network of asynchronous processes exchanging data chunks and applying transformations to them. FBP is a model ideally suited to IoT as its aim is to concentrate on the data and data streams first before deciding what processes are needed to convert between the different data streams. In object-orientated programming, you have to decide on the object classes, and then decide what messages each class should be able to respond to [6]. While it is possible to create real time interactive IoT applications using traditional programming tools, it can quite often be a difficult task.

One of the prime advantages of FBP is its modularity, meaning the degree to which a system's components may be separated and recombined. Nate Edwards of IBM [7] coined the term "configurable modularity" to denote an ability to reuse independent components just by changing their interconnections. A main characteristic of a system that exhibits "configurable modularity" is that you can build them out of "black box" reusable modules. While it is necessary to connect them together, they do not have to be modified to make this happen. The author of [8] discusses many of the inherent benefits with the data flow /flow-based programming paradigm, including implicit pipeline parallelism, exceptional composability, testability, inspectability and code re-use.

Based on the identified research opportunity, answering the following research questions is the aim of this work:

1) Can a multi-tier Flow Based Programming architecture enhance the distribution of vehicle sensor information in Smart cities?

2) What criteria should be used to determine where in the network processing should be performed?

The layout of this paper is as follows. Related work that informed and inspired this project is provided in Section II. A description of the evaluation of OBDII and FMS is provided in section III, followed by an overview of our reference architecture and implementation in section IV and V. We finish with a conclusion and future work plan in Section VI.

II. Related Work

Traditionally, IoT devices and wireless sensor networks (WSNs) were commonly designed to transfer data to remote servers and computing Clouds as discussed in [9], [10], and [11]. More recent work such as [12] propose a real-time job scheduler in Hadoop for Big Data. The scheduler aims to manage cluster resources in such a way that the real time jobs will not be affected by the long running (batch jobs), and vice-versa. The case study is applied as support for Smart City applications, taxicabs in particular. Although efficient in its design, all data is transferred to a single location in a completely centralized scheduler.

The authors of [13] propose an architecture for streaming spatio-temporal event processing, analysis and near real-time visualization. It is comprised of fully open source software and focuses on a use case involving a fleet of snow ploughs. Information on the plows is published to the public, as well as road coverage data. Technologies such as LocationTech GeoMesa, GeoTools, and GeoServer are used to enable geospatially-aware complex event processing (CEP) solutions. There are some challenges when it comes to processing stream geospatial events, such as handling differences in event and processing time. However, the inclusion of software such as Apache Storm [14]and Apache Kafka [15] into this architecture by the authors addressed such problems.

Hortonworks [16] demonstrated the simulation of bi-directional data communication between an on-vehicle platform and the cloud. This was achieved by loading Apache MiNiFi onto a custom Qualcomm modem located in a connected car, allowing the vehicles to transmit data to their HDF (Hortonworks Data Flow) platform [17]. The demo highlighted how to deliver critical capabilities for vehicle communication. The centralized HDF platform could process key data such as speed and geo-location in real-time. Minifi could manage how and when to transmit much larger but less time-relevant data, (system diagnostics, etc.) This data could be batched on the vehicle and sent in bursts over known Wi-Fi locations. This is an effective solution as bandwidth over LTE is expensive.

In [18] we propose a distributed data processing architecture for Edge devices in an IoT environment. Our approach focuses on a vehicular trucking use case. The traditionally centralized Storm processes such as calculating average speeds and aggregating driver errors are recreated on the Edge devices using a combination of Apache MiNiFi and the user's custom-built programs. This work focused on transmitting "un-normal" driving events such as speeding, lane departure etc, while storing all data on board which can be uploaded in bursts over known Wi-Fi spots. This was an effective use case that provided a solution in transmitting time dependent data in real-time yet storing all data locally. However, communication was only one directional, as information was not sent from the central server to the Edge devices.

There are a number of distributed middleware platform implementations of Edge-Computing that provide in network processing capability that leverages computation resources of

Edge devices [19], [20]. However, there are limitations in these works when it comes to scalability and mobility.

In recent years, several runtime environments such as [21], [22]and [23] have begun to implement FBP inspired approaches. UFlow is proposed in [24]. Uflow is a concept of data flow transformation closer to the source, on the devices with constrained resources. The authors analyzed two tier IoT architecture composed of devices and the Cloud. The scientific contribution of the paper as well as the concept of the data flow transformation is that the *UFlow* framework can be executed on a variety of resource-constrained embedded devices, and can be implemented on a NodeRED platform. In [25], a novel architecture called FogFlow is proposed. The aim is to ease the service orchestration and scalability for geo-distributed smart cities. Through the implementation of the dataflow programming model, developers in FogFlow only define a service topology (DAG) and decompose the IoT service into multiple processing units (black boxes).

Monitoring CAN-Bus signals has a wealth of literature over the past decade, focusing on driver behaviour, CO2 emissions and fuel economy in particular. Different variations of algorithms have been used for calculating fuel economy via OBDII [26], [27]. In [28], the authors performed a comparison between GPS and OBDII, reporting that GPS and smartphone sensor based techniques, combined with map and/or crowd-sourced data, can achieve greater than 94% correlation to OBDII information with regards to vehicle speed, acceleration etc. In [29] a methodology is developed to calculate, in real-time, the consumption and environmental impact of spark ignition and diesel vehicles from a set of OBDII parameters.

However, there are limitations in the literature regarding comparisons of vehicle data. A comparison of OBDII and FMS is presented in [30]. With a specific requirement in mind, the authors measure a limited number of parameters from both standards. Vehicle speed and RPM (Engine Revs per minute) are measured and compared, with only slight differences recorded. The only significant difference in FMS and OBDII was recorded when monitoring fuel economy. Testing shows the FMS fuel rate to be highly accurate compared to OBDII. However, only a basic algorithm using OBDII parameters was tested.

The same authors again stated in [31] that when comparing OBDII and FMS, differences in vehicle speed and RPM can only be noticed in short measuring time, up to 20 seconds. With regards to calculating fuel economy, corresponding calibration factors are required for OBDII. However, as previously mentioned, only one OBDII fuel algorithm was tested.

A system was proposed in [3] to log OBDII data and direct CAN data at the same time. This work primarily focused on the rates of data acquisition using a limited number of CAN and OBDII parameters for comparison. The OBDII adapter had a maximum request rate of 9HZ for one parameter, compared to 25 HZ for CAN. Although the authors state that the quality of analysis can be improved by having more information, results on improved accuracy due to an increase in data granularity were not provided.

III. OBDII & FMS Comparitive Evaluation

A significant comparison of both standards was evaluated over a three month period in which devices were installed in a vehicle that monitored OBDII and FMS data over the same trips. In total, 11 FMS parameters, and 22 OBDII parameters (decreased to 8 to improve granularity to 2Hz), were tested simultaneously. Monitoring basic parameters such as speed and RPM provide little difference between standards, however, FMS proved more accurate in stop and go traffic, as shown in Figure 1. OBDII had a tendency to drop to zero during slow speeds, which can greatly skewer statistical analysis, as shown in Table 1. FMS also outperformed OBDII in fuel consumption monitoring, data granularity and overall accuracy. Additional FMS parameters such as Clutch and Brake usage, and accelerator pedal position also give FMS an edge in monitoring driver behaviour. Table 2 gives an overview of the better performing standard in a series of tested scenarios.

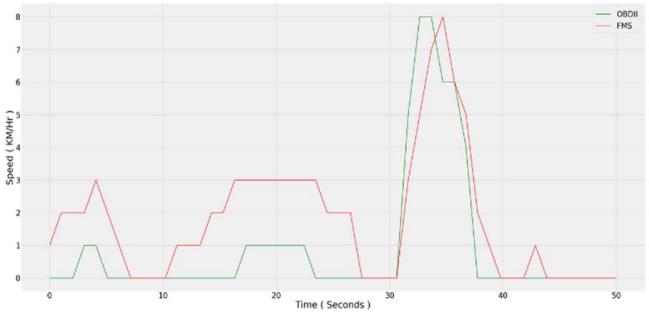

Figure 1: Stop and Go Traffic over a 50 second period. OBDII drops to zero for long periods, even when vehicle is moving slowly

Table 1: Statistical analysis on stop and go traffic as represented in Figure 1

Description	OBDII Speed	FMS Speed
Count	50	50
Mean	0.9	1.76
Std	2.06	1.92
Min	0	0
25%	0	0
50%	0	1.5
75%	1	3
Max	8	8

Table 2: Scenarios that were tested during comparison of OBDII and FMS

Scenarios	Preferred Standard	
Basic Parameters	FMS	OBDII
Urban Traffic	FMS	
Data Granularity	FMS	
Data Accuracy	FMS	
Fuel Monitoring	FMS	
Security	FMS	
Available Parameters	FMS	
CO2 Emission Calculations	FMS	
Cost of devices		OBDII
User Friendly		OBDII

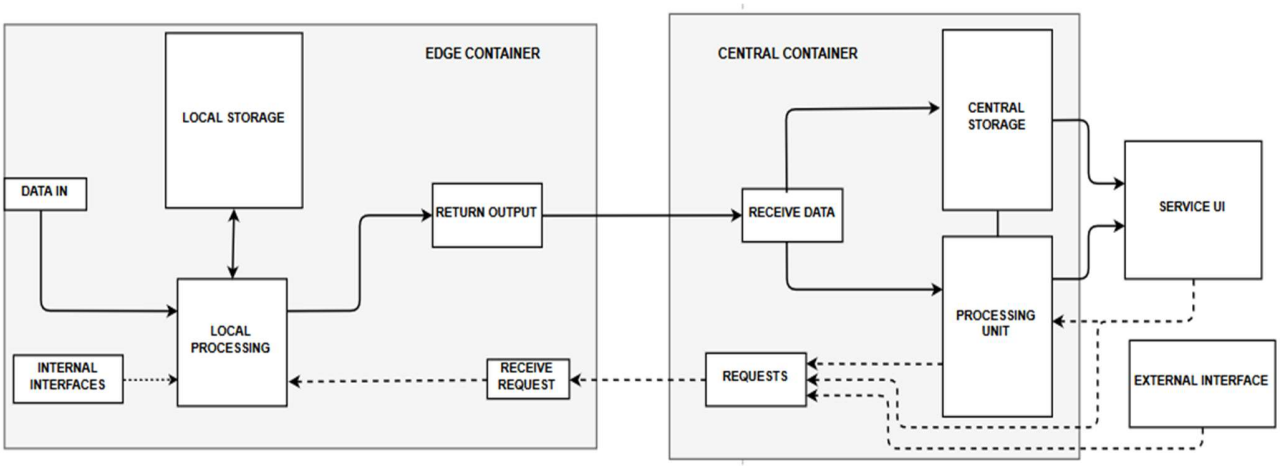

Figure 2: Current Reference Architecture showing dataflow and communication between edge and central container

For these reasons, FMS is the chosen standard for the proposed work. Precision is key in terms of monitoring traffic and driver behaviour in urban areas. While OBDII can adequately provide valuable information, at a cost effective price, it performs poorly in too many areas to be considered on a wide scale.

IV. Architecture

Previous publications [18], [32], and [33] were devoted to various implementations of our reference architecture. The architecture consists of three main components:

IoT Sensors – The IoT sensors create the data. In most cases, these will have limited resources, and transmit data to an edge gateway device for processing.

Edge Container – The edge container represents an edge device with local processing capability. This device acts as a gateway for IoT sensors. Custom programs of any language can be incorporated into the local processing unit. Data analysis can be influenced by internal factors such as network connectivity, CPU or RAM usage, external factors such as latency, or requests received from the central controller. The edge container can be viewed as an agent to the central container. In future work, the edge container is placed in the vehicles, and also RSUs, which allow for performing complex rules on data in motion to intelligently reduce, compress and process data throughout the network.

Central Container. The central container (or controller) can be viewed as a cloud server with large processing and storage capacity. Figure 2 represents an illustration of the architecture to date. The service UI allows the user to interact with the dataflows between the edge and central container. Through the service UI, the user can pass new logic to be performed on the edge and central container. Incoming results can be viewed in real-time through the UI.

The central container ingests data from the edge containers. Data is routed to a local database for storage, or the processing unit for analysis. The processing unit performs model building as it has access to a learning algorithm repository and the local database.

The central container acts as a coordinator for the edge devices/agents. It has the ability to send requests and information to its agents. This control data may be influenced by external factors or user requests directly from the service UI.

V. Implementation

Acquiring CAN-Bus data in a non-intrusive way can be problematic. Cutting or soldering connections to the vehicle CAN has potential to cause damage. However, recently, the development of a 'CAN-Bus clip', Figure 3, a non-intrusive reading device that clips to the outside of the cables, has addressed this issue. The clip is placed over the CAN-High and CAN-Low wires, and senses the data passing through the wires, directly from the CAN network. The CAN-Bus clip is safe, reliable and coded to understand each make & model. The clip is connected to a FMS gateway, which transforms the CAN signals into the FMS format. A Raspberry Pi, acting as the edge container, ingests the FMS data via Bluetooth, and performs local processing, Figure 4.

The Raspberry Pi is equipped with FBP technology, allowing for the creation of data flows that perform computation on the FMS information in real time. Figure 5 shows a sample of the logic that is performed on the FMS parameters, creating new parameters that are important in monitoring driving behaviour. Engine idle time for example, can be calculated from raw parameters such a vehicle speed and RPM. When RPM is above zero, but vehicle speed is zero, the engine is running and considered idle. Hard acceleration and deceleration have been

Figure 3: Contactless CAN clip. CAN-High and LOW wires pass through the clip, in which all CAN signals are sensed and transmitted to FMS Gateway

defined as an increase/decrease of 2.598 m/s^2 for the starting vehicles and 1.4705 m/s^2 when driving [34]. Therefore, hard acceleration/deceleration could then be calculated using the following equation:

$$A = (ms - ms.shift > 1.4705)$$

Where A is aggressive driving behaviour, ms is the current value for m/s^2 and ms.shift is the value of m/s^2 one second previous. Fuel rate is a parameter made available on the CAN-Bus, and is measured in litres per hour. However, fuel economy is commonly measured in volume of fuel over distance (Litres per 100km in Europe, or Miles per Gallon in the U.S). Fuel economy is calculated with the following equation:

$$Fuel\ Economy\ = 100\frac{FR}{VS}$$

Where FR is Fuel Rate and VS is Vehicle Speed.
All vehicle parameters are then transmitted to the nearest RSU via whatever communication protocol is in use (WIFI, 802.11P, also known as WAVE [35] , 4G or 5G).

In our proposed architecture, the RSU is also equipped with FBP technology that can further process incoming vehicle data. Multiple vehicle data may be aggregated by the RSU, as shown in Figure 6. The RSU will have bi-directional communication with nearby vehicles, giving it the capacity to change the computation that occurs on the vehicles if necessary. (For example, if network latency passes a certain threshold due to too many connected vehicles).

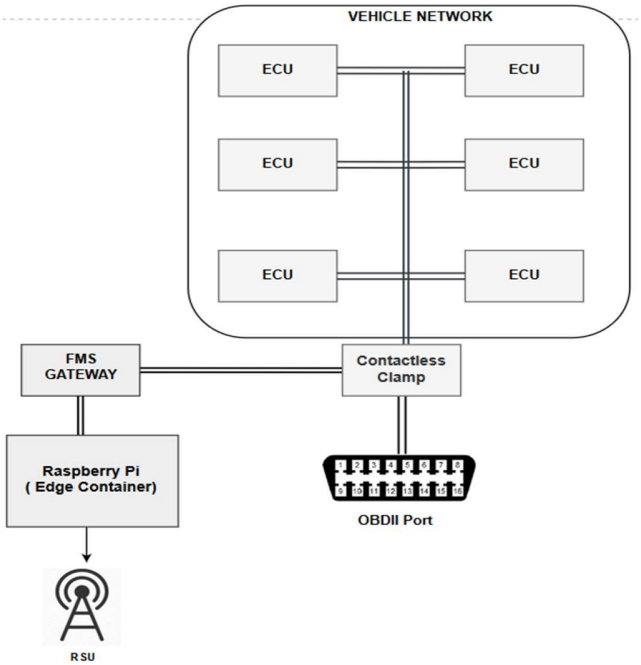

Figure 4: CAN-Bus signals intercepted by clamp and transmitted to FMS gateway. Edge Container performs local processing before transmission to RSU.

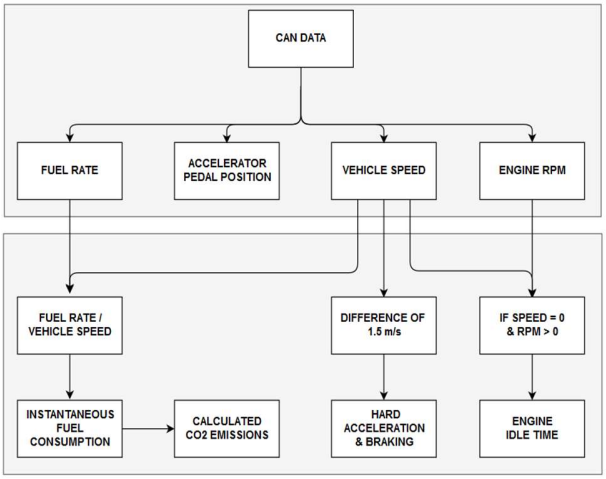

Figure 5: Sample of Dataflow on Edge Container. Ingesting raw data and performing logic locally before transmitting to RSU

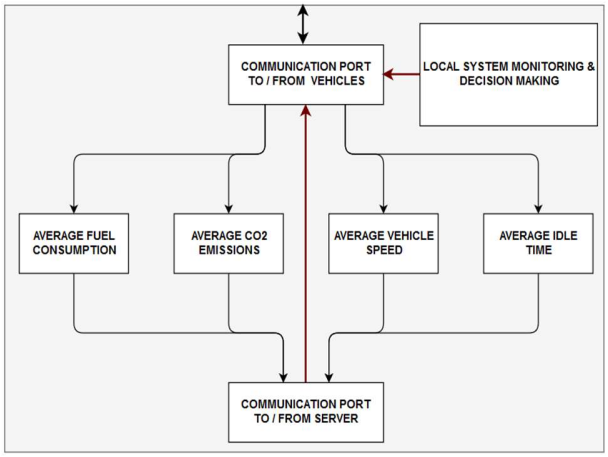

Figure 6: Dataflow and Logic performed on RSU. Tasks may be passed to vehicles from RSU or Central Controller

VI. Conclusions and Future Work

The contributions of this work thus far are as follows:

1) A comparative evaluation of OBDII and FMS standard, making available an open-source dataset to the research community [36] .

2) A standard-based programming model for vehicle telematics in smart cities. We extend a dataflow programming model across heterogeneous network nodes.

3) An efficient bi-directional vehicle telematics architecture capable of advanced local processing of sensor data.

To date, the focus has been on Edge to Central Container communication. The inclusion of RSUs into this scenario aims to further enhance the architectures role in the handling of vehicle telematics in a smart city. The future work plan is to test the scalability, and efficiency of the architecture. This will be achieved via simulation tools. Simulations will utilize the WAVE protocol, as it is likely to be implemeted in future Vehicle-to-Infrastructure scenarios. The aim will be to determine the role of the RSU. What level of Governance can

the RSU achieve over connected vehicles? In busy traffic scenarios, where latency becomes an issue, can the RSU change the level of local processing that occurs on the vehicle?

VII. REFERENCES

[1] O. Avatefipour and H. Malik, "State-Of-The-Art Survey on In-Vehicle Communication "Can-Bus" Security and Vulnerabilities," ArXiv Prepr, Michigan, 2018.

[2] "wikipedia.org/wiki/OBD-II_PIDs," [Online]. Available: https://en.wikipedia.org/wiki/OBD-II_PIDs. [Accessed 2 June 2018].

[3] K. Khosravinia, M. Hassan, R. Raman and S. Al-Haddad, "Improved Latency of CAN Vehicle Data Extraction Method," International Conference on Internet of Vehicles, 2018.

[4] "fms-standard.com/," [Online]. Available: http://www.fms-standard.com/. [Accessed 1 August 2018].

[5] J. P. Morrison, "Flow-Based Programming: A new approach to application development," CreateSpace, 2010.

[6] J. Morrison, "Comparison between FBP and Object-Oriented Programming," jpaulmorrison.com, [Online]. Available: http://www.jpaulmorrison.com/fbp/oops.htm. [Accessed 29 09 2017].

[7] N. Edwards, "The Effect of Certain Modular Design Principles on Testability," IBM Research Report, NY, 1974.

[8] "FBP inspired data flow syntax," bionics.i, 16 July 2016. [Online]. Available: http://bionics.it/posts/fbp-data-flow-syntax. [Accessed 20 September 2017].

[9] W. Kurschl and W. Beer, "Combining cloud computing and wireless Sensor Networks," Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services. ACM, Hagenburg, 2009.

[10] S. K. Dash, S. Mohapatra and P. K. Pattnaik, "A survey on applications of wireless sensor network using cloud computing," International Journal of Computer Science & Emerging Technologies 1, no.4, 2010.

[11] M. Yuriyama and T. Kushida, "Sensor-cloud infrastructure-physical sensor management with virtualized sensors on cloud computing," IEEE 13th International Conference on Network-Based Information Systems (NBiS), Tokyo, 2010.

[12] C. Barbieru and F. Pop, "Soft Real-Time Hadoop Scheduler for Big Data Processing in Smart Cities," IEEE, Bucharest, 2016.

[13] J. N. Hughes, M. D. Zimmerman, C. N. Eichelberger and A. D. Fox, "A Survey of Techniques and Open-Source Tools for Processing Streams of Spatio-Temporal Events," ACM, Charlottesville, 2016.

[14] R. Evans, "Apache Storm a Hands on Tutorial," IEEE, Urbana, 2015.

[15] HTC Global Services, "Apache Kafka – Your Event Stream Processing Solution," htcinc.com.

[16] Guest Author, "Hortonworks.com," Hortonworks, 8 June 2016. [Online]. Available: https://hortonworks.com/blog/qualcomm-hortonworks-showcase-connected-car-platform-tu-automotive-detroit/. [Accessed 11 February 2017].

[17] "HortonWorks DataFlow," Hortonworks, 2016.

[18] R. Young, S. Fallon and P. Jacob, "An Architecture for Intelligent Data Processing on IoT Devices," IEEE, Athlone, 2017.

[19] S. Cherrier and e. al, "D-LITe : Distributed Logic for Internet of Things sErvices," EEE International Conferences on Internet of Things, and Cyber, Physical and Social Computing, 2011.

[20] D. Alessandrelli, M. Petraccay and P. Pagano, "T-Res: enabling reconfigurable in-network processing in IoT-based WSNs," IEEE International Conference on Distributed Computing in Sensor Systems, Pisa, 2013.

[21] M. Blackstock and R. Lea, "Iot mashups with the wotkit," 3rd International Conference on the Internet of Things, Vancouver, 2012.

[22] H. Burgius, "Noflo–flow-based programming for javascript," 2015. [Online]. Available: http://noflojs. org.

[23] Node-RED, "nodered.org," nodered.org, October 2016. [Online]. Available: https://flows.nodered.org/flow/6fe183c197b3464a1fe4d89744e068ff. [Accessed 3 September 2017].

[24] T. Szydlo, R. Brzoza-Woch and G. C, "Flow-based programming for IoT leveraging fog computing," IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, Krakow, 2017.

[25] B. e. a. Cheng, "Cheng, Bin, et al. "FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities," IEEE Internet of Things Journal, 2017.

[26] R. Malekian, N. Moloisane, L. Nair, B. Maharaj and U. Chude-Okonkwo, "Design and Implementation of a Wireless OBD II Fleet Management System," IEEE Sensors Journal, 2017.

[27] J. Meseguer, C. Toh, C. Calafate, J. Cano and P. Manzoni, "DrivingStyles: A Mobile Platform for Driving Styles and Fuel Consumption Characterization," Journal of Communications and NEtworks, Vol. 19, NO. 2, 2017.

[28] R. Meng, C. Mao and R. Choudhury, "Driving analytics: Will it be OBDs or smartphones?," Zendrive Whitepaper, 2014.

[29] J. Meseguer, C. Calafate, J. Cano and P. Manzoni, "Assessing the Impact of Driving Behavior on Instantaneous Fuel Consumption," 12th IEEE Consumer Communications and Networking Conference (CCNC), Valencia, 2015.

[30] Z. Szalay et al, "ICT in Road Vehicles - Reliable vehicle sensor information from OBD versus CAN," Models and Technologies for Intelligent Transportation Systems, Budapest, 2015.

[31] D. Sik, T. Balogh, P. Ekler and L. Lengyel, "Comparing OBD and CAN Sampling on the go with the SensorHUB Framework," Procedia Engineering, Budapest, 2016.

[32] R. Young, S. Fallon and P. Jacob, "A Governance Architecture for Self-Adaption & control in IoT Applications," Codit, 2018.

[33] R. Young, S. Fallon and P. Jacob, "Dynamic Collaboration of Centralized & Edge Processing for Coordinated Data Management in an IoT Paradigm," AINA, 2018.

[34] E. Choi and E. Kim, "Critical Aggressive Acceleration Values and Models for Fuel Consumption When Starting and Driving a Passenger Car Running on LPG," International Journal of Sustainable Transportation, 2017.

[35] I.-S. S. Board, "Draft Guide for Wireless Access in Vehicular Environments (WAVE) Architecture," The Institute of Electrical and Electronics Engineers, Inc., 2016.

[36] R. Young, S. Fallon, P. Jacob and D. O'Dwyer, "Vehicle Telematics as a Key Enabler in the Development of Smart Cities," *Unpublished,* 2019.

[37] "A Survey of Techniques and Open-Source Tools for Processing Streams of Spatio-Temporal Events".

[38] A. Zavalko, "Applying energy approach in the evaluation of eco-driving skill and eco-driving training of truck drivers," Transportation Research Part D, 2018.