Combining Adaptive AUTOSAR and IoT Edge Processing for Next-gen Vehicles in High Performance Computers

Dr.-Ing. Sebastian Ohl[1], Dr-Ing. Matthias Beckert[1], Fabian Donath[1]

[1] Elektrobit Automotive GmbH, Am Wolfsmantel 46, 91058 Erlangen,
[sebastian.ohl|matthias.beckert|fabian.donath]@elektrobit.com, https://www.elektrobit.com

Keywords: Adaptive AUTOSAR, High-performance computer,
Amazon AWS IoT Greengrass, Edge computing

ABSTRACT

The automotive industry is heading towards an always connected future. Next-generation vehicles will be connected constantly to the external world through the internet. To achieve this, many functions will be partially or completely cloud-based. The automotive world is quite new to this setup. Traditional OEMs, who have mastered the process of vehicle series production, are starting to realize the disruption coming from the advancements in cloud technologies and the resulting new possibilities for their fleets. The shift in the methodology running applications at the edge and in the cloud can be supported by using a productized software platform that can be reused by different OEMs/Tier 1s and by featuring seamless data processing in the cloud as well as in the edge.
In this contribution, we will demonstrate how to realize a combination of cloud-based IoT edge processing hosted on an automotive high-performance ECU, running Adaptive AUTOSAR. Different base setups with containers and a hypervisor are discussed to provide different isolation levels. Communication between both domains is realized by an intelligent bridge translating between both worlds.

1 INTRODUCTION

With the wide availability of mobile handheld devices in the late 2000s, consumers got more and more used to a constant flow of updates for their devices. Features that were provided by third parties and fixes to already installed applications became normal for us all. The automotive industry, however, maintained their business model of selling ready-to-use products that usually do not change over their lifetimes. This also includes the absence of new technologies or use cases in a customer's car until they buy a new one. As the mean lifetime of a vehicle in Germany is about ten years, innovation happens rather slow.

With the entry of new market participants, e.g. Tesla, this started to change. Features can now be brought to a vehicle after purchasing, software packages are updated frequently, and even promised features, not available at time of purchase, can be delivered later on. As a result, all major OEMs are developing over-the-air update (OTA) mechanisms for their modern cars. However, having the possibility to update vehicle software also requires a feedback loop. Getting data from the vehicles in the field also needs to become much faster than downloading data every few months at a local car dealer. Modern vehicles are, therefore, required to be connected permanently or at least regularly (e.g. at the owner's Wi-Fi hot spot) to the internet.

Managing this IT infrastructure has never been the core competence of traditional automotive OEMs. With the rise of cloud technology in the beginning of the 2010s, established IT companies such as Amazon, Microsoft, or Google started to offer processing and storage data services without the need to run their own data centers. For traditional IT services such as web services, storage, server housing, or online-based applications, cloud services have prevailed over self-hosting. Cloud providers have built a strong ecosystem around the basic

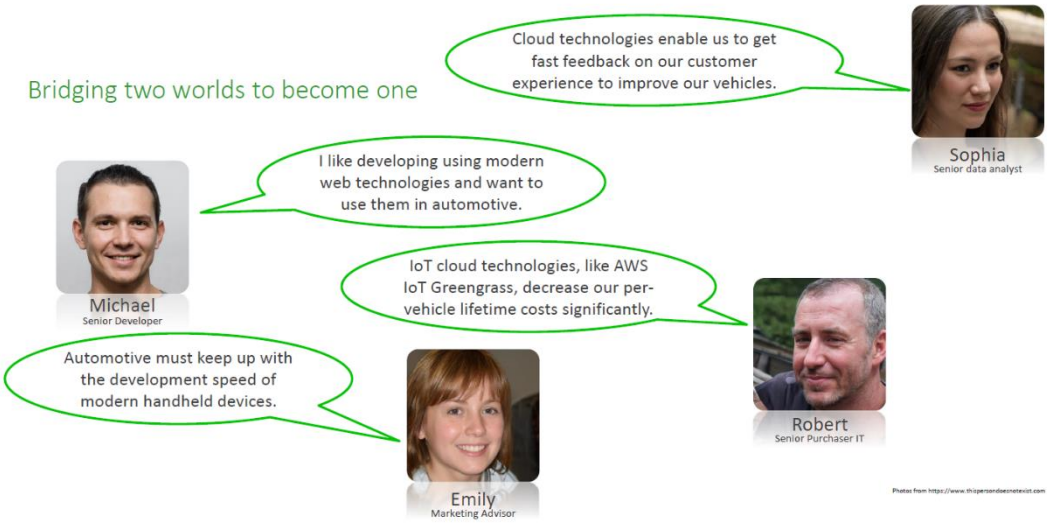

Figure 1: Statements from different stake holders.

services such as storage, hosting servers or services, and applications. They run data centers around the world and can provide a virtually uninterrupted service due to the capability to move services from one node to another without noticeable interruption. This also includes the distribution of software updates and fixes.

A strength of cloud services is their ability to scale in an instant. While scaling in self-hosted data centers requires buying and installation of new hardware, maybe even building up a new data center overseas, scaling in a cloud is fairly easy as the resources are already built up in the required region. All these attributes are technical. However, cost and technical skills are also big drivers towards cloud technology. When moving services to the cloud, costs are transferred from in-house data centers to the cloud provider. Whether the bill is cheaper or higher depends highly on the individual case. However, a big plus for cloud technology is the possibility to share resources. While you might run a dedicated machine in your own data center, a virtual machine, or a container might suffice in the cloud. For these solutions, the cloud user shares a physical machine with other users and can reduce costs. This shared machine is operated and maintained by the cloud provider. Security updates are installed regularly, software is kept up to date, and hardware failure is handled. All topics, a local IT employee usually puts a lot of work into.

Shifting services to the cloud gets even more compelling when it comes to specialized knowledge of a service. For example, a database service requires constant optimization and service to process data efficiently. Highly skilled personal is required to keep everything running. If a company runs multiple databases, they might be able to afford such a specialist. However, running a database in the cloud benefits from sharing these skilled personnel as it maintains many databases from other customers as well.

While the main area for cloud technology is traditional IT, the cloud providers are also investing in the area of internet of things (IoT) to connect data providers to their eco-system. These interfaces offer a chance for automotive companies to connect their vehicles and use the capabilities and scalability of the cloud technology.

2 SETUP

This contribution focuses on the integration of a cloud edge solution and AUTOSAR Adaptive Platform (ASR-AP) [2] in a software vehicle platform. The cloud edge solution provides an interface to deploy, update, and configure applications to an edge device as well as to connect it to a cloud. The advantages of connecting ASR-AP via an edge solution to the cloud are described by four stakeholders (see Figure 1):

- Robert is a Senior IT purchaser. He sees that the IT per-vehicle cost over the vehicle lifetime is not effectively managed. He has an interest in modeling these costs and bringing them down.
- Michael is a skilled developer originating from outside the automotive industry. He is interested to use modern web technologies for developing software applications.
- Emily works in Marketing and is a digital native. She sees the strong requirement that the automotive business needs to keep up with the development speed of other industries like mobile handhelds.
- Sophia is a data analyst. She is frustrated by getting data from vehicles in the field, which is not easy. Especially getting data that was not requested before the start of production. She wants to be as flexible as possible to request data whenever it is required for data analysis.

In the following fictitious scenario, an OEM wants to introduce cloud connectivity to their new vehicles. Their plan is to use Amazon AWS as cloud provider because the OEM has already good experience from another general IT project with this platform. Before starting to define exact specifications for this service, the OEM consults their employees about their requirements for cloud connectivity: As cloud connectivity provides the possibility to get data from the vehicles in the field, the IT department for processing customer feedback can provide valuable input. Their representative is Sophia, a data analyst. She is mainly interested in improving access to in-car data, especially, when it comes to retrieving data on demand. In past vehicle projects, all data that should be available for analysis later needed to be defined years before SoP. With the ability to run and alter a data gatherer after SoP with a reasonable effort, this becomes obsolete. Sophia wants to have the full vehicle network at her disposal. The system should have the ability to request data from all vehicles or just a subset. With the ability to

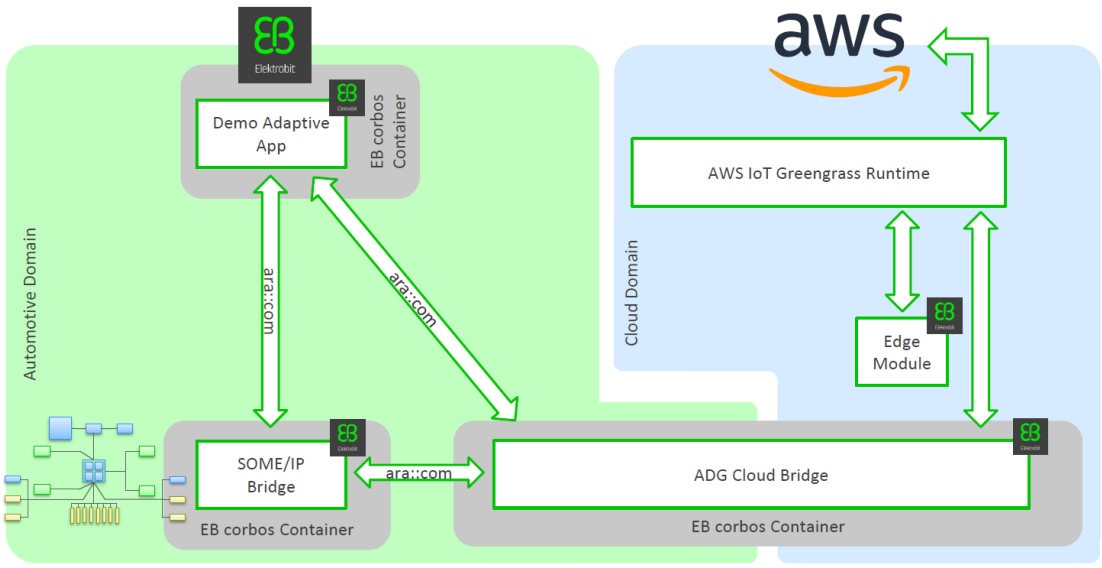

Figure 2 Solution integration architecture

deploy software easily to vehicles in the field, new possibilities for product design and marketing arise. Emily, a marketing employee, is looking forward to these options. She grew up with mobile devices and internalizes the development speed and spirit of this industry. Traditional automotive usually behaves in the exact opposite way. Deploying early prototypes to customers to get fast feedback could really improve the way Emily works. Digital products could be tested and improved not only in customer clinics but with real users in real-world scenarios. It would also be possible to deliver new features at short notice or to do marketing campaigns based on local or terminable events (e.g. an application helping soccer fans to get to a stadium). Time to market could be improved as features could be delivered more easily. Moreover, customer loyalty can be strengthened by offering the possibility to personalize a vehicle by software-defined behavior using apps. This also offers a new aftermarket revenue channel directly to the OEM that has been routed via local car dealers in the past. Bringing a vehicle into the cloud creates long-running IT costs per vehicle. Robert, an IT purchaser, is looking into IT costs for the OEM. These days, IT cost per vehicle is often not clearly defined. The corresponding development departments are only responsible for development and IT until a vehicle is sold. Costs that occur afterwards are hard to determine and, therefore, hard to optimize. Moving services to the cloud comes with the opportunity to define the aftermarket IT costs on a per-vehicle basis. Software development is one of the key success factors for future cars. Michael, a senior developer, is working on the software-defined car. He has a software industry background and is looking forward to more flexibility for software development in vehicles. In his opinion, three to seven years from planning until SoP are too long for releasing new software features. He proposes to separate vehicle-critical from non-critical functions and to generate a constant stream of differentiating features and improvements to vehicles in the field, even years after selling. This could be achieved by providing standardized APIs and the use of hardware-independent programming languages such as Java or scripting languages and a reliable way of over-the-air updating. A standard software platform is another prerequisite to achieve this independence from base software and user-experienceable applications. These views represent only a part of the story. There are more parties benefiting from a cloud-enabled vehicle. This contribution provides a solution for cloud integration of Amazon AWS into a vehicle infrastructure.

2 ARCHITECTURE

As depicted in Figure 2, the architecture is split into two different domains: an automotive domain, and a cloud domain. These domains differ from each other as different standards apply. The automotive domain follows all automotive standards for software development, testing, safety, and security regulations. The cloud domain is much more agile as its focus is gathering data and providing new functionality to the customers at a fast pace.

The safe and secure coexistence of these two domains is achieved by a cloud bridge application that exists in both domains. This bridge receives data from the automotive network or applications, converts it into a format that can be processed by the edge applications, and forwards the converted data to the cloud domain or vice versa. This cloud bridge also acts as a firewall between the two domains. It does not only convert all data but only data that has been defined during configuration time and according to the safe automotive processes. This also includes rate limiting or plausibility checks.

As described before, the automotive domain is driven by the AUTOSAR AP middleware. It communicates mainly over Ethernet using the SOME/IP protocol [3]. The used implementation of Elektrobit's EB corbos AdaptiveCore differentiates between two communication scenarios: intra-OS

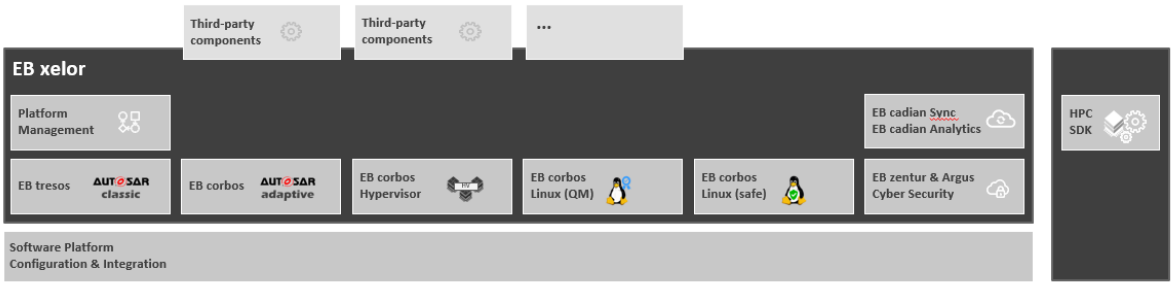

Figure 3 Software platform as basis for implementation

and inter-OS communication. The intra-OS communication is based on Unix domain sockets transferring SOME/IP serialized data packages. The inter-OS communication is based on Ethernet and handled by a SOME/IP bridge, bridging the intra-OS communication to the network and vice versa.

A similar approach is used with the cloud bridge applications, in the automotive domain it uses intra-OS communication to communicate with AUTOSAR AP application and the SOME/IP bridge to access the automotive network. The communication with the cloud domain can be realized via standardized protocols such as MQTT [1] or via proprietary cloud provider libraries to local edge applications or directly with the cloud.

2 DIFFERENT SOLUTIONS FOR IMPLEMENTATION

The concept is realized by using Amazon AWS IoT Greengrass [4] as cloud edge solution and EB xelor as software platform. EB xelor is an automotive software platform focusing on high-performance computers in automotive vehicles. It comes with different software packages, such as AUTOSAR Classic/Adaptive Platform, Linux, or a L1 hypervisor, combined to a holistic solution (see Figure 3). These software packages are completed by adding system level functionalities, such as inter-OS software updates, that are not standardized in automotive (e.g. via the AUTOSAR consortium). Such a package allows an OEM or Tier 1 to focus on differentiating features for their customers and not about the platform to execute these functions. Amazon AWS IoT Greengrass is one solution that directly comes with the EB xelor platform as part of the system features and enables the user to focus on their cloud edge applications.

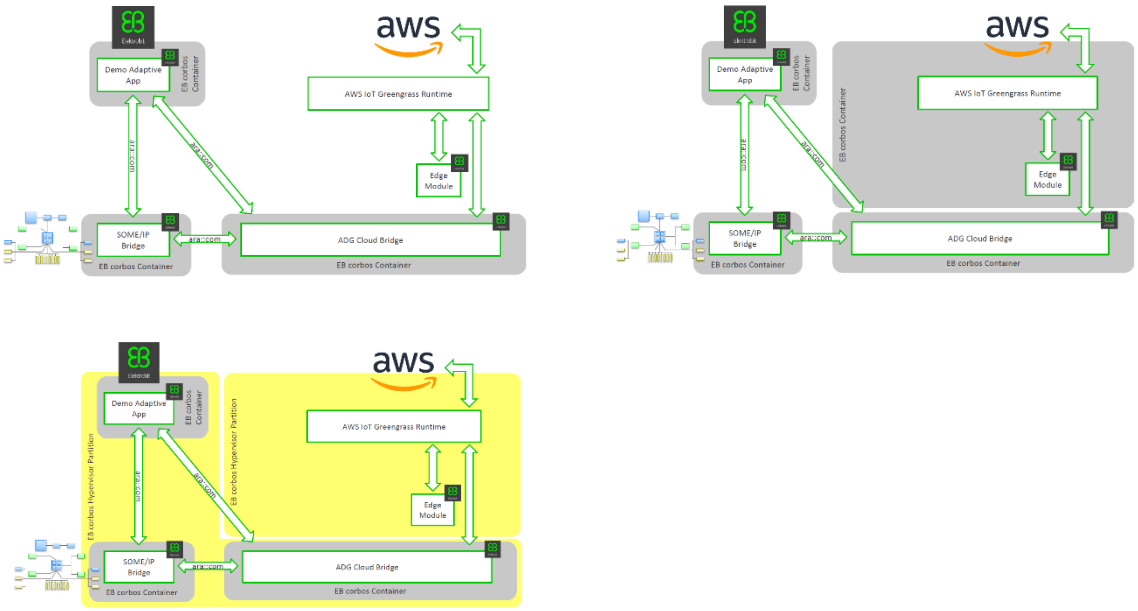

Figure 4 Different solutions for integration: direct integration into operating system (upper left), encapsulated into containers (upper right), encapsulated in hypervisor partitions (lower left)

Table 1 Comparison table of direct, containerized and hypervisor-partitioned integration setup

	Direct	Containerized	Virtualized
Runtime oerhead	Normal	Normal	Increased
Memory overhead	Normal	Slightly increased	Increased
Isolation	Only OS mechanisms	Only OS mechanisms	Hypervisor and hardware mechanisms
Updatability	Complex	Simple	Complex
Complexity	Normal	Slightly increased	Increased

Different integration setups of a cloud edge solution into a software platform can be discussed. In order to compare these, the following parameters will be used (an evaluation can be found in Table 1):

- Runtime overhead: What is the CPU overhead caused by the integration setup?
- Memory overhead: Additional usage of RAM and non-volatile (flash) memory caused by the integration setup?
- Isolation: Does the integration address safety-relevant topics such as freedom from interference?
- Updateability: To what extent can the Greengrass runtime easily be updated over the air?
- Complexity: What is needed to get the setup working and how easily can it be modified?

The first proposed integration setup is a direct integration on top of EB corbos Linux, side by side with the AUTOSAR AP stack and adaptive application. Figure 4 (upper left) shows such a setup. With respect to runtime overhead, the proposed setup does not introduce any other additional overhead than the actual execution of the Amazon AWS IoT Greengrass components. The edge components still need RAM and non-volatile memory for execution, but the integration setup does not add any additional memory usage to this. Since the edge components are executed on the same level as the other components, there is almost no isolation with respect to freedom from interference. Only OS-based isolation mechanisms can be used, but those are limited to their applicability to functional safety. The updateability of such a setup is limited. An edge module can be updated over the air without any problems, but the situation for the Greengrass runtime is different since it is part of the system's rootfs. In a default EB corbos Linux integration, the rootfs is read-only, therefore is is not possible to update only parts of it. If a complete update of the Greengrass runtime is needed, the entire rootfs needs to be updated. The integration is straightforward and not very complex. Access to resources or peripherals is only limited by the used kernel and device tree.

Next, an integration setup with a containerized AWS IoT Greengrass runtime is discussed, as shown in Figure 4 (upper right). Even though the Greengrass runtime already has its own isolation technique, the goal is to encapsulate the entire environment including applications in a runc container (EB corbos Container), as already used for EB corbos AdaptiveCore components and applications. For such a setup, there should not be any noticeable difference with respect to runtime overhead, compared to the direct integration without an additional runc container. The reason for this is that the execution is still performed directly on the underlying OS. For debugging purposes, the runc container might execute additional services (e.g. SSH), but those can be removed in the final integration. The overhead for non-volatile memory is increased slightly compared to the direct integration without container. This is due to redundant components in the container's rootfs, which cannot be mapped from the host's rootfs. The RAM usage should not change since the executed processes are still the same. In theory, the container might provide additional isolation since it can only access mapped resources. Nevertheless, it is questionable to which degree the container needs system access in order to execute the Greengrass runtime and the applications inside. In the end, it still relies on OS-based isolation mechanisms as for the direct integration. With respect to updateability, the containerized integration is beneficial. The runc container environment in EB corbos Linux always locates the containers in a region with write access. Since a container provides its own rootfs, the entire Greengrass runtime can be upgraded over the air without touching the read-only host rootfs. With introduction of the additional runc container, integration gets a little more complex since access to peripherals and resources need to be mapped to the container. Nevertheless, this additional effort should be minimal.

The third integration setup partitions the automotive and cloud domain in two different hypervisor virtual machines (VMs) running EB corbos Linux as guest OS shown in Figure 4 (lower left). For this setup the runtime overhead is significantly increased since each hypervisor VM is pinned to a set of cores, which are not shared among partitions. Additionally, each partition executes its own OS kernel which increases the overall number of processes running on the ECU. The memory overhead is also increased significantly. With respect to RAM usage, an entire second OS instance needs to be executed. And for non-volatile memory, an entire second rootfs is needed for an encapsulated execution of the two hypervisor VMs. With respect to isolation, the hypervisor is the only component that can provide real freedom from interference. The EB corbos Hypervisor is based on a small trusted computing base and relies on existing hardware/software virtualization mechanisms designed for functional safety. The updateability of such a setup is the same as for the direct integration. Since the rootfs of a VM is still read-only, it

Figure 5 Feedback from different stakeholders.

is only possible to directly update the Greengrass components. If a complete update of the Greengrass runtime is needed, the entire VM's rootfs needs to be updated. With the additional hypervisor, the integration setup gets a lot more complex. Before, there was only one OS, that needed to be configured properly. Even with the additional container this is still relatively simple since the hardware is not shared between multiple OS instances. Sharing access to hardware peripherals among multiple VMs without an additional arbitration layer in between is often not straightforward. Direct access to hardware peripherals is therefore often only allowed to the hypervisor or a prioritized VM. Whether it is possible to only grant a VM access to selected peripherals highly depends on the used hardware and the possible cross-dependencies (e.g. DMA) between the different peripherals. For some peripheral types such as network or block devices it is possible to emulate them based on VirtIO drivers. The actual access to the hardware is then managed by a single prioritized software component. Nevertheless, one might realize that the addition of a hypervisor clearly increases the integration complexity and the possibility for integration errors.

As shown above, all integration solutions have their own advantages and disadvantages. The virtualized solution might not be possible due to limitation in resources. The containerized solution might be out of the question because containers are not used in the overall setup, and security is not that much of an issue as the edge integration is hosted on its own chip. The decision for one or the other always depends on the specific requirements and the use case.

3 CONCLUSIONS

This contribution discussed different solution on integrating a cloud edge processing solution into an automotive use case. It was realized using Amazon AWS IoT Greengrass as cloud solution and EB xelor for providing the automotive base software platform. The result can be used to develop automotive edge applications. Referring to the defined personas from Section 1, their needs can be fulfilled with the provided solutions (see Figure 5).

Combined setups between traditional automotive applications and cloud-based edge applications will be seen in many future cars. These setups can only be achieved by providing already pre-integrated solutions that take care of the special needs of automotive such as the used automotive software platform EB xelor. Solving these tasks individually will cause security issues because the testing base is not broad enough, and security issues distract resources of OEMs and Tier 1s from developing differentiating features for their customers.

REFERENCES

[1] ISO/IEC 20922 (2016). Message Queuing Telemetry Transport (MQTT) v3.1.1. Standard, International, Organization for Standardization, Geneva, CH.

[2] Autosar Consortium (2019). Guideline for using Adaptive Platform interfaces, r19-03 edition.

[3] Autosar Consortium (2020). SOME/IP Protocol Specification, r20-11 edition.

[4] Amazon Webservices (2021). https://aws.amazon.com/de/greengrass/, visited 2021, June 22th