TrAF-Cloud State of the Art

Project TrAF-Cloud within Vinnova/FFI/EMK (Dnr 2018-05010)

Responsible authors Thomas Söderqvist, VGTT

Document status FINAL

Document date 2019-09-30

FFI Fordonsstrategisk Forskning och Innovation

VINNOVA Energimyndigheten TRAFIKVERKET

FKG VOLVO SCANIA VOLVO

Content

Authors

Name	Partner	Chapters/ Responsibilities
Thomas Söderqvist	VGTT	Quality Assurance
Brian Katumba	VGTT	Vehicle E/E Architecture, Development of On-Board Architecture over Time, Challenges
Tobias Sternvik	VGCS	Development of Off-Board Architecture over Time, Challenges
Amir Mohagheghzadeh	Combitech	API Ecosystem and Ecosystem Management
Philipp Leitner	Chalmers	Introduction, Cloud Computing, Challenges, Research Connection, Quality Assurance
Regina Hebig	Chalmers	Challenges, Research Connection, Quality Assurance
Niklas Mellegård	RISE	Challenges, Research Connection

Terminology

Term	Description

1. Introduction

Electronics and software systems play a fundamental role for a modern truck in order to operate effectively. With the transition of embedded systems from the traditionally isolated world towards interconnected devices and machines, and with the addition of smart analytics, we are experiencing a technological shift. Through the deployment of Internet of Things (IoT) traditional products become interconnected and thus heavily interwoven with other systems [Borgia, 2014]. This requires systems to be compliant with an environment following open standards. The border between an "embedded system", such as a truck, and internet-based applications is becoming fuzzy.

This also applies to all vehicles, including trucks, buses and construction equipment. Systems are becoming increasingly dependent on the interactions in a larger system context to be efficient and thus competitive. A prominent example of this development is the EU funded C-ROADS platform[1] which aims at coordinating deployment of cooperative services (C-ITS) enabled by vehicle-to-vehicle and -infrastructure communications. These and other connected services can enable more efficient and safer transport by contributing to and utilizing big data and the vast computational power enabled by the cloud. The vehicles then become part of a larger collaboration utilizing cloud services [Erl, 2013] for their operation. However, embedded onboard vehicle software is traditionally designed as isolated pieces of functionality and are typically seldom updated. To systematically leverage the benefits of being connected to a cloud infrastructure, it is necessary to fundamentally change the architecture of these onboard systems.

In this report we discuss the current state of the art within the context of the TrAF-Cloud project. We will provide an overview of both the current state of the art in industry with regards to electrical truck architectures, cloud connectivity, cloud services relevant to the automotive industry, and current API management, as well as the current state of academic research related to bringing cloud functions into automotive architectures. We also derive fundamental gaps in practice and literature, which will guide the project in the future.

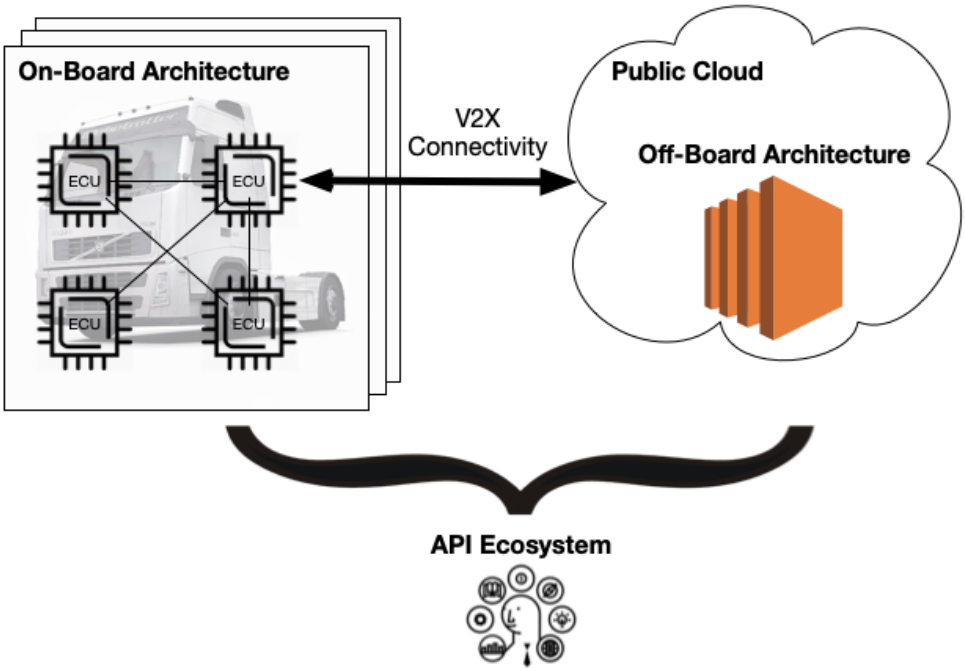
[1] https://www.c-roads.eu/platform.html

2. Software Architectures and the TrAF-Cloud Project

Following the definition of the IEEE, architecture is

> **'the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.'**

Following this definition, this document will attempt to illustrate the evolution of vehicle architectures up to and including the current state. Given the scope of the TrAF-Cloud project, a particular focus will be put on the interplay between the on-board architecture (i.e., the hard- and software running on vehicles) and the off-board architecture (i.e., supporting services running in a data center or in the cloud). Further, we will illustrate the current state of research and practice around the idea of API ecosystem management.



2.1 Vehicle E/E Architecture - Definition and Views

The term "Vehicle Electrical and Electronic Architecture" (E/E architecture) in the automotive industry is very wide and it has different meanings depending on who you ask and in what context you use it. In this project the IEEE definition of an architecture is used, i.e. "Architecture is 'the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution." [IEEE Std.610.12, I.S. Board. 1990]. Following the IEEE definition, The E/E architecture therefore defines the breakdown of the system into subsystems, the interfaces between these subsystems, the functional partitioning and the physical layout within the vehicle and their evolution over time. It also incorporates the overall design of the energy management, fault tolerance, cyber security, interface specifications, communication standards, diagnostics and rules and guidelines for designing the subsystems. From an industrial point of view, E/E architecture would be the system composed of function/application software architecture, platform software architecture, and electrical/ electronic hardware architecture. The function/ application software architecture addresses issues pertaining to application logic, application interfaces

etc. while the platform software represent the infrastructural parts of the system, like operating systems, middleware etc. The electrical/electronic hardware architecture address the fundamental organization of vehicle electrical and electronic components, such as ECUs, sensors, actuators, wiring, power distribution, onboard and wireless communication etc.

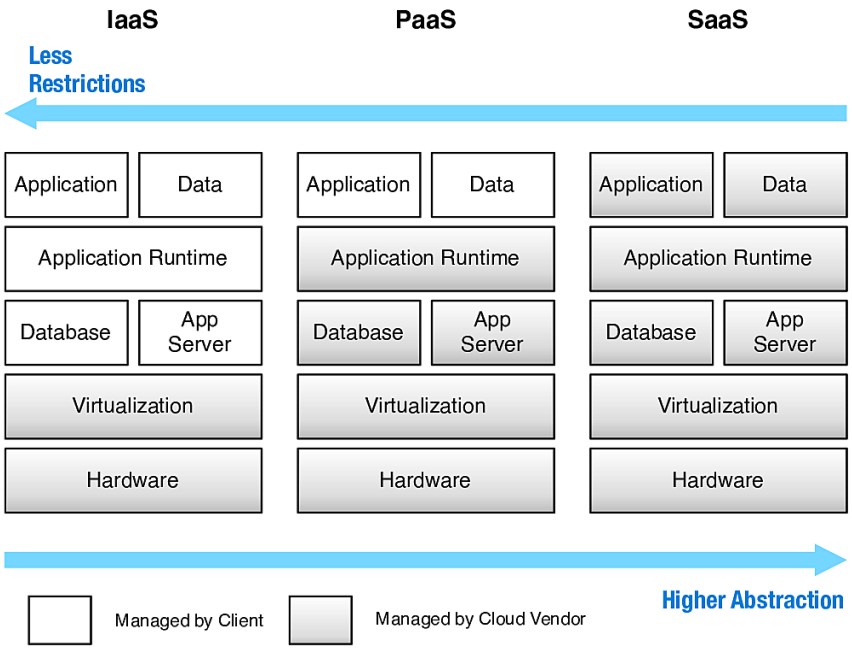
From a process perspective, Vehicle E/E architecture is also about the guiding rules for system design and integration, for how to build E/E systems that fulfill the requested functions and how to take care of current and future needs. Therefore, E/E architecture is the foundation of electrical, electronic and software system which, in turn, is the most important enabler of new functionality in the vehicle. In principle, for whatever new end user function in the vehicle that is devised, the electrical and electronic system is a key element of such an end-user function. In short, E/E architectures are of utmost important, but still complex to develop and deploy.

2.2 Vehicle E/E Architecture - Design

The design of an E/E architecture is a challenging job since it involves different departments, multiple vehicle types and usages, aftermarket, suppliers and production [M. Broy, 2006][K.V. Prasad, M. Broy, and I. Krueger, 2010][R. Schwabel, 2011]. It has to be reliable, flexible and scalable to fit into all vehicle types, ranges, segments, markets and brands. The complexity in the design of the E/E architecture requires the use of many advanced tools and following a number of standards for certification purposes. Standardization is a solution to reduce the complexity. Standardization can be found in the areas of communication (FlexRay, Ethernet, CAN, etc.), platform architecture (AUTOSAR) and diagnostics (e.g. ISO, SAE.). The drivers to have standardization of E/E architectures all comes back to optimization of resources, limit start costs, limit maintenance costs, reduce time to market etc. Most of the fundamental decisions on E/E architecture come from deep understanding of market demands, regulations and technical trends. Therefore E/E architecture design should be in consistence with the product positioning and technology roadmap of a company. This in the end requires a given automotive company to take into account the available resources, maturity of technologies, readiness of suppliers, and making good compromise between seeking immediate cost saving and obtaining long-term advantages. The E/E architecture design is a multi-objective puzzle that should fulfil vehicle and system quality attributes such as performance, reliability, safety, security, energy efficiency, packaging and weight etc., Moreover, an important consideration in E/E architecture design it should not only meet current requirements, but also accommodate future needs.

2.3 Cloud Computing

The term cloud computing is overloaded with multiple competing definitions. For the scope of this report (and TrAF-Cloud in general), we will adopt the NIST definition of cloud computing [P. Mell and T. Grance, 2011].



This definition considers three levels, each defined by the responsibilities of IT operations provided by the cloud vendor. In an Infrastructure-as-a-Service (IaaS) cloud, resources (e.g., computing power, storage, networking) are acquired and released dynamically, typically in the form of virtual machines or containers (e.g., using technologies such as Docker). IaaS customers do not need to operate physical servers, but are still required to administer their virtual servers, and manage installed software. Well-known examples of such services are Amazon's EC2 or ECS[2] services. One key tenet of IaaS services is that they are typically relatively straight-forward to migrate to, as customers can often do a "lift-and-shift" migration [D. Linthicum, 2017], i.e., a migration where the same application that a customer has previously hosted on their own servers ("on premise") is simply installed on resources provided by the cloud provider instead, with little required modifications.

With IaaS, the idea of Infrastructure-as-Code (IaC) has also started to gain momentum. IaC allows users to define and provision operation environments in version-controlled source code. Essentially, in an IaC project, the entire runtime environment of the application (e.g., IaaS resources, required software packages, configuration) is defined using scripts, which can then be executed by tools such as Chef[3] or Ansible[4]. IaC scripts allow entire test, staging or production environments to be started without manual interaction. The move towards IaC with its reproducible provisioning has become necessary since cloud applications often consist of a large number of machines that have to be configured automatically to scale horizontally.

Platform-as-a-Service (PaaS) services represent a higher level of abstraction and typically provide entire application runtimes as a service. The PaaS provider manages the hosting environment and customers only submit their application bundles (e.g., as JAR files, or as functions in a Function-as-a-Service system). They typically do not have access to the physical or virtual servers that the applications are running on. PaaS customers only communicate with the platform via its API and they are largely relieved from server administration and operation duties. Examples of such services

[2] https://aws.amazon.com/ecs/
[3] https://www.getchef.com
[4] https://www.ansible.com

include Google's Appengine[5], hosted Web runtimes such as Heroku[6], or Function-as-a-Service clouds such as AWS Lambda[7].

Finally, in Software-as-a-Service (SaaS), complete applications are provided as cloud services to end customers. The provider handles the entire stack, including the application. The client is only a user of the service, with little to no responsibility (or ability) to access and manage the underlying infrastructure. Examples of such systems are manifold but include well-known applications such as Dropbox or Salesforce. In TrAF-Cloud, such systems are largely out of scope. Instead, we will focus on IaaS and PaaS services, i.e., the services that are more geared towards software developers rather than end users, in the following.

It should be noted that modern clouds also provide access to a plethora of auxiliary services, which do not cleanly follow into this simple taxonomy. Examples include hosted AI services, authentication management, logging services, and many more. Extended taxonomies have been proposed to cleanly classify these additional services [S. Kächele, C. Spann, F. Hauck, and J. Domaschka, 2013]. However, so far, additional levels besides IaaS/PaaS/SaaS have not found widespread adoption. Hence, we will not use them in this report.

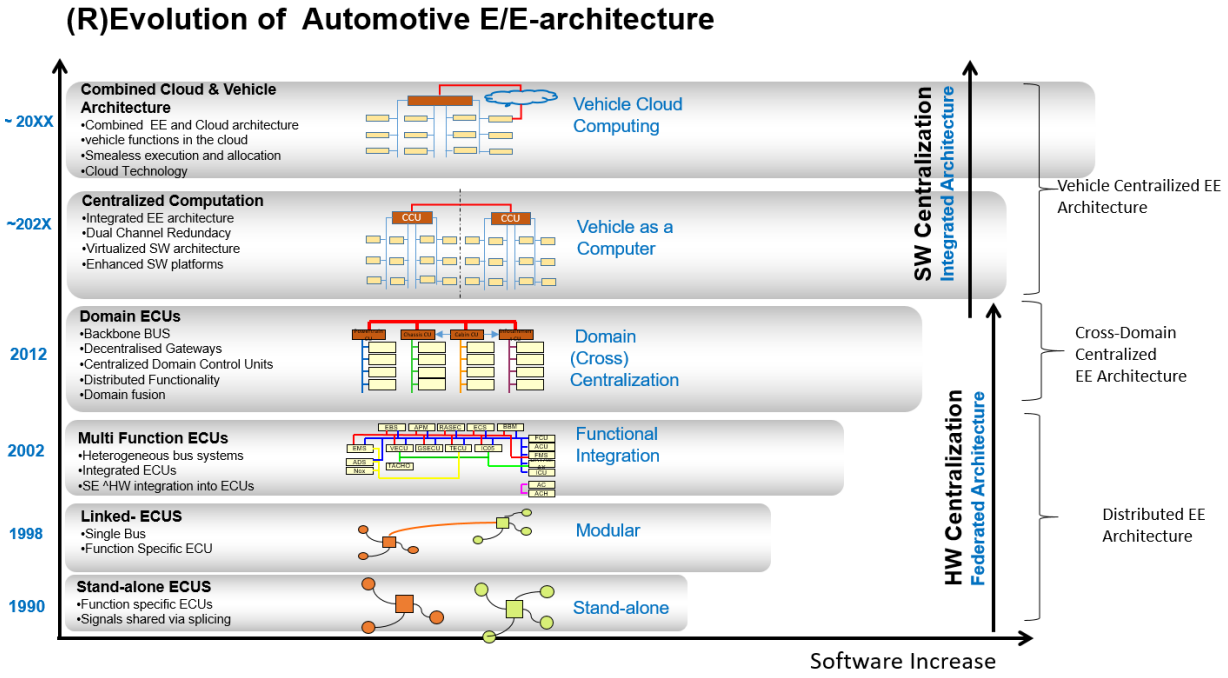
[5] https://cloud.google.com/appengine/
[6] https://www.heroku.com
[7] https://aws.amazon.com/lambda/

3. A Historical View on Automotive Services

We now provide a chronological overview of how on-board and off-board automotive architectures and services have developed. We discuss what factors drove key changes, and what the rationale between different iterations of the Electrical and Electronic architecture were. We conclude with the state of the art as observed today, and what challenges motivate and steer the TrAF-Cloud project.

3.1 Development of On-Board Architecture over Time

Over the last decade there has been an exponential increase in the functionality implemented in electronic systems in vehicles, and this is due to constant increase in customer demands and changing technologies [J. Axelsson, 2009]. This however affects both hardware and software and has increased the importance of system integration activities. For this reason, the importance of having reference architectures to aid in the design of these functions has become an integral part in the automotive product development [U. Eklund, Ö. Askerdal, J. Granholm, A. Alminger, and J. Axelsson, 2005]. The E/E-architecture has evolved over time to reflect and keep up with these changes.



The evolution is guided by the wish to improve functionality, customization, and quality while reducing cost and complexity. This evolution has ranged from stand-alone Electronic Control Units (ECUs) that used sensors and actuators in order to accomplish specialized functions, such as fuel injection or ABS-braking. The second step was to link the ECUs to each other but still function specific. With time the functions increased in number and also became more sophisticated since they needed information from other functions for their efficient operations. For example, to cut the fuel supply when the brakes are applied, the electronic brake system needed information from the fuel injector. This made ECU specific function design invalid but also making it complex for the electronics to operate. In addition, the software needed to be designed to fit the specific hardware. If either hard- or software was changed, the interface needed to be maintained. For example, if specific hardware was replaced by a supplier (standard time in production for ECU hardware is 2 years), the software needs to be verified with the new hardware. The result could be that the software needed to be adapted.

From Single-Purpose to Multi-Function ECUs (2002). Due to increasing customer needs, the functions in the vehicle increased over time, hence also increasing complexity. The automotive industry changed focus to introducing multi-function ECUs and functional integration for the ECUs. This means that one ECU could now be used for different functions. The communication between the ECUs was now replaced with a heterogeneous bus system, meaning that all signals are sent in the bus, which reduces the need for complex wiring. The function integration concept allowed different functions to share information and exchange messages.

In 2003, architecture evolved and instead of focusing on individual ECUs, E/E components and functions were categorized into various domains, such as chassis electronics, powertrain electronics, HMI, Infotainment, body electronics, passive safety, etc. leading to centralized domain control units and domain fusion.

The domain centralization seemed to be a very good concept and it greatly facilitated the function design and implementation, as functions that are closely related are developed and group together under one domain control ECU. However, the ever-increasing demand on vehicle performance has demanded more and more coordination among different domains to create new functions and features, or to improve existing ones. Many E/E systems could make use of information from other domains to optimize their own working performance and consequently achieve overall better vehicle performance. This has driven more interactions cross domains and the need for domain fusion.

AUTOSAR (2003). The changes in electrical designs also brought changes in the software platform architecture for the different ECUs. In 2003, software was designed to comply with the AUTOSAR (AUTomotive Open System Architecture) standard, an initiative supported by the world's leading automotive manufacturers, suppliers and tool providers [Senthilkumar and Ramadoss, 2011]. With AUTOSAR, the automotive industries could reuse architecture components and focus on developing applications instead of base technologies. In this, the AUTOSAR layer is designed between the application soft- and hardware. The communication between applications is managed by a runtime environment (RTE). AUTOSAR contains hard- and software specific components that are updated to communicate with different hardware, making the application software less dependent on ECU hardware. The aim was to reduce dependencies between hard- and software. With AUTOSAR, hardware can be updated without the need to constantly update applications. In other words, the objective for AUTOSAR was to specify an open standard for the software architecture and software interfaces for the most common functionality in ECUs.

The benefit of having AUTOSAR for automotive companies is to have standard functions such as "turning lights on/off" developed by suppliers, while having time to focus on core value adding functions such as safety and security. The potential benefits of re-using these standardized software modules are undisputed and make the use of AUTOSAR very attractive. However, when deploying AUTOSAR, both, the OEMs and Tier1 are confronted with significant challenges [Galla and Pallierer, 2011]. The challenges with the existing AUTOSAR (Classic AUTOSAR) have been linked to its inability to handle increasing complexity and dynamism of automotive functions [C. Jakobs, P. Tröger, M. Werner, P. Mundhenk, and K. Schmidt, 2018]. The current automotive architectures are distributed and therefore the distributed functions are limited by the network capabilities. An example is a CAN network which is based on standardized signal, implying all signals have to be standardized for each function.

Centralized Computation and Adaptive AUTOSAR (202X). With the influence of connectivity, electrification and automation, E/E architecture has taken a new shift from being a distributed architecture to a centralized computational integrated architecture [M. Traub, A. Maier and K. L. Barbehön, 2017][D. Yang, K. Jiang, D. Zhao, C. Yu, Z. Cao, S. Xie, Z. Xiao, X. Jiao, S. Wang, and K. Zhang, 2018]. In this architecture, a more and more adopted solution is to use a dedicated node to do the computation for the intelligence of functions while still having low intelligence input and output nodes. Such mechanisms relieve other ECUs from computation burden and facilitate function development and system integration. Even though the distributed architecture greatly enhanced the vehicle functions and improved

performance, they increased the number of ECUs which has increased E/E system complexity. With the advancement of semiconductor technologies in producing more powerful MCUs and System-on-Chip (SoC), the functions traditionally covered by multiple ECUs could be taken care of by one ECU or SoC. This implies that; the number of ECUs will significantly be reduced hence reducing complexity, mounting, wiring, weight and cost.

With the new applications such as automated driving, Car-2-X, software updates over the air, ADAS and Vehicles as part of the internet of things, they raise new requirements to a software platform for ECUs [J. E. Siegel, D. C. Erb, and S. E. Sarma, 2018]. In today's architecture almost all vehicle internal communication is done via a deeply embedded controller to meet OEM requirements, such as functional safety, all communications are described and standardized by classic AUTOSAR where information elements, known as signals, are standardized to and from certain ECUs [SAE J1939/1]. The automotive industry needs to have an architecture that is flexible as today automotive requirements, highly available and capable to adapt itself to specific application requirements at a given point in time.

Sidebar: In parallel to the industrial developments surrounding AUTOSAR, the academic community has conducted research on applying the ideas of service-oriented architectures (SOA) [T. Erl, 2005] to the automotive domain. Early works in this domain date back to 2004 (e.g., [I. Krüger, E. Nelson, and K. Prasad, 2004]), but even studies conducted as recently as 2017 have shown that widespread implementation of SOA is still 3 to 8+ years in the future [S. Kugele, P. Obergfell, M. Broy, O. Creighton, M. Traub, and W. Hopfensitz, 2017]. Most recently a first formal specification for a vehicle SOA was proposed together with a description on how to define (compose) services and their interfaces [V. Cebotari and S. Kugele, 2019]. A full adoption of SOA has long been seen, and still is, seen as a necessity for future automotive applications such as autonomous driving, but real-life implementation on top of existing E/E architectures remains a challenging proposition.

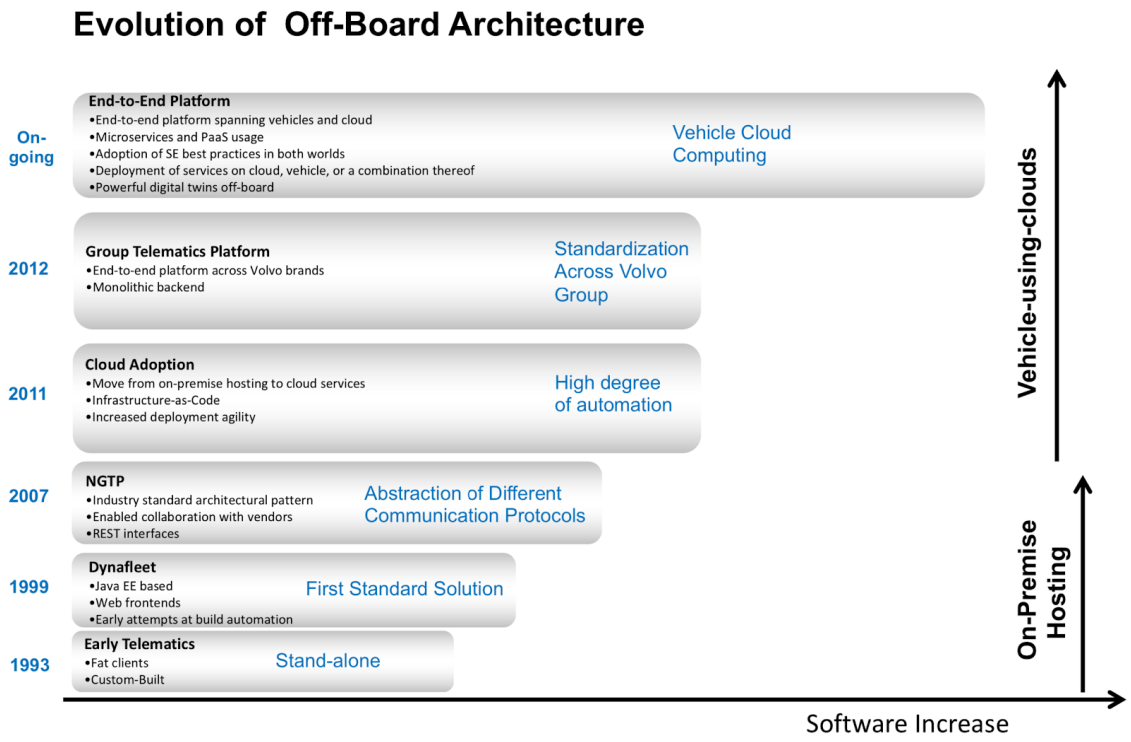
AUTOSAR being the leading standardization organization for in-vehicle software architecture bears this challenge and therefore paves the way of making the car an intelligent and adaptive vehicle. Starting with today's in-vehicle software platforms and their limitations regarding future requirements, the new AUTOSAR Adaptive Platform is aimed to support dynamic deployment of customer applications, to provide an environment for applications that require high-end computing power and to connect deeply embedded and non-AUTOSAR systems in a smooth way while preserving typical features originated in deeply embedded systems like safety, determinism and real-time capabilities [S. Fürst and M. Bechter, 2016]. The Adaptive AUTOSAR is built around existing standards such as POSIX; therefore, it complements automotive specific functionalities enabling the platform to run in an automotive network. To this end, the adaptive platform is aimed to fulfil the existing, changing and new requirements in the automotive domain. With the features provided by adaptive AUTOSAR such as integration of heterogeneous software platforms, software modularization, dynamic linking of services and clients at run time, and service-oriented communication, the adaptive AUTOSAR platform acts as an enabler for the execution of this project. It is therefore the purpose of this project to extend and build on the existing features of adaptive AUTOSAR platform to extend the execution of vehicle functionality in the cloud.

The new architecture is aimed to follow the core architectural principles that address the ever-increasing customer requirements. In the table below we make a comparison of how these principles will be addressed in the new architecture.

Core Architectural Principles	Past	Future
Centralized computation	Federation HW Centric	Integration/Control centralization SW Centralization
Platform virtualization	Low abstraction of hardware, software and networks	Higher abstraction of hardware, software and networks
Layered application services	Signal-based interaction between components implementing an end-to-end function	Provision and consumption of reusable application services Service Oriented Architecture
High integrity & fault tolerance	Limited requirements on integrity and lower expectations on fault tolerance	High integrity and fault tolerance especially for the Autonomous Drive System (ADS)
Information Ownership & Access	Limited exploitation of data and a comparably limited protection of data	Wide exploitation of data and a wider protection of data
Continuous delivery of services	New software delivered to vehicles a few times a year	Continuous Integration and deployment Modern Software Download
Development process	Waterfall based	Agile and Lean based methods

3.2 Development of Off-Board Architecture over Time

With the advent of widespread connectivity, the automotive industry has experienced a trend towards more and more off-board services (i.e., services operated not directly on the vehicle, but in a dedicated data centre or, more recently, in the cloud). We now sketch the historical development of these off-board services and the architecture that enables it.



Connected services: The growth of functionality within the electronic systems in the vehicle together with the utilization of telematics has led to an increased ability to create services targeting areas such as productivity and uptime. Today we refer to such services as connected services. They are since decades

an integral part of the infrastructure, driving high demands for availability, and support both the Volvo Group Customers and the Volvo Group company in conducting their business. Over the years, we have experienced exponential growth of transactions with more connected vehicles, more services per vehicle and more real-time behaviour for some services driving equally high demands on scalability. The change is driven by a constant growth of new or updated service and solution offerings. A global market and multi brand perspective have increased the demand for changeability and flexibility. Thus, there is a need for creating, updating, delivering, and operating the service/solution offerings for different customer segments over time.

A key driver behind this growth were large changes to the underlying telecom networks, going from 2G to 4G networks (with 5G now upcoming). In addition to general-purpose telecom networks, standards for V2X [C. Weiss, 2011] ("vehicle-to-X", which includes vehicle-to-infrastructure and vehicle-to-vehicle communication) have emerged and found use.

Sidebar: A mini-series of two related papers ([C. Weiss, 2011] for Europe, and [J. Misener, S. Biswas, and G. Larson, 2011] for North America) discusses the state of V2X adoption around 2011. Use cases around this time were mostly safety-related (e.g., vehicle-to-vehicle communication to avoid crashes). However, a lack of market penetration and "bootstrapping" as well as the costs of required specialized hardware have prevented a wide-spread adoption despite obvious benefits. Nowadays, there is a push towards adopting widely available standard wireless technology (e.g., 4/5G instead of custom V2X technology and protocols).

Regardless of the generation of networks and underlying technology used, the ever-increasing demand from the service perspective has constantly been challenging. Aspects such as range, capacity, robustness and cost force the connected service software development to constantly consider new technologies to achieve availability, scalability, performance and cost control.

The advent of telematics and connected services (1993): When the Volvo Group started its telematics / connected services journey in 1993 it was initially intended for Volvo Trucks. The initial system as proprietary fat client / server-based system using languages such as C and C++. Vehicle communication utilized the WAP protocol, and there was no real focus on scalability or reusability. The architecture was more or less built without proper modularization and layering principles. The entire area was initially seen as a green field development project specifically for isolated use cases in Volvo Trucks, and mainly driven from a technology research perspective. However, early on the benefits could be seen, and demand from customers for the ability to monitor and manage their vehicles remotely increased quickly. It also opened a door for Volvo to develop the ability to collect data, supporting areas such as warranty and product development.

Dynafleet (1999): In 1999 the solution reached its limits for Volvo Trucks as it could no longer manage the increasing number of customers and functional growth. This led to the release of a new generation of the commercial offering Dynafleet during 2000. The Dynafleet system was based on a three-layer architecture with data/integration, business logic, and user interface portal. The former fat client was replaced by a Web-based frontend. The technology changed, making use of standards such as Java and Java EE. Third party Java EE application servers were used to deploy the applications and an ANT-based build system was introduced to enable some first degree of automation during development.

Vehicle connectivity in the new solution was realized via telecom networks and satellites. The main principle was (and still is) that message loss is not acceptable. Communication cost, latency, and network instability drove the use of very compact data formats such as ASN.1 [G. Neufeld and S. Vuong, 1992] for data serialization. This also meant that TCP was not an option due to much larger size and time to

establish and maintain connection and communication. Instead, UDP was used. However, UDP is a fire-and-forget way of communication, and TCP-like acknowledgement behaviour was built on top of the used communication protocol. The vehicle communication was moved into a separate subsystem as it became more and more complex. To enable that, adapters were used for encapsulating differences in types and versions of telematics units and respective communication protocols.

The Dynafleet technology and architecture also enabled bringing connected services to other Volvo brands. However, although solutions for other brands were based on similar technology, the level of reuse ultimately was low, and silos were created from functional and organizational perspectives for each new solution. The monolithic architecture of Dynafleet drove complexity, making it hard to apply change in the pace needed to handle increased growth. The silo-based landscape often caused to need to solve similar problems in slightly different ways, which drove diversity. It also prevented multiple parties from collaborating and taking responsibility for different parts of a solution from end to end perspective. Ultimately, a need for standardization became apparent.

Next Generation Telematics Pattern (NGTP, 2007): This perceived need for standardization, and in order to open up the architecture for multi-party collaboration, Volvo collaborated through its company WirelessCar with BMW and Conexis to create the Next Generation Telematics Pattern. NGTP provided a distributed off-board architectural pattern for building flexible, scalable, and cost-efficient telematics backend systems. This pattern was based on a modular architecture and provided an interface specification for the off-board elements on a non-technical, semantic level. It was published under a Creative Commons license.

On a high level, NGTP separated vehicle communication and service management to abstract away the differences in communication protocols and patterns from different telematics ECUs. This allowed for a more uniform way of managing vehicle related services. A service integrator (bus) was introduced to connected vehicle communication with end user applications. This enabled a subscription-based data delivery methodology. To make off-board integrations easier, Restful HTTP [C. Pautasso, O. Zimmermann, and F. Leymann, 2008] became the default option for APIs.

In 2007, WirelessCar released the first solution based on NGTP. Another innovation in this step was the large-scale use of open source software (OSS). Build and packaging systems such as Maven were introduced for OSS dependency management. It was still hard to verify changes, which caused a fear of introducing problems that affect availability, reducing the speed of change. Tests needed to be repeatable and performed frequently, motivating the introduction of continuous integration principles.

Group Telematics Platform (GTP, 2012): With the increasing complexity and standardization of off-board architectures, a high level of dependencies between off- and on-board became problematic. An end-to-end architecture for telematics services was needed to enable modularity and reuse of software and services between Volvo Group Brands. In 2012 this was realized through the release of the Group Telematics Platform (GTP). This end-to-end architecture uses a layered and modular approach, which was extended into the vehicle. Further, a functional decomposition into submodules was implemented. The main scope was to support integration between the vehicle and other systems (including future cloud applications). This included several standard services caterings for e.g., vehicle positioning, dynamic vehicle data readout, remote software downloads and similar. Each service has an on-board and an off-board implementation, with a defined service protocol in between. With that, the journey of componentization continued. Due to limitations in how fast the then-prevalent on-premise infrastructure could be made available, the off-board system was deployed as a single system, and hence (from a physical perspective) became a “modular monolith”. It could in principle be deployed in a distributed fashion, but the lack of infrastructure capacity on demand prohibited this in practice.

Cloud-based hosting (2011): During 2011, cloud-based hosting was introduced to support the rapid growth in capacity needed for the continuous integration and delivery system. The ability to manage infrastructure-as-code drove the automation of environment and service provisioning forward. Learnings from that lead to the inclusion of a cloud enablement goal into the off-board target architecture for connected services. With the revision published 2014 this on-demand infrastructure capacity capability became the foundation for enabling an architectural style built on the concepts of component based, distributed, service oriented and event-driven architecture.

However, another aspect that still limited scalability and flexibility was the use of Java EE runtime containers as application servers. They were primarily targeting a multi-application or sub module deployment within each server. Java EE became too heavy-weight from a capacity perspective, and too costly in terms of licensing, to enable having multiple smaller applications deployed in a collaborative distributed environment. To arrive at a smaller application execution environment a shift from Java EE to the Spring framework was performed. To take this even further, new cloud service offerings such as cloud functions (built using Function-as-a-Service systems [P. Leitner, E. Wittern, J. Spillner, and W. Hummer, 2019]) were started to be taken into use. The traditional infrastructure components for storage, databases, and messaging deployed in an IaaS manner were, piece by piece, migrated to PaaS service offerings.

Since 2018, the off-board platform and has a clear cloud-first approach and with the functional separation and distribution of service it has enabled an increase decoupling of life cycles on the off-board side. The ability to use not just IaaS, but also for PaaS and similar higher-level services, enables more focus on developing value instead of building and maintaining infrastructure. Influenced by pioneers such as Netflix the microservices based architecture style [A. Balalaie, A. Heydarnoori, and P. Jamshidi, 2016] was adopted widely. In consequence, the off-board architecture as it stands today implements similar best practices as also seen in cloud applications built in other domains, such continuous deployment, infrastructure-as-code, and automated infrastructure at scale [J. Cito, P. Leitner, T. Fritz, and H. Gall, 2015].

Sidebar: academic research has discussed the idea of vehicular cloud computing in a somewhat different context than the Connected Services view presented here. [M. Whaiduzzamana, M. Sookhaka, A. Gania, and R. Buyya, 2014] presents a survey on vehicular clouds, which mostly discusses different use cases to utilize the processing capabilities of modern vehicles as a (mobile) data centre for general-purpose computing and data storage. [M. Jabbarpour, A. Marefat, A. Jalooli, and H. Zarrabi, 2019] introduces three broad classes of cloud usage in the automotive domain: vehicular clouds as discussed above, vehicle-using-clouds, and hybrid. In TrAF-Cloud, our focus is largely on the vehicle-using-clouds scenario.

Current State: To summarize, the off-board architecture today is based on Microservices style with functional and physical decomposition both in logic, storage and processing. Using cloud services for infrastructure as well as higher levels of services enables more scalability, availability, flexibility and reuse. A reference architecture supports standardized approaches to solving common problems through shareable libraries for common behaviours and functionality. This includes general processing, packaging and deployment, monitoring, logging, security, and similar services. Cloud systems provide much greater capabilities and opportunities on the off-board side, but we still struggle with extending this flexible platform into the vehicles. Vehicle communication today can still be considered non-intelligent with fixed command-based structures, dedicated logic in relation to signals read and functions to control. Control loops exists on both vehicle and cloud. However, they are hard-wired for specific purposes and highly static. Our ability to implement a more flexible IoT based end-to-end platform architecture is still limited.

Consequently, we are highly dependent on vehicle capabilities were we today are compensating for onboard flaws in the off-board service implementation. We need a more flexible, capable and more

intelligent on-board environment to fully link the capabilities available in the off-board cloud service environment.

Another challenge is that IoT PaaS service offerings are today available from the cloud providers (e.g., AWS Greengrass), but they still have many shortcomings that makes it hard to utilize them to a larger and more consistent extent in the context of automotive systems. Even if vehicles (from an off-board telematics perspective) can be viewed as a remotely connected "thing", the complexity of the vehicle, limited on-board capabilities and the context of it moving quickly through large areas with historically unreliable mobile network coverage creates problems. With a new approach and a full end-to-end view of the service execution on- and off-board we hope to be able to extend the cloud to the vehicle and vice versa.

However, we foresee that implementing this end-to-end view ultimately will bring additional challenges to consider. For instance, we may need better monitoring capabilities when the distributed service landscape on-board and off-board grows to manage more monitoring points and more data to be analysed. This can make harder to understand all collaborating parties and components, all of which will have to be managed. We certainly need to manage security threats as we increase the ability to remotely introduce vehicle behavioural changes easier [T. Zhang, H. Antunes, and S. Aggarwal., 2014]. We also see that cost control becomes extremely important to drive efficiency when the limits in capacity on-board is reduced. This means enabling to understand cost per service in its building blocks. We must thereby understand costs when moving control loops and logic out of the vehicle. When operating in a cloud environment, there is a direct link between execution time and (monetary) runtime costs [P. Leitner, E. Wittern, J. Spillner, and W. Hummer, 2019]. So far, this has not been a direct concern, as when executing functions in the vehicle, runtime costs are primarily reduced to energy consumption.

3.3 API Ecosystems and Ecosystem Management

Opening up for service-orientation and cloud thinking inside the truck implies a need to consider future truck service ecosystems. Initial cloud services may be built by the OEM themselves. However, in the long run, we envision that, a truck connected to cloud services will (have to) provide a platform for third-party service providers to develop their own vehicle services, akin to the app stores that rule today's smartphones. The ability to foster and regulate these future service ecosystems (in terms of policies, regulations, technical restrictions, and incentive or monetization models) is going to be a crucial factor for a truck manufacturer to remain competitive in the future. This can be compared to nowadays' situation, where users select smartphones to a substantial degree based on which apps are available for it. When it comes to approaching ecosystems thinking and empowering external contributions the firms need to balance the need to stimulate innovation through engaging third-party development with the need to sustain existing governance and control mechanism over the platform ecosystem.

App stores and third-party development as a competitive advantage: In July 2008 Apple decided to engage third-party application developers into the iOS platform's ecosystem by introducing App Store. Before this game-changing action, Apple was the main responsible for providing new services on iOS platform. However, this decision established a new role for Apple, which now started to act as innovation facilitator. Thanks to the participation of third-party developers, the App Store, is providing a variety of applications for iOS users, today. As Kim et al. put it: "An important underlying principle of App Store is to draw critical resources—applications—from the outside, instead of primarily relying on firm-centric planning." [H. J. Kim, I. Kim, and H. G. Lee, 2010]. The importance of third-party developers for the platform ecosystem is not limited to Apple's case. Other platform owners also tend to open up the doors for open innovation by involving third-party developers (like Facebook, Google Android, or YouTube).

Engaging third party developers and the Goldilocks Governance Problem: Considering the value of third-party developers for an ecosystem, platform owners should facilitate developers' contribution by

supporting them. Recent studies on open source-based ecosystems try to shed some light on barriers for third-party developers to engage in a new community, such as the need to follow "joining scripts" or limitations with regards to specializations [G. Von Krogh, S. Spaeth, and K. R. Lakhani, 2003]. Some other studies show different motivations and strategies for engaging more actors in the developer community. Interestingly, Von Krogh et al. found that institutions can be judged as constraining or supportive, depending on their adherence to social practices within the community [G. Von Krogh, S. Haefliger, S. Spaeth, and M. W. Wallin., 2012]. Platform owners may support third-party developers in different forms, e.g. by sharing knowledge and experiences, suggesting incentives, or creating discussion forums.

While third-party applications can significantly address innovation and creativity within the platform ecosystem, control and governance of such a huge group of actors in the platform ecosystem is not an easy task. Third-party developers and platform owners belong to two different social worlds. This means that they have different goals, work under different limitations, and may have conflicting goals. This leads to a strong need for governance over third-party developers to maintain the integrity of a platform. However, this can be a significant challenge for platform owners, as they need to find balancing mechanisms to also allow for generativity and autonomy of third-party developers [A. Ghazawneh and O. Henfridsson, 2013] [A. Tiwana, B. Konsynski and A. A. Bush, 2010]. Tiwana et al. call this the "Goldilocks Governance Problem" [A. Tiwana, B. Konsynski and A. A. Bush, 2010].

Boundary Resources: Addressing this dilemma in orchestrating platform ecosystem, literature suggests a couple of concepts to address the "Incomplete nature of translation" [N. Chrisman, 1999]: Trading zone and Boundary Objects. According to Galison [P. Galison, 1997] a 'trading zone' is a place, "where participants from different cultures communicate using a stripped-down interlanguage much as traders created pidgins or creoles that operated on the interface between social groups." [N. Chrisman, 1999]. Boundary objects are "common objects (that) form the boundaries between groups through flexibility and shared structure" [S. Leigh Star, 2010], and potentially engage in power relations [Jr. Boland, J. Richard, and R. V. Tenkasi., 1995]. The flexibility embedded in the boundary object allows enough room for affordances of various interpretations and usage in different contexts by diverse actors [I. Hutchby, 2001]. The concept of boundary objects has been used to theorize boundary resources theory in digital platform literature.

"Control over interfaces amounts to control over the platform and its evolution" [A. Tiwana, B. Konsynski and A. A. Bush, 2010, C. Baldwin, J. Woodard, and A. Gawer, 2009]. On the interface between the platform and third-party developers, platform owners offer resources that shift design capability to third-party developers and facilitate the use of core platform functionality. This enables third-party developers to tap into the platform and serve end-users through software applications that will be deployed and become part of the platform ecosystem. Ghazawneh and Henfridsson refer to these resources as platform boundary resources [A. Ghazawneh and O. Henfridsson, 2013]. A truck manufacturer that is acting as key actor in an ecosystem or even platform owner, can use such boundary resources to define which services third-party developers can create. An API designer holds the responsibility of supporting, updating and backing-up the APIs in proportion to the evolution of the service ecosystem [A. Ghazawneh and O. Henfridsson, 2013]. Considering the potential of platform resources in shifting design capability to external developers and outstretching the services in the ecosystem, boundary resources concept is a promising lens to explore and investigate the service ecosystem's requirements, for the purpose of this research project.

While Ghazawneh and Henfridsson divided platform boundary resources into technical and social [A. Ghazawneh and O. Henfridsson, 2013], Myllänrniemi et al. [V. Dal Bianco, V. Myllärniemi, M. Komssi, and M. Raatikainen, 2014] expand this model. According to them, in a software ecosystem, technical boundary resources can be categorized into application boundary resources (ABRs) and development boundary resources (DBRs), in addition to social boundary resources. ABRs are used by the applications directly and they enable applications to utilize the service of platform to provide valuable functionality for users. On the other hand, DBRs are program resources and tools that help developers in programming,

testing, debugging, deploying, and maintaining the applications [V. Dal Bianco, V. Myllärniemi, M. Komssi, and M. Raatikainen, 2014].

4. Open Challenges

We now summarize open challenges that emerge from the discussion of the current state of the art. These challenges will guide and drive the research and implementation work in the remainder of the TrAF-Cloud project.

Challenge 1: Integrating cloud functions with the EE Architecture

The on-board vehicle architecture has evolved over many years and has often grown organically as new needs arose. Still, a basic assumption has been that functionality executes locally, and mainly isolated with limited dependencies on external systems. The aim of the TrAF-Cloud project is to establish an architecture that supports seamless migration of functionality between an on- and off-board execution environment.

A challenge for the TrAF-Cloud project is to develop useful architectural support that enables generic, flexible and safe integration with the on-board vehicle systems, and that provides application developers with tools that allow focus to be put on the features to be developed. This includes questions like the following:

- How to define stable and useful APIs that allows third-party functionality to be seamlessly deployed on- and off-board?
- How to identify sensible abstraction layers in the on-board vehicle architecture?
- What shall be integrated with off-board functions?
- What communication protocols should be used?

Finally, a new architecture will come with the challenge to create guidelines on what functions need to take precedence in different situations.

Challenge 2: Manage on- and off-board consistency

Deploying parts of functionality off-board where the quality of the network connection (e.g. intermittent connectivity losses, varying latencies and speed) will risk leading to inconsistent states. Among foreseeable consequences are that the off-board function may operate on inaccurate data. Further, periodic synchronization between on-board services and the off-board digital twin (ref. Challenge 1) will be required to ensure that the two representations of the world will not diverge by too wide a margin.

Existing concepts known from data management and distributed systems (e.g., eventual consistency, gossiping) will be used to manage consistency in the system. Further, statistical modelling can be used to quantify confidence intervals and uncertainty.

Another aspect of this challenge is also to enable resilience in both, on-board and off-board services. Both services need to be able to gracefully react to degradation or loss of network connectivity between vehicle and cloud. We plan to adopt existing circuit breaker concepts, as known from the microservices architectural style [A. Balalaie, A. Heydarnoori, and P. Jamshidi, 2016].

Challenge 3: Public and internal API design

The APIs a platform provides are a central enabler for developing applications on that platform and needs to provide the right functionality to be useful, enough flexibility not to be limiting for new innovative functionality while allowing the platform provider to retain control. A well-known challenge is how to provide API stability while allowing the platform to evolve, including requirements for backwards compatibility and mandatory times to provide API parts as deprecated before removing them.

Furthermore, security and safety are paramount design constraints in the automotive domain, and levels of access to various systems may need to be defined and maintained. Thus, in this domain the well-known “Goldilocks Governance Problem” represents a very specific challenge. The ecosystem does not only

consist of platform provider and content developers, but of whole chains of suppliers. Finding the right level of granularity and deciding what options should be available via public API's is the first main challenge. Furthermore, the truck manufacturer needs to find ways to encourage or even ensure that certain processes and quality standards are followed.

Challenge 4: Digital twins in the cloud

One of the core ambitions of the project is to develop cloud services that can execute on-board, off-board, or in a combination thereof. In order to allow on-board services to also execute in the cloud, a digital twin concept [F. Tao, H. Zhang, A. Liu, and A.Y.C. Nee, 2018] [Q. Liu, B. Liu, G. Wang, and C. Zhang, 2019] needs to be developed. A digital twin is a virtual representation of a vehicle or fleet of vehicles and mirrors the current and future state (as well as core services) of those vehicles in the cloud. This allows off-board service to execute in a similar manner in both environments.

Existing cloud IoT solutions, such as AWS GreenGrass, already implement a basic version of this idea through device shadows. However, the current implementation is seen as too limiting for the goals of the project.

We envision a powerful digital twin (implemented on top of existing IoT solutions) to have the following properties:

- Be able to represent a single as well as a group (fleet) of vehicles.
- Provide access to rich monitoring data comparable to what is also available on-board, with low latency and high consistency (ref. also Challenge 2).
- Provide predictive capabilities. That is, the digital twin shall not only be able to represent the current state of the fleet, but also be able to make short-term predictions.
- Provide application developers with services for transparent interaction with the vehicle, ideally through the same API as also used on-board.

Challenge 5: Cost modelling of service off-boarding

Offloading the execution of services to the cloud has interesting implications regarding the monetary costs for the service provider (e.g., Volvo Trucks). Whereas the cost of execution for on-board service is essentially zero for the provider, computing in the cloud is billed (either per time interval, or, in the case of Function-as-a-Service clouds, per execution time / compute cycles).

In order to properly assess the business value of cloud services, a cost/benefit model is required. Establishing such a model is difficult, as modern cloud solutions often rely on a complex pricing model, combining charges based on time, number of requests, number of I/O operations, and hardware used. Further, a cost model is also required as part of the decision-making engine, to decide if it is "worth" to move a given service to the cloud to save provider costs.

5. Summary and Outlook

The vehicle architecture - on-board and off-board – underwent an enormous evolution in the last 25 years. The emergence of cloud architectures is accelerating this evolution today, opening the way for new business models and market places.

However, todays vehicle architectures are not yet fit for this development. We identified five main challenges that need to be addressed in order to enable Swedish truck-producers to advance their products and open them up to cloud-based innovations. These challenges concern the Integration of cloud functions with the EE Architecture, the management of consistency on- and off-board, the design and management of public and internal APIs, the use of digital twins in the cloud, as well as cost modelling of off-boarding services.

The TrAF-Cloud project aims at tackling these identified challenges through the collaboration of Industry and research, including the partners VGTT, VGCS, Combitech, Chalmers, RISE, Vector, and Amazon.

References

J. Axelsson, "Evolutionary architecting of embedded automotive product lines: An industrial case study," 2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture, Cambridge, 2009, pp. 101-110

C. Baldwin, J. Woodard, and A. Gawer. "Platforms, markets and innovation." The architecture of platforms: A unified view (2009): 19-44.

A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture”, IEEE Software, (Volume:33 , Issue:3 , May-June 2016)

I.S. Board. "Ieee standard glossary of software engineering terminology." *New York: The Institute of Electrical and Electronics Engineers* (1990).

Jr. Boland, J. Richard, and R. V. Tenkasi. "Perspective making and perspective taking in communities of knowing." Organization science 6, no. 4 (1995): 350-372.

E. Borgia. “The Internet of Things vision: Key features, applications and open issues”. Comput. Commun. 54, 1–31 (2014).

P. Brereton, B.A. Kitchenham, D. Budgen, M. Turner, and M. Khali: Lessons from applying the systematic literature review process within the software engineering domain. Journal of Systems and Software vol. 80, iss. 4, 571-583 (2007).

M. Broy, “Challenges in automotive software engineering,” in Proceedings of the 28th international conference on Software engineering, Shanghai, China, 2006, pp. 33–42.

V. Cebotari and S. Kugele, “On the Nature of Automotive Service Architectures,” in 2019 IEEE International Conference on Software Architecture Companion (ICSA-C), 2019, pp. 53–60.

N. Chrisman. "Trading zones or boundary objects: Understanding incomplete translations of technical expertise." In 4S meetings, San Diego. 1999.

J. Cito, P. Leitner, T. Fritz, and H. Gall, “The making of cloud applications: an empirical study on software development for the cloud”, Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE), 2015

V. Dal Bianco, V. Myllärniemi, M. Komssi, and M. Raatikainen. "The role of platform boundary resources in software ecosystems: A case study." In 2014 IEEE/IFIP Conference on Software Architecture, pp. 11-20. IEEE, 2014.

U. Eklund, Ö. Askerdal, J. Granholm, A. Alminger, and J. Axelsson. "Experience of introducing reference architectures in the development of automotive electronic systems." In Proceedings of the second international workshop on Software engineering for automotive systems (SEAS '05). ACM, New York, NY, USA, 1-6. 2005

T. Erl, R. Puttini and Z. Mahmood. “Cloud Computing: Concepts, Technology, and

Architecture”. (The Prentice Hall Service Technology Series from Thomas Erl) 1st Edition, 2013

T. Erl, “Service-Oriented Architecture. Concepts, Technology, and Design”. Prentice-Hall, 2005.

S. Fürst and M. Bechter, "AUTOSAR for Connected and Autonomous Vehicles: The AUTOSAR Adaptive Platform," 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W), Toulouse, 2016, pp. 215-217.

P. Galison. Image and logic: A material culture of microphysics. University of Chicago Press, 1997.

T. Galla, and R. Pallierer. “AUTOSAR – challenges and solutions from a software vendor’s perspective”, e & i Elektrotechnik und Informationstechnik, Vol 128: 6, 2011

A. Ghazawneh and O. Henfridsson. "Balancing platform control and external contribution in third-party development: the boundary resources model." Information systems journal 23, no. 2 (2013): 173-192.

I. Hutchby. "Technologies, texts and affordances." Sociology 35, no. 2 (2001): 441-456.

M. Jabbarpour, A. Marefat, A. Jalooli, and H. Zarrabi, “Could-based vehicular networks: a taxonomy, survey,and conceptual hybrid architecture”, Wireless Netw (2019) 25:335–354

C. Jakobs, P. Tröger, M. Werner, P. Mundhenk, and K. Schmidt, “Dynamic Vehicle Software with AUTOCONT,” in Proceedings of the 55th Annual Design Automation Conference, New York, NY, USA, 2018, pp. 101:1–101:6.

S. Kächele, C. Spann, F. Hauck, and J. Domaschka. “Beyond IaaS and PaaS: An Extended Cloud Taxonomy for Computation, Storage and Networking”, Proceedings of the 2013 Conference on Utility and Cloud Computing (UCC).

H. J. Kim, I. Kim, and H. G. Lee. "The Success Factors for App Store-Like Platform Businesses from the Perspective of Third-Party Developers: An Empirical Study Based on A Dual Model Framework." In PACIS, p. 60. 2010.

I. Krüger, E. Nelson, and K. Prasad, “Service-Based Software Development for Automotive Applications”. Proceedings of the Convergence International Congress & Exposition On Transportation Electronics, 2004.

S. Kugele, P. Obergfell, M. Broy, O. Creighton, M. Traub, and W. Hopfensitz, “On Service-Orientation for Automotive Software”, Proceedings of the 2017 IEEE International Conference on Software Architecture (ICSA).

S. Leigh Star. "This is not a boundary object: Reflections on the origin of a concept." Science, Technology, & Human Values 35, no. 5 (2010): 601-617.

P. Leitner, E. Wittern, J. Spillner, and W. Hummer, “A mixed-method empirical study of Function-as-a-Service software development in industrial practice”, Journal of Systems and Software, Volume 149, March 2019, Pages 340-359

D. Linthicum. “Cloud-Native Applications and Cloud Migration: The Good, the Bad, and the Points Between”. IEEE Cloud Computing. Volume:4 , Issue:5 , September/October 2017.

Q. Liu, B. Liu, G. Wang, and C. Zhang. "A comparative study on digital twin models." In AIP Conference Proceedings, vol. 2073, no. 1, p. 020091. AIP Publishing, 2019.

P. Mell and T. Grance. “The NIST Definition of Cloud Computing”. National Institute of Standards and Tech- nology (NIST), Gaithersburg, MD, Tech. Rep. 800-145, September 2011

J. Misener, S. Biswas, and G. Larson, “Development of V-to-X systems in North America: The promise,the pitfalls and the prognosis”, Computer Networks 55 (2011) 3120–3133

G. Neufeld and S. Vuong, “An overview of ASN.1”, Computer Networks and ISDN Systems 23 (1992) 393-415

C. Pautasso, O. Zimmermann, and F. Leymann, “Restful web services vs. "big"' web services: making the right architectural decision”, Proceedings of the 17th international conference on World Wide Web (WWW), 2008.

K. V. Prasad, M. Broy, and I. Krueger, “Scanning Advances in Aerospace Automobile Software Technology,” Proceedings of the IEEE, vol. 98, no. 4, pp. 510–514, Apr. 2010.

SAE J1939/1 Recommended Practice for Control and Communications Network for On Highway Equipment (and subsequent documents on http://www.sae.org/standardsdev/groundvehicle/j1939a.htm)

R. Schwabel, “Technical Challenges in Future Electrical Architectures,” presented at the SAE 2011 World Congress & Exhibition, 2011, pp. 2011-01–1021.

K. Senthilkumar and R. Ramadoss, "Designing multicore ECU architecture in vehicle networks using AUTOSAR," 2011 Third International Conference on Advanced Computing, Chennai, 2011, pp. 270-275

J. E. Siegel, D. C. Erb, and S. E. Sarma, "A Survey of the Connected Vehicle Landscape—Architectures, Enabling Technologies, Applications, and Development Areas," IEEE Transactions on Intelligent Transportation Systems, vol. 19, no. 8, pp. 2391–2406, Aug. 2018.

F. Tao, H. Zhang, A. Liu, and A.Y.C. Nee. "Digital twin in industry: state-of-the-art." IEEE Transactions on Industrial Informatics 15, no. 4 (2018): 2405-2415.

A. Tiwana, B. Konsynski and A. A. Bush. "Research commentary—Platform evolution: Coevolution of platform architecture, governance, and environmental dynamics." Information systems research 21, no. 4 (2010): 675-687.

M. Traub, A. Maier and K. L. Barbehön, "Future Automotive Architecture and the Impact of IT Trends," in IEEE Software, vol. 34, no. 3, pp. 27-32, May-Jun. 2017.

G. Von Krogh, S. Haefliger, S. Spaeth, and M. W. Wallin. "Carrots and rainbows: Motivation and social practice in open source software development." MIS quarterly 36, no. 2 (2012): 649-676.

G. Von Krogh, S. Spaeth, and K. R. Lakhani. "Community, joining, and specialization in open source software innovation: a case study." Research policy 32, no. 7 (2003): 1217-1241.

C. Weiss, "V2X communication in Europe – From research projects towards standardization and field testing of vehicle communication technology", Computer Networks 55 (2011) 3103–3119

M. Whaiduzzamana, M. Sookhaka, A. Gania, and R. Buyya, "A survey on vehicular cloud computing", Journal of Network and Computer Applications 40 (2014) 325–344

D. Yang, K. Jiang, D. Zhao, C. Yu, Z. Cao, S. Xie, Z. Xiao, X. Jiao, S. Wang, and K. Zhang., "Intelligent and connected vehicles: Current status and future perspectives," Sci. China Technol. Sci., vol. 61, no. 10, pp. 1446–1471, Oct. 2018.

T. Zhang, H. Antunes, and S. Aggarwal. "Securing connected vehicles end to end." No. 2014-01-0300. SAE Technical Paper, 2014.