Combining Eclipse IoT Technologies for a RPI3-Rover along with Eclipse Kuksa*

Robert Höttger[1], Mustafa Özcelikörs[1], Philipp Heisig[1], Lukas Krawczyk[1], Pedro Cuadra[1], and Carsten Wolff[1]

***Abstract*— Developing open source software has proven to provide benefits to customers and OEMs vise versa [1]. IoT and Cloud development activities have already been around for decades and huge communities have grown among an immense variety of applications. However, the vehicle domain just recently approaches the connected era and still faces many challenges stemming from security, safety, reliability, real-time, or more demands. Such demands are also the reason for mandatory adaptations of existing technologies. Hence, Kuksa, i.e. an open-source platform, addresses such specific demands of the connected vehicle era and further uses and extends existing technologies to ease development, analysis, and activities for IoT and Cloud-based approaches for vehicles and provide a basis for new application fields. This paper presents a comprehensive description of technologies and their specifics towards secure data transmission and device management services.**

I. INTRODUCTION

IoT and Cloud platforms are estimated to count more than 1000 worldwide [2]. Many platforms comprise huge communities and continuously advance existing or add new technologies to cope with increasing demands of the respective domain. The vehicle industry is one of the domains that just recently enters the open source IoT and Cloud market. Eclipse IoT [3] is one of the biggest communities that hosts a huge variety of protocol implementations (REST, MQTT, etc.), cloud back-& front-ends, m2m management, device management, OTA update methodologies, security and authentication approaches among others. Eclipse Kuksa [4] is a recent project to fill the missing gaps in order to introduce respective applications and technologies to vehicles.

The remainder of this paper is organized as follows. Section II describes the Eclipse Kuksa technologies briefly upon their utilization, adaption, or extension as well as their interaction with each other. Section III then presents the IoT stack of Eclipse Kuksa in more detail. Afterwards, Section IV outlines specifics about the RPI3 Rover, its hardware, main components, and applications. Finally, Section V concludes the current status of the implementations and provides insights into results that could be obtained from applications across a use-case with three Rovers. Furthermore, the section presents issues and topics to be addressed in upcoming work.

*This work was been supported by the German BMBF under funding.No. 01—S16047D

[1]All authors are with IDiAL Institute, Dortmund University of Applied Sciences and Arts, Otto-Hahn-Str. 23, 44227 Dortmund, Germany robert.hoettger@fh-dortmund.de

II. ECLIPSE KUKSA

Eclipse Kuksa [4] has its origin in the publicly funded APPSTACLE project [5] that started in early 2017. Various companies, suppliers, and research institutes therefore investigate appropriate ways to introduce adaptable cloud and IoT technologies to the vehicle domain and avoid proprietary and commercial solutions.

Figure 1 presents the initial Kuksa architecture that shows a threefold structure consisting of (a) the in-vehicle platform based on automotive grade linux (AGL) [6], (b) the cloud front- and back-end with several technology utilization, and (c) the developer-IDE that is capable to adapt and extend both (a) and (b) towards the developer's specific needs.

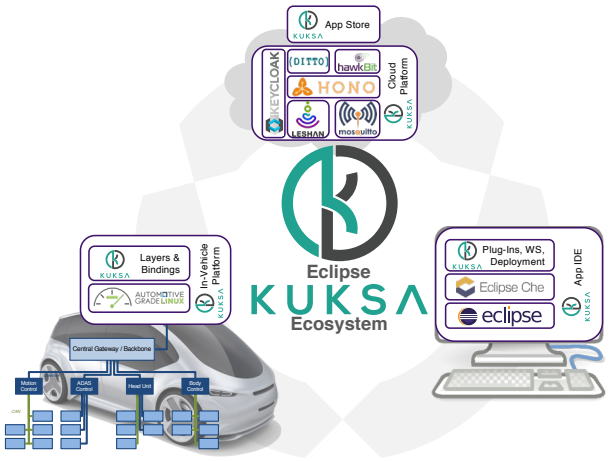

Fig. 1. Kuksa Abstract Architecture

Figure 2 shows the repository structure of Eclipse Kuksa in order to either build AGL adapted Kuksa images, Spring Boot applications applicable to the cloud, or integrate technologies across the connected vehicle domain. The shown repositories are yet under development and may change in the course of upcoming Eclipse Kuksa releases. Figure 2 groups repositories to (i) in-vehicle technologies; (ii) cloud technologies; (iii) in-vehicle application development; and (iv) cloud application development respectively. Having this structure provides a modular design, adaptable continuous integration possibilities, comprehensive bug and issue tracking, sophisticated library usage, and a holistic approach towards the dynamic nature of application evolution of connectivity approaches and vehicle development activities. Developers can therewith address either the used technologies themselves

in isolated repositories or design and implement applications for an arbitrary technology stack within the single Che-based Kuksa IDE. Nevertheless, Eclipse Kuksa has to continuously check and test versions and updates on the repositories in order to guarantee stable development and build processes.

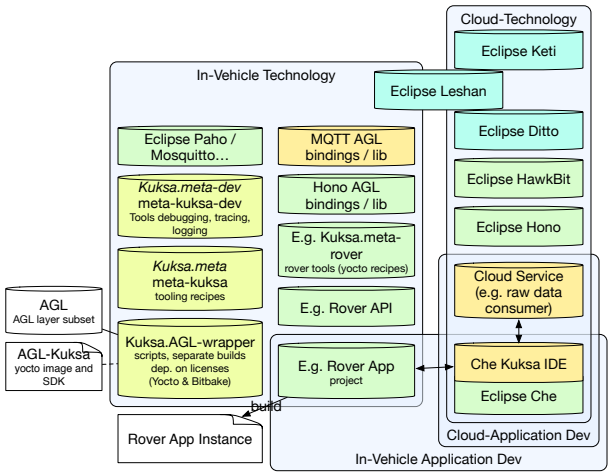

Fig. 2. Development Repositories and Technologies

Figure 3 outlines the Grafana dashboard as a typical cloud front-end visualization tool to access data stored along with the cloud back-end storage database and visualize data via a variety of charts and diagrams configurable by the user. While Grafana is nothing automotive specific, it provides a brief variety of visualization tools for metrics in form of time, series, histogram, bar charts, line charts, stacks, and further customization possibilities.

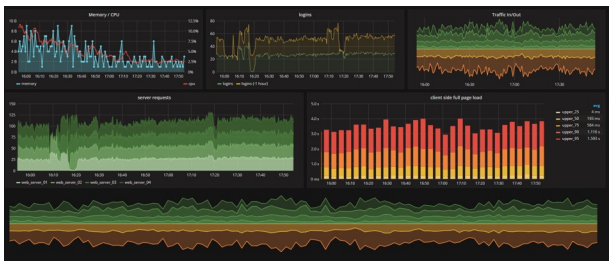

Fig. 3. Grafana Cloud Visualization Front-End

Finally, Figure 4 gives an overview of the layers necessary for the Kuksa in-vehicle platform. The four layers *Boot Loader*, *Operating System*, *Middleware*, and *Secure App Runtime* are accompanied by *Quality of Service Monitoring* and *Authentication and Encryption* methodologies. Since the operating systems has been defined by AGL, several drivers and interfaces are required to implement connectivity standards like 5G, LTE as well as in-vehicle inferfaces such as CAN, LIN, automotive ethernet among others. The middleware layer further consists of in-vehicle and ex-vehicle protocol implementations such as MQTT, AMQP, LWM2M, or eSOC, SOME/IP and others. In addition, network IDS, communication services and the APPSTACLE API are mandatory functionalities to guarantee automotive-adequate technologies to address respective requirements. Furthermore, the Secure APP Runtime contains several applications for OTA management or IoT connectivity but also the applications themselves as well as a corresponding intrusion detection application.

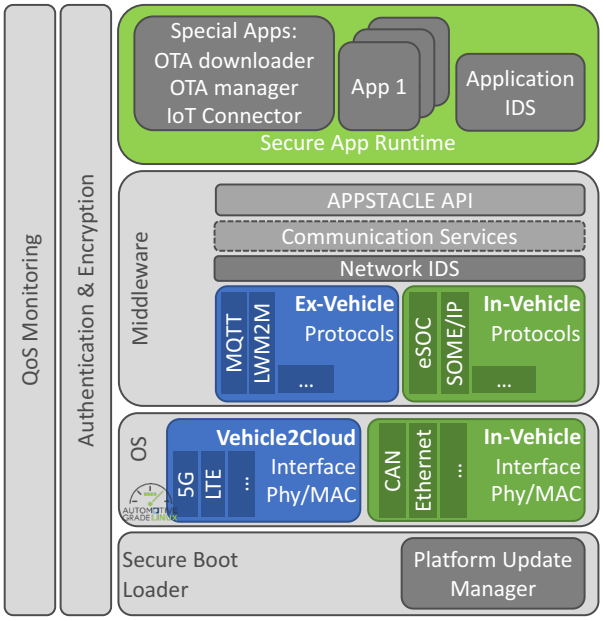

Fig. 4. AGL-based In-Vehicle Architecture

III. Relevant IoT Stack

The following subsections outline the Eclipse Kuksa relevant open source technologies and the cloud platform architecture.

A. Eclipse Ditto

Within the IoT, classical embedded software development and web development come together. However, both disciplines have a very different culture: While embedded software development focus often on reliability and safety, developing web application development requires speed and feature richness. Thus, integrating both worlds introduces several problems. In order to cope with them, the digital twin metaphor has been proposed. A digital twin allows to have a digital representation of all capabilities and aspects that each physical device connected to the internet consists of. This offers the possibility to access and alter the state of a device in a controlled manner. Eclipse Ditto [7] provides a platform to realize the digital twin metaphor. More precisely, it provides functionality to address the following aspects:

- Device as a Service: Higher level API to access devices
- State management for digital twins including notification of state changes
- Digital Twin Management: Provides meta-data based support to search and select digital twins

The API of Eclipse Ditto is realized as REST-API, which allows to have a backend-less realization of IoT applications. In this way, software developers for IoT application can

concentrate on the business logic and user experience without the hassle to integrate different protocols and device types.

B. Eclipse HawkBit

Eclipse hawkBit [8] is a backend framework released under the EPL to roll out software updates to constrained edge devices. With the challenge of a safe and reliable remote software update process in mind, the project aims to provide an uniform update process. This allows to avoid duplicate work for implementing mechanisms separately in each software component. Therefore, hawkBit provides a backend server that can be deployed in any cloud infrastructure. It helps managing roll out campaigns, e.g. by defining deployment groups, cascading deployments, emergency stop of rollouts, and progress monitoring. Further, it offers several device management interfaces on which management messages and updates can be exchanged. However, hawkBit does not provide a client for edge devices by default. To connect certain devices, an adapter implementation that understands the protocols is needed. Those protocols in the Device Management Federation (DMF) API are AMQP, ODA-DM, and LWM2M. Also software can be delivered to edge devices through a REST API. At the cloud side, hawkBit ships a web-based UI for management purposes. Within the UI, all management functionalities are ready to use with a few clicks. In regard of the rising IoT cloud service infrastructure also interfaces for integrating hawkBit into other applications are accessible. Currently, a REST API exposes the functionality of the backend server towards other applications. For a more convenient use, hawkBit also helps managing roll out campaigns, e. g. by defining deployment groups, cascading deployments, emergency stop of rollouts, and progress monitoring. One of the features on the roadmap is the integration of Eclipse Hono (cf. Section III-D) as DMF provider.

C. Eclipse Leshan

Eclipse Leshan [9] is an implementation of the LWM2M protocol in Java. It provides implementations for the client as well as for the server side. Leshan uses Eclipse Californium to communicate via CoAP and relies on Eclipse Scandium for establishing a datagram transport layer security. Furthermore, there are libraries available that help people to implement their own LWM2M server and client side.

D. Eclipse Hono

The Eclipse Hono project provides a platform for the scalable messaging in the IoT by introducing a middleware layer between back-end services and devices. Thereby, the communication to back-end services takes place via the AMQP protocol. If devices can speak this protocol directly, they can transparently connect to the middleware. Otherwise, Hono provides so called protocol adapters, which translate messages from the according device protocol to AMQP. In this way, Hono's core services are decoupled from the protocols that certain applications are using. Via AMQP 1.0 endpoints, Hono provides APIs that represent four common communication scenarios of devices in the IoT:

- Registration
- Telemetry
- Event
- Command & Control

Eclipse Hono consists of different building blocks. The first are the protocol adapters, which are required to connect devices that do not speak AMQP natively. Currently, Hono comes with two protocol adapters: One for MQTT and the other for HTTP-based REST messages. Custom protocol adapters can be provided by using Hono's API. Dispatch router handles the proper routing of AMQP messages within Hono between producing and consuming endpoints. The dispatch router in Hono is based on the Apache Qpid project and designed with scalability in mind so that it can handle potentially connections from millions of devices. As such it do not takes ownership of messages, but rather passes AMQP packets between different endpoints. This allows a horizontal scaling to achieve reliability and responsiveness. Event and commands messages, which need a delivery guarantee, can be routed through a broker queue. The broker dispatches messages that need some delivery guarantees. Typically, such messages are mainly from the command & control API. The broker is based on the Apache ActiveMQ Artemis project. While devices are connecting to the Hono server component, back-end services connecting via subscribing to specific topics at the Qpid server [10].

Among the routing of messages, Hono consists of a device registry for the registration and activation of devices and the provision of credentials as well as an Auth Server to handle authentication and authorization of devices. By using an InfluxDB and a Grafana dashboard, Hono comes also along with some monitoring infrastructure. Due to its modular design, also other AMQP 1.0-compatible message broker than the Apache ActiveMQ Artemis can be used.

E. Eclipse Keti

Eclipse Keti is an application that allows to authorize the access of users to resources by applying the attribute based access control principle. In general, it is used to secure the communication of RESTful APIs. Whether a user can access a resource is decided based on evaluating rules with attribute values as input. An attribute either refers to a user, resource, or environment variable. Multiple rules can be combined into a policy that comprises (i) a target that matches a given request to corresponding policy; (ii) a condition that contains the specific authorization logic; and (iii) an effect that specifies the impact an authorization decision.

Eclipse Keti is realied as a Spring Boot application and consists of three major components:

- Policy evaluation: Conducts the evaluation of every authorization request.
- Policy management: Allows to update and maintain the set of policies.
- Attribute store: Maintains the user and resource permissions captured using attributes.

In addition, Eclipse Keti relies on spring-security-oauth to secure its own endpoints.

F. Keycloak

Keycloak is an application that allows to manage the identity of users (authentication) and their access to resources (authorization). When building an application, Keycloak allows to delegate the user authentication process to substantially reduces the related implementation overhead for the application. It provides an user with single sign on to access all applications registered at a particular Keycloak instance. In addition to serving as an identity provider, Keycloak is also capable of incorporating existing providers, e. g. social networks. Keycloack supports authentication processes that rely on the protocols OpenID Connect and version 2.0 of the security assertion markup language. Authorization is conducted using version 2.0 of the Open Authorization (OAuth) protocol. By using Keberos, Keycloak allows to link user information from other identity management systems, such as LDAP or Active Directory servers. Authentication and authorization is organized within Keycloack via so-called realms. A realm comprises a set of registered users as well as applications. The authentication within a realm is either realized by supplying a separate identity service or enabling the access to a set of existing identity providers. Users that are part of the realm may be federated with other data by attaching additional identity management systems. The access to resources is steered by defining roles and assigning them to users. An admin console provides means to modify the realm settings, while users can manage their accounts by using a separate interface. There are a set of Keycloak client adapters that enable the communication employing the protocols mentioned above. Depending on the protocol, also implementations in various programming languages exists, such as Java and JavaScript. Some of them provide tight integration with specific platforms, for example Spring Boot and WildFly.

G. AGL

AGL (Automotive Grade Linux) is a Linux Foundation Workgroup dedicated to creating open-source software solutions for automotive applications. The initial target for AGL is to provide an IVI (In-Vehicle-Infotainment) system [15]. For providing this kind of systems, AGL uses Yocto Project's building system. Thereby, being able to produce a Linux System image and a Cross-Developement toolchain [16].

However, in order to integrate Kuksa-specific tools, software, and functionalities, AGL's building system was extended by including three layers on top of AGL:

- meta-kuksa: contains Kuksa's code base and all its dependencies that are not included in AGL, e.g. MQTT libraries.
- meta-kuksa-dev: contains extra packages that are useful for the development- and testing-only, e.g. Valgrind, gtest.
- meta-rover: contains all packages needed to enable the RPI3 Rover's runtime software. Basically, it provides recipes for rover-app [12], rover-web [13], and their dependencies in [14].

Furthermore, the Kuksa.AGL-wrapper [17] is an EPL licensed repository that holds the scripts to integrate Kuksa-specific layers with the building system already provided by AGL.

H. Eclipse Che

Eclipse Che is an open source EPL licensed multi-user cloud Integrated Development Environment (IDE) and workspace server. Typically, it can either be utilized by its Browser IDE (Frontend, cf. Fig. 5), or by directly connecting to the resp. workspaces that are realized as customized docker containers, which bring their complete runtime environment, e.g. an Ubuntu based installation with Java, Maven, and/or a C/C++ tool chain. In contrast to typical IDEs, the concept of having workspaces with runtime stacks allows to skip the setup time for end-developers by sharing proper configurations, e.g. with example projects and tutorials. In Kuksa, Eclipse Che has proven to be a valuable asset for developing a variety of applications and projects, such as AGL (services, drivers, or applications such as ACC) for different hardware platforms, or Spring Boot.

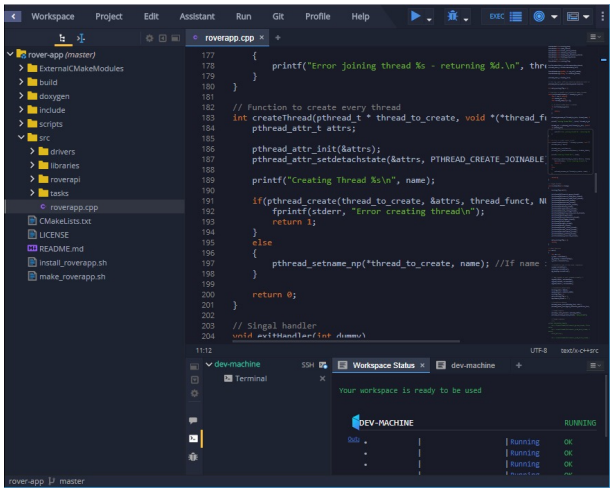

Fig. 5. Eclipse Che development environment

IV. RPI3 Rover

To investigate, demonstrate, and evaluate the aforementioned open-source technologies along with the standards and cloud infrastructure proposed with the Eclipse Kuksa project, an open-source test platform is developed in form of a Raspberry Pi 3 based Rover that is shown in Figure 6 and described in the following.

The RPI3-Rover [14] is a mobile robot that has several features addressing various tasks such as actuation, sensing, communication, and visualization. The Rover's Raspberry Pi 3 features AGL as the operating system, while sensing is accomplished by getting input from several sensors, such as infrared proximity sensors, ultrasonic proximity sensors, a compass, accelerometer, temperature sensor, and a camera.

Fig. 6. RPI3-Rover

The Rover also consists of two brushless DC motors for actuation and an OLED display for visualization. With the developed software, i.e. the rover-app, the Rover is able to achieve several tasks such as following another robot in adaptive cruise control mode, being controlled via a cloud platform, a wifi-based client, or a bluetooth-based client, camera streaming, image and sensor processing, sensor information visualization etc. in a multi-threaded manner. Additionally, the Rover's application programming interface (Rover API) [11] provides developers with a comprehensive library for user application development such as the already developed aforementioned applications.

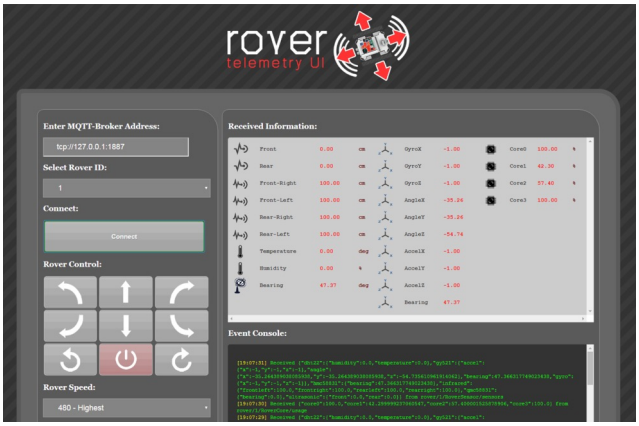

Fig. 7. Rover Telemetry User Interface

The Rover has been developed with a complete open-source work-flow in parallel with Eclipse Kuksa developments. For its release, the AGL software development kit (SDK) has been adapted using embedded Linux tools Yocto build system and BitBake build engine. By using these tools, necessary dependencies for the Rover are built and installed for AGL from custom recipes. Thus, the Rover's compatibility with AGL and other embedded Linux distributions is significantly achieved. Furthermore, Eclipse Che allows to configure Rover's AGL SDK for easy open-source development activities.

Rover is developed especially for developers who want to address cloud-based communication in cyber-physical scenerios. Figure 8 shows the test environment setup prepared for Rover using Rover's API and relevant open-source cloud-related technologies. Rover API utilizes Eclipse Paho MQTT client implementations in an object oriented manner for MQTT-based cloud communication. Using this implementation Rover is able to connect to topics on a cloud instance that is running Eclipse Hono. Similarly, Rover's telemetry interface (shown in Figure 7), a visual MQTT client implementation that is based on node.js and socket.io, also makes use of Eclipse Paho MQTT client implementations in order to connect, publish, and subscribe to the Eclipse Hono instance. This communication allows multiple Rovers and telemetry interfaces talk with each other, drive rovers, and visualize sensor and core utilization information.

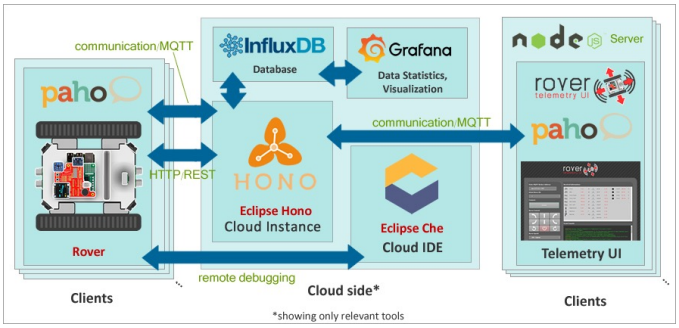

Fig. 8. Technologies and communication infrastructure for the Rover

In the cloud server side, one of many available applications is data statistics visualization. Eclipse Hono takes care of this by storing information that is sent to the "telemetry" topic in an InfluxDB database. Although Grafana's focus is to visualize cloud data statistics rather than data itself, a Grafana dashboard is created to filter the raw data itself for visualization. Thus, Grafana is able to grab that information from the InfluxDB and display the results in JSON-configurable dashboard. The dashboard configured for the Rover is given as an example in the Figure 9, which involves the sensor data visualization over time.

Fig. 9. Grafana dashboard showing Rover sensor data

V. CONCLUSIONS

The Eclipse Kuksa initial public contribution is expected during Q1 2018. Local tests and use case implementations along with the Rover have shown that...

- ...the Rover image is extensively smaller (250MB for Kuksa-AGL compared with 4GB using the Raspbian OS)
- ...new technologies can be integrated easily without adjusting the meta-kuksa layer too much
- ...soft real-time applications can be deployed
- ...versioning, issue tracking, and configuration can be achieved on a modular and fine grained level
- ...browser-based development activities significantly ease teamwork and leverage the use of required configurations
- ...the utilization of Eclipse open source IoT technologies is highly beneficial and allows concentrating on specifics and innovative constructs

In the later course of the project, a holistic API will be added to the *kuksa-dev* layer in order to provide easy access to the various existing and upcoming technologies. The authors are looking forward to leverage synergies with established communities as well as technologies in order to bring automotive based application development activities to the next level. Therefore, cloud concepts and Spring Boot applications can perform complex analyses and simulations as well as upgrade and update applications for vehicles and therewith make future driving more beneficial for end-users, developers, tool suppliers, or even OEMs via a common easy to use architecture.

ACKNOWLEDGMENT

The authors would like to express their appreciation to the APPSTACLE consortium for sharing experience, expertise, and knowledge. Special thanks to Johannes Kristan and Tobias Rawald from Bosch Software Innovations for their industrial insights into the various Eclipse IoT projects used in Eclipse Kuksa.

REFERENCES

[10] Eclipse Hono: https://www.eclipse.org/hono/, access February 2018

[1] The advantages of open source tools, *Kayla Matthews*, jaxenter 21.11.2017, https://jaxenter.com/advantages-open-source-tools-139026.html, access February 2018

[2] Create Your Own Internet of Things: A survey of IoT platforms, *Kiran Jot Singh and Divneet Singh Kapoor*, IEEE Consumer Electronics Magazine (Volume: 6, Issue: 2, April 2017)

[3] Open Source Software for Industry 4.0, *An Eclipse IoT Working Group collaboration*, October 2017, The Eclipse Foundation, Inc., Made available under the Eclipse Public License 2.0 (EPL-2.0), Online: https://iot.eclipse.org/resources/white-papers/Eclipse%20IoT%20White%20Paper%20-%20Open%20Source%20Software%20for%20Industry%204.0.pdf, access February 2018

[4] Eclipse Kuksa: https://projects.eclipse.org/proposals/eclipse-kuksa, access February 2018

[5] ITEA3 APPSTACLE Project: https://itea3.org/project/appstacle.html, access February 2018

[6] Automotive Grade Linux: https://www.automotivelinux.org, access February 2018

[7] Eclipse Ditto Project Proposal: https://projects.eclipse.org/proposals/eclipse-ditto

[8] Eclipse hawkBit: https://projects.eclipse.org/proposals/hawkbit, access February 2018

[9] Eclipse Leshan: https://www.eclipse.org/leshan/, access February 2018

[11] Rover API: https://app4mc-rover.github.io/rover-app, access February 2018

[12] Rover APP: https://app4mc-rover.github.io/rover-app, access February 2018

[13] Rover Web: https://app4mc-rover.github.io/rover-web, access February 2018

[14] Rover Documentation: https://app4mc-rover.github.io/rover-docs, access February 2018

[15] Automotive Grade Linux Requirements Specifications: http://docs.automotivelinux.org/docs/architecture/en/dev/reference/AGL_Specifications/agl_spec_v1.0_final.pdf, access February 2018

[16] Yocto Project Software Development Kit (SDK) Developer's Guide: http://www.yoctoproject.org/docs/2.1/sdk-manual/sdk-manual.html

[17] AGL-kuksa: https://gitlab.idial.institute/appstacle/agl-kuksa