GUARD KNOX

UNTANGLING VEHICLE SOFTWARE WITH AUTOMOTIVE SOA

FREEDOM TO EVOLVE

TABLE OF CONTENTS

ABSTRACT

Software-Defined, Connected and Autonomous Vehicles require a fresh look at software development in the automotive industry. With increasingly complex software, the industry must move away from today's inflexible monolithic blocks of code to a software design methodology based on service oriented approaches.

This White Paper starts by reviewing the history of automotive software development, and it discusses the emerging requirements for automotive software today and in the future.

It explains why today's approaches are not scalable and how Automotive SOA, derived from the existing SOA concept, can support the entire software lifecycle in the automotive industry.

The introduction of SOA in the automotive industry will not happen as a revolution, but in evolutionary steps compatible with today's processes and legacy models.

The benefits for drivers and passengers as well as OEMs, Tier 1s and the rest of the supply chain are clearly identified.

INTRODUCTION

The automotive industry is experiencing a dramatic paradigm shift. Where previously vehicles were built with complex hardware-oriented designs, new models are designed as Software-Defined Vehicles (SDV) with a software-oriented design. The majority of functions previously implemented in hardware (HW) or with simple electronics are increasingly implemented using software (SW) on computing platforms with different performance and cost positions.

This paradigm shift is driven by changing consumer demands and advances in technology. Modern drivers want their car to perform like their smartphone with flexibility for SW customization and connectivity to the surrounding environment. Technological advances in Advanced Driver Assistance Systems (ADAS) have been introduced to relieve the driver of some of the boring tasks while in stop-and-go traffic or on long highway drives. This trend will continue pushing to the industry goal of the introduction of autonomous vehicles.

Part of this evolution is the emergence of the Connected Vehicle which communicates closely with its environment, to increase driving safety and efficiency, to receive entertainment content, to support its passengers' multimedia communication with the outside world, and to facilitate remote maintenance of the vehicle. Today's vehicle architectures only support these new requirements to a small degree. The automotive software architecture and the development processes have evolved over a long period, without the stringent efficiency standards of the non-automotive SW industry. Automotive SW is still created and deployed as monolithic blocks of code that are very difficult to customize and to maintain. For every change all potential side effects have to be taken into consideration which makes the process of software development, testing, deployment and maintenance slow and error-prone.

In order to overcome these inefficiencies automotive SW development has to adopt the concept of a Service-Oriented Architecture (SOA) that was introduced in enterprise SW long ago. All the functions inside a vehicle are defined as services - Software Components (SWC) that can be implemented and tested individually and independently. They communicate across a virtualized communication infrastructure using well-defined interfaces.

GuardKnox has developed a secure Automotive SOA framework to support the automotive industry in its move towards a service-oriented SW design methodology. This framework will facilitate the entire SW lifetime process from architecture definition, over development, test, deployment, all the way to continuous SW maintenance and upgrades after a car has left the production line.

EVOLUTION OF AUTOMOTIVE FUNCTIONS

History

Cars have historically been designed as complex systems of sophisticated hardware (HW) with a small amount of software (SW) to control their (relatively) few electronic components.

Some of the first electronic functions deployed in cars provided increased comfort for drivers and passengers with features such as power windows, air conditioning systems, interval control of windshield wipers and central lock systems.

Requirements for more efficient engines led to the introduction of electronic ignition and digital engine control, and the need for improved driving safety led to ABS (Anti-lock Braking System) and ESC (Electronic Stability Control) systems.

The demand for more safety functions and comfort led to the introduction of numerous additional devices, like distance sensors for parking or emergency braking, cameras to watch the surroundings of the vehicle, read speed limit signs, or to stay on track.

The standard car radio morphed into a highly sophisticated infotainment system that not only provides entertainment, but also includes navigation, external communication, and convenient control of many vehicle functions.

The evolution towards connected and autonomous vehicles will lead to more and more electronic functions being implemented in software on computing platforms in the vehicle with increasing performance.

The Path to Autonomous Vehicles

The Society of Automotive Engineers (SAE) has defined 6 autonomy levels on the path towards autonomous driving with increasing reliance on electronic functions. Levels 1 and 2 are already widely available today.

Level - 0 No Driving Automation (manually controlled): The human provides the dynamic driving task although there may be systems in place to help the driver. An example would be the emergency braking system—since it technically does not drive the vehicle, it does not qualify as automation.

Level - 1 Driver Assistance: Cruise control is an automated system for driver assistance, keeping a car at a constant speed. Adaptive cruise control, where the vehicle is kept at a safe distance behind the next car, qualifies as Level 1 because the human driver monitors the other aspects of driving such as steering and braking.

Level - 2 Partial Driving Automation: With Advanced Driver Assistance Systems (ADAS) the vehicle can control both steering and accelerating/decelerating. This is not considered self-driving because a human sits in the driver's seat and can take control of the car at any time. Many modern cars have ADAS systems implemented that can keep them in the lane and keep a safe distance to the car in front. This relieves the driver in common stop-and-go driving situations.

Level - 3 Conditional Driving Automation: Level 3 vehicles have environmental detection capabilities and can make informed decisions for themselves, such as accelerating past a slow-moving vehicle, but they still require human override. The driver must remain alert and ready to take control if the system is unable to execute the task.

Level - 4 High Driving Automation: The key difference between Level 3 and Level 4 automation is that Level 4 vehicles can react if things go wrong or there is a system failure. Thus, these cars do not require human interaction in most circumstances. However, a human still has the option to manually override. Level 4 vehicles can operate in self-driving mode. But until legislation and infrastructure evolves, they can only do so within a limited area (usually an urban environment where top speeds reach about 50km/h). This is known as geofencing.

Level - 5 Full Driving Automation: Level 5 vehicles do not require human attention—the dynamic driving task is eliminated. Level 5 cars will not even have steering wheels or acceleration/braking pedals. They will be free from geofencing, able to go anywhere and do anything that an experienced human driver can do.

From the description of the levels it is obvious that a large number of additional distance sensors and cameras are required to analyze the vehicle's environment. Enormous computing power is needed to process all the inputs and make decisions for steering, accelerating/decelerating and braking. Gradually, Artificial Intelligence (AI) mechanisms will be introduced.

Autonomous vehicles do not only analyze their environment, but also communicate with it. They talk to other vehicles, traffic lights, pedestrians and more. Communication architectures between vehicles and their environment are summarized under the term V2X (Vehicle to everything). This will be discussed in more depth below.

The Connected Vehicle

Communication with the environment has to be very secure. False information can cause a vehicle to make wrong – and potentially fatal – decisions. Wireless networks to supply this information will never be 100% available in every locale. Therefore, externally supplied information can only serve as context information to improve decision-making, but an autonomous vehicle's safety must not rely on this information, but on its own sensors and stored maps.

Consumers are used to downloading and installing apps on their computers, smartphones or tablets, customizing their devices to their preferences. For the generation who grew up with customization as a basic offering in most consumer-facing software, it is an expectation to be able to customize their cars in the same ways. They want to add new functions (e.g. new ADAS functions, games, videoconferencing SW, additional performance (horse power), etc). The physical infrastructure that would allow greater interactivity and communication similar to personal computers or smartphones (e.g. screens, powerful computing platforms and external connectivity) is often available in modern cars. However, the SW architecture of today's vehicles typically does not support using the hardware for additional purposes beyond specific predefined use cases decided by the automakers.

Expectations for car customization will grow with the continued progress towards Level 5 autonomous vehicles. Their interior will resemble a living room or home office environment instead of today's vehicle interior or a cockpit. Audio and video-based entertainment functions, video conferencing and professional workstation features will require support via large screens and high-speed wireless external communication.

Nowadays, the majority of cars are shared only amongst family and perhaps friends, but this is going to change with commercial car sharing becoming much more popular, in particular autonomous ones. Cars will be easily adapted to meet the preferences of any user. Some users want to be entertained while commuting in their vehicle while others have different preferences such as working or resting while travelling. Portable digital identities or profiles will customize the car to a particular user as soon as they log their profile into the car.

Software Updates

An unfortunate reality of software development is that software in any computing environment can always contain bugs. The number of residual bugs typically grows as the size of the SW packages increases. We have grown accustomed to regular software updates to correct bugs and add new functionality in our everyday environments and devices, but not for our cars.

If a bug is severe enough to impact driving or functional safety, automakers traditionally initiate a very expensive and reputation-impacting recall program to get the software fixed in an authorized garage with a physical connection to the vehicle - usually only happening as a last resort.

The standard Over-the-Air (OTA) updates approach used in our smartphones and computers is not available in a vehicle today where it could be abused as an entrance to safety-critical functionalities. However, as the SW complexity grows, OTA updates for cars will become mandatory because of the higher frequency of required updates, and to support upgrades for customization.

HISTORY OF AUTOMOTIVE SW DEVELOPMENT

How Did We Get Here?

In early vehicles, there was a simple 1:1 relationship between sensors and actuators. A switch was connected via a direct cable to the lights (sometimes using a relay in between to increase the allowed current), the ignition lock was connected directly to the starter relay. Today, this is no longer the case as drive by wire sub-systems proliferated. For example, the acceleration pedal does not mechanically control a carburetor. Instead, it has a position sensor that provides its information to the engine control electronics, which in turn, combine it with information from ADAS systems. Direct electronic control of the brakes, beyond ABS, will likely be approved in the near future as well.

With more and more functions and sophistication that have been introduced into vehicles the complex interconnection of all the sensors and actuators no longer allows any direct wiring in the vehicle. Functions are now performed by Electronic Control Units (ECUs), computing platforms using either microcontrollers (MCUs) or microprocessors (MPUs), depending on the performance requirements. In any modern car, more than 100 ECUs are distributed across the vehicle with functionalities determined by their respective applications. These ECUs are connected to centralized control systems (gateways and domain controllers) through digital communication media.

ECUs can be as non-critical and rarely used as those for seat adjustment, while ECUs for brakes or steering are highly safety-critical and have to operate deterministically.

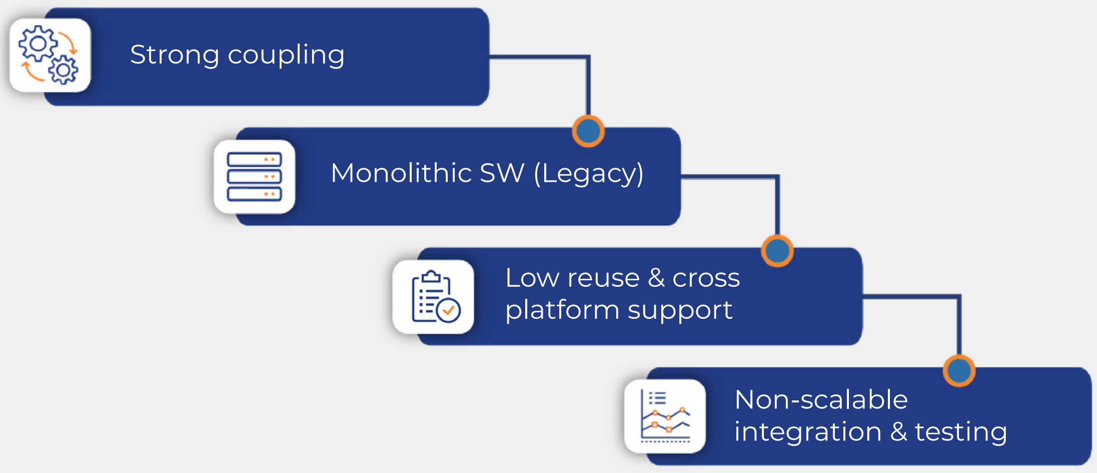
The trend to connected and autonomous vehicles means that a lot more computing platforms, interconnected sensors and actuators will be required for optimal functioning and performance.

When ECUs were first introduced SW development was seen as a necessary evil to control these devices. More and more ECUs enabled more functions, and vehicle functionality became largely SW-defined.

With the number of ECUs in a car exploding, SW development became a key capability for OEMs and Tier 1 suppliers who split their responsibilities roughly between overall architecture and specific implementations of ECUs and domain controllers. All the SW components (SWCs) are then integrated by the OEM, tested, and deployed on the production line.

Where is the Problem?

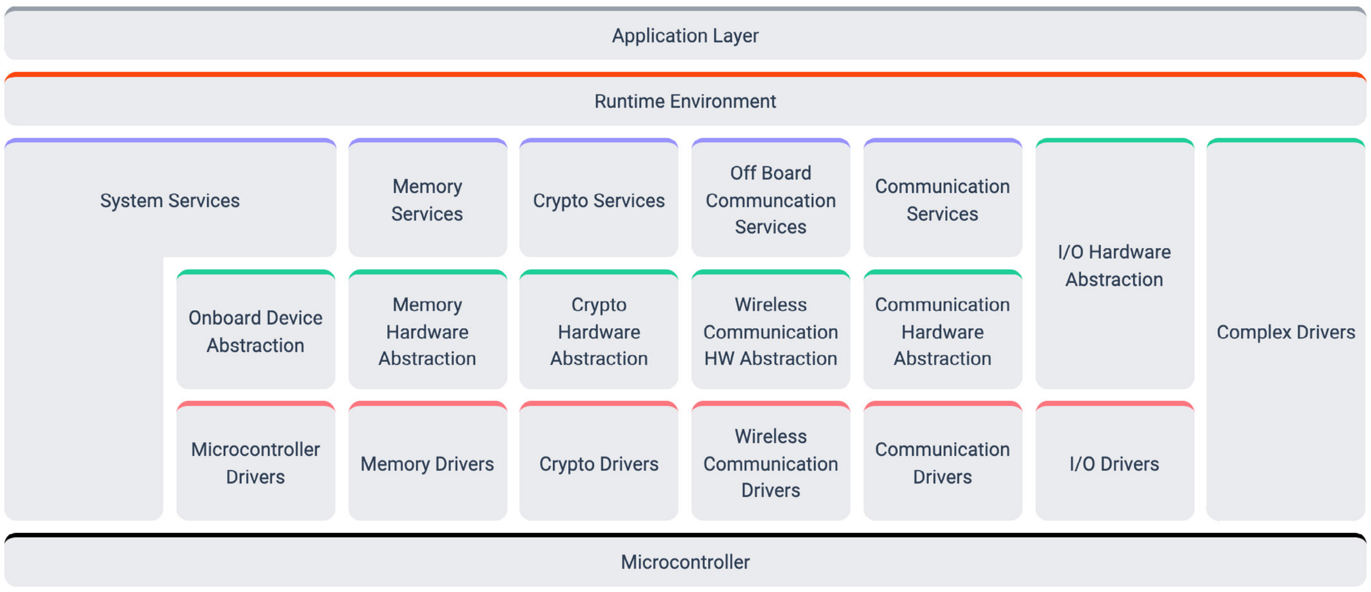
Initially the SW in centralized or domain controllers was directly addressing peripheral ECUs by their network addresses and the logical interfaces determined by the makers of the ECUs. ECU addresses had to be predetermined and hard-coded into the SW. As the complexity of the system grew, it became more error-prone and every modification caused numerous side effects that became more and more complex to resolve.
This strong coupling between software and hardware components has led to the monolithic SW system. Whenever one SWC is changed a complete re-integration and re-testing of the entire system is required rather than just testing this single SWC, to make sure no unexpected side effects will occur.
Customizing the vehicle SW to specific customer desires is virtually impossible in this system as it would require high cost for development and deployment on the production line or once the car is already on the road. Software updates or aftermarket enhancements are very complex and have to be deployed physically in the shops of authorized dealerships, after significant development, integration and testing by the OEM that might take many months.
Every SWC is written for a specific SW system on a specific computing platform. Therefore, reuse of SWCs and cross-platform support is difficult to achieve and the same function has to be implemented over and over again if required in a different context.
With the increasing complexity of SW systems, we have reached the scalability limits for integration and testing with this legacy approach.


Strong coupling
Monolithic SW (Legacy)
Low reuse & cross platform support
Non-scalable integration & testing

Waterfall of problems in current SW development process

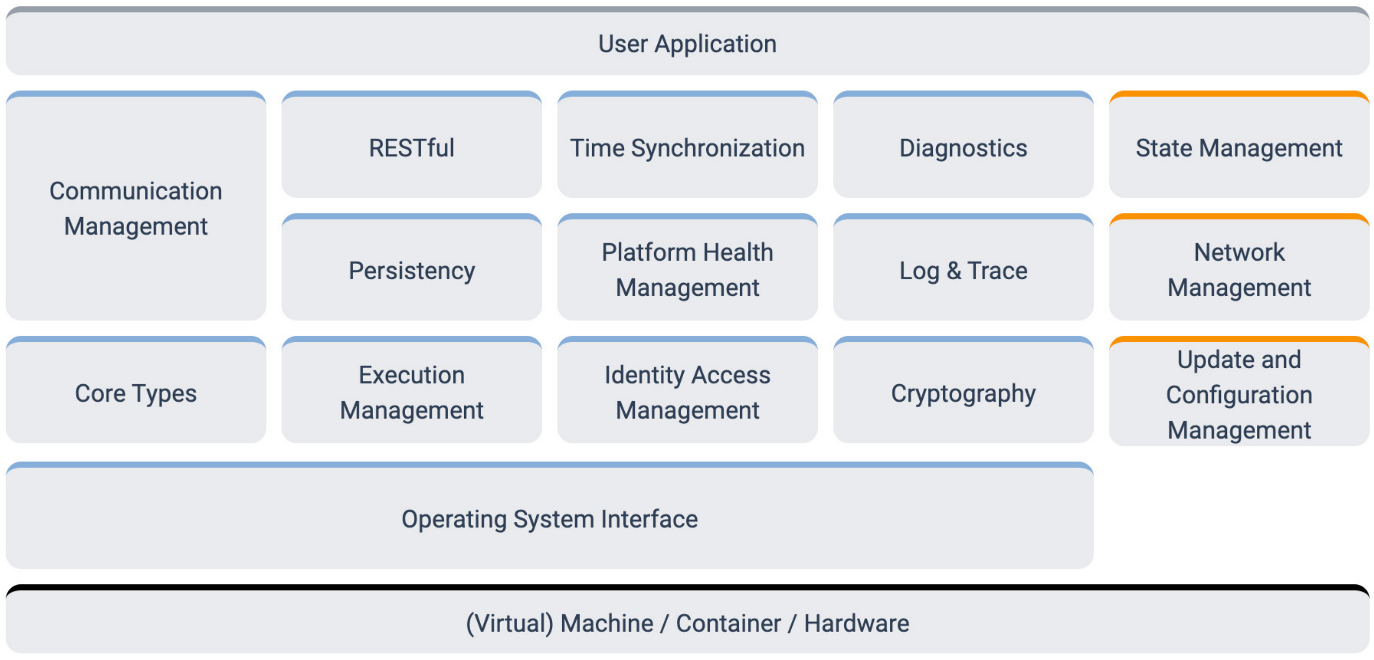
What Has Been Done Until Now

Starting in 2004, the automotive industry developed an approach to create some level of abstraction for the ECUs' interfaces through the AUTOSAR consortium. The AUTOSAR Classic definitions have allowed a certain standardization and complexity reduction, but the SW system in a vehicle remains a monolithic block of code.



AUTOSAR Classic structure (From AUTOSAR website)

Recognizing the fact that the increasing SW complexity requires a more modular approach, AUTOSAR published its first version of AUTOSAR Adaptive in 2017. It has been an attempt by the automotive industry to introduce a service-oriented architecture.



AUTOSAR Adaptive structure (from AUTOSAR site)

Its core is an operating system based on the POSIX standard. The operating system can be used by applications via a subset of POSIX. The communication protocol used for the in-vehicle networking using the Adaptive Platform is SOME/IP on top of Ethernet.

Two types of interfaces are available - services and application programming interfaces (APIs). The platform consists of functional clusters which are grouped into services and the AUTOSAR Adaptive Platform foundation.

Adaptive Platform services include:

- Update and Configuration management
- State Management
- Network Management
- Diagnostics

The AUTOSAR Adaptive Platform contains both specification and code. In comparison to the Classic Platform, AUTOSAR Adaptive develops implementations to shorten the validation cycle and illustrate the underlying concepts. These implementations are available to the AUTOSAR partners.

While AUTOSAR adaptive can be considered a step into the right direction, to reach a truly Service-Oriented Architecture, much remains to be done and will be discussed in later sections.

COMMUNICATION REQUIREMENTS

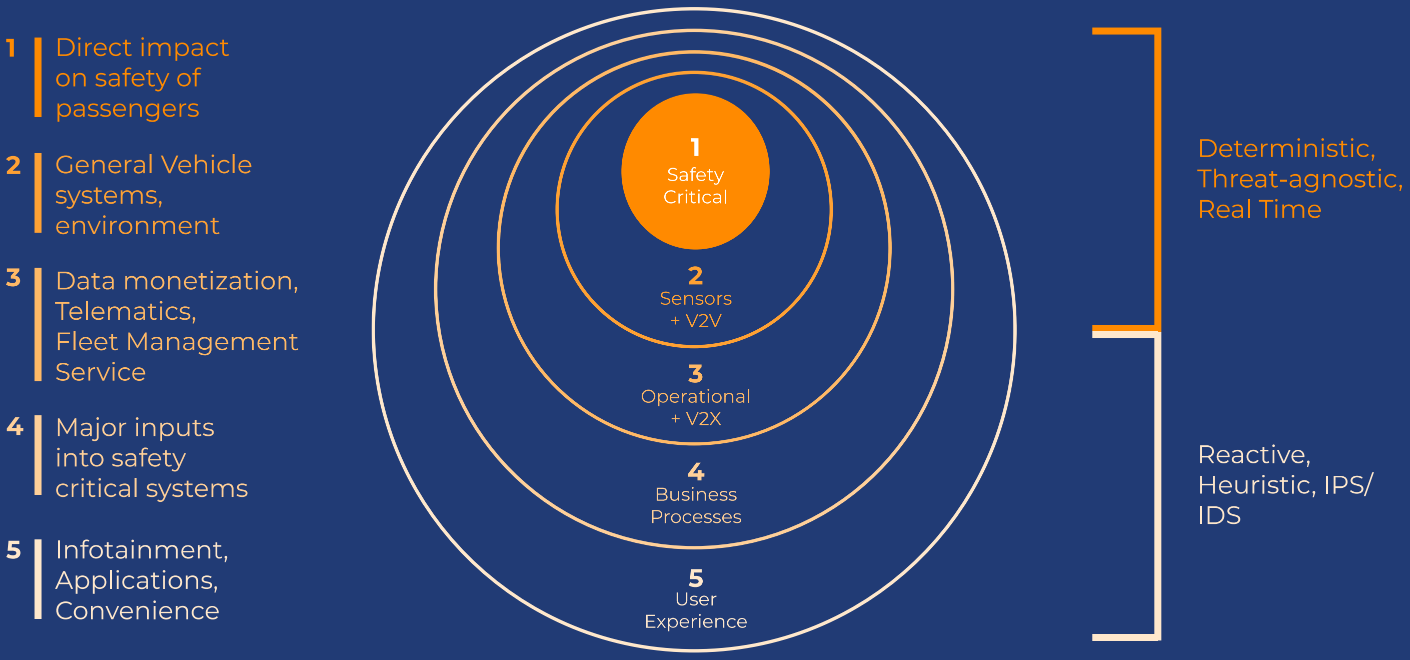
Internal Communication

Communication between the ECUs in a vehicle today is largely based on legacy communication networks, of which CAN is the most popular. CAN versions exist with different speeds. HS (High-Speed) CAN provides a gross data rate of 1 Mbps, LS (Low-Speed) CAN of 125 kbps, and CAN FD (Flexible Data rates) of 2 or 5 Mbps. While these speeds are sufficient for many applications they are not enough for use-cases like transfer of uncompressed video which is important in the context of autonomous vehicles. High data rates are also required to achieve low latency for time-critical applications, like ADAS functions. Furthermore, the device addresses on a CAN bus have to be defined up-front and have to be hard-coded in the vehicle SW.

The automotive industry has started to replace CAN and other legacy automotive communication systems with Automotive Ethernet, which is available in speed classes from 10 Mbps up to 10 Gbps and beyond, and in various topologies, like star and bus arrangements. However, there are still many legacy ECUs that continue to be used within cars in production. Any new SW architecture, therefore, has to support these legacy ECUs and their communication mechanisms. This involves gateways to CAN, mapping message formats, abstracting their physical interfaces and treating their communication with the adequate levels of security.

Autonomous vehicles will rely strongly on the integrity and proper function of ECUs for safety-critical services, like steering, braking, acceleration/deceleration. In order to protect these components from being compromised by faulty or malicious SWCs, their communication has to be protected by sophisticated security mechanisms.

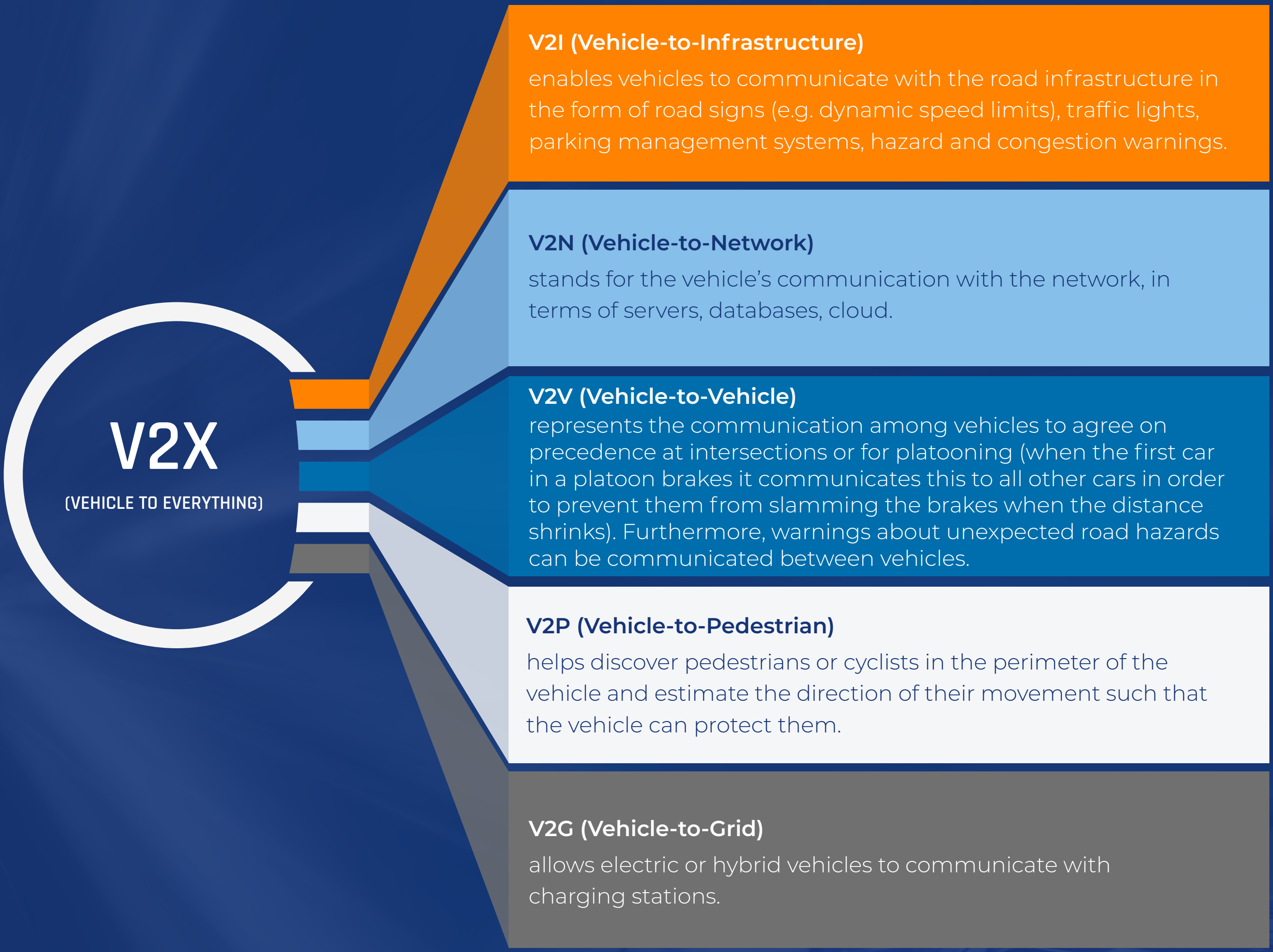
Layered Security Approach


1 Direct impact on safety of passengers
2 General Vehicle systems, environment
3 Data monetization, Telematics, Fleet Management Service
4 Major inputs into safety critical systems
5 Infotainment, Applications, Convenience
1 Safety Critical
2 Sensors + V2V
3 Operational + V2X
4 Business Processes
5 User Experience
Deterministic, Threat-agnostic, Real Time
Reactive, Heuristic, IPS/ IDS

FREEDOM TO EVOLVE

External Communication

Driving safety, traffic efficiency, energy savings, and convenience in connected cars will be improved by providing vehicles with information about their environment. Communication architectures between vehicles and their environment are known as V2X (Vehicle to everything). Communication mechanisms include cellular mobile radio, WLAN, as well as some dedicated radio interfaces and protocols, depending on the location of the vehicle.



V2I (Vehicle-to-Infrastructure)
enables vehicles to communicate with the road infrastructure in the form of road signs (e.g. dynamic speed limits), traffic lights, parking management systems, hazard and congestion warnings.

V2N (Vehicle-to-Network)
stands for the vehicle's communication with the network, in terms of servers, databases, cloud.

V2V (Vehicle-to-Vehicle)
represents the communication among vehicles to agree on precedence at intersections or for platooning (when the first car in a platoon brakes it communicates this to all other cars in order to prevent them from slamming the brakes when the distance shrinks). Furthermore, warnings about unexpected road hazards can be communicated between vehicles.

V2P (Vehicle-to-Pedestrian)
helps discover pedestrians or cyclists in the perimeter of the vehicle and estimate the direction of their movement such that the vehicle can protect them.

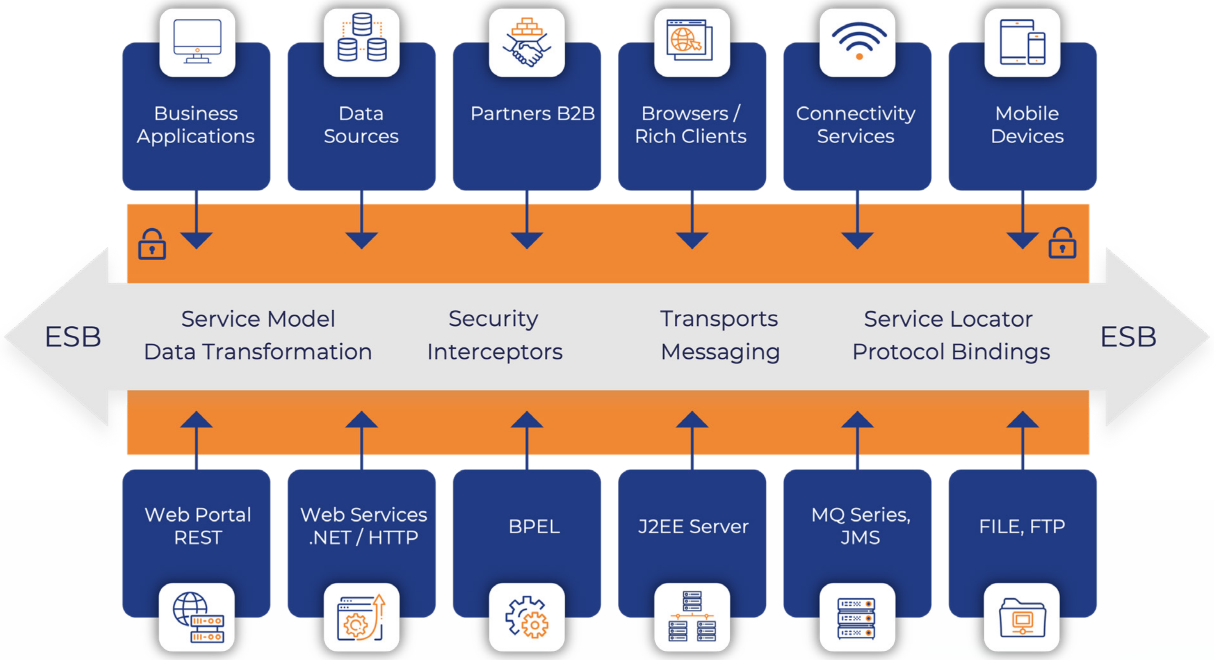
V2G (Vehicle-to-Grid)
allows electric or hybrid vehicles to communicate with charging stations.

Furthermore, external communication is required for Over-the-Air (OTA) download of SW updates, 3rd party applications, or map updates, or simply for web browsing. Streaming or downloading entertainment content might use the same mechanisms or its own cellular modem connectivity.

THE SOA CONCEPT AND VIRTUALIZATION

The Service Oriented Architecture (SOA)

Years ago enterprise SW was faced with the problem of large numbers of different applications, each with their own specific interfaces, OS requirements, programming languages, etc. Integrating these applications into a common system became very difficult and involved complex translations of input and output data. Interactions between applications had to be explicitly programmed. A solution to the problem became known as Service-Oriented Architecture (SOA). It is based on the concept of self-contained services that can communicate with other services through an Enterprise Service Bus (ESB), a virtual bus structure interconnecting all the services. The ESB is a significant part of the middleware that forms the basic SW for the integration of services into an overall system.



General SOA concept

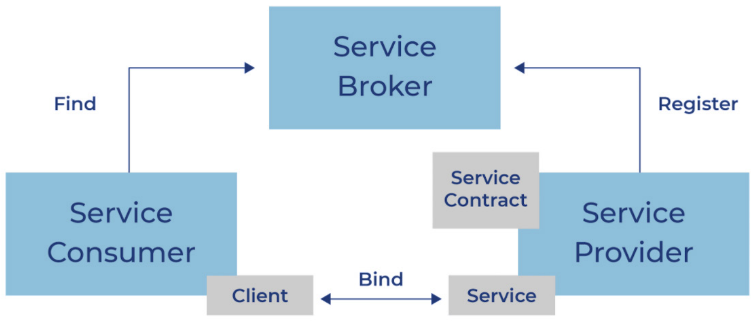
A service has four properties:

1. It logically represents a repeatable activity with a specified functionality and outcome.
2. It is self-contained.
3. It is a black box for its consumers, meaning the consumer does not have to be aware of the service's inner workings.
4. It may be composed of other services.

FREEDOM TO EVOLVE

Each SOA building block can play any of the three roles:

1. Service provider: It creates a service and provides its information to the service broker registry.
2. Service broker, service registry or service repository: Its main functionality is to make the information regarding the service available to any potential requester.
3. Service requester/consumer: It locates entries in the broker registry using various find operations and then binds to the service provider in order to invoke one of its services.



SOA building blocks and their relationships

An example for the specification of an object / service-oriented middleware is CORBA (Common Object Request Broker Architecture). Its core is the Object Request Broker (ORB) and an Interface Definition Language (IDL) for the interface definition between objects/services and the middleware.

Virtualization

Another important concept is virtualization. It has been introduced in the context of data centers where an increasing amount of SW is hosted today. Applications are usually implemented on the basis of various operating systems. For commercial applications these are mostly different versions of Windows and Linux.

For efficiency reasons multiple applications have to run on the same server, but each application typically expects to run on its own computing platform, along with its OS. Therefore, servers have to be virtualized. This means that a piece of SW, called a Hypervisor, presents a virtualized version of the CPU to the different OSs which, in turn, support the respective applications. In this way, applications can be moved flexibly across the servers within a data center without impacting the applications. OSs and applications can be updated individually, independent of other instances.

FREEDOM TO EVOLVE

AUTOMOTIVE SOA

Requirements

For automotive SW development, we require the following capabilities:

1. **HW abstraction** - masks the complexity of the hardware with all the sensors and actuators below an abstraction level and allows applications to find them in a directory instead of programming their interfaces into the code base, essentially decoupling the software from the hardware.

2. **Flexible Service Allocation** - allows applications to run on any computing platform / ECU that has the necessary performance characteristics. The distribution of applications to ECUs does not have to be pre-determined during the design phase. If additional or more demanding applications are required, existing resources in the car can be reallocated to match the computing requirements. If this is not enough, then further ECUs can be added or existing ECUs can be replaced with higher performance models during the lifetime of the car. In case of HW failures, critical applications can be re-deployed to remaining functional compute platforms providing resilience and self-healing qualities.

3. **Consolidation** - consolidates multiple application-specific ECUs into fewer higher-performance generic ECUs that act as computing platforms, thus reducing overall cost and complexity, and adding High Performance Computing (HPC) capabilities.

4. **OTA update / upgrade** - allows to update / upgrade the SW during the lifetime of the vehicle, either by an authorized dealership or a third party, over a wired connection or OTA, using WLAN or mobile radio.

5. **Security** - provides the necessary level of security for every communication inside and outside of the vehicle, and for every installation of an application.

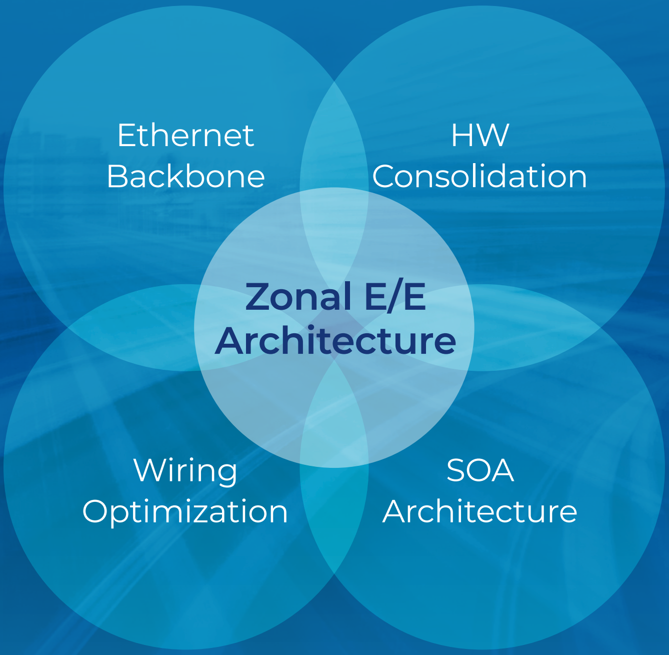
6. **Service orientation** - defines all the functions in a vehicle as services provided by certain devices or applications and consumed by other services. Services can be combined to provide a required function, feature or capability.

AUTOMOTIVE SOA

The Architecture

The Automotive Service-Oriented Architecture is also part of the evolution towards a Zonal E/E Architecture which brings:

1. **Hardware Consolidation** - Consolidating multiple functions that are today served by separate ECUs into ECUs that are multi-functional and provide HPC functionality.
2. **Ethernet Backbone** - Moving from the legacy bus architectures to a modern high-speed Ethernet communication network.
3. **Wiring Optimization** - Consolidation of ECUs together with new topologies for the vehicle networks to reduce the needed cabling length, weight, and cost to a fraction of what it currently is.
4. **Software-Driven Service-Oriented Architecture** - An evolved modular software architecture to accommodate the needed flexibility, security and agility for the new software-defined cars.



Zonal E/E Architecture

Considering the nature of today's automotive SW development it is quite obvious that a service-oriented approach will greatly facilitate development, testing, deployment and maintenance of automotive SW. GuardKnox has developed a framework of components to build an automotive middleware. This middleware provides an ESB among the services that represent the applications. It decouples the individual SWCs from the underlying HW, thus enabling SW portability / SWC 'lift and shift' inside a vehicle's architecture.

Automotive SOA takes the fundamental SOA concept of service providers and service consumers and translates them to the automotive software environment. Every ECU uses layers of abstraction to hide the complexities of network topology, communication, and HW implementation. Interactions between software components are therefore no longer hard-coded. Every new service is represented in a directory from where its functions and services can be offered and where it can find the services that it needs to consume. Binding between services across the ESB is not fixed and can be changed dynamically.

Introducing the SOA approach in the automotive industry enables an unprecedented degree of modularity. Services can be implemented and tested independently. Any change in a service implementation does not require renewed integration and testing of the entire software system as long as its interfaces do not change.

With automotive SOA a SW architect defines the services and the parameters to be exchanged through the standard interfaces using the language OMG IDL (Interface Definition Language defined by the Object Management Group). A tool then creates descriptions and code templates for each service. This description is called a manifest. The developer of a certain service takes the manifest and completes it with the code representing the logic of the service, like reading a sensor, processing input values, invoking an actuator, transmitting information or alerts, etc. The resulting SWC is then fed into the code repository and made available to other services through the service directory. Since the interfaces of the SWCs have been created automatically and uniformly, the communication with other services is guaranteed. If during integration tests an SWC shows bugs it can be corrected and rebuilt without impacting any other SWC.

The SOA Framework creates a uniform SW environment, similar to today's common PC or mobile platforms where new or updated applications do not have to be tested against all other applications. As for PCs, smartphones or tablets, with automotive SOA developers can build applications in an abstract manner, without platform constraints. This greatly simplifies the development process and enables multiple developers to work largely independently on a common project.

Multiple different operating systems can run in parallel on each ECU, supported by a hypervisor. Popular OSs like Windows, iOS, Android, etc. can be supported to enable the installation of standard applications and 3rd party apps on microprocessor-based ECUs.

Generic Framework

The below figure displays the generic framework of the Automotive SOA approach. On top of the HW the Secure Separation Kernel (SSK) is located. A separation kernel is defined as a SW layer which creates an environment that is indistinguishable from that provided by a physically distributed system. It must appear as if each regime is a separate, isolated machine and that information can only flow from one machine to another along known external communication lines. The secure separation kernel adds sophisticated security features to the separation functions.

On top of the SSK is the hypervisor that creates a Virtual Machine (VM) environment decoupling the SW environment from the HW. For this purpose it emulates a HW platform on which multiple guest partitions with their respective Operating Systems (OSs) can run. This means that for a Guest OS it is indistinguishable whether it runs on top of a hypervisor or directly on top of a processor HW. Every guest partition contains a SOA Node Manager on top of the OS which manages this particular partition or node. Every application contains an SOA Port which is responsible for the unified communication (ESB) of the SWCs.

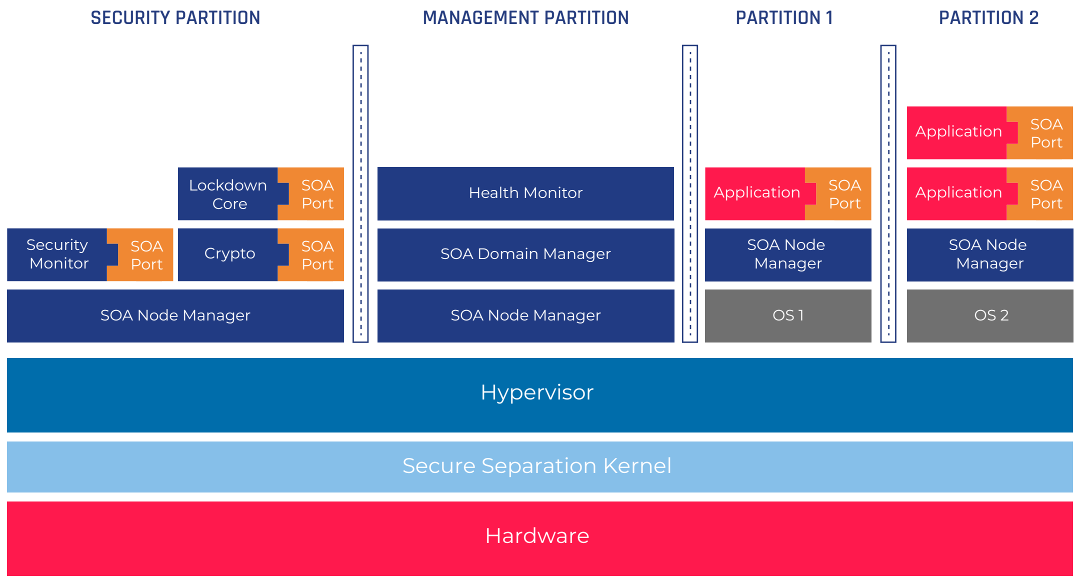
There are two additional partitions which are part of the SOA framework:

- **The Management Partition** with the SOA Domain Manager and the Health Monitor
- **The Security Partition** with Security Monitor, Crypto Module and Lockdown Core Applications

A SWC can be manually and automatically deployed or redeployed, by the SOA framework, into a compatible partition. When the SWC's source was compiled using a supported compiler for a deployment target compatible OS, it can be seamlessly shifted between partitions.

A POSIX-compliant SWC can be activated on top of any OS and hypervisor which is POSIX compliant.

Access to the virtualized communication infrastructure (ESB) is seamlessly mediated through the SOA framework. This infrastructure allows for virtualized RPC and data exchange.



Generic GuardKnox SOA framework

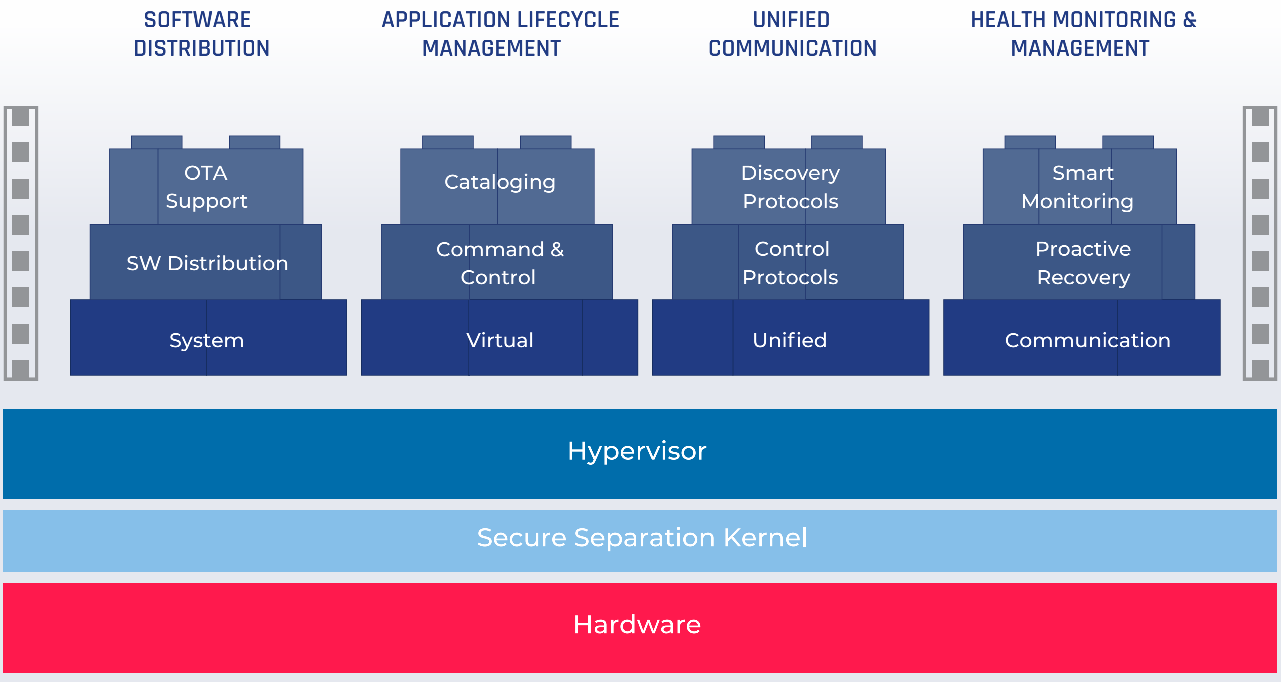
The SOA Domain Manager has four responsibilities:

1. **Software Distribution** is responsible for bringing new SW components into the different ECUs. The OTA Agent receives a SWC through secure communication. The Software Verifier / Activator verifies and activates the SWC, potentially involving a license from a cloud management server. From the SW repository the deployment location is determined and the SWC sent to the right partition. Within the partition the SOA Node Manager initializes the SWC and starts its service.

2. **Application Lifecycle Management** employs an Applications Catalog listing all available Services, and optimizes their locations and communications. The decisions about the SW life cycle – from deployment to uninstallation –including all the operations which are supported by the framework use AI mechanisms and are performed in real time.

   - Install
   - Instantiate
   - Initialize
   - Start
   - Stop
   - Shutdown
   - Teardown
   - Uninstall

3. **Unified Communication** is in charge of the communications infrastructure represented by the Enterprise Service Bus (ESB), based on CORBA, DDS, SOME/IP. It is aware of the network connectivity, along with the paths to all the different ECUs.
4. **Health Monitoring and Management** guarantees the proper functioning of the entire system. It monitors the flow of messages in the system, initiates recovery actions in case of failures and logs events.



Management partition

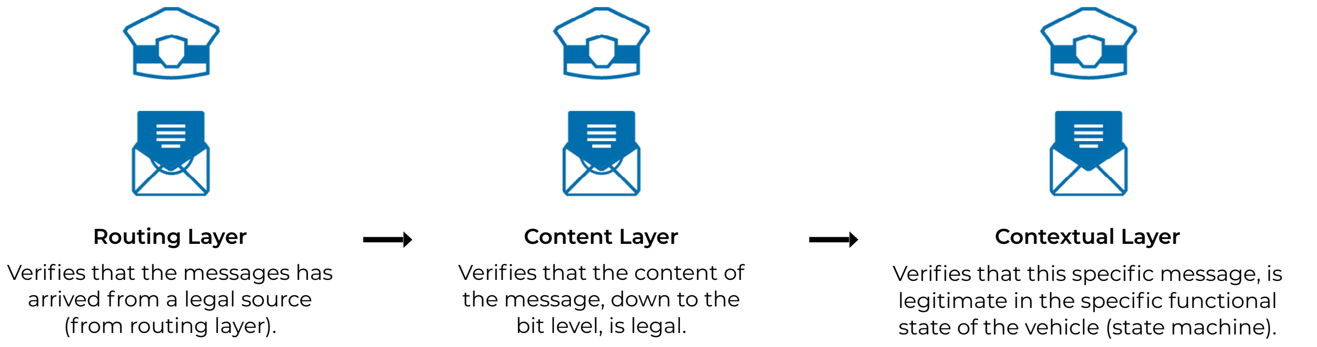
FREEDOM TO EVOLVE

Security

A connected vehicle's safety depends strongly on its secure internal and external communication. Therefore, communication between services is restricted to mandatory interactions. Everything else is blocked. Highly secure OTA download and deployment of SWCs prevent malicious or accidental attack vectors from impacting the safety of the car.

A unique feature of the GuardKnox SOA Platform is its patented Communication Lockdown™ approach for providing holistic vehicle cybersecurity. Using the vehicle's communications matrix and OEM's specifications of the vehicle, GuardKnox builds a state machine that is used to inspect activity on three layers:

- **Routing Layer** - verifying that messages originate from the appropriate network or sub-segment
- **Content Layer** - verifying that message content is permissible down to the bit level
- **Contextual Layer** - verifying that each message is legitimate within the context of the vehicle's specific functional state (e.g., braking and accelerating at the same time)


Routing Layer
Verifies that the messages has arrived from a legal source (from routing layer).
Content Layer
Verifies that the content of the message, down to the bit level, is legal.
Contextual Layer
Verifies that this specific message, is legitimate in the specific functional state of the vehicle (state machine).

The three layers of inspection ensure that if the external vehicle network is compromised by a message from an external source, the internal vehicle network remains fully protected from the propagation of malicious activity.

The Communication Lockdown method is agnostic to known, unknown and future cyber attacks since the proper behavior of all messages has been fully modeled by the communications schema and certified by the OEM.

This also enables the solution to become fully autonomous after installation and operate deterministically without the need for frequent software or firmware updates— unlike traditional Intrusion Detection / Intrusion Prevention Systems (IDS / IPS) or firewalls. This approach removes the possibility of an undetectable hidden backdoor that is possible in all AI/ML based approaches.

If the OEM changes the vehicle's technical specifications or its configuration, a new Communication Lockdown™ schema can be generated, certified and installed via secure OTA update or via a wired connection (such as the standardized OBD port).

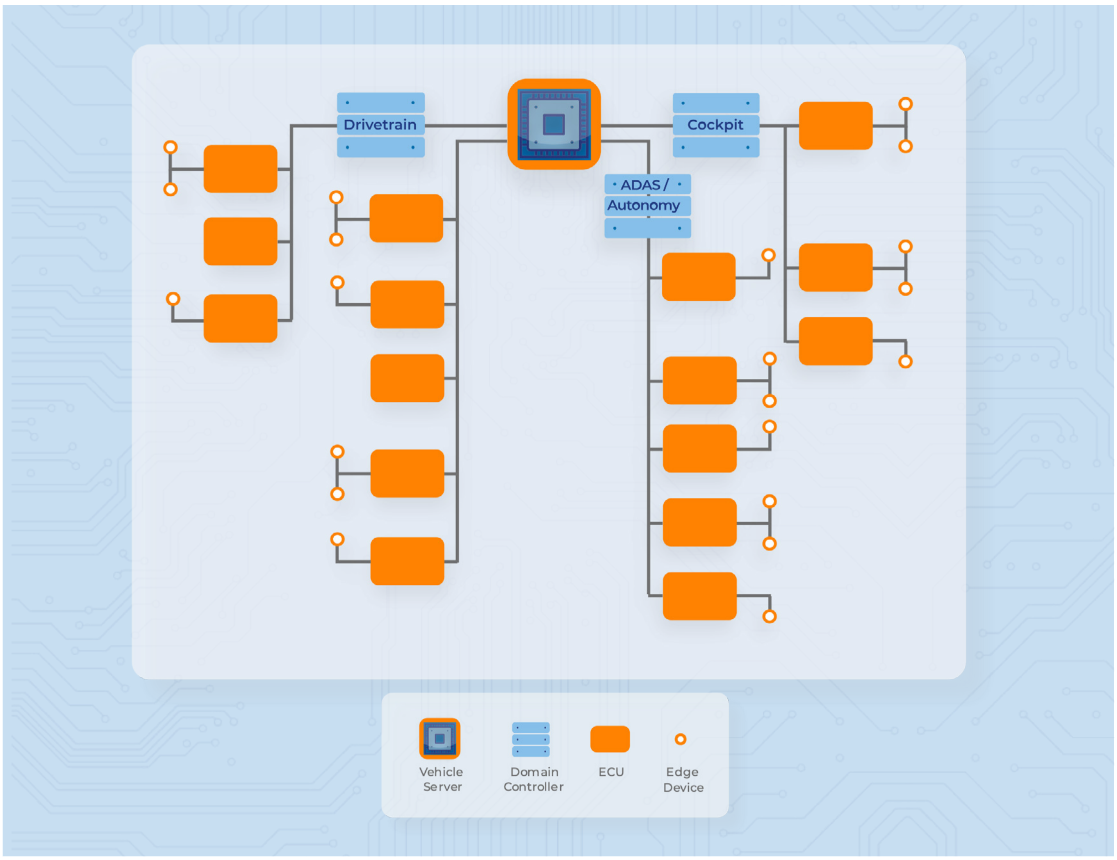
For the strongest protection of the vehicle's SW system a domain controller can be designed which runs on a dedicated HW featuring all the necessary external communication interfaces. Using the Communication Lockdown method, it examines all the communication with the environment and blocks any unauthorized traffic. It supports the OTA download of updates and upgrades to the vehicle SW and of 3rd party apps, and it ensures they are installed in the correct ECUs and partitions and that their rights for internal communication are restricted to the necessary interactions. Where functions require cloud support, the secure domain controller maintains secure cloud access. GuardKnox has designed a Communications Engine FPGA to implement these functions.

PHASED INTRODUCTION OF AUTOMOTIVE SOA

Types of ECUs

The ECUs installed in new cars today can be roughly classified into two categories.

1. **High-performance microprocessor (MPU)-based** computing platforms that support the infotainment systems, run navigation systems, perform external communications, or support vehicle cameras. They use Ethernet and/or legacy networking interfaces.
2. **Low-cost microcontroller (MCU)-based** devices which have been optimized for their respective applications and which are manufactured in large quantities. They use mostly legacy interfaces like CAN and others.



Example for a modern vehicle's internal communication architecture

The high-performance ECUs, like in PCs and Smartphones, will need more processing power as applications become more demanding and many new SW-based capabilities are introduced in Software-Defined Vehicles (SDV), especially autonomous ones. Consequently, such new ECUs are regularly inserted in the vehicle production even during subsequent production runs of a particular model. In the architecture diagram they represent the Vehicle Server and the Domain Controllers.

The low-cost devices support simple functions that will only change slightly. For example, a microcontroller-based ECU that operates seat adjustment is idle most of the time and has no strict real-time requirements. These ECUs are very stable, sometimes spanning even several vehicle model generations. It is highly likely that they will continue to be used going forward and will continue to be supported by future E/E architectures. Any shift to new, cost optimized generic devices that support a native SOA approach and use Ethernet interfaces will happen gradually. This will then allow consolidation of these MCU-based ECUs if it makes sense within the network topology.

The Steps to an Automotive SOA Future

In order to reach the Automotive SOA vision both new hardware and software solutions have to be brought into the automotive development environment in well-defined steps.

- **Starting with the powerful computing platforms**

  An initial step will be the introduction of universal MPU-based ECUs. Such ECUs have sufficient performance and memory size to support fully-featured automotive SOA implementations.

  This allows applications or services to run on top of standard operating systems with a central management partition that organizes the overall SOA framework. Support for AUTOSAR Adaptive will be provided from the beginning to ensure legacy compatibility and these ECUs typically support high-speed Ethernet versions (e.g. Gigabit Ethernet) supporting fast, new network-demanding applications and low latencies.

  The high-performance universal ECUs, on the one hand, allow a consolidation of these devices and lead to a significant cost reduction. On the other hand, additional ECUs can be introduced relatively easily to scale the overall computing performance, due to the SOA framework approach, even after a car has left the manufacturing plant.

- **Message router supporting legacy ECUs**

  To support existing low-performance ECUs, their interfaces have to be adapted, their message formats converted, and the messages routed properly.

  For example, message routing protocols up to layer 5 have to be handled when using AUTOSAR. This function can also be implemented as part of other ECUs, using a Communication Engine FPGA.

- **Domain controller for external communication**

  All communication to the external world has to be highly secure in order to prevent malicious actors and applications from compromising the vehicle's functionality and impacting driving safety. To perform these secure external functions, a dedicated domain controller can be introduced that is physically separate from other ECU implementations. This domain controller will support both wired (Ethernet) and wireless (WLAN, cellular mobile radio) external communication. It is based on the Communication Engine FPGA.

- **Low-cost MCU-based ECUs**

  Over time, the legacy low-cost ECUs will become technologically obsolete. More powerful microcontrollers with more integrated functions, like interfaces and more memory, will become available at a low cost which will enable the replacement and consolidation of these ECUs.

  This will allow a consistent implementation of the SOA framework also in these devices. When this is achieved, along with the implementation of Ethernet interfaces across the entire E/E network, the communication infrastructure will become significantly simplified.

BENEFITS FOR THE SUPPLY CHAIN

Automotive SW development is not confined to the OEM, but there is a complex work-split between the OEM and a large number of suppliers, with each player providing their ECUs along with their specific SW or Firmware (FW) implementations. This entire set of HW and SW needs to be integrated and tested to build the overall system. The lack of modularity today makes any modifications to the SW system a complex and slow task which requires re-building and re-testing, resulting in today's lengthy time to market.

The complexity reduction resulting from Automotive SOA will lead to significant development cost reductions and much quicker integration cycles. Fewer errors will be created, and, e.g., the number of testers might also be substantially reduced. This will lead to a significant acceleration of correction cycles, reducing them from several months to just a few weeks, greatly improving time to market.

The Automotive SOA development approach facilitates simulation of the SW system. With well-defined interfaces of all the services there is no need to run these on the target platform. Instead, a simulation environment can be created to test the interactions of the services with emulated HW components like sensors and actuators. For certain functions some services may not be required, so they will not be addressed by the other services, although they might already be represented in the services directory, running some dummy code.

The SW configuration for each particular car is dynamically created at startup of the system. Therefore, for any car model, all available services are stored in a repository and published in the directory. The binding of the services is dynamic and can change due to HW and SW modifications in the car. Modifications of services during the manufacturing process can be made on the fly, while the car is still on the production line and even after it has left the factory.

The work-split between OEM and suppliers will be redefined: the main responsibility of the OEM remains the definition of the architecture, and the services and their interactions, while the service implementations can be performed by any suitable party.

3rd party application developers, who may not be part of the classical automotive ecosystem, can create new services for the vehicle without in-depth knowledge of the HW configuration. Opening the app market to new entrants will increase competition and improve capabilities and number of offered applications instead of relying on a limited number of OEM or pre-defined developers.

BENEFITS OF GUARDKNOX' SOA FRAMEWORK FOR DRIVERS AND PASSENGERS

Depending on drivers' and passengers' needs and preferences, sophisticated driver assistance, safety and comfort functions can be installed in any car. A car can be customized based on portable profiles in such a way that a shared or rented vehicle's functions and interior can be adapted to each users' requirements.

Consumers will be able to enjoy a similar experience in their vehicles as they do with their smartphones resulting from a broadening developer market. They will be able to buy their apps from the app stores of the OEMs or from independent after-market app stores. Consumer apps and specific ADAS functions will be downloaded via WLAN or cellular mobile radio. The vehicle's safety is maintained by the strict security mechanisms built into Automotive SOA.

SW updates will no longer require a visit to an authorized garage but will be performed OTA, so severe bugs can be fixed without undue delay and with significantly reduced cost.

GUARDKNOX' PRODUCT OFFERING

GuardKnox' Secure Automotive SOA product consists of the core framework and customized solutions. Which elements will be generic or customized depends very much on the target solution.

Deliverables that are part of the product include

- A SW package of components that constitute the basic core framework
- Interfaces and APIs
- Binaries and libraries required to run the framework on the target environment
- Technical documentation for developers and system architects
- Tools required for efficient development of applications running on the framework
- Services
  - Professional engineering services to customize the framework
  - Integration services
  - Certification packages
- Source code under specific commercial conditions, otherwise source code will be on escrow

The design is open and extensible. It features cross-platform support, where a platform consists of CPU architecture + hypervisor + partition OS. Several MPUs (e.g. ARM-VA) and MCUs are supported, as well as several partition OSs (e.g., Linux, Android, RTOS, ...). GuardKnox' SOA framework supports AUTOSAR Adaptive in order to reuse existing concepts and implementations.

GuardKnox' toolchain allows system architects to utilize multiple pre-existing components and automatically create a SWC shell with IDL interfaces utilizing manifests.

The framework supports multiple ESBs concurrently, and it can accommodate middleware implementations based on CORBA, DDS, SOME/IP.

CONCLUSION

The automotive industry has always been driven by unique innovation. The current paradigm shift focuses on functionality and on fundamentally changing the way consumers interact with and drive their vehicles. Purely mechanical machines transporting goods or people from point A to point B are long gone and the way a vehicle is designed and built is already evolving.

The development of automotive software needs to change to reach the goals of a customizable personalized experience for each driver. The current method of using a monolithic block of code is bulky and inflexible, leading to a lengthy time to market and rigid update protocols through a physical presence even for those fixes that could be done over-the-air with the proper software infrastructure.

Automotive SOA offers a way for vehicles to evolve even after they are on the road through app stores and aftermarket enhancements with no risk to vehicle performance or safety. Development time, errors and frustrations will be reduced immensely thanks to the decoupling of software and hardware. 3rd party vendors will be able to create unique software customized to drivers that could apply to any vehicle using the SOA Framework.

The GuardKnox SOA Framework offers the following key benefits:

- Automatic management of the software lifecycle within the vehicle, down to every ECU – automatic deployment, initialization, start, stop, teardown and removal of SW components
- Flexible unified communication between SWCs and services, where the underlying transport middleware can be easily changed, allowing for multiple ESBs to co-exist seamlessly together
- Cross-platform support: Platform = CPU architecture + Hypervisor + Partition OS
- Support for service deployment decisions and health monitoring
- Support integration of AUTOSAR Adaptive Platform in our framework
- Streamlined software developer tool suite including:
    - Modeling Tool that lets architects develop car E/E network and components
    - Analysis Tool that lets developers, integrators, and testers verify the software packages deployment and overall functionality

Automotive SOA Enables:

1. Vehicle app store leading to new revenue streams
2. Large infotainment screens for pleasure, work and gaming
3. Secure autonomous driving
4. Downloadable ADAS functionalities

LIST OF GUARDKNOX PATENTS

At GuardKnox, we are proud that our expertise in E/E Architecture and automotive cybersecurity has led to many international patents in the USA, EU, Japan and China, including:

Patent #9,866,563: Specially programmed computing systems with associated devices configured to implement secure communication lockdowns and methods of use thereof

Patent # 10,009,350: Hardware components configured for secure physical separation of communication networks in a vehicle and methods of use thereof

Patent #10,055,260: Service-Oriented Architecture (SOA) for vehicle ECUs, including Secure SOA and efficient implementation of in-vehicle SOA

Patent # 10,129,259: Installment configurations within a vehicle and interoperability of devices configured to implement secure communication lockdowns, and methods of use thereof

Patent # 10,191,777: Distributed SOA to enable services not solely related to a single ECU within a vehicle

Patent # 10,776,169: Centralized services ECU based on Service- Oriented Architecture and methods of use thereof

LIST OF ABBREVIATIONS

ABS	Anti-lock Braking System	ORB	Object Request Broker
ADAS	Advanced Driver Assistance System	OS	Operating System
AI	Artificial Intelligence	OSI	Open Systems Interconnection
API	Application Program Interface	OTA	Over the Air
AUTOSAR	AUTomotive Open System ARchitecture	PC	Personal Computer
CAN	Controller Area Network	POSIX	Portable Operating System Interface
CORBA	Common Object Request Broker Architecture	RPC	Remote Procedure Call
CPU	Central Processing Unit	RTOS	Real-Time Operating System
DDS	Data Distribution Service	SAE	Society of Automotive Engineers
E/E	Electrical / Electronic	SDV	Software Defined Vehicle
ECU	Electronic Control Unit	SOA	Service-oriented Architecture
ESB	Enterprise Service Bus	SOME/IP	Scalable Service-Oriented Middleware over IP
ESC	Electronic Stability Control	SSK	Secure Separation Kernel
FW	Firmware	SW	Software
HPC	High-Performance Computing	SWC	Software Component
HW	Hardware	V2G	Vehicle to Grid
IDL	Interface Definition Language	V2I	Vehicle to Infrastructure
IDS	Intrusion Detection System	V2N	Vehicle to Network
IPS	Intrusion Prevention System	V2P	Vehicle to Pedestrian
MCU	Micro-Controller Unit	V2V	Vehicle to Vehicle
MPU	Micro-Processor Unit	V2X	Vehicle to Everything
OBD	On-Board Diagnostic	VM	Virtual Machine
OEM	Original Equipment Manufacturer	WLAN	Wireless Local Area Network

GUARDKNOX: THE LEADING CYBERTECH TIER VENDOR

GuardKnox is a technology and engineering company specializing in E/E products and solutions for the automotive market. The automotive industry's first Cybertech Tier vendor, GuardKnox empowers OEMs, Tier 1 suppliers, and the aftermarket to deliver the next generation of secure, software-defined and service-oriented vehicles. GuardKnox's secure, flexible and scalable solutions enable added connectivity, Zonal E/E Architecture, hosted applications, high-speed routing (including network recovery and service discovery functionalities) and vehicle personalization.

By partnering with GuardKnox, OEMs, Tier 1s and aftermarket suppliers can quickly provide the next generation of drivers with the high-performing vehicles that they expect powered by the latest technologies and advanced app capabilities.

The company's pioneering approach to automotive innovation is inspired by the founders' experience in missile defense systems and the aviation industry and has led to multiple technology patents for automotive architecture. The GuardKnox leadership team's vast experience in developing and testing cybersecurity for military systems places them in the unique position to apply these solutions to similar civilian challenges in vehicles.

The challenges of today's automotive E/E Architecture are similar to those the team faced with fighter jets and solved by decoupling functionalities from the hardware and implementing a Zonal Architecture. This patented approach uses SOA methodology to enable any ECU, domain controller or gateway to serve as a vehicle-wide computing platform.

GuardKnox is based in Israel, with subsidiaries in Stuttgart, Germany and Detroit, Michigan.

Please feel free to contact us at info@guardknox.com for more information!

Let us empower you with the

FREEDOM TO EVOLVE

GUARDKNOX