26th IMEKO TC4 International Symposium
24th International Workshop on ADC and DAC Modelling and Testing
IMEKO TC-4 2023
Pordenone, Italy / September 20-21, 2023

Rapid Prototyping of Vehicle Software Defined Functions

Jan Sobotka[1], Jiří Novák[1], Jiří Pinkava[1]

[1]*Czech Technical University in Prague, Prague, Czech Republic, jan.sobotka@fel.cvut.cz*

***Abstract* – A modern car is made up of a considerable amount of software running on many Electronic Control Units interconnected by communication infrastructure. Together it creates a rigid system, which is not easy to modify for running modified or custom code to implement new vehicle functions or gathering scientific data. The paper presents a modified (electric) vehicle architecture allowing rapid prototyping. The architecture enables control of particular vehicle functions by an independent computer, arbitrary vehicle data acquisition, and offers an interface for communication with the driver. Car-dependent and independent layers are used to provide an abstraction and hide details of a specific car model.**

I. INTRODUCTION

Passenger cars are object of research activities focused on many research areas. They include, e.g., vehicle stability control, passenger comfort, simple and advanced driver assistance systems, and so on, up to autonomous driving technologies. Implementing a new algorithm into a standard production car to validate it is often as challenging for an independent researcher as the research itself. Today passenger cars mostly use a domain-oriented architecture of in-vehicle electronic systems, which relies on a high number of Electronic Control Units (ECUs) interconnected by different networking technologies [1]. Details about particular ECUs functionality [2], as well as data communication [3] over the in-vehicle networks, are subject of public access restrictions. In addition, each car manufacturer uses its own architecture of internal electronic systems and in-vehicle networking. The complexity and lack of information for implementation on a real car often lead to the validation of research results on models of the vehicle or its subsystem [4]. These models, either intentionally or by mistake, often miss important aspects of reality, and the validity of the research validation is then limited. In some cases, the research results should be validated by the driver and passengers (user experience); here, the actual implementation of the research result is essential. To provide solution, the HW/SW framework was designed and developed, which hides the vehicle-dependent implementation details. It provides high-level API allowing information acquisition from the vehicle (e.g., vehicle speed, internal and external temperature, seat occupation, air condition settings) and simultaneous control of selected vehicle functionalities (e.g., air condition settings, adaptive cruise control settings, infotainment settings, drive mode settings) in a vehicle (and possibly the vehicle manufacturer) independent way. A new prototype functionality can thus reside on the top of the framework, utilize the data available from vehicle information systems, and control vehicle settings according to the researcher goals.

II. E/E ARCHITECTURE MODIFICATION ANALYSIS

Electronic functionalities are implemented in various ways. It depends on the used E/E (Electrical/Electronic) architecture [1]. Typical vehicle architecture evolves from several isolated systems over communicating specialized ECUs towards general ECUs over time. This section analyzes modification possibilities for the rapid prototyping of new software-defined functionalities.

A. ECU Software Modification

A natural way for rapid prototyping of a new feature is a modification of ECU software. The situation is complicated by two facts. First, running own code on an ECU without manufacturer support is a challenging task [5], as often even the car manufacturer does not have the original source codes available. Second, I/Os are distributed over multiple ECUs; thus, code modification in a single ECU might not be sufficient. For rapid prototyping, it is necessary to have the ability to extend existing code for a new feature. Running its own software on ECU hardware thus leads to the implementation of complete ECU software, including communication with the rest of the car. This is nearly impossible for an independent researcher, so the following methods can be used.

B. Physical Signals Modification

Every vehicle electronics system uses some sensors and actuators. Modification of input signals, as well as output signals, is the first possibility for alternating the system behavior. This approach is widespread in Hardware-in-the-loop (HIL) testing [6]. The disadvantage of this technique is a limited generality. Different types of sensors and actuators require specific solutions. Also, the concurrent usage of I/O for an existing and prototyped function could be problematic.

C. Communication Modification

Modern passenger cars reside on in-vehicle communication. Several communication standards are used. The most commons are CAN/CAN FD, LIN, Automotive Ethernet, and FlexRay. In the rest of the paper, CAN/CAN FD technology is assumed. A widespread scenario is a communication among the ECUs based on the exchange of entities called signals. An example of the signal is outside temperature measured by an ECU and broadcasted to other network nodes by CAN bus [7]. Switch status (pressed/released) checked and transmitted in respective CAN message by one ECU can be another example. The message is received by other ECU and used to control the output as required. To change the switch state, the dedicated communication device that filters and/or modifies the original message and sends a modified one toward the ECU of interest should be used.

D. Service-Oriented Architectures

Not only signal communication is used in modern vehicles. Service-oriented protocols are becoming an integral part of modern cars. These protocols can either be proprietary or open. An example of an open protocol is Scalable serviceOriented MiddlewarE over IP (SOME/IP) [8] as a part of AUTOSAR. A representative of a proprietary protocol can be BAP (Bedien und Anzeigeprotokoll), used in Volkswagen Group vehicles. The advantage of services-oriented communication is that requests for service provided by ECU can come from any other ECU. And thus, the service requests can be generated using a suitable communication interface connected to a bus.

III. CAR ARCHITECTURE FOR RAPID PROTOTYPING

The presented platform for rapid prototyping of new vehicle functionalities uses the principles described above in paragraphs C. and D. Overall architecture is depicted in Fig. 1. The central element is the industrial computer Advantech ARK-3520P. Used technologies require some up-to-date desktop Linux distro. In our case, it is Ubuntu 22.04.

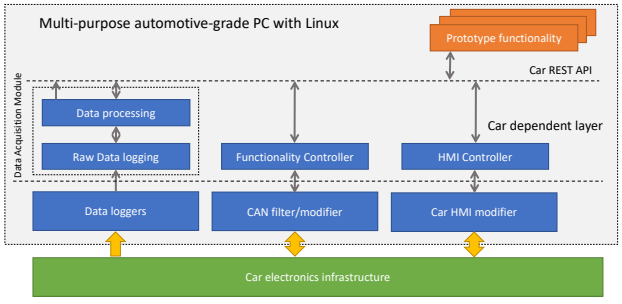

Fig. 1. Car Architecture for Rapid Prototyping

All the platform services are available at the Car REST API [9], which is vehicle (and vehicle manufacturer) independent. The car-dependent layer consists of three modules, providing for vehicle data acquisition, required vehicle functionality control, and access to vehicle infotainment screen (user-defined screens, driver touch responses). Each of these software modules is supported by the hardware modules that physically interfere with the car's functions (see Fig. 2).

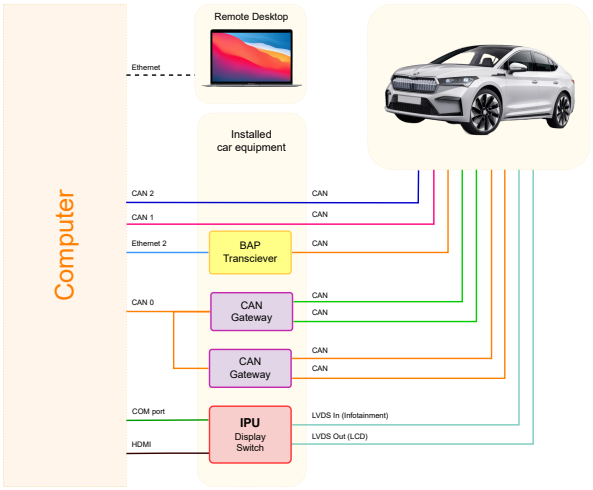

Fig. 2. Car Hardware Setup

A. Data Acquisition Module

The Data Acquisition module is implemented using Data loggers HW layer, consisting of external CAN FD/USB interfaces. They are supported by the native Linux CAN interface SocketCAN. DAQ module receives CAN messages from an in-vehicle network and interprets them using industry-standard .dbc definition file into the physical quantities. Simultaneously the BAP protocol messages (carried in specific CAN messages) are parsed, providing complete information about the vehicle state and activities. The stream of acquired vehicle signals (in JSON format) is pushed to the TCP endpoint configured (using the Car REST API) from the prototyped application. Each signal value is accompanied by its timestamp.

B. Functionality Controller Module

The preferred way to control the car functionalities is real-time manipulation of in-vehicle communication networks. The data manipulation system structure is shown in Fig. 3. To independently control selected vehicle functionalities, the CAN Gateway (GW) modules (providing real-time manipulation with CAN message) are inserted into the internal vehicle communication paths. Their activities (which are independent of specific vehicle functionality that should be initiated) are controlled by a Func-

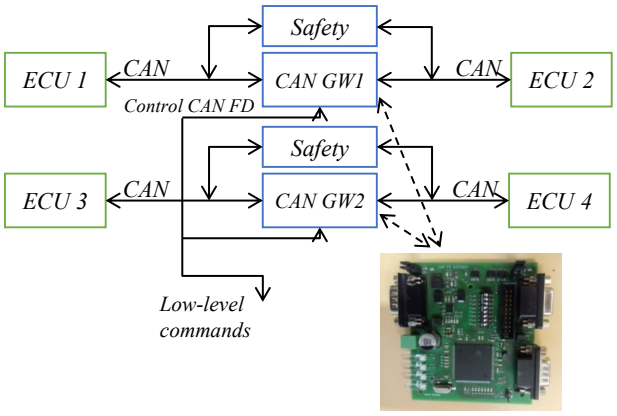

Fig. 3. CAN Gateway Application Principle

tionality Controller module. The CAN Gateway allows blocking/passing selected messages, modifying their content (with respect to some application layer constraints), and transmitting/monitoring chosen messages. The CAN Gateway functionality is vehicle and function independent. Thus the list of supported functionalities can be extended without the GW firmware modification (only the Functionality Controller module will require a software update in such a case). Service oriented communication used to control specific car functionalities is also implemented within this SW module. Each CAN Gateway is equipped with an emergency circuit – in case of any problem, the CAN Gateways are bridged, and the original CAN interconnection is restored to keep the driver and passengers safe.

C. HMI Controller

Human-Machine Interface (HMI) Controller module provides for communication with driver/passengers. It relies on video inserter and CAN GW hardware support. The overall HMI subsystem structure is shown in Fig. 4.

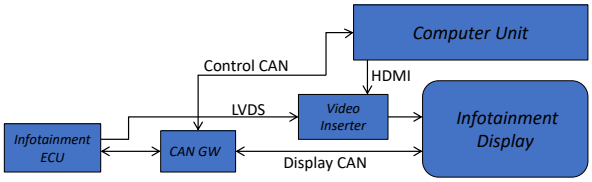

Fig. 4. HMI Control Application Principle

Video inserter is placed between the vehicle Infotainment ECU and its display. It allows switching between the standard video source (Infotainment ECU) and the user prototype generated video (HDMI output of Linux computer). In case the prototype SW needs to use the car display, the display is switched to the prototyped screen, and the user response is awaited (see Figure 5). The display is connected by the FDP-Link interface. The display unit supports touch-screen functionality – user actions are reported from the display to the Infotainment ECU by CAN messages. The CAN GW module is used here again, allowing redirection of touch screen message of interest, when the response on prototype generated screen is expected. Simultaneously it blocks the touchscreen messages transport from display to Infotainment ECU when prototype software generated screen is active (to avoid false car Infotainment ECU reactions).

IV. RESULTS - PLATFORM USAGE EXAMPLES

Rapid-prototyping platform allow its user to design and validate applications that may change the way how drivers use their cars. The application can use information from in-vehicle networks (Data Acquisition subsystem), control selected car functionalities (Functionality Controller subsystem), and use internal user interface (HMI Controller subsystem), all available at the vehicle-independent car REST API. To evaluate the platform's functionality and to show its ease of use, we have designed and implemented two groups of examples. Within the first group, there are classical (rule-based) applications, and within the second, the machine learning (ML) based applications. All the below-described applications can be configured using car HMI.

A. Classical Application Examples

The first example application uses car-provided information about the actual location and external map sources to get information about the tunnels. It implements automated activation of internal air circulation when the car approaches the tunnel and switches it back when the tunnel is left. This functionality can be configured as fully autonomous or user-confirmed. The second example application uses car-provided information about the internal temperature and seats occupation. If the temperature is below the limit, occupied seat heating is activated. The temperature limits and heating level can be configured individually for particular seats. This application can easily be extended to support automated control of seat ventilation if supported by car hardware.

B. Machine Learning based Examples

The third example application again uses the car-provided information about the actual location. Simultaneously it acquires information about the ACC (Adaptive Cruise Control) activity and required car speed (in case the ACC is active). During the training phase, the ML model learns the way the driver uses ACC. When trained, its predictions are used to (de-)activate the ACC and set the required speed (after the simple confirmation from the driver (see Fig. 5). The fourth implemented example is focused on car infotainment. On weekday mornings, the driver

regularly takes the children to school. They always ask him/her to activate the function of playing songs from their mobile phone via infotainment. An ML-based application receives information about the date/time, seat occupation, and infotainment setting. It learns the usage pattern and autonomously activates the audio source to the children's phone when they get on and back to the driver's favorite radio station when they get off.

Fig. 5. Prediction Based ACC Activation Offer

V. CONCLUSION

The platform for rapid prototyping of new vehicle features was presented. It addresses the problem of rapid prototyping or evaluating new software-defined functions, algorithms, and data acquisition on production vehicles. The main capabilities are data acquisition of information available on in-vehicle networks, independent control of selected vehicle functionalities, and HMI access. The used approach can be adapted to vehicles of different manufacturers, but quite detailed information is required. The presented platform was incorporated into Škoda Kodiaq and Škoda Enyaq Coupé RS iV. Thanks to the shared vehicle platforms, it is pretty straightforward to implement the proposed solution into any car of the VW group.

Further rapid prototyping platform development is planned to focus on three areas. First is the usage of SOME/IP services over the Automotive Ethernet within the Functionality Controller. Second is the extension of the platform hardware layer with LIN Gateways. They provide similar services as CAN Gateways described above, but for the LIN networks. Finally, the HMI will be improved to support voice input (instead of touchscreen response, which distracts the driver much more).

ACKNOWLEDGMENT

This research has been realized using the support of Technology Agency of the Czech Republic, programme National Competence Centres, project #TN01000026 Josef Bozek National Center of Competence for Surface Transport Vehicles. This support is gratefully acknowledged.

REFERENCES

[1] A. G. Mariño, F. Fons, and J. M. M. Arostegui, "The future roadmap of in-vehicle network processing: A hw-centric (r-) evolution," *IEEE access*, vol. 10, pp. 69 223–69 249, 2022.

[2] J. Van den Herrewegen, "Automotive firmware extraction and analysis techniques," Ph.D. dissertation, University of Birmingham, 2021.

[3] M. Zago, S. Longari, A. Tricarico, M. Carminati, M. G. Pérez, G. M. Pérez, and S. Zanero, "Recan–dataset for reverse engineering of controller area networks," *Data in brief*, vol. 29, p. 105149, 2020.

[4] X. Pan, C. Zivkovic, and C. Grimm, "Virtual prototyping of heterogeneous automotive applications: matlab, systemc, or both?" in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 544–549.

[5] P. Nasahl and N. Timmers, "Attacking autosar using software and hardware attacks," *2019 Embedded Security in Cars USA*, 2019.

[6] D. d. S. A. Loura, F. F. V. de Melo Ferreira, and R. C. da Costa, "Hardware-in-the-loop alternative approach for an esp verification," SAE Technical Paper, Tech. Rep., 2022.

[7] S. Yong, Y. Ma, Y. Zhao, and L. Qi, "Analysis of the influence of can bus structure on communication performance," in *IoT as a Service: 5th EAI International Conference, IoTaaS 2019, Xi'an, China, November 16-17, 2019, Proceedings 5*. Springer, 2020, pp. 405–416.

[8] D. Martin, L. Völker, and M. Zitterbart, "A flexible framework for future internet design, assessment, and operation," *Computer Networks*, vol. 55, no. 4, pp. 910–918, 2011.

[9] K. Relan, *Building REST APIs with Flask: Create Python Web Services with MySQL*, 1st ed. USA: Apress, 2019.