Proc. Of the 5th International E-Conference on Advances in Engineering, Technology and Management - ICETM 2021

ISBN: 978-1-63248-192-4 DOI: 10.15224/ 978-1-63248-192-4-11

The Need for New Software Architectures for Next-generation Vehicles

Johannes Büttner, Cristina Alonso-Villa, Pere Bohigas Boladeras, Markus Kucera and Thomas Waas
Faculty of Computer Science and Mathematics
Regensburg University of Applied Sciences
Regensburg, Germany

***Abstract*—Assuming that future connected cars with automated driving functions will require even more computing power and communication bandwidth, the current network infrastructure as well as the existing individualized control units are not a profitable option for such vehicles. In addition, changing user expectations demand flexible architectural patterns and upgradeability of software components without the need to visit the workshop. However, the current statically developed and configured ECU architecture does not offer any practicable possibilities for this. For these reasons, the research for a new dynamic and flexible architecture is necessary. This new type of system architecture is expected to meet future requirements in terms of space, cost, performance, energy efficiency and number of required computing units in the vehicle, which will arise as a result of the implementation/inclusion of new automated driving functionalities, and due to the changes in user expectations. Solutions to this issue can be found in the field of enterprise IT (cluster computing), in which technologies such as Ethernet, container-based virtualization and flexible software architectures have proven themselves to be very efficient for years. Relevant infrastructures, for example from cloud computing providers, have commonly been used in high-performance or high-availability applications. Thus, in the research project A^{3}F has been investigated which of these concepts and methods can be applied to modern vehicle system architectures. One of the main goals is to assess the synergy potential of the two sectors, information technology and automotive industry, which to date have very different orientations. However, this synergy is expected to grow strongly in the course of the developments mentioned above.**

In the following pages, the necessary changes related to hardware and software will be discussed briefly und will be compared to concepts and possible solutions from the IT world.

I. Introduction

Nowadays, there's a computer – in the broadest sense of the word – almost everywhere, in every gadget that automates some aspect of daily life. Cars and other vehicles are no exception to this. Since the 1970s, the amount of existing connections in vehicles has steadily increased until a decade later the first field-buses were introduced, which allowed for a decrease in the number of cables needed and a rise in the number of control units [1]. As the number of electronic components for driver assistance functions increased, so did the complexity of the architecture, which has since become increasingly difficult to control.

In addition, the latest technological developments and trends are challenging the current system architectures of vehicles: *Over-The-Air updates* (OTA-Updates), *self-driving cars* and the so-called *connected car* [2].

Concerning *OTA-Updates*, one first needs the ability to deploy updates to the existing software by means of a backend that provides the updates and a network connection that can load these updates, but even more importantly, the control units themselves must be upgradeable, i.e., they must provide a way to install new software in an automated manner. Subsequently, it is necessary to consider the granularity with which the software is to be updated and how modular and dependent the individual systems are on each other.

Self-driving cars require massive computing power whilst combining a large number of input signals from sensors all over the vehicle. To achieve this, control units and sensors are typically connected via a bus. However, the bandwidth of current vehicles will presumably be insufficient to process the necessary amount of sensor signals.

Connected cars (V2V, V2X, ...), on the other hand, require a high network bandwidth both on the internal network and to the outside world. Again, this requires increasing the network bandwidth of current vehicles.

As the demands on the hardware side continue to increase, the software side should also be reconsidered.

A good way to illustrate the escalation of the amount of software present in vehicles is to observe the increase in the percentage of production costs derived from software, from 20% in 2000 to 37% in 2010. Thus, as Doherty et al. [3] point out, cars became more dependent on software, which lead to the implementation of methodologies characteristic of the IT world.

This has produced an increase in the complexity of the systems and thus an increment in the possibility of software-related errors. Therefore new solutions should be focused in providing a fail-safe stability.

One way to address this problem and meet the requirements of future vehicles is through exploring the IT sector: modularity, scalability, and fail-safety are key approaches to service delivery.

Precisely this is the goal of the project Fail-Safe Architecture for Self-Driving vehicles (A^{3}F): to test the suitability of these technologies from the IT branch to solve some of the most impelling pitfalls in the development of a fail-safe software platform for autonomous-driving vehicles.

ISBN: 978-1-63248-192-4 DOI: 10.15224/ 978-1-63248-192-4-11

The A³F Project is a collaboration between Continental AG and the University of Applied Sciences of Regensburg (OTH Regensburg). Whereas the former does research and executes analysis on hardware and network communication, the latter focuses on software architechtures and finding solutions on an application level. The present paper has been written in the framework of the OTH Regensburg and thus deals mainly with issues pertaining software.

II. The Challenges of the Upcoming Vehicle Architecture

As discussed above, the main design considerations of an architectural change amount to the following:

1) Significantly more data must be exchanged between control units and sensors, hence the network bandwidth must be increased.
2) It must be possible to reinstall or update the software on the individual control units during operation.
3) The software on the control units must be fail-safe.

To increase the network bandwidth between the control units and sensors, one could simply install more cables. However, this also increases the weight of the vehicle, which is likewise undesirable. Thus, this type of architecture does not scale well.

Another approach is to group the control units more closely together and integrate them on a few high-performance units. This means that a large amount of the signals that would otherwise be sent over the network are already processed within the control unit. This approach is already being practiced and the trend has become visible in recent years, for example in the so-called domain controllers or domain architecture.

The next item appears to be more challenging, since previous systems are configured in a very static fashion. There are not yet any suitable ways of deploying updates “over the air”; instead, direct physical access to the control units is required. In addition, the interaction of the individual control units – although each unit has its own software – is developed as an overall system and as such is either updateable as a whole or not at all. For example, introducing a new feature to the vehicle and thus a new software component requires revising the configuration of the entire system in terms of resource consumption, performance and safety. This makes it extremely impractical to not only deploy updates over the air, but even install them in the first place.

On top of that, the software must also be fail-safe, which is the biggest constraint on the flexibility of the system. As already mentioned, the system as a whole is tested for safety before it is rolled out, so it is impossible to replace or update individual components without performing the test on the whole system again. However, this does not mean that the software should not be developed in a modular fashion. It would be conceivable, for example, to test various combinations of software components already at the OEM, and if these are successful, to roll them out to the vehicles over the air. Each update would therefore need to be tested beforehand in the overall system at the OEM. Once this is the case, however, safety is also guaranteed for the end customer.

III. Software Architecture

Software architectures provide the structure for the development of software and allow both abstract and concrete definition of modularization, dependencies, and interfaces. They also have great influence on scalability and extensibility of software.

Both the IT industry and the automotive industry anticipate the need to divide software into defined modules and subsystems in order to properly manage complex systems. In this spirit, it is very likely that the current OEM-supplier model will evolve in such a way that componies previously knows as hardware vendors will supply software modules in the future. This means that the OEM must provide a platform that allows the integration of such modules. The software architecture of this platform must in turn be suitable for this. This approach shows similarities to the service-oriented architecture that has been established in IT for years. In more recent years, another interesting trend has emerged, namely microservices.

1) Service-Oriented Architecture (SOA): SOA is a concept that allows for a great flexibility and interpretation. For this reason, there are many definitions that include different components and different views. For instance, Josuttis [4] defines SOA as a paradigm that diminishes the complexity of a company by dividing its technical structure. Driven by the use of distributed systems based on the fact that in a company exist a great amount of areas, each with their own systems. In addition, these systems can be distinguished by their own specifications and by the differences in hardware and software.

Erl [5] on his behalf considers SOA as an “open, agile, expandable, unified, componentized architecture consisting in autonomous, [...] capable of quality-of-service, possibly reusable services [...]. SOA can be considered an abstraction of the business logic and technology that leads to a loose link between the domains.” For Erl, SOA depends on the companies that implement it. In A³F we have worked with Erls interpretation of SOA, as it is the more productive perspective.

A detailed look at the definition allows us to list a number of characteristics:

Main concepts of SOA:

- **Modularity**
  One of the central concepts of SOA. The systems have to be build with a modular structure, in order to be as independent as possible and thus make the deployment and testing easier. The modularity is achieved through the implementation of services.
- **Loose coupling**
  A requisite that allows the integration of both individual and standard software acquired from different manufacturers. The goal is to improve the flexibility, the error tolerance and the scalability. By reducing the interdependency of the different parts of the system or between the services, chain failures in the system can be avoided.
- **High interoperation**
  A quality that makes it possible to combine a number

ISBN: 978-1-63248-192-4 DOI: 10.15224/ 978-1-63248-192-4-11

of heterogeneous systems. Some examples of this are interfaces and the Enterprise Service Bus (ESB).

Another important part are the services. Richter defines them as "clearly defined services that can be employed as elements of a bigger or multiple processing sequences."

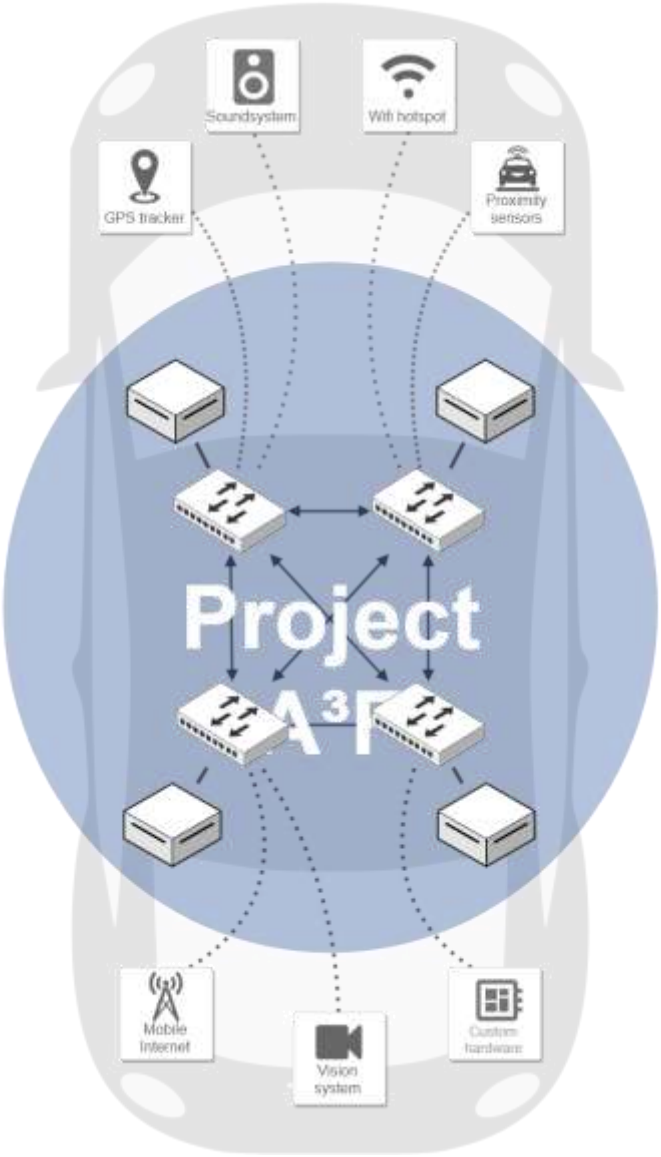

Figure 1. New architecture based on a computer cluster with redundant network connection, investigated within the project A^{3}F.

Components:

- **Basic services**
  Represent the minimal service unit. A basic service cannot be divided in smaller ones.
- **Composed services**
  Composed by the arrangement of a number of basic services. The build a higher level of abstraction and can carry out more task in the scheduled order.
- **Process services**
  A combination of composed services. They depict a whole business process or long-term workflow that maintains its state after many calls.
- **Interfaces and contracts**
  Interfaces guarantee the possibility to build independent services and systems. Through the definition of interfaces it is possible to communicate the function of the service without disclosing the details of the implementation.
  A contract irs a detailed specification of a service between an specific provider and an specific user (Josuttis, 2008). It contains the already defined interface plus details about the resources that need to be submitted.
- **Infrastructure**
  Also referred to as ESB, the infrastructure describes a backbone that carries out task along the whole system. In order to fulfill this role, it offers a series of methods and technologies, such as the connection among other services and components, conversion of data, routing, security, fail-safety, management of services, monitoring and logging.

The implementation of the SOA highly depends on the characteristics, the individuality of the company and the goals set for the architecture. For this reason, SOA provides descriptions of a series of criteria that need to be taken into account in order to achieve the desired flexible and expandable structure.

2) Microservices: Microservices can also be conceived manifold. As Fowler [6] explains, they represent an approach to the development of single applications as a set of services that run in single processes, and communicate through lightweight mechanisms. Fowler also says, that Microservices can be defined based on their characteristics. Wolff [7] employs also in his definition the characteristics of the Microservices, and for this reason calls this approach "concept of modularization".

Main concepts of Microservices:

- Modularization and functionality
- Independence
- Lightweight communication
- Definition of interfaces

Characteristics:

- Scalability
- Deployment, Replaceability and Scalability
- Responsiveness
- Reusability

In contrast to SOA, the microservices perspective does not aim to completely change the structure of the company, but to achieve a system that is expandable and maintainable. The business logic needs to be partitioned so that a service can issue a special request to other services with as few dependencies as possible. The microservices approach is additionally accompanied by concepts that move the operation of software closer to the development of the software. This particular aspect does not seem to get as much attention in the SOA approach.

As far as the development of future architectures in vehicles is concerned, characteristics from both approaches are promis-

ISBN: 978-1-63248-192-4 DOI: 10.15224/ 978-1-63248-192-4-11

ing. Due to the modularity and independence of the software in SOA, this approach could be transferred relatively easily to the automotive world, while preserving the previous OEM-supplier model. However, due to the importance of failure safety, topics from the microservices approach also come into play, as operations are treated as a primary problem here.

The concrete design of such an architecture as well as further overlaps from the IT area will be investigated in future work within the A^{3}F project.

IV. Conclusions

In this paper, we have analyzed the shortcomings of current automotive system architectures and identified key future challenges. However, with the help of modern IT principles, the automotive industry can also develop further here and adopt promising approaches. Service-oriented architecture is a groundbreaking concept here, which is presumably relatively easy to implement due to the OEM-vendor structure.

Greater challenges exist on the part of the technical implementation of the proposed cluster, particularly with regard to the compatibility of flexibility and fail-safety. Further investigations will have to follow in this regard as part of the A^{3}F project.

Acknowledgments

The authors gratefully acknowledge the financial support of the Bavarian Ministry of Economic Affairs, Energy and Technology (Bayerisches Staatsministerium für Wirtschaft und Medien, Energie und Technologie [STMWI]), ot the funding program “Information and Communication Technology Bavaria (Informations- und Kommunikationstechnologien” as well as the support by project management organization VDI/VDE Innovation + Technik GmbH.

References

[1]H. Richter, “Elektronik und datenkommunikationim automobil,” *IfI Technical Report Series*, vol. Institut für Informatik, Technische Universität ClausthalJulius-Albert Str. 4, 38678 Clausthal-Zellerfeld, Germany, p. 15, IfI-09-05 May 2009, ISSN: 1860-8477. [Online]. Available: http : / / www. in . tu - clausthal . de / forschung / technical - reports/.

[2]R. Grave, “Autonomous driving – from fail-safe to fail-operational systems,” TechDay, Dec. 2015, [Online]. Available: https : / / d23rjziej2pu9i . cloudfront . net / wp - content / uploads / 2015 / 12 / 09163552 / Autonomous - Driving- From- Fail- Safe- to- Fail- Operational- Systems_ TechDay_December2015.pdf (visited on 06/17/2017).

[3]P. Doherty, P. Haslum, F. Heintz, T. Merz, P. Nyblom, T. Persson, and B. Wingman, “A distributed architecture for autonomous unmanned aerial vehicle experimentation,” in *Distributed Autonomous Robotic Systems 6*, R. Alami, R. Chatila, and H. Asama, Eds., Tokyo: Springer Japan, 2007, pp. 233–242, ISBN: 978-4-431-35869-5 978-4-431-35873-2. DOI: 10 . 1007 / 978 - 4 - 431 - 35873 - 2 _ 23. [Online]. Available: http://www.springerlink.com/index/10.1007/978-4-431-35873-2_23 (visited on 09/17/2019).

[4]N. M. Josuttis, *SOA in practice*, 1st ed. Beijing ; Sebastopol: O’Reilly, 2007, 324 pp., OCLC: ocm77796085, ISBN: 978-0-596-52955-0.

[5]T. Erl, *Service-oriented architecture: concepts, technology, and design*. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference, 2005, 760 pp., ISBN: 978-0-13-185858-9.

[6]M. Fowler and J. Lewis, “Microservices: Nur ein weiteres konzept in der softwarearchitektur oder mehr?” *Objekt Spektrum*, vol. SIGS DATACOM GmbH, no. 1, 2015.

[7]E. Wolff, *Microservices: Grundlagen flexibler Softwarearchitekturen*, 1., korrigierter Nachdruck. Heidelberg: dpunkt.verlag, 2016, 376 pp., OCLC: 915162854, ISBN: 978-3-86490-313-7 978-3-86491-841-4 978-3-86491-842-1 978-3-86491-843-8.