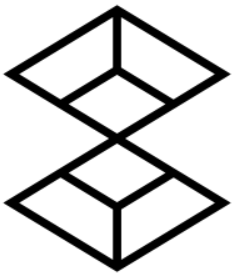

SOAFEE

SOAFEE Architecture

SOAFEE Members

May 11, 2023

CONTENTS:

CHAPTER

ONE

INTRODUCTION

1.1 Software Defined Vehicle

Mobility undergoes a significant digital transition towards becoming connected, autonomous, shared and electric (CASE). This transformation bases majorly on Software Technologies and the Software Defined Vehicle is characterized by managing its functionality and its operations to a large extent by Software throughout its entire lifecycle.

1.2 Centralization of Compute

With a software-centric focus, the electronic architectures of vehicles transform towards greater centralization of compute which is happening due to both technical and business reasons. This trend in consolidation of compute in automotive allows for improvements such as the reduction of wiring complexity and weight amongst other factors. These improvements allow for supply chain and manufacturing optimisations with the promise of easier and less expensive product development.

1.3 New Business Models

The move towards Software Defined Vehicles enables an OEM to enhance the vehicle's capabilities over time as new features are expressed in software that can be delivered over the air. This will allow for enhanced business models whereby the end users can purchase additional features after the vehicle has left the dealership.

1.4 Separation of Hardware and Software

With vehicle functions getting updated or enhanced throughout the entire lifecycle of a vehicle, it becomes essential to deploy identical Software Components to different vehicle computer variants and therefore achieve a separation of Hardware and Software.

1.5 Virtual Development and Validation

To shorten the time-to-market for new Software Functions and allow for scalability of Software Validation with the number of vehicle and therefore hardware variants, use of Virtual Development and Virtual Validation becomes essential. Fast and Frequent Software Deployments With the penetration of vehicles being connected to the digital consumer ecosystem, the need for fast and frequent Software Deployments is majorly driven by Cyber Security requirements, which currently get increasingly important. In addition to technical and business aspects, legal standards have evolved significantly. Additionally the lifecycle of hardware and software will be decoupled and the latest version of a software might be installed on vehicles, which are already in operation for several years.

1.6 Impacts

There are many benefits to the industry migrating to a Software Defined Vehicle model, but there are differentiating and non-differentiating challenges with this migration. The first issue is that traditionally the vehicle has been seen as an embedded platform, where the software has been hardware-dependent to a large extent. This means that the software for one system may not be easily transferred to another without potential large up-front engineering costs. In a Software Defined world, anything that adds barriers such dependencies add to overall development costs reduces attractiveness in a competitive market. The second issue arises with the increased complexity of the centralized compute modules. These may include heterogeneous compute islands for application, real-time and microcontroller based workloads, each needing to support different profiles ranging from QM to ASIL-D, from low to high security and from best-effort to hard

real-time profiles, including combinations thereof. The configuration and deployment of workloads to these SoC's has the potential to become very system specific, again reducing the ability for workloads to easily migrate from one system to another. These two primary complexity angles are potential barriers that will challenge the pace at which the industry migrates towards the Software Defined Vehicle. Increasing the pace of migration to a Software Defined Vehicle in automotive will rely on industry agreeing on open standards and methodologies to help manage and mitigate these challenges. The SOAFEE Special Interest Group (SIG) is addressing these challenges with a standardized system architecture specification. This specification will mature and evolve over time as open standards and best known methodologies evolve.

CHAPTER

TWO

ARCHITECTURE OVERVIEW

Automotive software is increasingly complex as more of the final product is expressed in software. This complexity is found at many levels. Automotive OEMs have stated that the cost of developing board support packages and new System on Chip (SoC)/platform integration is unnecessary and very much a non-value added expense. SOAFEE intends to change this by adopting standards in both cloud native and edge computing allowing automotive OEMs to focus on their core competencies and increase the reusability of software. SOAFEE will adopt and enhance current standards used in the cloud-native world today to help manage the software and hardware complexities of the automotive software defined vehicle architecture.

2.1 SOAFEE Architecture Overview

The primary objective of SOAFEE is to define a framework that supports cloud-native development and vehicle edge platform deployment of vehicle applications and functions. The framework allows to integrate different middleware and application Software stacks and focuses on the essential elements for building service oriented architectures in Automotive Use Cases. In addition, the SOAFEE architecture will enable cloud-native development of all workloads including those with functional safety, security, temporal partitioning, spatial partitioning and real-time requirements. These are important characteristics of next generation in-vehicle embedded edge platforms.

2.2 SOAFEE Architecture Guiding Principles

It is essential that real-world, deployable automotive use-cases drive the system architecture requirements for SOAFEE. The SOAFEE System Architecture Working Group (WG) has established a workflow by which automotive use-cases drive the architecture analysis, refinement, and allocation of requirements to the SOAFEE architecture planning process. Long-term, it is the goal of the SOAFEE System Arch WG to analyze and decompose all proposed use-cases to ensure that use-case requirements in the following areas are addressed by the SOAFEE System Architecture:

2.2.1 Safety

It is fully expected that the SOAFEE architecture will support use-cases that execute safety-critical (micro)services alongside non-safety-critical ones. Therefore, the execution platform needs to be reliable and safe to ensure the functional safety of such electrical/electronic (E/E) systems. The application of the ISO 26262 safety standard in the automotive domain helps to ensure the correct functioning of the safety-critical platform elements and to keep the residual errors acceptably low. Since the development of the entire platform compliant to a safety standard is not practical, the strategy is to develop (or qualify) only safety-critical elements according to ISO 26262 and isolate them from the non-safety-critical elements accordingly to ensure spatial, temporal and communication Freedom From Interference (FFI).

2.2.2 Security

Security analysis of all use-case must be a common practice. All implementations shall pass security checks and follow a set of best practices, including:

1. Threat model: Platform and Applications. A threat model for the platform is to be created. All application decomposition should be associated with a bare minimal threat model.
2. Design principle: Principle of least privilege to be followed for the infrastructure (container runtime, orchestrator etc) and the applications.
3. Support of common security services such, but not limited to cryptography, attestation, etc.
4. Robust security APIs to be used by applications. SOAFEE should include a standardized API. Not to define its own but use available standards.
5. Architecture agnostic deployment. It should be possible for applications to consume security services from the platform independent of how they are supported (SW, HSM, TEE etc.).
6. Discoverability of security services and their robustness. E.g., an application may be deployed only if the underlying HW provides crypto service from HSM.
7. Application authentications: How do we verify a deployed container is trusted? How do we verify a container when starting is not modified in storage?
8. Coding guidelines and static analysis. Propose that SOAFEE platform is to follow CERT coding guidelines (where applicable) and run static analysis to avoid vulnerabilities.
9. Security verifications: A basic minimal set of testing covering privilege checks, fuzzing.

2.2.3 Real-Time

SOAFEE is targeting use-cases that will require time constraints on the execution of the workloads. Real-time requirements must be considered from the point where workloads are analyzed and decomposed, throughout the design of the SOAFEE software stack, and finally when considering software / hardware interactions.

CHAPTER

THREE

CHALLENGES AHEAD

Cloud native technologies were not designed to run inside a vehicle. Therefore most of the available cloud native technology is missing several aspects, which are crucial for running inside the car. Multiple challenges are already identified for automotive cloud native.

3.1 Safety certification of cloud native technologies

A majority of the cloud native technologies, which are intended to bring the functionality inside the vehicle, are not safety certified. Often the used programming language itself is not suitable to ensure important features such as memory safety or determinism. Therefore, crucial components such as runtimes or orchestrators may need to be audited or rewritten.

3.2 Compatible safety certified system software

Mixed critical orchestration is about management of applications with different safety requirements. Especially for functions with safety requirements the whole software stack underneath the orchestrator needs to be certified. To date, no automotive safety certified linux with container support is available. On the other hand, several certified OS systems don't support containers.

3.3 Startup

When starting a vehicle, multiple applications get started and requirements of availability are typically in the range of seconds. This implies that you need to have a prioritized order to start containers and the container runtime needs to start 10-100 containers in this time period. This relates to startup times lower than 10-50 ms per container (this presents a challenge for currently available runtimes).

3.4 Real-time constraints & determinism

For automotive functions it is often mandatory to keep timing requirements during execution. Processing of signals chains (e.g., from sensors to applications control units and back to actuators) needs to be guaranteed to ensure important reaction times in order to avoid accidents. For this cloud native technology needs to be enhanced to support:

- Deterministic scheduling
- Deterministic processing of signals / guarantee for latency
- Priority handling

3.5 Automotive networks

The automotive network has time sensitive networking (TSN) enabled to achieve deterministic timing for data communication. Currently TSN is not supported in cloud native technologies. Resource constraints: The high-performance computers inside the vehicle are constrained environments. Containers are only allowed to consume a defined amount of resources (CPU, memory, IO, etc.). This should be well handled by available OS mechanisms but needs to be ensured during the development process.

3.6 Dynamic functions and homologation

A big benefit of an orchestrator is the capability of adding new applications during runtime. On the other hand automotive regulations require a homologation process of software before deployment. To unleash the full potential of orchestration in the vehicle a process to add or update applications dynamically needs to be established.

3.7 Workload Deployment

Automotive applications have different implications, restrictions and lifecycle when it comes to both releasing and deploying a new version of an end user application or a (operating) system level update. Existing DevOps methodologies can be applied to enhance this workflow with some refinement for the Automotive land.

3.8 Workload partitioning and clusters

To enable mixed safety systems often a hypervisor or microkernel is used to partition the hardware resources into domains running several Operating Systems in the car. The orchestrator has the ability to manage applications across separate partitions by adding agent nodes.

3.9 Cloud based Testing

Modern DevOps processes have robust testing infrastructure build into their testing and deployment pipelines. Currently this infrastructure from the cloud-native world lacks the ability to test and validate real-time workloads, or workloads that are targeting real-time or microcontroller based architectures. We need to enhance the cloud based tooling to meet the validation needs of safety and real-time domains.

Addressing these and other challenges ahead will require industry cooperation and collaboration.

CHAPTER

FOUR

DEVOPS IN THE CLOUD, DEPLOY AT THE EDGE

The DevOps methodology is well established for cloud workflows. There is a huge opportunity for cloud development workflows and practices to be re-used in the automotive industry. This is not just about how to do continuous testing/building in the cloud or automated deployments on edge devices but it also implies how end-user applications will be developed. Adopting a DevOps model in automotive implies that most embedded software developers will no longer need to develop, build, and test applications using a host development system alongside a target/edge platform for testing. While it is possible to inherit most DevOps practices from a cloud-based environment, deploying systems and applications on edge devices have its unique set of challenges. A few of these challenges are listed below:

- When and how should application updates be deployed?
- How will the system deal with an update of a running application?
- How to identify if an application has enough resources to run, keeping in mind it is not limited to the usual computing resources but also car features/components.
- How will I test applications in an automated manner in an emulated environment in the cloud.

Essential to support application development in the cloud is the requirement for a base operating system that can run not only in the cloud for testing but must also run on the edge platform for on-device testing. Sticking to well-known cloud technologies such as OCI compatible containers are essential in developing/testing applications in the cloud, allowing the same portable container to also be deployed at the edge for example, in a vehicle. To help facilitate this workflow and provide repeatable examples, the SOAFEE SIG has introduced SOAFEE Blueprints and the SOAFEE Integration Lab.

4.1 SOAFEE Blueprints

SOAFEE Blueprints are example reference application software solutions guided by specific automotive software defined use-cases used to validate SOAFEE architectural concepts and accelerate product development and prototyping. Blueprints are intended to demonstrate concepts such as DevOps in the Cloud and Deploy at the Edge, however it is not a requirement for a blueprint to cover the complete end-to-end DevOps cycle.. This will allow SOAFEE members to validate application-level use-cases and showcase edge deployments.

4.2 SOAFEE Integration Lab

SOAFEE Blueprints will be key to validating the SW defined model defined by SOAFEE. Blueprints will be used to validate the cloud native development, integration and test workflows in the cloud. Once the cloud native development workflow has been developed/matured for the blueprint the next step is to test/validate on the edge compute platforms. This is where SOAFEE Integration Labs will play an important role. This is a lab that will make available supported edge platforms for on-device testing of workloads. These labs provide the hardware to test the "Deploy at the Edge" workflow on embedded edge hardware. The Integration Lab will enable the testing and integration of compliant hardware platforms, both virtual and physical, with SOAFEE compliant implementations such as the the Edge Workload Abstraction & Orchestration Layer (EWAOL), a SOAFEE reference implementation, and stacks from commercial software vendors, with workloads such as Open AD Kit that conform to the SOAFEE architecture as outlined in this document. The combination of the SOAFEE Blueprints and the SOAFEE Integration Lab will create a quick start tool for third parties to build functional solutions quickly and easily on SOAFEE compliant solutions.

CHAPTER

FIVE

ARCHITECTURE

5.1 Overview

Considering the key factors and the motivation for SOAFEE, it becomes obvious that adopting concepts and solutions from clou

- Functionality and interfaces for scaling and orchestrating services should work in the same way in the vehicle and in the cloud and follow cloud native technologies. This however needs to consider the Automotive-specific needs for safety and limited resource footprints.
- Development / Testing / Deployment workflows, which enable service development and testing in a cloud based virtual environment (simulation) before deploying to the physical hardware. We can move more test and validation scenarios to the cloud, if the execution environment in the virtual system is similar or equivalent to the one in the physical embedded system (environmental parity).

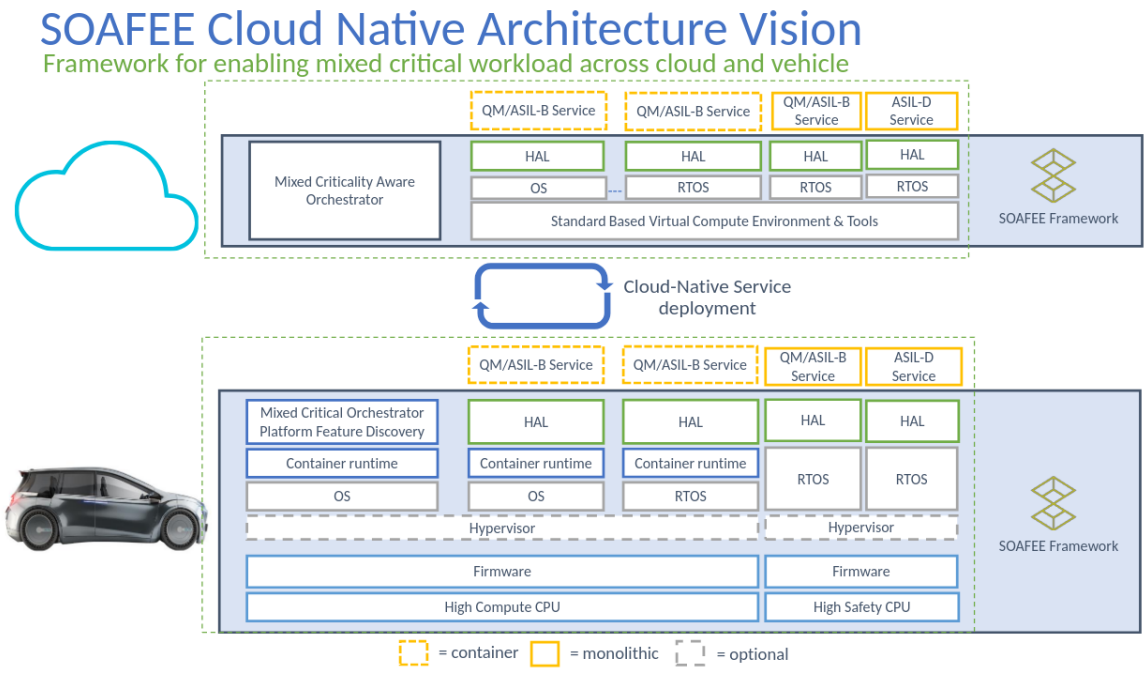

Figure 1 - SOAFEE Architecture Vision

5.2 SOAFEE Software Architecture

At this stage in the SOAFEE journey to support software defined vehicles the SOAFEE software architecture can be represented by the elements in the following diagram.

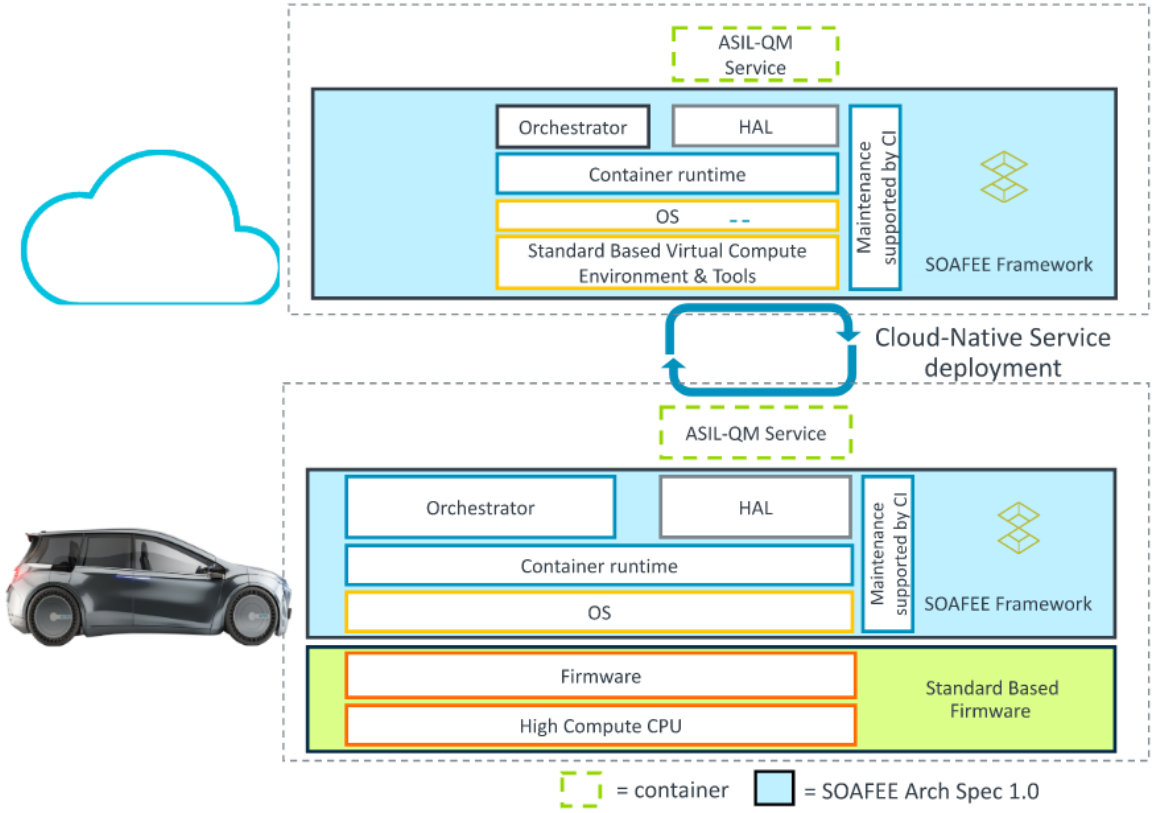

Figure 2 - SOAFEE Architecture v1.0

The required elements for this release of the SOAFEE architecture must include:

- OCI compliant container engine and runtime: OCI Runtime Specification.
- Kubernetes compatible container workload orchestration: Kubernetes API.
- Develop in Cloud, Deploy at the Edge (native cloud development)
- Reference Implementation Maintenance supported by CI: meta-ewaol CI.
- Runs unmodified on Platforms with Standard Based Firmware

The reference implementation of SOAFEE, EWAOL is an expression of this release of the architecture. The release numbers of EWAOL reflect the release of this document.

5.3 Future Work

The following topics need to be detailed in order to continue the adoption of cloud native technologies into the Automotive Domain:

- Observability and Analysis: The paradigm of environmental parity, which underlines one of the central pillars of SOAFEE needs to be proven by dedicated observations and analytics. The major efficiency effect of a virtual development and validation in the cloud can only be leveraged, if we can prove that the results are comparable.
- Mixed Critical Safety Orchestration: The paradigm shift for cloud native in Automotive needs to support the complete range of mixed criticality. Migration of applications towards Microservices requires significant investment. This is only acceptable, if the gains justify it. Harmonizing and simplifying the programming models in safety, non-safety, vehicle and cloud environments therefore is essential. The problem of mixed-criticality comprises considerations on appropriate tooling to allow flexible deployment of safety services during the entire lifecycle of the vehicle.
- Cloud Native Trail: The adoption of Cloud Native in Automotive requires collaboration and contribution of many different stakeholders. It can only be driven by SOAFEE. To indicate a clear and stepwise approach for a migration, a clear trail needs to be described (similar to the cloud native trail map).
- Partitioning Recipes: There are different options to achieve partitioning of compute (Containers, Hypervisor, upcoming technologies e.g. via eBPF). A clear comparison and guidelines are needed for a selection of the best possible solution for a dedicated use-case.

CHAPTER

SIX

HOW TO GET INVOLVED

For more information please go to:

- General Information: https://soafee.io
- Source code repository: https://gitlab.com/soafee
- Join SOAFEE: https://soafee.io/community/join/
- Public calendar: Web view, ICAL