SELECTIVE SOFTWARE UPDATES WITH IN SITU MONITORING OF NON-HOMOGENEOUS AUTOMOTIVE ELECTRONIC CONTROL UNITS

Andrew Koerner[1], Björn Hendriks[2], Michael Kürschner[3]

[1]Andrew.Koerner@dlr.de
[2]Bjoern.Hendriks@dlr.de
[3]Michael.Kuerschner@dlr.de
German Aerospace Center (DLR), Institute of Transportation Systems
Lilienthalplatz 7, 38108 Braunschweig, Germany, dlr.de/ts

Keywords: Cyber-Physical-Systems, Deployment, Updating, Monitoring, Contracts

ABSTRACT

The number and complexity of in-vehicle electronic control units (ECUs) is rapidly increasing with the adaption of autonomous driving capabilities and the integration of advanced driver assistance systems (ADAS). The ECUs in a modern vehicle contain numerous software components forming a complex network of dependencies, while today's rapid development of software creates the need for more and more updates of such components during the lifetime of a vehicle, for example, to adapt to security threats. Currently, an update of a safety-critical component requires extensive re-validation of the whole system.

We have developed a prototype to test, deploy, and monitor updates of single components with minimal effort while ensuring that safety and other requirements of the entire system are still fulfilled. The safety and other properties of each component are defined as formal contracts. Virtual integration tests make sure that the contracts of all components together always satisfy the safety and other requirements of the entire system. Subsequently, it is sufficient to test an updated component only against applicable contracts. This is tested before deployment as well as monitored, in the field, during runtime.

This paper describes an implementation of these techniques by leveraging existing and mature technologies to update a safety-critical vehicle ECU.

1 INTRODUCTION

Modern vehicles contain an increasing number of more and more complex electronic control units (ECUs), caused by the growing computational demand of advanced driver assistance systems (ADAS) and autonomous driving capabilities [2, 6, 7]. ECUs are an amalgamation of non-homogeneous computing resources that can include, but are not limited to, general-purpose processors, programmable logic devices (PLDs), single-purpose processors, systems on a chip (SOCs), and application-specific integrated circuits (ASICs). With this increased demand for capability on ECUs, automotive software and configurations are becoming more complex [2, 6, 7].

Diversity in the composition of ECU compute resources means that software and configurations can take many forms such as embedded firmware, bitstreams and configurations, operating systems, shared libraries and programs to name a few. The computing diversity, software complexity, enhanced interconnectivity and networked nature of ECUs gives rise to a unique and critical importance for secure delivery of software updates to these ECUs in order to adapt to evolving security and safety threats [2], and to expand and extend product life. This paper offers a possible solution to selectively and securely deliver software updates to non-homogeneous vehicle ECUs, as well as, in situ software monitoring by leveraging existing and mature technologies.

In the following sections the bases for a system that can deliver and monitor software updates to an automotive ECU will be described. This will begin with a primer on the request tools and technologies, which will include an introduction to contract-based-design. This will be followed by a brief aggregate description of the update service provider system that brings all of the discussed tools and concepts together. Finally, the future prospects and impacts of presented concepts will be summarized and discussed.

2 BACKGROUND

2.1 Automotive vehicle software updates

Updating automotive ECUs is a significant challenge [5]. As automobile ECU complexity has increased the need for software maintenance as part of a proper overall vehicle maintenance regime becomes necessary. Providing software and configuration updates to vehicles offer some of the following important benefits:

- Extend product life through added, enhanced, or evolving capability
- Lower cost and increase sustainability
- Providing bug fixes
- Enable adaptation to emerging security threats
- Adapt to regulatory changes

The option for software maintenance for automobile owners traditionally used to be to take their vehicle to a certified mechanic or dealer [6], while some manufacturers already offer over-the-air (OTA) updates by mobile telecommunications [6] or wireless local area networks (WLAN). For this paper, it is assumed that in the future OTA updates will become increasingly ubiquitous.

2.2 Vehicle telemetry

Another emerging demand as automotive ECUs become more complex and interconnected is the need for scalable fleet monitoring and telemetry in order to ensure and maintain long term safety and security of a fleet, to identify software runtime errors, faults, failures, and defects with on-board diagnostics, as well as, to take fleet inventory of software and hardware configurations. Automotive vehicle telemetries can offer significant benefits to vehicle manufacturers, as well as, vehicle operators. Although it is possible to forward telemetry data during a service or garage visit, we assume it will be increasingly necessary for vehicle telemetry data to be forwarded via OTA connections.

2.3 Contract-based design

Due to the safety-critical nature of vehicles as cyber-physical systems all software changes must be rigorously checked, verified, and tested at every step of their life cycles. Considering that vehicles are comprised of many hardware and software configuration variations contract-based design offers an efficient approach providing rigor to design, verification, and testing [4].

In a practical sense for this paper contract-based design is a methodology for formally controlling, defining, and formalizing interfaces [1] and providing formalized guarantees on module/component behavior having a wide breadth of applications.

The following list contains just a few benefits that can be gained by use of contract-based design according to [1]:

- Enabling of parallelization/concurrent development

- Facilitating subcontractor and team interactions through rigorous formalization of software interfaces
- Removing interface ambiguity
- Tracing of requirements at all stages of the software life cycle
- Complexity management and mitigation
- Provide guarantees about module or component behavior

This paper will apply contract-based design to ensure that updated components do not violate safety and other requirements of the whole system. To achieve this, every component is related to contracts whose combination will undergo a virtual integration test to proof that the system of contracts ensures the overall requirements.

When updating a component its contracts will be propagated to the target ECU for in situ verification, and also, retroactive analysis.

2.4 Research questions

OTA connections suffer from sporadic, intermittent, unreliable, or unavailable connectivity, which is important to consider when designing an update system, as well as, a vehicle telemetry system. Based on that the main research questions this paper addresses are:

1. How can software and configuration updates be reliably and securely delivered to vehicles?
2. How can a vehicle be monitored in situ or in operation to ensure successful software updates?

3 PROTOTYPE

To tackle the research questions, we have developed a prototype vehicle software update system latter referred to as the “update service provider”. In order to demonstrate this update service provider we came up with a test case to deploy a software change in an ECU.

The primary objective of the test case is to implement a software change and propagate it over-the-air to the ECU controlling the vehicle brake lights resulting in a change in their behavior. This behavior change was to accommodate brake pedal force in order to modulate the brake light blinking pattern to indicate an emergency brake situation when high brake force was applied. A secondary objective for the test case is to implement a basic software contract, which in turn could be applied during testing campaigns, as well as, propagated to the vehicle in order to do in situ contract verification This was accomplished using an abbreviated continuous integration/continuous deployment (CI/CD) pipeline.

3.1 Update service provider prototype

We designed and implemented a prototype update service provider system to provide software and configuration updates over-the-air to a vehicle with the updates specifically targeted at the brake light ECU. This brake light ECU prototype consists of a general purpose computing resource running Debian 10 and a Nortic Semiconductors nRF52840 microcontroller acting as an embedded peripheral to control the brake lights. The prototype update service provider system provides means to monitor vehicles via a vehicle telemetry system, software packaging and delivery capability, and finally an implementation of contract-based design.

3.2 User roles and scenarios

There are three user roles in the system: software engineer, system operator, and vehicle owner/operator. A software engineer can manage and modify software code and release it to the update process. Software that has been released to the update process can then be deployed to vehicles by the system operator in the form of an update campaign through a user interface dashboard. Finally, software that has been made available on the vehicle via an update campaign can be installed by a vehicle owner/operator with consent and on demand by interacting with an in-vehicle display.

3.3 Technical Background

3.3.1 Requirements engineering supplemented with contracts

The first step realizing the benefits of contract-based design is deriving contracts from requirements. For example take a requirement that was used for the brake light ECU: *brake light must be extinguished after no greater than 100 ms after the brake pedal is released*. This requirement implies the post conditions of a brake pedal event and facts regarding the pressing of the brake pedal, e.g., the event can take no longer than 100 ms.

3.3.2 Contract derivation

Given the previous example requirement the following derived implication could be used as the basis of a contract: `if` *the brake pedal is released* `then` *the brake light should be off*. Contracts can be specified as broad or as granular as needed in order to fully encapsulate a given requirement and apply the desired level of rigor and formality to a given requirement. A single requirement can have many subsequent contract specifications that capture it giving flexibility to system engineers. With the previous example requirement two subsequent contracts were derived (shown in Figure 1). Contracts can be grouped based on context, applied broadly across many requirements, or be applied in a highly discriminatory and specific manner.

#	Contract ID	REQ ID	Description	Assumption	Guarantee
1	AP6.1-1.2-C1	AP6.1-1.2-1.1	The brake light has to be turned off after at most 100 ms after the brake pedal was pushed	execution_time > 0 and execution_time != None	execution_time <= 100
2	AP6.1-1.2-C2	AP6.1-1.2-1.1	If the brake pedal is released then the brake light should not be on	brake_force = 0	brake_light_state = 0

Figure 1: Derived brake light contracts related to releasing the brake pedal

As a supplement to a typical requirements engineering process a contract could be specified and associated with the previously discussed requirement and derived contracts can then be captured using the same tools used to capture and trace requirements. During procedural evaluation of the contract against vehicle telemetry the following logical implication is evaluated:

`if` ***assumption*** `then` ***guarantee***

3.3.3 Contract serialization

With a specified set of contracts derived from requirements the technical benefits of contract-based design can be realized. The curated contracts can be serialized or converted into a consumable data structure. For our prototype JavaScript Object Notation (JSON) was used, however other serialization formats such as XML would also be appropriate. The data structure (shown in Figure 2) is used to serialize contracts for this prototype. The core elements of the data structure (shown in Figure 2) are the *contract assumption*

and *contract guarantee*. The other elements namely *precondition* and *postcondition* are an extension to the contract concept and support contract evaluation by enforcing certain data types for inputs and outputs. The *execution time* definition enforces timing constraints on the component. Once contracts have been derived and serialized they can be versioned and controlled using software version control such as *Git* alongside their associated safety-critical software components. Finally, versioned contracts can also be packaged and delivered alongside their safety-critical software components to target systems for in situ contract evaluation and enforcement.

Figure 2: Contract data structure

3.3.4 Vehicle telemetry

In order for safety-critical software components to take advantage of contracts in the prototype system they must log relevant telemetry for later contract verification and telemetry forwarding. This logging can occur in one of two places. Firstly, within the unit or function, which requires modification of the function or unit to add necessary log messages. The second option is to wrap the unit or function in a monitoring function call. The benefit of the second approach is that the unit would not require direct modification.

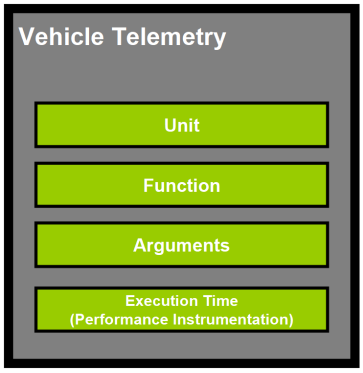

Figure 3: Vehicle telemetry data structure

Necessary parameters that must be logged for contract verification are shown in Figure 3. Some of the parameters logged by the monitoring function in our prototype are the function name, unit name,

version, as well as, data types, return values, and arguments. Another critical parameter that is captured and logged by the monitoring function call is performance instrumentation in the form of execution time or how long the target function took to call and return.

3.3.5 Contract enforcement and evaluation

The two ingredients necessary in order to apply and enforce contracts are: 1) a contract specification in the form of a serialized data structure, which was previously described, 2) telemetry containing the applicable unit, function, and run-time environment. From this point on, contract enforcement becomes a trivial procedural check by first selecting a contract and procedurally applying it to all applicable and available telemetry. The contract verification procedure (shown in Figure 4) begins with a telemetry pointand a contract specification. Function preconditions and postconditions are checked if specified in the provided contract followed by the contract implication. This procedural check on telemetry can be executed on the vehicle in situ or retrospectively against a historical telemetry database.

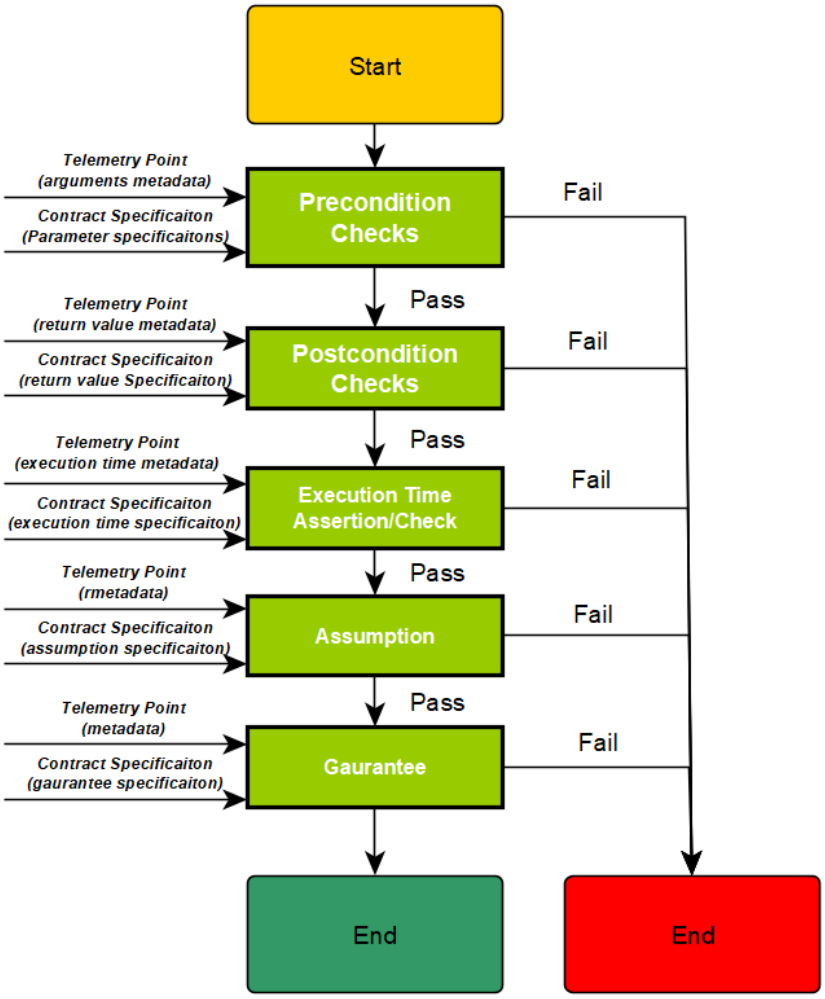

Figure 4: Contract Evaluation Procedure

3.3.6 Vehicle telemetry system

Previously we mentioned that logging of telemetry data is done on the vehicle. This technical concept will be expanded on to form the basis for the vehicle telemetry system used in the prototype. For logging

we apply rsyslog. Rsyslog is a scalable high-performance logging system that can log to local systems or forward logs to remote infrastructure requiring minimal processing power [9]. Rsyslog also runs on a wide variety of systems including embedded systems. Some key features required by a vehicle telemetry system that rsyslog offers are log rotation, log filtering/transformation, and secure log forwarding over IP networks. Vehicles will have sporadic internet connectivity as discussed in a previous section and for this reason log rotation is an important feature of a telemetry logging system. Log rotation is critical for vehicle ECUs due to limited physical storage and log filtering/transformation is important for a vehicle telemetry system due to the potential high-volume log message generation.

In an example where an event occurs thousands or tens of thousands of times with identical parameterization instead of persisting multiple thousands of log messages a single message can be filtered, persisted, and forwarded. This is especially important for the resource-constrained environment of an ECU. Finally, rsyslog offers log forwarding over IP networks, which is the basis of the telemetry system in the prototype and chosen due to the ubiquity of rsyslog in IT for most popular operating systems, as well as, for embedded and real-time systems.

For example the network (shown in Figure 5), which is representative for the in-vehicle network of a modern vehicle with two primary networks, namely the Controller Area Network (CAN) bus and the Ethernet [6]. To realize the test case for the brake light ECU for this prototype it is resident on the Ethernet side of the in-vehicle network. During operation of the brake light ECU firmware/software for both installation and runtime, rsyslog is used to log status and telemetry by the firmware. This in turn is forwarded on the in-vehicle network to the gateway for vehicle wide log/telemetry collection from non-homogeneous ECUs.

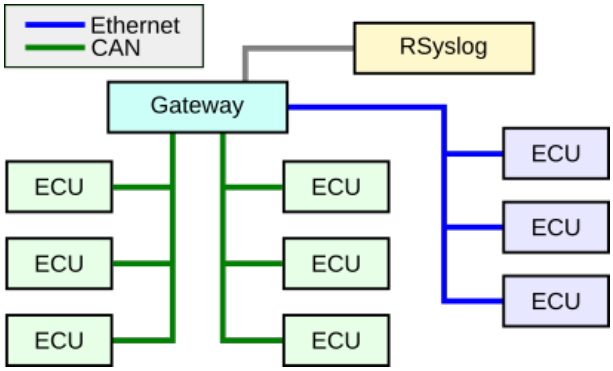

Figure 5: In-vehicle network

With this basic telemetry architecture centered around rsyslog, telemetry collected by a wide array of vehicle types and infrastructure can be forwarded via a Wide Area Network (WAN) for collection, retention, processing, and analysis on a large scale. The flexibility and configurability of rsyslog is particularly suitable for automotive use cases especially with low bandwidth networks. Messages can be collected and forwarded asynchronously in a highly discriminatory manner. This is useful for environments where physical storage and network are a premium resource. The WAN (shown in Figure 6) shows a network setup with multiple end-point types (vehicles, infrastructure, et cetera) that can forward telemetry to a centralized collection service. Having a comprehensive and representative vehicle telemetry database has many positive benefits including the ability for root-cause analysis and validation testing against real historical data.

3.3.7 Packaging and package management systems

The complexity of software package management systems is high. They provide a wide range of features and capabilities, which include: dependency management, configuration management, variant management, define and control target system architecture, package priority and urgency management, software

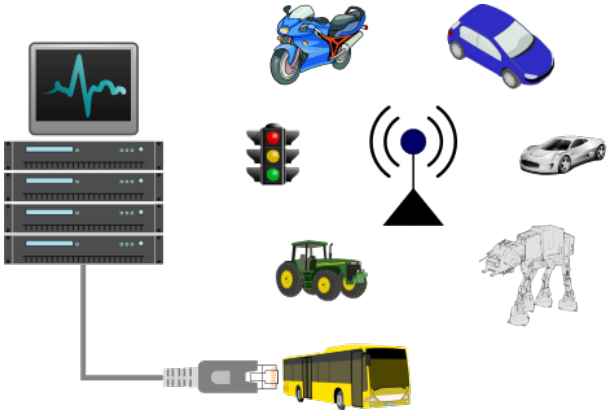

Figure 6: Example for a vehicle wide area network collecting telemetry data with multiple end-point connections

delta distribution, package integrity and secure package distribution [8]. Package management systems are available for mostly every operating system and even embedded systems with some examples such as: dpm, rpm, opkg, ipkg, et cetera. For the update service provider of this prototype the Debian Package Management System (dpm) was leveraged to provide software and configuration changes to the vehicle brake light ECU. Although, dpm was utilized for the prototype this approach could be applied to any package management system.

The Debian package management system, utilized in our prototype update management system, is capable of delivering arbitrary configuration, binaries, and file system changes to a target system, over a secure channel [3]. In our case the target system was the brake light ECU. Our brake light ECU consists of a general purpose computer resource running Debian 10 and a microcontroller acting as an embedded peripheral. With the Debian package management system it was possible to deliver contract specifications, drivers, embedded firmware for the microcontroller, as well as, monitor installation status via the previously described telemetry system, and roll back to a previous state on failure.

To implement a custom package management system is a significant development effort with many possible pitfalls. With this prototype it has been demonstrated that existing IT package management systems can be leveraged for automotive and are even appropriate to deliver embedded firmware to “deeply”embedded systems. Furthermore, any software development project that can be built with the *make* or *CMake* build system can immediately take advantage of many existing package management systems.

4 UPDATE SERVICE PROVIDER

Thus far all of the necessary technical components required to deliver software and configuration updates to a vehicle ECU and from an update service provider have been described. This includes a vehicle telemetry system based on rsyslog and package management via Debian Package Management System (dpm) to deliver software and configuration for complexity management and in situ verification of software changes. These tools can be an extension to any standard or existing build pipeline in order to exploit the benefits of contract-based design.

Before the update process begins, standard change management and requirements engineering occur with the added step of contract capture, derivation, and serialization and deployment as previously described. Following this, and coupled with the telemetry system, any software or configuration change can be verified against applicable contracts either in situ or during other stages of the software life cycle, with the benefit of a historical telemetry database.

4.1 The update deployment process

With the right tools in place a basic continuous deployment process could look like the following:

1. A software change has been approved and committed by a change control process.
2. A change delta becomes available to the update system for release by an update campaign.
3. Approved change is selected and released (via a management dashboard GUI) to vehicles by a release campaign.
4. Automated testing is performed on the change delta on a simulated vehicle. This includes execution of available test suites, generation of vehicle telemetry, and telemetry verification against a contract specification and historical vehicle telemetry database.
5. Change is packaged, with applicable contracts, and published as update to a package repository.
6. Update is installed on the vehicle (via in-vehicle GUI dashboard). This includes installation of drivers, shared libraries, applications, applicable contracts, and flashing firmware to embedded systems.
7. Run-time telemetry is generated and forwarded by the telemetry system for future testing, and failure detection, correction, and analysis.

Due to the ubiquity and flexibility of rsyslog, integrating it into existing mobility infrastructure and subsequently a CI/CD pipeline becomes a straight forward configuration process often requiring no or little modification of existing software systems. Together with the previously described process of design-by-contract a powerful engineering tool becomes available. This system provides a means to define formalized behavior as described by a given requirement and traces it to subsequent impacts and effects in situ. This can improve design rigor, as well as, provide confidence and predicability that a software change will not have unintended consequences.

Leveraging existing package management systems also has significant benefits in terms of software reuse and it has been demonstrated with this prototype that it is possible to apply dpm to automotive systems. This includes delivering changes in the form of drivers, firmware, programs, contract specifications, and configurations to a non-homogeneous ECU.

5 CONCLUSION

In this paper, a practical example of delivering monitored software and configuration updates to a non-homogeneous automotive ECU was outlined applying contract-based design. The key novelties in the provided approach are leveraging existing IT domain technologies to form the basis of a vehicle telemetry system, as well as, leveraging existing packaging management systems to deliver software, configuration, firmware, and contract specifications to an ECU. Problems from the information technology (IT) domain directly map to the automotive domain and with ECU complexity and interconnectivity/networking on the rise the line between the domains is becoming blurred. The result of this is that the automotive domain can immediately and directly benefit from decades of progress within IT.

ACKNOWLEDGEMENTS

This work has been funded by the German Federal Ministry of Education and Research (BMBF) in the project Step-Up!CPS (Förderkennzeichen: 01IS18080C). The responsibility for the content remains with the authors.

REFERENCES

[1] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Thomas Henzinger, and Kim Guldstrand Larsen. *Contracts for System Design*. Research Report RR-8147. INRIA, Nov. 2012, p. 65. URL: https://hal.inria.fr/hal-00757488.

[2] David J. Coe, Jeffrey H. Kulick, Aleksandar Milenkovic, and Letha Etzkorn. "Virtualized In Situ Software Update Verification: Verification of Over-the-Air Automotive Software Updates". In: *IEEE Vehicular Technology Magazine* 15.1 (Mar. 2020), pp. 84–90. ISSN: 1556-6080. DOI: 10.1109/MVT.2019.2954302.

[3] *Debian package management*. Apr. 2020. URL: https://www.debian.org/doc/manuals/debian-reference/ch02.en.html.

[4] Houssem Guissouma, Carl Philipp Hohl, Hannes Stoll, and Eric Sax. "Variability-Aware Process Extension for Updating Cyber Physical Systems Over the Air". In: *2020 9th Mediterranean Conference on Embedded Computing (MECO)*. 2020, pp. 1–8. DOI: 10.1109/MECO49872.2020.9134339.

[5] Houssem Guissouma, Andreas Lauber, Amir Mkadem, and Eric Sax. "Virtual Test Environment for Efficient Verification of Software Updates for Variant-Rich Automotive Systems". In: *2019 IEEE International Systems Conference (SysCon)*. 2019, pp. 1–8. DOI: 10.1109/SYSCON.2019.8836898.

[6] Byungjoo Kim and Sungkwon Park. "ECU Software Updating Scenario Using OTA Technology through Mobile Communication Network". In: *2018 IEEE 3rd International Conference on Communication and Information Systems (ICCIS)*. Dec. 2018, pp. 67–72. DOI: 10.1109/ICOMIS.2018.8645019.

[7] Yutaka Onuma, Yoshiaki Terashima, Sumika Nakamura, and Ryozo Kiyohara. "A method of ECU software updating". In: *2018 International Conference on Information Networking (ICOIN)*. Jan. 2018, pp. 298–303. DOI: 10.1109/ICOIN.2018.8343129.

[8] Diomidis Spinellis. "Package Management Systems". In: *IEEE Software* 29.2 (Mar. 2012), pp. 84–86. ISSN: 1937-4194. DOI: 10.1109/MS.2012.38.

[9] *The rocket-fast Syslog Server*. Apr. 2020. URL: https://www.rsyslog.com/doc/master/.