On deployment of Eclipse Kuksa as a framework for an intelligent moving test platform for research of autonomous vehicles

Harri Hirvonsalo
Faculty of Information Technology and Electrical Engineering / Empirical Software Engineering in Software, Systems and Services (M3S Group) / University of Oulu
Oulu, Finland
harri.hirvonsalo@oulu.fi
https://orcid.org/0000-0002-5503-510X

Pertti Seppänen
Faculty of Information Technology and Electrical Engineering / Empirical Software Engineering in Software, Systems and Services (M3S Group) / University of Oulu
Oulu, Finland
pertti.seppanen@oulu.fi
https://orcid.org/0000-0002-4289-2487

***Abstract*—After an era of huge propagation within the field of mobile communications, digitalization has been spreading during the latest years at an accelerating speed to automotive technology and business. Following the developments experienced earlier in the mobile communications branch, open solutions and platforms are emerging in the automotive industries, challenging the traditional proprietary systems. Like in mobile communications, open platforms enable development of a wide variety of novel applications and systems that speed up the digitalization of the automotive and traffic ecosystems and offer the developers new attractive business opportunities. One of such open approaches is the Eclipse Kuksa framework developed in a consortium of European research institutions and automotive industries.**

In this study, we explored how the Kuksa framework could be used as a technology basis for an automotive data system combining in-vehicle functionality to cloud-based services. The study was carried out as a part of the SMAD research project of the University of Oulu aiming at an intelligent moving test platform for research of autonomous vehicles built on top of a Toyota Rav4 hybrid car. Our study covered subsystems of the Kuksa framework, the in-vehicle subsystem, cloud subsystems and the data communications between them.

Our results indicate that the Kuksa framework is a feasible basis for the development of open, intelligent automotive data systems, though with a considerable learning needs. The results and experiences of our study provide besides additional knowledge for our continuing research on automotive software, also new, valuable contribution to the Eclipse Kuksa community and the practitioners planning to deploy open approaches in their automotive software development.

Keywords—automotive software, open systems, cloud services, internet-of-vehicles, Eclipse Kuksa

I. Introduction

Following the developments of smart mobile device business and technology, open solutions have gained increased interests in the automotive industry [43]. Started in 2016, the University of Oulu conducted research of the open solutions for automotive industries in a three-year research project ITEA 3 APPSTACLE (open standard APplication Platform for carS and TrAnsportation vehiCLEs) [1], which further continued as Eclipse Foundation development project, Eclipse Kuksa [2]. A key contribution of these projects was an application development and testing framework called Eclipse Kuksa [3]–[7], according to the traditional wooden drinking cup of northern Finland's hunters and fishermen.

The Kuksa framework introduced an in-vehicle software platform, an internet-of-things cloud platform, a cloud-based IDE and an application store that was connected to an automotive with a specific hardware interface, providing software developers with an integrated environment for developing, testing, and offering for use in automotive applications.

In 2019, University of Oulu started a new research project, SMAD (Smart and mobile testbed for automated and assisted driving), funded by the European Regional Development Fund, aiming at creating an intelligent moving test platform for automotive research [8]. During this two-and-a-half-year project, eight research units of University of Oulu, built a broad palette of research and test infrastructures for supporting autonomous driving research, utilizing two Toyota Rav4 Hybrid cars as moving carriages of the test platform [9].

Open solutions were seen as an important option also in the research of autonomous driving, and Kuksa was opted as the framework for software and application development of the SMAD project. Being outside the scope of the SMAD project, the application store of Kuksa was left fully out of the focus of this research. Kuksa IDE was covered only to the extend necessary to address the targets set in the SMAD project plan.

In this paper, we report the work conducted on Kuksa framework in SMAD project; the results of implementing a moving test platform utilizing Kuksa, gained experiences regarding to Kuksa, and contributions to Eclipse Kuksa community. In this research, we define Kuksa framework as a combination of a vehicle-level software in connection to Kuksa development and testing hardware and the data communications and cloud services software defined and built in the APPSTACLE and Eclipse Kuksa projects. From the perspective of Kuksa framework we define the intelligent moving test platform, the SMAD environment, as a Toyota Rav4 Hybrid car connected to Kuksa cloud platform utilizing mechanisms provided by Kuksa framework.

The research was conducted by deploying the methods of the Design Science Research (DSR) as defined by Hevner et al. [10], Hevner [11] and Hevner & Chatterjee [12].

In Section II, an overview on Kuksa framework is summarized. In Section III, the research problem is defined. In Section IV, the research methodology is presented. In Section V, building of the SMAD-specific Kuksa is presented. In Section VI, the results and experiences are presented. In Section VII, the conclusions are drawn.

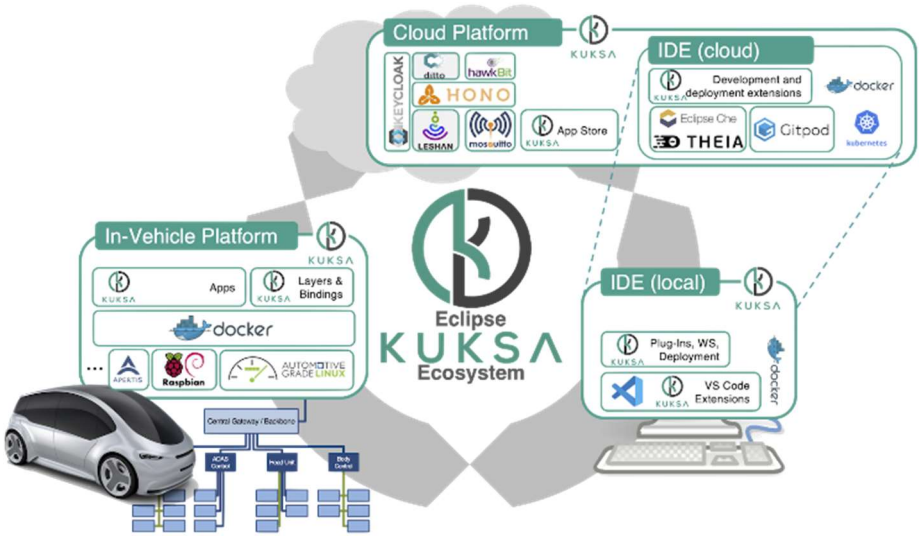

Fig. 1. Kuksa architecture [2]

II. Summary of Kuksa framework

In this section a short summary of Eclipse Kuksa framework subsystems, referenced in Fig. 1 as platforms, the in-vehicle subsystem [4] and cloud subsystem [5], is presented to the extent that was relevant for the SMAD project targets [9]. Application store functionality of cloud subsystem [13] and Kuksa integrated development environment subsystem [6] are excluded from this summary. In addition, a short summary of data communication between in-vehicle subsystem and cloud subsystem is presented.

A. In-vehicle subsystem

Kuksa framework in-vehicle subsystem, a gateway for connecting to in-vehicle devices and data sources, is composed of both software and hardware that integrate into a vehicle. It provides in- and ex-vehicle data access mechanisms, application platform and secure gateway to the cloud [14]–[16]. As depicted in Fig. 2, Kuksa in-vehicle subsystem architecture is three-layered, including an OS layer, a middleware layer, and an application layer.

Application layer provides a sandboxed and secure runtime environment for Kuksa in-vehicle specific applications such as over-the-air update functionality. Custom applications deployed in the in-vehicle subsystem are run on this layer. Middleware layer provides libraries and APIs to enable interaction and communication with in-vehicle subsystem hardware, the vehicle itself and cloud subsystem. OS layer with the use of Automotive Grade Linux Unified Code Base - Linux distribution [17] (later referenced in this study only as AGL UCB) of Automotive Grade Linux - project [18] (later referenced in this study as AGL) provides typical operating system services and scheduling, and for example, device drivers needed to interact with the hardware in-vehicle subsystem is deployed to. Kuksa in-vehicle specifications recommend that OS should be booted by hardware supported secure boot mechanism [14].

Utilizing in-vehicle subsystem requires use of AGL UCB supported hardware or that a custom build of AGL UCB is done in order to make it compatible with the hardware in-vehicle subsystem is being deployed on; different CPU architectures need to be taken into account and although drivers for hardware needed for ex- and in-vehicle communication could be distributed and loaded separately, our interpretation is that in-vehicle subsystem documentation suggests that drivers should be distributed together with AGL UCB in manner of custom build [19]. Similarly, in-vehicle subsystem specific software, software requirements of custom software and applications, and the applications themselves can be distributed with AGL UCB, by including them in the target environment specific custom AGL UCB build. By our interpretation, Kuksa in-vehicle subsystem documentation implicitly recommends packaging all but custom applications into a custom AGL UCB build [19] [20]. Packaging the operating system, drivers for hardware, in-vehicle subsystem software, and software requirements of applications into a single software image, enables to update whole software stack of in-vehicle subsystem over-the-air [20].

AGL utilizes OpenEmbedded build framework [21] and Yocto project [22] resources and best-practices to enable modularity and customizability of AGL UCB builds [23]. Kuksa in-vehicle subsystem utilizes the same build tool of OpenEmbedded, BitBake [24], to include in-vehicle subsystem specific software in-vehicle specific custom build of AGL UCB [19].

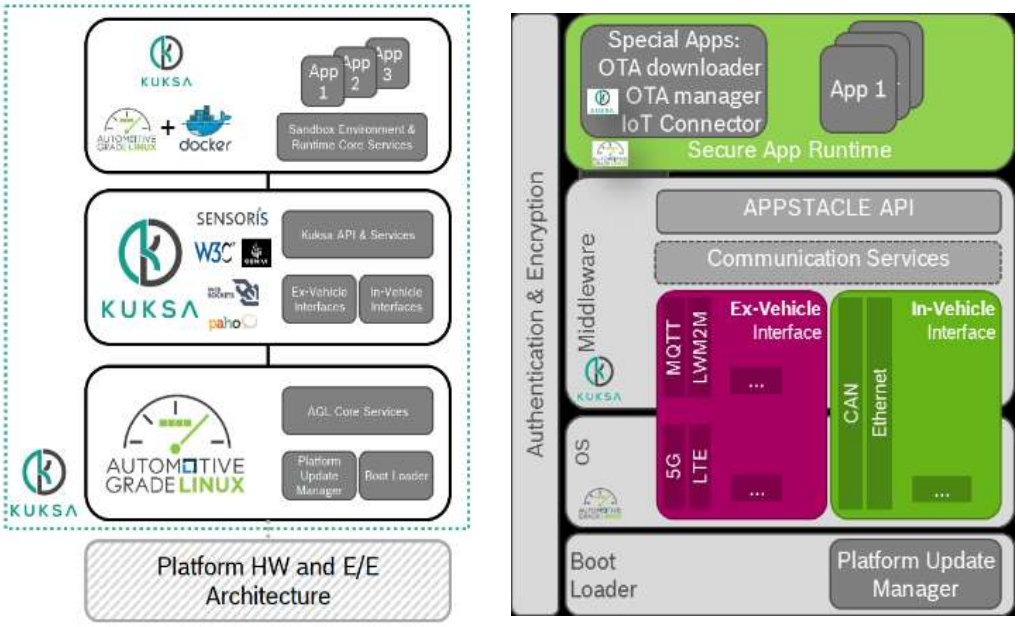

Fig. 2. Kuksa in-vehicle subsystem layer architecture [16], [71]

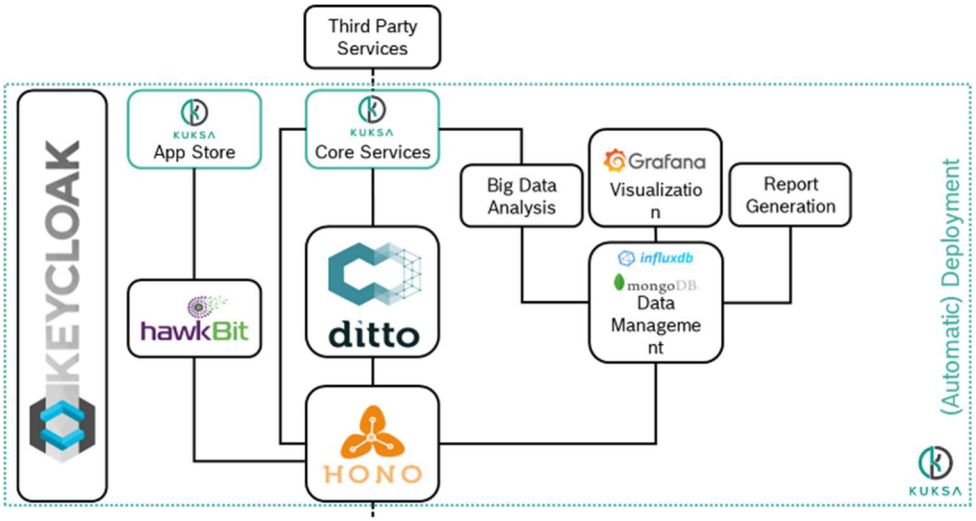

Fig. 3. Architecture of Kuksa cloud subsystem [15]

Regarding hardware, Kuksa in-vehicle subsystem specification in [14] lists two hardware platforms for in-vehicle subsystem, but technically in-vehicle subsystem software does not require that a specific type of hardware, e.g. from a specific vendor or with a specific CPU architecture is used to provide hardware part of in-vehicle subsystem. However, software specifications of in-vehicle subsystem implicitly set some requirements to hardware, such as the capability to run AGL UCB and recommendation for hardware supported secure boot.

Kuksa offers free and open schematics for a development and testing hardware board build around capabilities offered by Raspberry Pi Computing Module [7]. Device includes a wireless network card interface and SIM card slot for 4G or 5G connectivity and a CAN bus compatible interface chip, STN2120 [25], to enable communications with control units of a car through car's on-board diagnostics (OBD) port. Device can be extended with peripherals through USB connectors. Rendered image of the device is presented in Fig. 4.

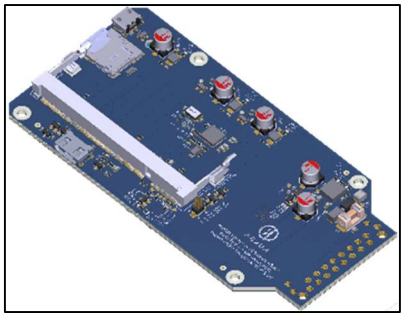

Fig. 4. Rendered image of Kuksa hardware dongle

B. *Cloud subsystem*

Kuksa cloud subsystem is comprehensively summarized by Banijamali et al in following way:

> The Eclipse Kuksa cloud platform (EKCP) sends and receives different types of messages from and to various sources, such as vehicles, devices, and third-party services. In general, messages include "telemetry messages" that depict data stemming from vehicles, devices, and sensors and "commands and controls messages" that are dedicated to the vehicles and device management components [26, p. 460].

Kuksa is not limited to cloud centered communication model, as its in-vehicle subsystem can be extended to support, for example vehicle-to-infrastructure (V2I) and vehicle-to-vehicle (V2V) communication [14] [27]. However, Kuksa cloud subsystem should be viewed as cloud centric IoT architecture [28] and given its automotive context it can be seen as an internet-of-vehicles architecture (IoV). In general, IoT architecture such as Kuksa cloud subsystem consists of the following building blocks according to the different reference architectures studied in APPSTACLE [29]:

- Message gateway: A component for sending and receiving data to and from an arbitrary amount of (constrained) devices via different kind of protocols.

As this component is the central point of interaction with the cloud backend, the message gateway transforms data after ingress to events and act as broker by redirecting the events to other components for further processing. Eclipse Hono service [30] presented in Fig. 3 realizes responsibilities of this component in Kuksa.

- Data storage and management: A component for persisting data within different types of databases. In Kuksa, InfluxDB [31] and Mongodb [32] presented in Fig. 3 are used to provide functionality of this component.
- Data analytic and visualization: Components for analyzing existing data including big data analyses and visualizing data in a suitable and valuable way. In Kuksa, Grafana service [33] presented in Fig. 3 is used for visualization. Apache Flink service [34] is used as the big data analysis component presented in Fig 3.
- Device management: A component for device management allows to authenticate, configure and control, monitor, maintain, and update devices. Eclipse hawkBit service [35] presented in Fig. 3, together with Eclipse Hono provide functionality of this component in Kuksa.
- Application and service integration: Components that support the development and provision of applications and services within the cloud backend. In general, all components of Kuksa cloud subsystem provide some functionality that enables Kuksa to provide responsibilities of this component; for example, Eclipse Ditto service [36] presented in Fig. 3 enables use of digital twins and Eclipse Hono supports wide variety of communication protocols.
- Security: Components that realize authentication, authorization, privacy, and a secured communication. In Kuksa, Keycloak service [37] presented in Fig 3 and Eclipse Hono provide this functionality.

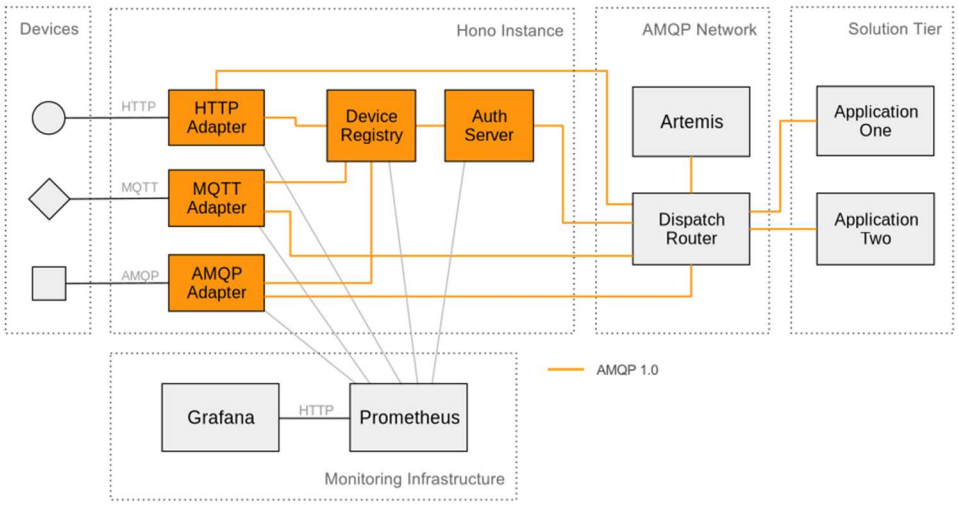

Fig. 5. Component view of Eclipse Hono architecture [30]

Although Kuksa cloud subsystem does not strictly require specific runtime system, available Kuksa cloud subsystem deployment documentation [38] and [39] imply that Kuksa cloud subsystem is meant to be deployed in a cloud-native fashion to a Kubernetes [40] cluster using Helm [41] deployment tool or alternatively to OpenShift [42]. Deployment examples included in Kuksa cloud subsystem in [38] instruct how subsystem can be deployed to Azure Kubernetes Service (AKS) [44] using various automation scripts utilizing Azure CLI tool [45]. Additional Kuksa cloud subsystem related deployment documentation is available through documentation of some of its components such as Eclipse Hono in [46]. Hono documentation explicitly mentions that deployment to Kuberenetes runtime is the only deployment model Hono officially supports. Similarly, as Kuksa cloud subsystem, Hono deployment examples in [46] instruct how Hono can be deployed to AKS, utilizing Azure CLI and, in addition to Kuksa documentation, Azure Resource Manager (ARM) templates.

C. *Data communications between Kuksa in-vehicle and Kuksa cloud subsystems*

Data communication between Kuksa in-vehicle and Kuksa cloud subsystems happens via gateway components, which essentially provide remote service interfaces for connecting vehicles and devices to cloud subsystem (cloud subsystem gateway) and enable data transfer from vehicles systems, i.e. CAN bus (in-vehicle subsystem gateway) [39]. Eclipse Hono, which is the cloud subsystem gateway component, uses AMQP 1.0 protocol to broker messages between in-vehicle and other Kuksa cloud subsystem components and external applications. This communication is considered to be egress or northbound traffic from point-of-view of Hono.

For communication between in-vehicle subsystem and Hono, i.e. ingress or southbound connections, MQTT, HTTP, CoAP, AMQP 1.0 and LoRa protocols are supported. Hono provides support for southbound protocols via protocol adapters and northbound connections via AMQP 1.0 to Dispatch router component as depicted Fig. 5.

Southbound connection requests, i.e. devices connecting to Hono protocol adapters are authenticated by Hono device registry, or rather its credentials service. Authentication of northbound communication requests are delegated to Authentication service of Hono. Hono's internal components use the same Authentication service to authenticate and authorize each other. Authentication mechanisms for authentication of devices supported by Hono are hashed passwords, Pre-Shared Keys (PSK) for Transport Layer Security (TLS) connections and X.509 certificates [47]. Authentication mechanism depends on the used communication protocol; not all protocols support all authentication mechanisms. Kuksa utilizes Keycloak for user authentication of App Store functionality [48].

III. Problem Definition

While defining the targets of the SMAD project [8], the M3S, as one of the eight research teams, figured out several intelligent services utilizing Eclipse Kuksa framework [3]–[7], such as distant monitoring of the vehicle and on-line functionality and software update to the vehicle.

During the project it became evident, that Kuksa framework, as developed in the APPSTACLE [1] and Eclipse Kuksa [2] projects, was not an off-the-shelf product, but a research framework developed and integrated by several project consortium members to address various member-specific targets. Wide use of open-source software packages had enabled the APPSTACLE consortium members to set up and finetune slightly different combinations of software components and services and to our best knowledge, none of them would directly (i.e. without customization of the framework and its components) fit to the target setting of the SMAD project, including the test Toyotas and their technical details, the data communications solutions, and the intelligent services planned in the SMAD project plan [9].

With this reasoning we understood that our research will focus on both implementation of the test platform and finding out how one can utilize Kuksa framework; how Kuksa framework needs to be customized, in order to utilize it in realizing the intelligent moving test platform of SMAD project, the SMAD environment. Since main outcome of SMAD project is the SMAD environment we decided to focus our research questions to build process of this test platform.

RQ1: How to build an intelligent moving test platform utilizing the Kuksa framework?

RQ2: How to customize the Kuksa framework for a case-specific, intelligent automotive data system?

RQ3: What are the obstacles of utilizing the Kuksa framework for a case-specific, intelligent automotive data system?

Description of SMAD project targets and details how Kuksa framework was customized to implement the SMAD environment, will be presented in chapter 5.

IV. Research methods

Because of the target setting of the SMAD project aiming at building a test platform, we decided to utilize the Design Science Research (DSR) approach as defined by Hevner et al. [10], Hevner [11] and Hevner & Chatterjee [12].

Hevner, [11] defines a three-cycle model of Design Science model, as presented in Fig. 6.

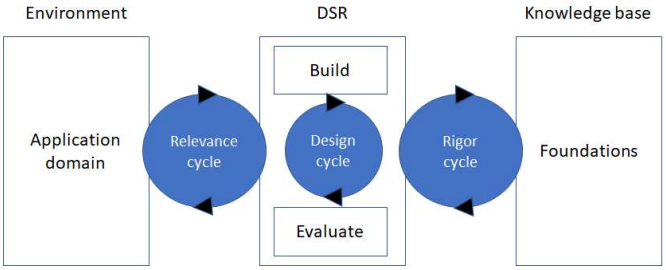

Fig. 6. Hevner's three-cycle model of Design Science Research [11].

In the relevance cycle, the entities of the application domain, people, organizational systems, technical systems and, and the related problems and opportunities are utilized as the basis for the requirement identification and field testing of the artifacts and processes built and evaluated in the DSR process.

In the rigor cycle, foundations of the DSR process, scientific theories & methods, experiences & expertise, and meta-artifacts are used for grounding the results of the DSR process, which, in turn provide the knowledge base with additional knowledge and experiences.

When mapping Hevner's three-cycle model of DSR we were able to identify the following mappings to our problem domain, validating our research methodology selection:

1) The application domain consisted of the M3S research team and its researchers, having a task to find solutions to the problems defined in section 3. The requirements for the artifacts to be built and the validation criteria were naturally derived from the problem domains of the SMAD test platform [9].

2) The knowledge base consisted of the Kuksa framework documentation, of the experiences and expertise gained in the APPSTACLE project in our university, and of the documentation and code of the open-source software and services of the Kuksa framework.

3) The artifact to be built and evaluated by following the DSR process was an integrated Kuksa from a Toyota test car to the cloud services relevant for addressing the target setting of the SMAD project.

V. Building SMAD-specific Kuksa

To support realization of the intelligent moving test platform, i.e. the SMAD environment, we built a Kuksa system addressing the problem domain presented in section 3. While building such SMAD-specific Kuksa we utilized, besides the overall idea of Kuksa framework, the sub-systems and solutions that were designed and built in the APPSTACLE [1] and Eclipse Kuksa [2] projects. In the following sections, we describe how the work was carried out deploying Hevner's three-cycle process model [11].

A. Identifying the requirements, the relevance cycle work

Focus of SMAD project was on two larger themes; research of autonomous driving itself and on various aspects needed to enable and support autonomous driving, and implementation of a moving development and testing platform, i.e. the SMAD environment, to support future research on autonomous vehicles. Research topics range from connectivity and communication research, such as 5G network connectivity and vehicle-to-everything (V2X), to ubiquitous integration of car to smart transportation systems.

Implementation of the SMAD environment was less research oriented, but still very much a research task; to our best knowledge no such system exists, and as such there are lot of unknows to deal with when implementing such system.

Implementation of the SMAD environment would happen simultaneously with the research activities and SMAD environment would be used to support research when possible.

Following requirements were extracted by us from SMAD project description [9]:

- SMAD environment needs to enable future research and development of use cases such as vehicle as a sensor, online functionality and software updates of the vehicle, traffic situation updates and route booking.
- SMAD environment will provide both the hardware and the software to support automotive research. It will provide the vehicles to which needed research instruments will be installed to.
- The vehicles themselves are also research instruments and as such, SMAD environment needs to provide means to gather data from various sensors and systems of the vehicles.
- SMAD environment needs to provide means to interact with the vehicle and support interacting with the research instruments installed in the vehicle when applicable.
- SMAD environment needs to be able to integrate to instruments and systems outside of the vehicle.

In essence, SMAD environment is complex system where communication between various elements, both inside and outside of the system's core, is the central functionality provided by the SMAD environment. To enable this, core of SMAD environment is depicted as a system consisting of cloud environment and a software environment installed to vehicles. Research instruments in the vehicle will connect to the cloud backend via means provided by the in-vehicle software environment. In-vehicle environment will enable installation of software research artifacts to the vehicle and since systems and sensors of the vehicle must also be utilized by the test platform and research instruments, in-vehicle environment must provide means to do so. This changes the nature of in-vehicle environment from software only environment to system which also provides hardware capabilities to interact with the vehicle and research instruments connected to the vehicle. External systems and services will integrate to SMAD environment through the cloud environment, apart from V2X communication, where vehicle and its environment could in some cases interact directly without connection to cloud environment.

SMAD environment is a service that will be utilized by researchers and companies doing automotive related software research and development. From a service point- of-view following requirements were deducted for SMAD environment core based on SMAD project description:

- SMAD environment core and its software and services must be managed according to best practices suitable for chosen deployment model.
- SMAD environment core needs to be operated and maintained, in best case as a production grade service. Design of the system must take this into account.
- Operation metrics of the SMAD environment core need to be gathered and monitored in order to identify service level degradation.
- Software development and maintenance of SMAD environment core components must be trustworthy; utilized development processes must provide, for example, traceability of software and its dependencies, consistent and repeatable compilation results, version control and other best practices.

B. *Utilizing the results of the APPSTACLE and Eclipse Kuksa projects, the rigor cycle*

Eclipse Kuksa framework was identified as a key enabler for building SMAD environment core and for research targets of the SMAD project, but as described in section 3, we identified that Kuksa framework would have to be customized in order to utilize it in context of SMAD project. We started our SMAD work by identifying parts of Kuksa framework architecture that must be modified or parts that could left out to build a customized, SMAD-specific Kuksa system to realize SMAD environment core, better suited to SMAD project targets. We identified three areas for change along the system-level architecture of Kuksa framework: the in-vehicle subsystem, the cloud subsystem, and the data communications between the subsystems. Each subsection describes how corresponding subsystem of Kuksa framework was to be adapted to form a SMAD-specific Kuksa. Table I presents a summary of design decisions which reflect how SMAD-specific Kuksa will differ from Eclipse Kuksa.

1) In-vehicle subsystem

AGL UCB [17], OS choice of Kuksa in-vehicle subsystem, includes lot of functionality such as application framework [49] and human-machine interface framework (HMI) [50] which were not needed in SMAD project; no SMAD project task required implementation of graphical user interface in the in-vehicle subsystem and software that will be run in in-vehicle subsystem would not benefit from the use of application framework, since security features provided by application runtime of in-vehicle system can be utilized without utilizing application framework itself. Apart from HMI and application framework most of the generic operating system functionality provided by AGL UCB and its underlying use of OpenEmbedded [21] resources were seen as a necessity for SMAD-specific Kuksa. AGL UCB is stable and maintained operating system, which through use of BitBake [24] build tool offers flexibility to make use-case specific changes to the OS. With this reasoning, it was decided that application framework and HMI framework of AGL UCB would not be used in SMAD-specific Kuksa. However, it was also decided that, if time permits, application framework could be evaluated and used if evaluation revealed that application framework would benefit SMAD project targets. Customization of AGL UCB would happen with BitBake tool.

Most of the functionality provided by middleware layer of Kuksa in-vehicle subsystem is not needed in SMAD project. However, some functionality such as vehicle abstraction layer (VAL) [51] were identified to be useful, as it provided an easy-to-use and ready-made implementation for transforming vehicle manufacturer specific CAN bus messages to GENIVI Alliance Vehicle Signal Specification (VSS) [52] data model. By this rationale only vehicle abstraction layer of Kuksa in-vehicle subsystem was decided to be included in SMAD-specific Kuksa.

Kuksa development and testing hardware [7] was selected as the hardware platform for SMAD-specific in-vehicle subsystem, since we had one available from APPSTACLE project. Other hardware platforms could be utilized in the future. Kuksa development and testing hardware does not contain necessary hardware for secure boot and as such, using a secure boot loader as defined by [14] was not a viable option. To address the need of boot time security, which provides the base for runtime security, we identified that secure boot like functionality could be achieved on Kuksa development and testing hardware via use of special SD memory card such as Swissbit PS-45, which when used with a custom bootloader, allows to store secure boot key material protected by on-demand decryption and write-protection of SD-card partitions [53]. Given that SMAD is time constrained project and we did not have previous experience on utilizing Swissbit PS-45 or similar products, we decided that we will first implement a version of SMAD-specific Kuksa without secure boot capabilities on development and testing hardware of Kuksa and if project schedule allows, we will either try to implement a secure boot mechanism based on Swissbit SD-card or test SMAD-specific Kuksa in-vehicle subsystem on another hardware platform which supports secure boot out-of-the-box.

2) Cloud subsystem

SMAD project tasks did not involve any digital twin related activities, so Eclipse Ditto [36] service was decided to be removed from SMAD-specific Kuksa. If future use-cases of SMAD environment involve digital twin activities, support for Eclipse Ditto service could implemented into SMAD-specific Kuksa.

Similarly, Kuksa Appstore service [13] was also outside of context of SMAD project and it was decided to be left out from SMAD-specific Kuksa. Application management practices for in-vehicle subsystem, similar to those of Kuksa Appstore could be implemented to SMAD-specific Kuksa in the future.

Since development SMAD-specific Kuksa will continue after SMAD project, we had the opportunity to narrow down the expected use of Eclipse hawkBit [35] compared to Kuksa. It's importance and usefulness in over-the-air (OTA) updates of in-vehicle system is without question, but in first version of SMAD-specific Kuksa, it will only be used to perform OTA updates of whole software image of in-vehicle subsystem, i.e. update the AGL UCB distribution combined with all necessary software of the SMAD-specific in-vehicle subsystem. As with Kuksa, hawkBit service would be used to update and install new, user specific software on in-vehicle subsystem, but this feature would be developed in the future, after SMAD project.

With similar justification of future development activities, we decided that during SMAD project, SMAD-specific Kuksa won't provide data gathering or data visualization features. Instead, an example integration to external data gathering system using northbound AMQP connection of cloud subsystem would be done. During SMAD project an external integration is better option, as we do not know if SMAD environment should even offer data gathering as an integrated service. We would find this out during SMAD project. Development of data gathering and visualization features integrated services might happen in the future, outside of SMAD project. For now, data gathering would only involve gathering of operational data of cloud subsystem components. Grafana service would be used for visualization of this data.

At start of SMAD project, we did initial investigation on deployment of Kuksa cloud subsystem. It seemed that deployment of various Kuksa cloud subsystem services was not uniform; deployment instructions, examples, and models of cloud subsystem services differed from each other and automated set up of infrastructure seemed lacking [38]. We decided to implement our own, more uniform infrastructure deployment automation. All infrastructure-as-a-service (IaaS) related automation would utilize Terraform tool [54] and Helm tool [41] deploying to Kubernetes cluster [40] set up at the selected IaaS cloud service. Necessary secrets, storage and authentication and authorization management would be implemented using Terraform and Helm, using other tools only when strictly necessary. Some automation scripts utilizing Helm and Terraform had already been done in context of Kuksa cloud services deployment [38]. Infrastructure set up part of these scripts was not adequate, but service deployment part of the scripts could have been used in SMAD. However, scripts deployed unnecessary components for SMAD-specific Kuksa in interlinked way, which might have required lot of changes to the deployment scripts, to deploy a SMAD-specific Kuksa with them. We decided to use these scripts for reference when needed.

Since Eclipse Hono [30] was identified to be key enabler for SMAD related work, we decided to focus our initial effort on deployment of Hono. Deployment of other services of SMAD-specific Kuksa, would be adopted based on our experienced gained with Hono deployment. From investigation of Hono deployment scripts and documentation [47] we learned that there is quite good monitoring and tracing support available in Hono, but this was not mentioned in Kuksa documentation.

3) Data communication

Data communication options for devices connecting to cloud subsystem provided by Kuksa were more than adequate for SMAD. In SMAD-specific Kuksa we would only utilize mutual transport layer security [55], i.e. mTLS, protected MQTT protocol for communications between vehicle and Kuksa cloud subsystem. From security point-of-view, this meant that support for unnecessary protocols would need to be disabled in Hono. In case we would see a need to expand selection of supported protocols this could be done by enabling disabled protocol adapters or if needed, implement new ones.

Regarding to communication between Kuksa cloud subsystem and external applications, Kuksa supports only AMQP 1.0 protocol. We identified that this limitation might affect usefulness of Kuksa for potential users of SMAD environment. After further investigation of Eclipse Hono architecture it was decided that we won't add support for other protocols for northbound connections during SMAD project. This would be too big task and since Hono is under active development we wanted to see if new development would bring support for new protocols.

V2X research activities were included in SMAD project targets, but it was not clear if this V2X research would benefit from use Kuksa in-vehicle subsystem and to what extent we could even add support for direct vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) into in-vehicle subsystem. To address this, we decided that V2V or V2I would not be included in SMAD-specific Kuksa at all.

TABLE I. SUMMARY OF DESIGN DECISIONS FOR SMAD-SPECIFIC KUKSA

#	Decision	Rationale	Remarks
In-vehicle subsystem			
1	AGL HMI and application framework will be removed.	No use for HMI since no GUI in SMAD. No added security value from application framework.	Evaluate other benefits of application framework if time permits.
2	SMAD-specific AGL UCB customized with BitBake tool.	BitBake build process provides software traceability	-
3	Only Kuksa.val be included from Kuksa	No added value gained from other Kuksa middleware.	-
4	No secure boot in 1st version	Kuksa hardware doesn't support secure boot	If time permits evaluate Swissbit PS-45 secure boot like solution
Cloud subsystem			
5	Eclipse Ditto and Kuksa Appstore will be removed.	No use in SMAD project context.	If digital twin use cases arise in the future, Ditto support will be implemented.
6	Only operational metrics will be gathered. Visualisation of metrics using Grafana	SMAD Kuksa integrates into external data analysis services. Hono monitoring supports Grafana.	-
7	1st version focuses on custom deployment scripts for Hono instead Kuksa	Hono identified to be most important component. Custom deployment needed	Terraform and Helm will be used to implement deployment automation.
8	hawkBit will be used to install only complete SMAD-specific AGL UCB images.	Schedule restrictions prevent implementing OTA support for individual applications.	OTA install of individual application will be implemented in future.
Data communication			
9	Only mTLS MQTT support in 1st version	Mutual TLS increases security.	Other protocols will be supported in future.
10	Devices authenticate with X.509 certificates	Improves security, enables identity management of devices	Smallstep CA and tooling will be used.
11	Identity management integrated into AGL UCB build process	Improves security and traceability when compared to manual identity management activities	-
12	No additional northbound protocols will be implemented.	Hono is actively developed. More northbound protocols might come with new Hono versions.	-
13	Keycloak will be removed	Only Kuksa Appstore utilized Keycloak	Keycloak support could be implemented in future, for non-Appstore purposes.

Authorization and authentication in Kuksa are handled by Keycloak [37] and Hono services. Eclipse Kuksa reference implementation implies that Appstore is the only service utilizing Keycloak and Appstore service is not going to be included in SMAD-specific Kuksa, we decided that Keycloak will removed. We did identify need for authentication service such as Keycloak might arise in the future, especially on authentication of northbound AMQP connections. Apache Qpid, the AMQP message broker used in Kuksa has good support for multiple types of authentication mechanisms as documented by [56]. As with the decision to remove data gathering support from cloud subsystem, we would gain more information about requirements for northbound connection authorization during SMAD project. This also supported the decision to remove Keycloak from SMAD-specific Kuksa at least for now.

We decided that device authentication of SMAD-specific Kuksa happens via X.509 certificates, as we could do identify management and access control of devices with certificate management practices. Neither Kuksa or Eclipse Hono define or recommend any practices or tooling for X.509 certificate management, so we considered that Smallstep CA software [57] was a good candidate for certificate management in SMAD-specific Kuksa. Smallstep CA provides tooling [58] to issue client certificates for devices and utilize Smallstep CA certificate authority and possible intermediate certificates in Hono to manage authorization of devices. We envisioned that we will integrate identity management of devices into SMAD-specific in-vehicle subsystem build process; we would generate a unique certificate signing requests (CSR) during the build, include the CSR in the built image and once the device flashed with the resulting is powered on it would exchange the CSR to a proper X.509 client certificate issued for this specific device.

C. *Building and evaluating SMAD-specific Kuksa, the design cycle*

We started development of SMAD-specific Kuksa by defining an implementation plan based on requirements deducted from SMAD project proposal, description of Kuksa framework based on APPSTACLE project deliverables, documentation and source code available for Eclipse Kuksa reference implementation [4], [5], [7], and our design decisions described in chapter 5.2 and summarized in Table I.

As we started our first implementation effort, we run into issues which forced us to rethink our implementation plan from a minimum-viable-product (MVP) point-of-view [59]. Based on refined implementation plan we implemented a SMAD-specific MVP cloud subsystem and MVP in-vehicle subsystem and integrated our test vehicle to the MVP subsystems. Each subsection describes how corresponding implementation effort was done.

1) Implementation plan

Our initial plan was to first implement the cloud subsystem, verify that the implementation works and then continue to implement in-vehicle subsystem. If possible, software from in-vehicle subsystem would be used to verify that cloud subsystem works as expected and during development of in-vehicle subsystem, cloud subsystem would be utilized to identify possible problems in the integration of the two subsystems.

To verify our SMAD-specific Kuksa implementation, we needed to define a suitable test case. We first defined the overall objective of test case; enable transfer of data read from sensors of the vehicles first to cloud subsystem and then to an application reading the data from the cloud subsystem. At this point, we did not exactly know what protocols we would use for data transfer between in-vehicle and cloud subsystem and we did not know what data we would read from the car or what software would be used to read the data. Given the number of unknowns, the test case would be built incrementally. Test case would first verify cloud subsystem implementation; data can be sent to southbound APIs and same data can be read from northbound APIs. Then the test case would verify in-vehicle subsystem implementation; this essentially meant verifying that our in-vehicle subsystem could send data to our cloud subsystem. In addition, test case would also be used to verify integration of in-vehicle subsystem to our test vehicle; this meant verifying that data can be read from vehicles sensors utilizing our in-vehicle subsystem implementation software and hardware.

Given the requirement to aim for production grade operation and maintenance we decided to focus on implementing robust IaaS set up and service deployment automation. Self-hosted Kubernetes was not an option for us and as we generally wanted to avoid tight vendor locking with any cloud provider, we decided to clearly separate IaaS and service deployment automation from each other. There would be strict separation of concerns; IaaS set up automation would utilize Terraform tool and based on our experience gained on implementing the IaaS set up automation, Helm tool or combination of Terraform and Helm would be used for service deployment. Service deployment would not depend on the selected IaaS provider and in case there is a need to deploy SMAD test platform to some other cloud provider than the initially selected Microsoft Azure, this would only mean implementation of new IaaS set up automation. Service deployment should work as it works on any Kubernetes cluster despite the underlying infrastructure. As we were drafting the implementation plan, we learned that SMAD-specific Kuksa is going to be utilized in university study courses. This introduced a need to make our deployment automation support a use-case where multiple cloud subsystems would be deployed to Azure on-demand and destroyed after the corresponding university course had ended.

2) 1st implementation effort – Refined implementation plan

After initial implementation plan had been defined, we progressed towards actual implementation by first creating minimal Terraform automation to set up Azure Kubernetes Cluster (AKS) [44] and then started developing our SMAD-specific deployment automation using Kuksa deployment documentation as reference. We soon realized that this would be too big task with high risk of failure. As we had observed before deployment instructions and scripts of Kuksa seemed complex and lacking and modifying them from SMAD-specific Kuksa without having a working deployment as a reference point for modifications, could be very time-consuming process.

We decided that deployment scripts would be implemented incrementally following the incremental implementation of our test case; we would not try implement deployment automation for complete SMAD-specific cloud

subsyste, in one go, but rather implement absolutely minimum required to evaluate our test case and as test case evolves our deployment scripts would evolve. We felt that this approach would be best supported by defining a minimum viable product (MVP), which defined only the strictly necessary to realize our overall test case objective. This meant that functionality such as identity management with X.509 certificates, support for OTA update of in-vehicle subsystem software image and use of mTLS was postponed until MVP would be finished.

3) 2nd implementation effort – cloud subsystem

As Eclipse Hono was identified as the most important component in the MVP of SMAD-specific Kuksa we started closer investigation of Hono and its deployment documentation. In contrast to Kuksa, Eclipse Hono documentation was generally very comprehensive, and its deployment documentation gave a clear instructions on how one deploys Hono. Once we successfully deployed Hono on a local development instance of Kubernetes, we felt confident that at this point no further investigation of Hono was necessary. Hono documentation gave detailed and explicit instruction on deploying Hono on AKS, which to us meant that we could focus our efforts on implementing Terraform scripts for cloud infrastructure set up and then work out possible small incompatibilities with deploying Hono on AKS infrastructure set up by our automation scripts. Hono deployment would eventually work, IaaS automation was more unknown. Given that we were committed to incremental building, following requirements of SMAD-specific Kuksa MVP, we decided that we would implement set up of access control, container registry, persistent volume management, secrets management and other production grade features only to extent required to support out current MVP implementation.

After we successfully deployed Hono on AKS set up by our Terraform scripts, we needed to implement our test case. Since at this point test case would only be able test the minimum viable implementation of SMAD-specific Kuksa cloud subsystem, i.e. Hono, and we had committed to use in-vehicle subsystem software for evaluation of cloud subsystem, we decided that our test case would replicate some parts of a newly released kuksa.val demo [60]. By re-creating key parts of the demo as we would advance to in-vehicle subsystem implementation and vehicle integration, we could iteratively develop our test case as we had planned. We deployed kuksa.val software stack on laptop and user mocked GPS data which kuksa.val transformed the GPS data into GENIVI VSS messages and sent them to Hono instance in AKS using MQTT protocol. Node-RED [61] dashboard received the messages from Hono over AMQP protocol and displayed them in a web browser UI. Implementation of our test case was fast and relatively easy as we extensively re-used source code available from [51]. We successfully verified our cloud subsystem implementation and progressed to implement SMAD-specific in-vehicle subsystem.

4) 3rd implementation effort – in-vehicle subsystem

Implementation of in-vehicle subsystem started with examination of AGL UCB and its build process. We tried to replicate build of Kuksa specific AGL UCB image and failed. Build instructions of Kuksa in-vehicle subsystem in [19] made an outdated reference to AGL UCB version not available anymore. This reference was easily fixed, and we successfully built an AGL UCB image. However, even though we had successfully built an AGL UCB image with Kuksa in-vehicle software included, we could not get this image to start on Kuksa development and testing hardware. The image started on Raspberry Pi 3 device, but since we had the impression that it should start on Kuksa development and testing hardware, we, despite successfully building an image, could not trust that we had built a working image; essentially we didn't have a working reference point to compare changes we were about to make. Since considerable amount of time was used to get to this point, we did not want to continue troubleshooting the build process of AGL UCB, as we were not confident that we would find out why the image was not starting in timely manner. Failing to do so waste considerable amount of SMAD project development effort. We decided to follow MVP approach and remove need for AGL UCB and BitBake and use a ready-made Raspberry Pi OS [62] (formerly Raspbian). We made clear separation between AGL UCB and BitBake; if time permits, build a Raspberry Pi OS based SMAD-specific in-vehicle subsystem image with BitBake and keep the repeatability and software traceability offered by BitBake. Use of AGL UCB and use BitBake would be implemented later, possible as future development. It is worth noting that by examining and troubleshooting the build process, we gained valuable experience about BitBake and customization of AGL UCB architecture. We also experimented with build process additions using QEMU [63], Packer tool [64] and Packer builder ARM -plugin [65] and defined support for device specific customizations in a pre-deployment process and an initialization process, i.e. first boot customization process, which allows integration of the planned identity management processes to build process of SMAD-specific in-vehicle subsystem. This information would be utilized in future development of SMAD-specific Kuksa.

After we changed AGL UCB to Raspberry Pi OS it was very straightforward to install kuksa.val into in-vehicle subsystem and successfully verify our SMAD-specific in-vehicle subsystem MVP implementation. Details in Table II

5) 3rd implementation effort – in-vehicle subsystem

After implementing our in-vehicle subsystem MVP we decided integrate test vehicle to SMAD-specific subsystems. In essence, the integration meant that we would need to implement everything required to evaluate the test case we defined at the beginning of implementation. Since data transfer from cloud subsystem to external application and from in-vehicle subsystem to cloud subsystem were verified to be working, our main goal was to read data from sensors of test vehicle.

We decided to use vehicle's existing CAN busses to access sensor data of the vehicle. Kuksa development and testing hardware contains an integrated CAN bus interface IC (integrated circuit), STN2120 [25], we decided to utilize that for interacting with the vehicle. Kuksa development and testing hardware is designed to utilize standard on-board-diagnostic connector (i.e. OBD connector) and as such it would be connected OBD connector of Toyota for CAN bus interaction.

We evolved our test case accordingly. Instead of mocked GPS data live data from CAN bus would be read and send to cloud subsystem as suitable GENIVI VSS message.

For the software level interaction with CAN bus, we needed to consider utility of our MVP in context of whole moving test platform. MVP is not a throwaway prototype and

we choices we make now should be useful when the system is developed further. We considered two options, ELM command protocol [66, p.10] and SocketCAN protocol [67]. kuksa.val contained some example code for interacting with CAN bus interface chip using ELM command protocol and as STN2120 supports this protocol, example code from kuksa.val could be used to fetch data from CAN busses of Toyota. ELM command protocol is a request-response protocol aimed for onboard diagnostics (OBD) of vehicles. It can be used for general purpose interaction over CAN bus, but protocol is designed to support especially OBD use-cases. When using ELM command protocol to interact with a CAN bus, one requests the CAN bus interface IC to send or receive specific CAN frames. There is also a special monitoring mode where the IC tries to capture all CAN frames in the bus. Communication with the IC happens over serial port using ELM specific AT commands. SocketCAN, in contrast to ELM protocol, can be seen as a subclass of standard network interface like an Ethernet or a Wi-Fi connection. Interface used to interact with the CAN interface, e.g. serial or USB port, is abstracted away and SocketCAN exposes CAN interfaces as network interfaces through use of Berkeley sockets API and Linux network stack. CAN interface, when interfaced through SocketCAN, can be utilized with abundance of features and tools available for interacting with Linux network stack. There are lot of open-source software libraries for communicating over ELM command protocol, but we felt that SocketCAN with well-known and mature interface provided by Berkeley sockets and Linux network stack, provides more interoperability over various vendors and products. Communication model of SocketCAN differs greatly from ELM command protocol, since by design one does not request specific messages from the CAN interface IC but filtering of CAN frames happens via functionality provided by Linux kernel. ELM command protocol is a half-duplex protocol, which means that simultaneous receiving and sending of CAN frames is impossible. SocketCAN does not have such restriction as long as the underlying CAN bus interface IC supports full-duplex communication.

We decided to utilize SocketCAN but unfortunately vendor support for SocketCAN in ICs that utilize ELM protocol is, to our best knowledge, non-existing and STN2120 was not an exception. There is an open-source implementation of Linux kernel driver, elmcan [68], which exposes ELM protocol using CAN bus interface devices through SocketCAN protocol exists. This implementation, forced by limitations of the ELM protocol utilizing ICs, has many shortcomings compared to CAN bus interface IC with vendor provided SocketCAN support. This essentially meant that if we were to use the STN2120, we could not utilize SocketCAN in a trustworthy way. To overcome this, we decided that we will evaluate trustworthiness and performance of STN2120-elmcan combination by comparing output of CAN bus readouts done with STN2120-elmcan to readouts done with Kvaser Leaf II CAN bus interface [69]. Kvaser provides vendor level SocketCAN support, and we felt that it offers a practical reference point to evaluate performance and trustworthiness (i.e. correctness of interaction with CAN bus) of CAN traffic reading with STN2120-elmcan combination.

Since our test case required to send GENIVI VSS messages to cloud subsystem, we had to transform the sensor data contained in a CAN frame to a GENIVI VSS message. SocketCAN does not decode content of CAN frames it receives or sends; CAN frame decoding into meaningful messages happens outside of SocketCAN, in the software utilizing SocketCAN. As kuksa.val already provided means to decode CAN frames to human-readable format with CAN bus database files (i.e. DBC files) and to transform the message to GENIVI VSS messages via manual mapping, it was a natural choice to use kuksa.val for CAN frame decoding. Since DBC files are typically vehicle manufacturer or even vehicle model specific we needed to obtain or implement our own suitable DBC file for decoding CAN bus traffic of the test Toyotas. Opendbc project of comma.ai [70] contained a Toyota specific DBC file suitable to be used for our MVP implementation.

When we tried utilizing STN2120 with kuksa.val we immediately found out that we cannot communicate with the STN2120. In our research on Kuksa described in section 5.2, we utilized Kuksa development and testing hardware documentation to determine the capabilities of the device and now during the implementation we used STN2120 data sheet to find out how communication with STN2120 should be done. STN2120 datasheet contained some remarks about correct communication baud rates, which we tested without success. It was clear that STN2120 on our Kuksa development and testing hardware had been configured to use different baud rate from those mentioned in STN2120 datasheet. At the time of troubleshooting, Kuksa development and testing hardware documentation didn't contain information about software configuration of the hardware and even with significant search effort we couldn't find correct baud rate setting. We reverted to deduce correct baud rate by monitoring serial communication lines of STN2120 with an oscilloscope during power up of the IC. We later learned by chance that correct baud rate setting could have been found from configuration files of kuksa.val. Once we had a correct baud rate, we successfully configured elmcan to interface with STN2120.

After communications problems with STN2120 were solved we started interacting with the Toyota through OBD connector. After first test we realized that OBD connector in our test Toyota was probably directly wired to gateway module which prevented us from reading CAN frames containing data from vehicle's sensors. Although we could not find a definitive information on this matter, we strongly feel that Toyota only supports ISO 14229-1, i.e. unified diagnostic services (UDS) protocol to communicate with the gateway module. This essentially meant that we would have to fetch vehicle sensor data in request-response manner, and we could not utilize kuksa.val CAN frame decoding functionality easily. Implementing UDS support would be too big effort since we wanted to finish our MVP implementation and verify our complete test case during SMAD project. As an alternative approach, we connected Kuksa development and testing hardware directly into CAN busses available in Toyota wiring harness, bypassing the gateway module between OBD connector and CAN busses. This way we could observe CAN bus traffic without explicitly requesting specific sensor values with UDS protocol.

Once necessary wiring harness modifications were done, we indeed observed much more CAN bus traffic than from OBD connector and kuksa.val CAN frame decoding functionality produced meaningful human readable sensor data.

We did initial comparison of CAN bus readouts made with STN2120 through elmcan SocketCAN implementation and Kvaser Leaf. We could see that CAN Frames received with

elmcan appear in the same order as the corresponding CAN Frames appear when received with Kvaser Leaf. Thorough comparison of trustworthiness of using STN2120 through elmcan SocketCAN implementation and Kvaser Leaf was left to further studies.

Table II summarizes our implementation process by describing functionality provided by the MVP SMAD-specific Kuksa system after each implementation effort.

Due to challenges we faced in the implementation process in-vehicle build process customization, identity management with X.509 certificates, support for OTA update of in-vehicle subsystem software image and use of mTLS will left for future development.

TABLE II. EVOLUTION OF SMAD-SPECIFIC KUKSA MVP

Cloud subsystem	**In-vehicle subsystem**
After cloud subsystem implementation effort (chapter 5.3.3)	
Eclipse Hono, deployed to AKS using custom made automation scripts, receives mocked device telemetry messages through MQTT and sends the messages to external application over AMQP	Kuksa.val deployed to a laptop sends mocked GPS data as GENIVI VSS messages to cloud subsystem over MQTT protocol through LAN connection.
After in-vehicle subsystem implementation effort (chapter 5.3.4)	
Eclipse Hono, deployed to AKS using custom made automation scripts, receives mocked device telemetry messages through MQTT and sends the messages to external application over AMQP.	Raspberry Pi OS deployed to Kuksa development and testing hardware. Kuksa.val, installed to Kuksa development and testing hardware, sends mocked GPS data as GENIVI VSS messages to cloud subsystem over MQTT protocol through LAN connection.
After test vehicle integration effort (chapter 5.3.5)	
Eclipse Hono, deployed to AKS using custom made automation scripts, receives device telemetry messages through MQTT and sends the messages to external application over AMQP.	Raspberry Pi OS deployed to Kuksa development and testing hardware. Kuksa.val, installed to Kuksa development and testing hardware, sends live data read from vehicle sensors over SockerCAN as GENIVI VSS messages to cloud subsystem over MQTT protocol through LAN connection.

VI. RESULTS AND LESSONS LEARNED

Building the SMAD-specific Kuksa system for the SMAD environment was started on the basis of available Eclipse Kuksa framework documentation [3]–[7] but there were a number of unknow details as presented in section 5.3.1. Some of the components of Kuksa framework turned out to be unnecessary in the context of the SMAD project, and some relevant components required modifications to address SMAD-specific needs. During the implementation of SMAD-specific Kuksa, we also identified some shortcomings in the Kuksa framework documentation, which lead us to drastically change our design during the implementation.

The design changes during implementation caused further that we were not able to build a moving test platform within the timeline of the SMAD project to the extent defined in the SMAD project plan. However, we were able to build a baseline Kuksa solution to be used in the projects following SMAD. The incremental approach and focusing on implementing an MVP [59] to evaluate our test case turned out to be a correct way to progress, as presented in section 5.3.2.

As Eclipse Hono [30] was identified as the most important component in the MVP, we started closer investigation of Hono and its deployment documentation [47]. In contrast to Kuksa framework, Hono documentation was generally very comprehensive, and its deployment documentation gave clear instructions on how one deploys Hono. We succeeded in implementing deployment automation and the container registry, persistent volume management, secrets management, other production grade features only to extent required to support the SMAD-specific cloud subsystem MVP implementation, and the test case relevant for the MVP, as presented in section 5.3.3.

For the implementation of the SMAD-specific in-vehicle subsystem, a closer examination of AGL UCB [17] and its build process was necessary. We successfully built an AGL UCB image with Kuksa in-vehicle software included, but we didn't manage to start this image on Kuksa development and testing hardware. Due to the project's timeline we decided to omit use a ready-made Raspberry Pi OS image instead, because the hardware was not tied to a specific version of Linux. After change from AGL UCB to Raspberry Pi OS it was straightforward to implement the rest of the SMAD-specific in-vehicle subsystem MVP and successfully verify it. We will utilize build process and tools of [17] in future development of SMAD-specific in-vehicle system.

After our in-vehicle subsystem MVP was built, we integrated the solution to a test vehicle. The vehicle integration had a set of low-level technical problems to be solve, as presented in section 5.3.5. We did manage to successfully integrate SMAD-specific subsystems with test vehicle using Kuksa development and testing hardware.

As a summary, our results and experiences indicate that despite its documentation shortcomings, the Eclipse Kuksa open-source framework provides new users and projects with a usable basis for building case-specific solutions for vehicle-to-environment communication and control, combining in-vehicle solutions, data communications, and cloud services.

VII. DISCUSSION AND CONCLUSIONS

In this study, we explored how to build a case-specific, intelligent automotive data system utilizing the Eclipse Kuksa framework [4], [5], [7] developed in ITEA 3 APPSTACLE [1] and Eclipse Kuksa projects [2]. The study was carried out as a part of the SMAD research project of the University of Oulu by following the guidelines of Design Science Research (DSR) as defined by Hevner et al. [10], Hevner [11] and Hevner & Chatterjee [12]. The key target of the SMAD project was to build an intelligent mobile test platform for automotive research of our university and, thus, this study was a kick-off for possible deployment of the Kuksa framework in our future research on open automotive software systems.

By following the guidelines of DSR we were able to build minimum-viable version of automotive data system, the SMAD-specific Kuksa, addressing a set of requirements of an intelligent moving test platform as defined in the SMAD project plan. We estimate that current implementation fulfills criteria for Technology Readiness Level 3 as some original project requirements were dropped, for schedule and resource circumstance of the project. Section 5 presents the building process in detail providing an answer to the research question RQ1. Our work-in-progress implementation of SMAD-specific Kuksa MVP can be found in [72].

Kuksa turned out to be an open and modular framework, customizable for a target system that was different from the ones developed in the APPSTACLE project. Needed customizations varied between different parts of the framework. As an answer to research question RQ2 we summarize our design decisions in table I.

Building a novel, case-specific automotive data system based on the Kuksa framework can be estimated to be demanding, though the framework was technically modular and customizable. This was expectable result as the framework was built in a distributed manner in the APPSTACLE project - a research project with 21 partners having various research interests and targets. The biggest obstacles for deployment (RQ3) were our limited initial experience and knowledge of the framework, shortages in the Kuksa documentation, and technical problems we encountered during implementation as presented in detail in section 5.

The easiest part to reuse turned out to be the cloud system, mostly of excellent documentation of Eclipse Hono and dropping out the most challenging use cases of the SMAD project plan. In-vehicle subsystem in combination with Kuksa hardware turned out to be the most difficult. That was expectable because the custom-build hardware and use of AGL UCB, were novel for our researcher team.

Although we didn't actively participate in Eclipse Kuksa community in course of this study, we feel that becoming an active member of Eclipse Kuksa community, would have helped us in our SMAD-specific Kuksa implementation, as we would have been able to ask details not covered by Eclipse Kuksa documentation. Active role in Kuksa community might have also helped us to coordinate our work with ongoing development effort of Eclipse Kuksa community and possibly contribute back to community more than we now did alone. We feel that active community of an open-source project fosters active and productive development effort, and this would foster evolution of Eclipse Kuksa framework.

We summarize the results of our study by noting that Kuksa is a customizable framework, deployable in different case-specific automotive data systems, but requiring lots of low-level technical knowledge on how to configure, build and use its open-source software packages. Customization needs of Kuksa are strongly context dependent, generalized requirement guidelines for customization might be achievable with further studies.

Our targets for future research on the Kuksa framework cover solving of the technical problems that were left unsolved in the SMAD project, examining scalability of the built system, and improving transferability of our experiences. Once ready, SMAD-specific Kuksa will be utilized to support our and other future research on automotive software.

ACKNOWLEDGMENT

This study was funded by European Regional Development Fund, Oulu Civil Engineering Foundation, BusinessOulu and Finnish Transport and Communications Agency. Part of the described work has been done in context of Arctic 5G project. We want to thank all colleagues and partners who worked in the SMAD project, especially student group N. Lunden, J. Holmi, J. Kosola, M. Saarinen and S. Wickström, who worked with us on implementing monitoring and tracing support for SMAD-specific cloud subsystem.

REFERENCES

[1] APPSTACLE Project, "APPSTACLE project page", ITEA3, [Online], Available: https://itea3.org/project/appstacle.html , [Accessed: Apr. 11, 2021].

[2] "Eclipse KUKSA community website", Eclipse Foundation, [Online], Available: https://www.eclipse.org/kuksa/ , [Accessed: Apr. 12, 2021].

[3] "Eclipse Kuksa documentation", Eclipse Foundation, [Online], Available: https://www.eclipse.org/kuksa/documentation/ , [Accessed: Apr. 22, 2021].

[4] eclipse/kuksa.invehicle, Eclipse Foundation, 2021, [Software], Available: https://github.com/eclipse/kuksa.invehicle , [Accessed: Apr. 18, 2021].

[5] eclipse/kuksa.cloud, Eclipse Foundation, 2021, [Software], Available: https://github.com/eclipse/kuksa.cloud , [Accessed: Apr. 18, 2021].

[6] eclipse/kuksa.ide, Eclipse Foundation, 2020, [Software], Available: https://github.com/eclipse/kuksa.ide , [Accessed: Apr. 18, 2021].

[7] eclipse/kuksa.hardware, Eclipse Foundation, 2021, [Software], Available: https://github.com/eclipse/kuksa.hardware , [Accessed: Apr. 23, 2021].

[8] "SMAD project homepage", [Online], Available: https://www.smad.fi/ , [Accessed: Apr. 16, 2021].

[9] "SMAD project proposal", [Online], Available: https://www.eura2014.fi/rrtiepa/projekti.php?projektikoodi=A74419&lang=en , [Accessed: Apr. 22, 2021].

[10] A. Hevner et al., "Design Science in Information Systems Research", Manag. Inf. Syst. Q., vol. 28, p. 75, Mar. 2004.

[11] A. Hevner, "A Three Cycle View of Design Science Research", Scand. J. Inf. Syst., vol. 19, Jan. 2007.

[12] A. Hevner and S. Chatterjee, "Design Science Research in Information Systems", Des. Res. Inf. Syst., pp. 9–22, 2010, doi: 10.1007/978-1-4419-5653-8_2.

[13] eclipse/kuksa.cloud – Appstore, Eclipse Foundation, 2020, [Software], Available: https://github.com/eclipse/kuksa.cloud/tree/master/kuksa-appstore , [Accessed: Apr. 18, 2021]

[14] APPSTACLE project, "Deliverable 1.1 - Specification of In-car Software Architecture for Car2X Applications", [Online], Available: https://itea3.org/project/appstacle.html , [Accessed: Apr. 11, 2021]

[15] R. Höttger, "Driving the Future Connected Vehicle with Eclipse Kuksa", Eclipse IoT Day Grenoble 2019, [Presentation], Available: https://wiki.eclipse.org/Eclipse_IoT_Day_Grenoble_2019 , [Accessed: Apr. 22, 2021].

[16] M. Wagner, J. Tessmer, "An Introduction to Eclipse Kuksa", Webinar on Eclipse Kuksa, 2019, [Presentation] Available: https://at.projects.genivi.org/wiki/pages/viewpage.action?pageId=34963516#Cloud&ConnectedServices-History&MinutesofBoFdiscussions , [Accessed: Apr. 22, 2021].

[17] "AGL Unified Code Base", Automotive Grade Linux, [Online] Available: https://www.automotivelinux.org/software/unified-code-base/ , [Accessed: Apr. 22, 2021].

[18] "Automotive Grade Linux project homepage", Automotive Grade Linux, [Online], Available: https://www.automotivelinux.org/ , [Accessed: Apr. 22, 2021].

[19] "eclipse/kuksa.invehicle - AGL build instructions", Eclipse Foundation, 2019, [Online], Available: https://github.com/eclipse/kuksa.invehicle/blob/master/agl-kuksa/README.md, [Accessed: Apr. 16, 2021]

[20] "eclipse/kuksa.invehicle - Firmware-over-the-air update", [Online], Eclipse Foundation, 2019, Available: https://github.com/eclipse/kuksa.invehicle/blob/master/kuksa-appmanager/wiki/fota.md , [Accessed: Apr. 16, 2021].

[21] "OpenEmbedded homepage", [Online], Available: https://www.openembedded.org , [Accessed: Apr. 22, 2021].

[22] "Yocto Project homepage", [Online], Available: https://www.yoctoproject.org/ , [Accessed: Apr. 22, 2021].

[23] "Build Process Overview - AGL Documentation". Automotive Grade Linux, [Online], Available: https://docs.automotivelinux.org/en/master/#0_Getting_Started/2_Building_AGL_Image/0_Build_Process/ , [Accessed: Apr. 22, 2021].

[24] openembedded/bitbake, OpenEmbedded, 2021, [Software], Available: https://github.com/openembedded/bitbake, [Accessed: Apr. 19, 2021].

[25] "STN2120: OBD-II, SW-CAN, MS-CAN Interpreter IC", OBD Solutions, [Online], Available: https://www.obdsol.com/solutions/chips/stn2120/ , [Accessed: Apr. 22, 2021].

[26] A. Banijamali, P. Jamshidi, P. Kuvaja, and M. Oivo, "Kuksa: A Cloud-Native Architecture for Enabling Continuous Delivery in the Automotive Domain", in Product-Focused Software Process Improvement, vol. 11915, X. Franch, T. Männistö, and S. Martínez-Fernández, Eds. Cham: Springer International Publishing, 2019, pp. 455–472.

[27] APPSTACLE project. "Deliverable 2.1 - SotA Research with regard to Car2X Communication, Cloud and Network Middleware and corresponding Security Concepts", ITEA3, [Online], Available: https://itea3.org/project/appstacle.html , [Accessed: Apr. 11, 2021].

[28] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions", Future Gener. Comput. Syst., vol. 29, no. 7, pp. 1645–1660, Sep. 2013, doi: 10/f427k4.

[29] APPSTACLE project, "Deliverable 3.1 - Specification of Data Management, Cloud Platform Architecture and Features of the Automotive IoT Cloud Platform", ITEA3, [Online], Available: https://itea3.org/project/appstacle.html , [Accessed: Apr. 11, 2021].

[30] Eclipse Foundation, "Eclipse Hono", 2021, [Software], Available: https://www.eclipse.org/hono/ , [Accessed: Apr. 11, 2021].

[31] "InfluxDB Time Series Platform", InfluxData, 2021, Available: https://www.influxdata.com/products/influxdb/ , [Accessed: Apr. 22, 2021].

[32] "MongoDB", MongoDB, 2021, [Software], Available: https://www.mongodb.com , [Accessed: Apr. 22, 2021].

[33] "Grafana", Grafana Labs, 2021, [Software] Available: https://grafana.com/grafana/ , [Accessed: Apr. 22, 2021].

[34] "Apache Flink: Stateful Computations over Data Streams", Apache Foundation, [Software], Available: https://flink.apache.org/ , [Accessed: Apr. 22, 2021].

[35] Eclipse hawkBit Project, "Eclipse hawkBit", Eclipse Foundation, 2021, [Software], Available: https://www.eclipse.org/hawkbit/ , [Accessed: Apr. 22, 2021].

[36] "Eclipse Ditto", Eclipse Foundation, 2021, [Software], Available: https://www.eclipse.org/ditto/ , [Accessed: Apr. 22, 2021]

[37] "Keycloak", 2021, [Software], Available: https://www.keycloak.org/ [Accessed: Apr. 22, 2021].

[38] eclipse/kuksa.cloud – Deployment, Eclipse Foundation, 2020, [Online], Available: https://github.com/eclipse/kuksa.cloud/tree/master/deployment , [Accessed: Apr. 18, 2021]

[39] A. Banijamali, P. Kuvaja, M. Oivo, and P. Jamshidi, "Kuksa: Self-adaptive Microservices in Automotive Systems", in Product-Focused Software Process Improvement, Springer International Publishing, 2020, pp. 367–384, doi: 10/gjpm6d.

[40] "Kubernetes - Production-Grade Container Orchestration", Kubernetes, 2021, [Software], Available: https://kubernetes.io/ , [Accessed: Apr. 22, 2021].

[41] "Helm", 2021, [Software], Available: https://helm.sh/ , [Accessed: Apr. 22, 2021].

[42] "OKD - The Community Distribution of Kubernetes that powers Red Hat OpenShift.", [Software], Available: https://www.okd.io/ , [Accessed: Apr. 22, 2021].

[43] "Bosch pursues an open source strategy to transform IoT", Bosch, 2018, [Online], Available: https://blog.bosch-si.com/bosch-iot-suite/bosch-pursues-an-open-strategy-to-transform-iot/ , [Accessed: May 24, 2021].

[44] "Azure Kubernetes Service (AKS) | Microsoft Azure", Microsoft, 2021, [Online], Available: https://azure.microsoft.com/en-us/services/kubernetes-service/ , [Accessed: Apr. 23, 2021].

[45] Azure/azure-cli, Microsoft Azure, 2021, [Software], Available: .https://github.com/Azure/azure-cli , [Accessed: Apr. 22, 2021]

[46] Eclipse Hono Project, "Deployment :: Eclipse Hono", [Online], https://www.eclipse.org/hono/docs/deployment/ , [Accessed: Apr. 22, 2021].

[47] Eclipse Hono Project, "Documentation :: Eclipse Hono", [Online], Available: https://www.eclipse.org/hono/docs/ , [Accessed: Apr. 22, 2021].

[48] eclipse/kuksa.cloud – Appstore, 2021, [Online], Available: https://github.com/eclipse/kuksa.cloud/tree/master/deployment/helm , [Accessed: Apr. 18, 2021].

[49] "agl-distro:app-framework [Automotive Linux Wiki]", Automotive Grade Linux, [Online], Available: https://wiki.automotivelinux.org/agl-distro/app-framework , [Accessed: Apr. 23, 2021].

[50] "hmiframework [Automotive Linux Wiki]", Automotive Grade Linux, Available: https://wiki.automotivelinux.org/hmiframework ,[Accessed: Apr. 23, 2021].

[51] eclipse/kuksa.val, Eclipse Foundation, 2021, [Software], Available: https://github.com/eclipse/kuksa.val [Accessed: Apr. 17, 2021]

[52] "GENIVI Vehicle Signal Specification", GENIVI, [Online], Available: https://genivi.github.io/vehicle_signal_specification/ [Accessed: Apr. 11, 2021].

[53] "Secure Boot with PS-45u DP – Swissbit", Swissbit, [Online], Available: https://www.swissbit.com/en/products/security-technology/security-products/secure-boot/ [Accessed: Apr. 13, 2021].

[54] "Terraform by HashiCorp", HashiCorp, 2021, [Software], Available: https://www.terraform.io/ , [Accessed: Apr. 23, 2021].

[55] T. Dierks, E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", [Online], Available: https://tools.ietf.org/html/rfc5246 , [Accessed: Apr. 23, 2021].

[56] "Apache Qpid - Authentication Providers", Apache Foundation, [Online], Available: https://qpid.apache.org/releases/qpid-broker-j-7.0.7/book/Java-Broker-Management-Managing-Authentication-Providers.html , [Accessed: Apr. 15, 2021].

[57] "Smallstep step-ca", Smallstep, 2021, [Software], Available: https://smallstep.com/docs/step-ca , [Accessed: Apr. 15, 2021].

[58] smallstep/cli, Smallstep, 2021, [Software], Available: https://github.com/smallstep/cli , [Accessed: Apr. 23, 2021].

[59] E. Ries, "The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, Illustrated edition". New York: Currency, 2011.

[60] "Eclipse Kuksa.val DBC Feeder Demo", Eclipse Foundation, 2020, [Video], Available: https://www.eclipse.org/kuksa/blog/2020/08/18/2020-08-18-dbc/ , [Accessed: Apr. 23, 2021].

[61] "Node-RED", JS Foundation, 2021, [Software], Available: https://nodered.org/ , [Accessed: Apr. 23, 2021].

[62] The Raspberry Pi Foundation, "Raspberry Pi OS", Raspberry Pi Foundation, [Software], Available: https://www.raspberrypi.org/software/ , [Accessed: Apr. 23, 2021].

[63] "QEMU", 2021, [Software], Available: https://www.qemu.org/ , [Accessed Apr. 23, 2021].

[64] "Packer by HashiCorp", HashiCorp, 2021, [Software], Available: https://www.packer.io/ , [Accessed: Apr. 23, 2021].

[65] M. Kaczanowski, mkaczanowski/packer-builder-arm, 2021, [Software], Available: https://github.com/mkaczanowski/packer-builder-arm , [Accessed: Apr. 23. 2021].

[66] "ELM327 datasheet", ELM electronics, [Online], Available: https://www.elmelectronics.com/products/dsheets/ , [Accessed: Apr. 23, 2021].

[67] "SocketCAN specification", Available: https://www.kernel.org/doc/Documentation/networking/can.txt , [Accessed: Apr. 23, 2021].

[68] norly, norly/elmcan. 2019, [Software], Available: https://github.com/norly/elmcan, [Accessed: Apr. 23, 2021].

[69] "Kvaser Leaf Light HS v2", Kvaser, [Online], Available: https://www.kvaser.com/product/kvaser-leaf-light-hs-v2/ , [Accessed: Apr. 23, 2021].

[70] commaai/opendbc, comma.ai, 2021, [Online], Available: https://github.com/commaai/opendbc , [Accessed: Apr. 23, 2021].

[71] M. Wagner and S. Schildt, "Innovation durch Offenheit: Das Open Source Connected Vehicle Framework Eclipse Kuksa", [Innovation through Openness - The Open Source Connected Vehicle Framework Eclipse Kuksa], Bordnetz Kongress, Sep. 29, 2018

[72] SMAD software repository, 2021, [Software], Available: https://github.com/smaddis, [Accessed: Apr. 24, 2021].