Service-Oriented Communication Between ADAS and IVI Domains in Automotive Solutions

Dušan Kenjić and Marija Antić, *Members, IEEE*, Dušan Živkov

Abstract—**As the complexity of automotive systems has grown, it has become necessary to cluster various vehicle components into several domains, based on a specific function they perform. This approach has facilitated the development of domain-specific features, as it allows to create communication standards and common libraries that meet the requirements of the particular domain. On the other hand, it has created the redundancy in the resource consumption required to perform similar tasks in different domains, which leaves the room for further optimizations. This is most notable if we analyze the functionalities of the two fastest growing domains: autonomous driving assistance (ADAS) and in-vehicle infotainment (IVI), which are both developing simultaneously, and may benefit from the option of providing features and services to each other. This paper will examine and propose a solution for interconnection between ADAS and IVI domains by utilizing state-of-the-art mechanisms of the service-oriented architecture (SOA) paradigm. The examination of SOA utilization rationale will be presented, as well as the crucial challenges and limitations of the possible approaches, derived mainly from the discrepancy of service-oriented architecture implementation and mapping in different standards. Various features and use-cases will be discussed, that would be good candidates for cross-domain implementation.**

Index Terms— **in-vehicle domains, ADAS, IVI, interconnection, SOA in automotive.**

I. Introduction

The transition to centralized domains in the automotive system design and development was necessary due to the increasing number of Electronic Control Units (ECUs) in the modern vehicle. With this approach, the system is organized into several domains based on the features and the tasks ECUs within the domain perform. By splitting the whole system into a set of specialized domains, it was possible to create standards and abstractions that facilitate the development of features specific to the particular domain, without the need for developers to constantly solve the problems of connectivity and resource sharing. Two domains which are constantly improved and require powerful resources are ADAS - Advanced Driver System Assistant domain and IVI - In-vehicle Infotainment domain. The ADAS domain is responsible for safety-critical features and algorithms using various types of sensors in order to enable safe, comfortable and cost-effective driving. On the other hand, the IVI domain is oriented towards passenger entertainment, as well as towards providing useful information about the driving conditions and the state of the vehicle. Although these two domains perform different tasks, there is a set of features and sensors of the same type which are commonly used in both of them. However, in the current architecture of modern vehicle, these two domains do not share any of the hardware resources nor results of the data processing algorithms. This creates an implementation overhead, as similar functionalities need to be implemented in both of the domains, and the hardware cost is constantly increasing. This represents the main motivation to design an approach for resource sharing between domains as a first step towards the unified platform which shall control the entire system.

In this paper, we will present the results of the initial phase of a research project aiming to create the solution for the inter-domain communication and resource sharing in the automotive solutions. First, we will present the summary of the state-of-the-art research and commercially used approaches for the inter-process communication and service-oriented architecture in automotive industry. Then, we will propose the architecture of the solution connecting ADAS and IVI domains, provide some practical details and discuss the examples of the use-cases which would benefit from the resource sharing between these two domains. Finally, we will discuss the implementation challenges of the proposed approach.

Dušan Kenjić is currently working toward the Ph.D. degree with the University of Novi Sad, Serbia, (e-mail: dusan.kenjic@uns.ac.rs).

Marija Antić is currently the Assistant Professor with the University of Novi Sad, Serbia, (e-mail: marija.antic@rt-rk.uns.ac).

Dušan Živkoc is with the RT-RK Institute for Computer based Systems, (e-mail: dusan.zivkov@rrt-rk.com).

II. IPC And SOA In Automotive Systems

A. *Service-Oriented Middleware in Automotive*

Traditionally, automotive systems use a conventional signal-based communication approach, which provides a deterministic data transfer, and enables the processes to run in the predefined schedule [1][2]. However, such an approach does not support the desired scalability of the system, which is required to satisfy the requirements of emerging applications and scenarios. Therefore, in order to provide the flexibility and a more dynamic and scalable system, service-oriented architecture (SOA) was introduced to the automotive system design. Service-oriented communication approach has been adopted from the domains of web applications, cloud and information systems, where it has already proven its flexibility for functional services implementation [3]. SOA represents an efficient way to encapsulate the job done by the specific component into a service. This way, resources can be distributed to the clients interested in the information which the service provides, the service implementation can remain obscured from the clients and modularity and repetitiveness can be achieved. Additionally, the unified communication mechanism facilitates the interoperability between heterogenous system components, which otherwise represents a time consuming and challenging problem that needs to be solved during the application development.

The first step towards the integration of SOA principles within an automotive system is to create a platform and define a protocol which can support this integration. Such platform must be compatible with other automotive solutions and protocol must be suitable with the automotive requirements [4]. Scalable service-Oriented Middleware over IP (SOME/IP) is a Remote Procedure Call (RPC) mechanism [5] specialized for the usage in the automotive systems. It consists of three modules: SOME/IP, SOME/IP Service Discovery (SD) and SOME/IP Transformer. SOME/IP fundamental module is managing the serialization and deserialization of transmitting data, SD module enables the connection establishment and service discovery procedure and the Transformer module specifies automotive/embedded data serialization [4].

There are multiple protocols which can be used for in-vehicle cross-domain communication such as DDS, HTTP, MQTT, web sockets, etc. Besides the abovementioned fact that the SOME/IP is created for the automotive industry there are several functional benefits that made it our choice for such use-case. First, the mandatory configuration of the communication over SOME/IP enables somewhat more deterministic behavior in contrary to another protocols and mechanisms used to implement SOA in the web and cloud computing such as the HTTP for example. Additional benefit is that SOME/IP provides multiple types of communication. Comparing to the HTTP, which allows only request-response communication initiated from clients, SOME/IP provides both request-response and publish-subscribe approaches. Furthermore, SOME/IP does not require communication establishment for each data exchange, but only for initial client-service connection and it can rely on both, TCP and UDP protocols in contrary to the communication implementing Representational State Transfer (REST) principles.

Although having different architectures, diverse software platforms use the SOME/IP communication stack based on the similar concept as depicted in Fig. 1.

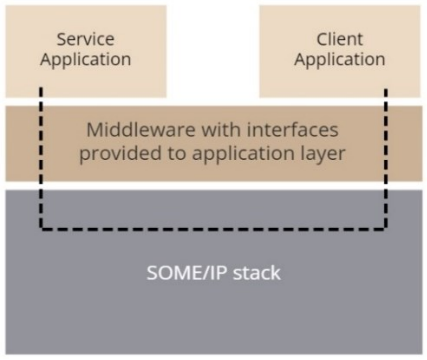

Fig. 1. Concept of SOME/IP implementation in automotive platforms

Usually, the middleware which provides the applications with the particular interfaces based on the determined configuration parameters is implemented by the standard. Depending on whether it is event, method or field that is defined by the configuration the data exchange would be performed by publishing the information to subscribed client when a logic on the service side determines so, client requesting the execution of a method on service side and getting the response if needed and getting, setting or notifying about the changed state of a field, i.e. attribute on the service side respectively.

Services in automotive SOA need to meet strict requirements regarding the service discovery and startup latency time [6]. Authors in [3] even propose dividing and isolating secured and exposed subnetworks in order to accomplish more reliability, since the service discovery mechanisms cannot guarantee that the service will be provided at the needed time. However, this aspect will not be examined in this paper, but another one instead – how SOA is implemented within available architectures and how it can be used for inter-communication between ADAS and IVI domains.

B. IPC Standards in IVI Domain

Modern vehicles are currently competing to meet the requirements driven by the consumer technology, especially in the infotainment domain. Inside the vehicle, the passengers expect the experience they have when using everyday portable devices, such as tablets and mobile phones. They are used to being able to install and use various types of applications developed by different vendors. In order to meet these requests, it is necessary to utilize the globally accepted standards for building scalable and portable platforms.

1) GENIVI approach

The former GENIVI (currently COVESA) alliance drives the development of open standards and technologies used in automotive systems. Their goal is to address the challenges which the in-vehicle infotainment components are facing when reaching to the outside world (cloud services, other vehicles, etc.) and communicating with other in-vehicle infotainment components as well. They offer the CommonAPI [8] – an inter-process communication middleware based on the FRANCA framework, which provides service-oriented mechanisms. It is designed to split the applications implementation apart from the communication mechanisms used between the implemented application components.

Since the only purpose of this middleware is to provide interfaces between lower (platform services and protocols) and upper (applications) layers, its implementation is generated mainly from the FRANCA Interface Definition Language (FIDL) to make its utilization easier. Applying the specified interface definition language – FIDL, it enables flexible deployment models. This way, the dynamic behavior of an API is specified by defining client/server interaction interfaces, states and transitions between them [7]. The communication itself is performed by using the generated Stub and Proxy classes relying on the CommonAPI middleware within the Service and Client applications respectively. This way, the concept from Fig. 1 is kept since the entire CommonAPI stack including the Stub and Proxy provides applications with interfaces for usage of the SOME/IP mechanisms.

Additionally, COVESA semantically differentiates between the two realms: Common-API Core, which does not depend on the communication protocol itself, and CommonAPI Binding which is protocol-specific [8]. Currently, the CommonAPI support two RPCs, D-Bus and the SOME/IP. In order to set deployment parameters for chosen protocol, the FRANCA Deployment (FDEPL) files are used along with the FIDL.

2) Android approach

Android is an open-source operating system mainly utilized for mobile devices. It enables deployment on wide range of hardware platforms and supports third-party applications development [9]. Currently, the automotive industry is facing a similar requirement for the possibility for third-party application development and utilization, therefore the automotive community is more interested in the Android platform [10].

Android platform has the mechanisms for feasible handling of the Inter-Process Communication (IPC) via its proprietary interface definition language called AIDL. It provides a programming interface utilized by both the client and the service using the IPC to communicate with each other [11]. Although AIDL has similar functionality as other IDLs, its utilization does not rely on the same paradigm as it is the case with the FIDL and COVESA's Common-API service-client communication model. Additionally, the SOME/IP had not been supported in Android until vsomeip version 3 was released. The possible correlation between CommonAPI and Android and more details about the AIDL paradigm and its communication mapping to other mechanisms will be addressed in the Section 4.

C. IPC standards in ADAS domain

The previously described standards are used for the implementation of the application for the in-vehicle infotainment part of the automotive system. On the other hand, ADAS domain is faced with the challenges driven by different requirements, as it considers safety-critical algorithms and modules. Nevertheless, ADAS domain implies the integration of functionalities provided by machine vision and sensor fusion. Lots of these algorithms are used in consumer technologies, i.e., in the IVI domain also. Therefore, the benefit of exchanging resources between two mentioned domains is obvious, since there is a set of functionalities they share. A standard that has become a convention for the implementation of ADAS domain functionalities is AUTOSAR.

The AUTOSAR standard considers both safety host and performance host implementations. Safety hosts are referring to ECU's cores with safety and security control features specialized for the automotive industry. Classic AUTOSAR platform is designed for the fully deterministic, deeply embedded standardization of safety hosts. Furthermore, the Adaptive AUTOSAR platform is offering more flexibility by addressing operability and communication mechanisms more suitable for high-performance computing devices called performance hosts. Since the performance host resources and algorithms complexity are more similar to the ones in the IVI domain, we will focus on the sharing resources and features of the Adaptive AUTOSAR platform.

User applications are running on the top level, right on top the AUTOSAR Runtime Environment for Adaptive Applications (ARA). The main component of ARA is ara::com, a middleware controlling the communication within a system. It provides the interfaces to the user applications which allow data exchange with both local and remote applications and ARA services [12].

Equivalent to the FIDL, ara::com interfaces in ARA-API are defined by the ARXML. Interfaces are provided to applications with the exact same purpose as it is the case with CommonAPI, to decouple the applications development from the communication mechanism. It is done by utilizing two artifacts - Skeleton and Proxy which implement the SOA paradigm, i.e., the service-client communication, likewise it is the case with the Stub and Proxy in CommonAPI. Skeleton represents the generated instance which provides service calls functionalities. On the other hand, Proxy is a generated instance which provides the client calls functionalities.

III. CURRENT CROSS-DOMAIN RESOURCE SHARING SOLUTIONS

Most of the research in the field of interconnecting different automotive domains focuses on the modelling and implementation of multi-ECU system using a single standard. Since meeting the safety and latency requirements for ADAS is critical, it dictates the approach to use Adaptive AUTOSAR for both the ADAS and IVI realm. This way, for the sake of connectivity between different domains, neither the CommonAPI nor Android are used, although they are a better fit for IVI domain, since the development is forced to a single standard approach which must fit ADAS requirements. The authors then try to deal with the shortcomings that the AUTOSAR standard provides in terms of UI as an important aspect of in-vehicle infotainment [13]. Authors in [14] presented challenges of modelling ADAS components for camera resource sharing. However, it is needed to perform further research on the most suitable communication channel for the transmission of sensing data and data streams along with the research on the most suitable communication mechanisms by considering the entire, end-to-end communication context for such resources sharing between domains in automotive, the SOME/IP is not the most effective solution for such use-cases. Furthermore, taking into consideration the variety of operating systems on the other side, such implementation cannot be taken "as is".

COVESA alliance recognized this challenge and tried to attain the adaptation between Adaptive AUTOSAR and the CommonAPI by creating FARACON generator [15]. This generator is used to translate the interface definition files from one standard to another. This can be considered as a first step towards the mapping of the features between standards. However, it does not solve the cross-domain heterogeneity issue, which is somewhat more complex.

There are not many papers that provide the actual proposition for interconnection between ADAS and IVI domains utilizing different standards. i.e., following AUTOSAR on ADAS and CommonAPI or Android on IVI side. The existing solutions have recognized the need for such binding, but are also typically reduced to simple utilization of socket-based communication with no actual research background on the available protocols and state-of-the-art SOA principles [16]. Additionally, the inter-process communication paradigms diversity when considering the various platforms standards is not actually covered even in papers which provide the extensive solution for heterogenous in-vehicle environments [17]. Hence, there is no comprehensive project dealing with all aspects of this topic.

Since this topic is substantive and our project is still in the development, some of the challenges will not be covered by this paper but will be addressed in future work instead.

IV. Proposed Solution

In this section, we will discuss the possible approaches that allow remote procedure calls and exchange of data between ADAS and IVI domains of the vehicle. Our goal is to provide the connectivity, without compromising the functionality of the IVI domain offered by the CommonAPI or Android, or the safety features provided by the AUTOSAR in ADAS domain. We will design our solution using the service-oriented architecture principles, which fit perfectly into the scenarios we want to support. Our focus is on allowing IVI domain applications to use raw measurement data from ADAS sensors, as well as the results of some of the algorithms that run on the ADAS side. The opposite direction of integration is not possible, due to potential safety issues.

There are several examples of use-cases where the proposed cross-domain inter-connection can be beneficial. For example, inputs from cabin camera which is commonly used for driver monitoring on ADAS side can be shared for video calls and other applications using camera in IVI domain. This way, the cost of providing redundant hardware components would be avoided. On the other hand, data from sensors monitoring tire pressure, engine temperature and other crucial components of the vehicle could be easily transferred and handled by the applications in the IVI domain. These applications could then not only inform the driver, but also provide the better user experience by searching for the recommendations and manuals on the Internet, or help by finding the route to the nearest mechanic service. The results of the data processing algorithms such as traffic sign detection and recognition or driver drowsiness monitoring could also be used by the IVI domain applications, to propose rest stops, provide tourist information, etc.

To connect the components of the two domains in the proposed solution, Ethernet-based communication will be used. Recently, Ethernet has taken on the role of the vehicle communication backbone because of its bandwidth, scalability, flexibility and prevalence. In all of the aforementioned terms, Ethernet is generally superior to other in-vehicle buses, which are designed and optimized to fit only specific use-cases. For example, CAN provides the reliability which Ethernet cannot achieve because of the different transmission media access strategies. On the other hand, CAN is the automotive specific technology which means that Android, as a standard that was not created solely for the automotive industry, does not support CAN bus module natively. Similarly, other in-vehicle buses are created to meet

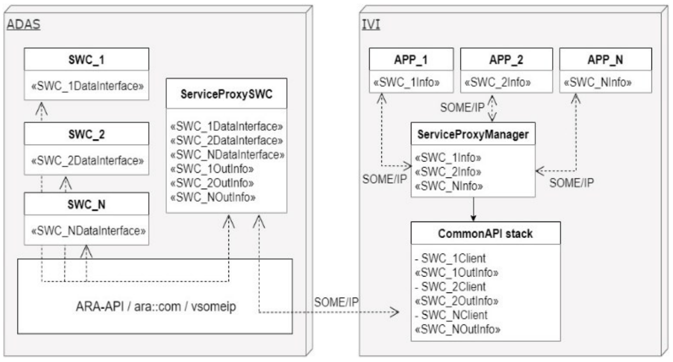

Fig. 2. Centralized interconnection approach with POSIX OS on IVI side

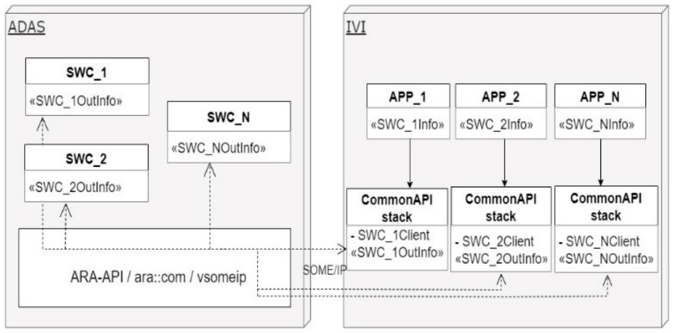

Fig. 3. Distributed interconnection approach with POSIX OS on IVI side

the requirements of automotive signal-based communication, where priority is the price and the determinism of the communication mechanism, not the bandwidth itself. On the other hand, Ethernet is widely used technology which makes it suitable for interconnection of different domains. To exchange data between the domains, we will use SOME/IP, from the reasons already discussed in Section 2, and it can be used over the Ethernet network.

Typically, IVI solutions can either run on Linux operating system and use CommonAPI mechanisms for the inter-process communication, or they can be Android-based. For both of these cases, we will propose the solution architectures in the following sections.

Since Adaptive AUTOSAR and CommonAPI both implement the SOME/IP communication interfaces, this is the easiest way to establish the communication between the two domains in the SOA manner. The ADAS side is implemented by following Adaptive AUTOSAR standard and the IVI domain uses the CommonAPI middleware running on the native operating system such as Linux. This scenario is depicted in Fig. 2 and Fig. 3. Communication in Adaptive AUTOSAR is handled by ara::com which natively supports vsomeip as a library that implements SOME/IP standard. The same vsomeip implementation is utilized in CommonAPI SOME/IP stack. This means that serialization and deserialization of data shall be handled in the same way, so both sides will be able to interpret data properly.

Information from components on ADAS side are initially given to the Service Proxy SWC via SOME/IP implemented within the ara::com module. This data is furtherly forwarded to the corresponding CommonAPI clients grouped together on the IVI side in a single Service Proxy Manager instance (Fig. 2). Such inter-domain transfer is performed over SOME/IP on demand of IVI applications or when the event/change is captured.

The type of communication between the ADAS and the IVI does not necessarily have to match the communication between the ADAS Service Proxy and other SWCs which means that IVI applications can request the data through the Service Proxy Manager instance over method mechanism, but the sharing information on the ADAS side can be sent to the Service Proxy SWC from the actual service component as an event for example.

Another approach is to implement separate services for each CommonAPI client (Fig. 3) so the Service Proxy components on both sides are unneeded and the information will be provided from ADAS ara::com services to the IVI CommonAPI clients included in particular application. The first approach is easier to scale and can be used with the variable number of application instances. Also, it can be favorable from the safety perspective since it can contain mechanisms to protect from other SWCs from being jeopardize by IVI applications. On the other hand, the second is superior in terms of reliability, because there is no single central node which distributes the data between the applications. This way, the malfunction of one service does not affect the operability of others. Furthermore, the monolithic design is harder to maintain, as even minor changes require the entire integration cycle. The speed of access to information is also one of the factors that is on the side of the distributed approach.

As already said, Android has recently become the operating system of choice for IVI applications, as most of the users are familiar with it and it is available on a very large variety of hardware. The interconnection of the ADAS domain with the IVI domain running on the Android platform is a bit more challenging for the implementation. Namely, Android itself does not have mechanisms to implement SOME/IP client which can communicate with ADAS side. Therefore, the CommonAPI must also be used in this scenario in the exact same way it was the case when non-Android OS was examined, as it is presented in Fig. 4 and Fig. 5.

The CommonAPI clients are included within an Android native service and provided information can be transferred to both, custom applications and HAL modules over AIDL. The entire CommonAPI stack can be built within an AOSP (Android Open Source Project maintained by Google) with the soong build system. Still, the vsomeip itself has some dependencies, such as boost library, which can cause issues while building within the AOSP. Further options are to build CommonAPI client beyond the AOSP, with the Native Development Kit – NDK, or even to use another implementation of SOME/IP standard instead of vsomeip, which would eliminate the dependencies such as the afore-mentioned boost library. Nevertheless, CommonAPI clients must be included in Android services so the data from ADAS can be provided to applications or other services in IVI domain.

Additionally, the mapping of SOME/IP service-client communication paradigm from CommonAPI/AUTOSAR to Android represents a challenge. Namely, AIDL files used for

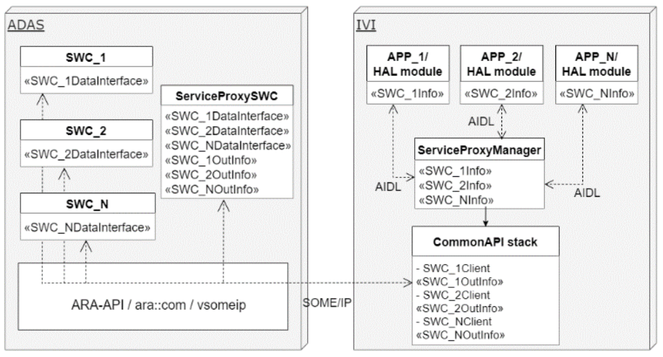

Fig. 4. Centralized interconnection approach with Android on IVI side

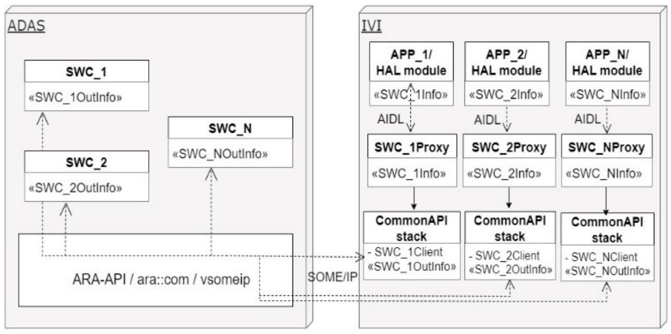

Fig. 5. Distributed interconnection approach with Android on IVI side

interface generation provide the inter-process communication by marshaling the object instances through the binder. This is not suitable for the event-triggered traffic. Event-triggered communication from service to clients within a SOA is performed in a way that the client itself is only subscribed to the events from service. This specific case cannot be covered by using regular AIDL, because AIDL always assumes that the communication is initiated from the client side

Our approach was to incorporate the receiving (client) side for broadcasts and events in the Android native service, and further distribute this information to the interested applications. The easiest way to achieve this is to set properties based on the information received by the Android native service. The interested applications can then read that particular property. This approach has a big limitation since the data can only be used to transfer flags and states since properties do not exist to be used as IPC mechanisms.

We went for the another, a slightly more demanding way for implementation. It assumes the creation of a helper AIDL, which will pass the interface object as a parameter from the applications to Android native service, in order to enable the Android native service to react on event-trigger signals from CommonAPI by invoking methods from the passed object like it is a client to the application. Furthermore, in order to avoid forming a list of registered applications which methods will be invoked when event-triggers we created additional helper service in Java with which it will be communicated via that helper AIDL and which will furtherly provide Intents to the applications. Additionally, it is even possible to have stand-alone Java service which will use the CommonAPI via Java Native Interface – JNI. JNI is necessary in this scenario to enable inter-operability between Java and C/C++ code, since the COVESA provides CommonAPI middleware in C++

programming language.

V. Conclusion and Further Work Directions

This paper presented both, the theoretical and the practical aspects of proposal for service-oriented communication between ADAS and IVI domains. Background and motivation for such binding are provided, along with the key challenges and limitations as it is summarized in Table 1. Several approaches were elaborated in order to satisfy system heterogeneity. Additionally, the beneficial use-cases are discussed in order to emphasize the value of bonding itself.

TABLE I
Major challenges and limitations

Challenges	Limitations
SOA paradigm mapping	Implementing broadcast/events with AIDL principles
Centralized or distributed approach	Prioritization, robustness, bandwidth
Data transfer channel	Performance evaluation
Safety	Enable safety solution for Android
Generation of inter-communication	Verification and tool qualification

In our future work, we will focus on the evaluation of the latency, bandwidth and robustness in order to present comprehensive comparison of the centralized service proxy manager approach with the distributed approach, to determine the optimal design.

The performance of data transfer channel shall be furtherly examined too by considering the Audio Video Bridging (AVB) and other mechanisms for big data integration. It is needed to determine the exact use-cases where the data shall be transferred only within SOME/IP request/response, and where it is more suitable to open additional channel for data transfer. Several aspects regarding data size and safety shall be analyzed in order to define the optimal approach.

Safety requirements are maybe the most complex of all challenges that we plan to address. Safety analysis implies the detail examination on the system level too. It is not enough only to implement mechanisms for Android native service to control which applications can use it based on the given permissions and to properly handle dead listeners and multiple registrations which is done by now. Hazard analysis on the system level involves hardware and OS safety competence and certain communication determinism (Time-Triggered Ethernet or Time-Sensitive Networking). Android itself currently cannot have any Safety integrity level but QM [18]. From that reason, it is mandatory to involve the hypervisor if the communication must be initiated from the Android [19].

The final challenge will be to automate the entire process of providing resources from ADAS to IVI. This means that our goal will be to generate the translation between ARXML, FIDL and AIDL, as well as the generation of Android service along with the code that is responsible for providing resources from service on ADAS side to the IVI realm.

Acknowledgment

This research (paper) has been supported by the Ministry of Education, Science and Technological Development through project no. 451-03-68/2022-14/ 200156 “Innovative scientific and artistic research from the FTS (activity) domain”.

References

[1] P. Bajaj, M. Khanapurkar, “Automotive networks based intra-vehicular communication applications. New Advances in Vehicular Technology and Automotive Engineering”, pp. 207-230, (2012).
[2] B. Glas, J. Guajardo, H. Hacioglu, M. Ihle, K. Wehefritz, A. Yavuz, “Signal-based automotive communication security and its interplay with safety requirements.”, In Proceedings of Embedded Security in Cars Conference, 2012
[3] M. Bellanger, E. Marmounier, E, “Service Oriented Architecture: impacts and challenges of an architecture paradigm change”, In 10th European Congress on Embedded Real Time Software and Systems, (2020).
[4] G. L. Gopu, K. V. Kavitha, J. Joy, “Service oriented architecture based connectivity of automotive ecus”, In 2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT), pp. 1-4, (2016).
[5] “Example for a Serialization Protocol (SOME/IP)”, [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-1/AUTOSAR_TR_SomeIpExample.pdf”, last accessed 2022/10/1.
[6] J. R. Seyler, T. Streichert, M. Glaß, N. Navet, J. Teich, “Formal analysis of the startup delay of SOME/IP service discovery”, In 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 49-54, (2015).
[7] "Welcome to FRANCA!” [Online]. Available: https://github.com/franca/franca, last accessed 2022/10/1.
[8] “CommonAPICppUserGuide”, [Online]. Available: https://usermanual.wiki/Document/CommonAPICppUserGuide.1126244679/html, last accessed 2022/10/1.
[9] G. Macario, M. Torchiano, M. Violante, M., “An in-vehicle infotainment software architecture based on google android”, In 2009 IEEE International Symposium on Industrial Embedded Systems, pp. 257-260, (2009).
[10] N. Pajic, M. Bjelica, “Integrating Android to Next Generation Vehicles”, In 2018 Zooming Innovation in Consumer Technologies Conference (ZINC), pp. 152-155, (2018).
[11] “Android Interface Definition Language(AIDL)” [Online]. Available: https://developer.android.com/guide/components/aidl, last accessed 2022/10/1.
[12] S. Fürst, M. Bechter, “AUTOSAR for connected and autonomous vehicles: The AUTOSAR adaptive platform”, In 2016 46th annual IEEE/IFIP international conference on Dependable Systems and Networks Workshop (DSN-W), pp. 215-217, (2016).
[13] S. Aust, ”Paving the way for connected cars with adaptive AUTOSAR and AGL”, In 2018 IEEE 43rd Conference on Local Computer Networks Workshops (LCN Workshops), pp. 53-58, (2018).
[14] M. Kotur, N. Lukić, M. Krunić, G. Velikić, “One solution of camera service in AUTOSAR ADAPTIVE environment”, In 2020 IEEE 10th International Conference on Consumer Electronics, pp. 1-5, 2020.
[15] “Franca/ ARA:COM Interoperability”, [Online]. Available: https://at.projects.genivi.org/wiki/download/attachments/16026116/GENIVI%20Franca-ARA-COM-tech-brief-20181219.pdf, last accessed 2022/10/1.
[16] K. Omerovic, J. Janjatovic, M. Milosevic, T. Maruna, “Supporting sensor fusion in next generation android In-Vehicle infotainment units”, In 2016 IEEE 6th International Conference on Consumer Electronics-Berlin (ICCE-Berlin), pp. 187-189, (2016).
[17] M. Milosevic, M. Z. Bjelica, T. Maruna, N. Teslic, “Software platform for heterogeneous in-vehicle environments”, In IEEE Transactions on Consumer Electronics, pp. 213-221, (2018).
[18] L. Perneel, H. Fayyad-Kazan, M. Timmerman, “Can Android be used for real-time purposes?”, In 2012 International Conference on Computer Systems and Industrial Informatics, pp. 1-6, (2012).
[19] M. Bjelica, Z. Lukac, “Central vehicle computer design: software taking over”, IEEE Consumer Electronics Magazine, 8(6), 84-90, 2019.