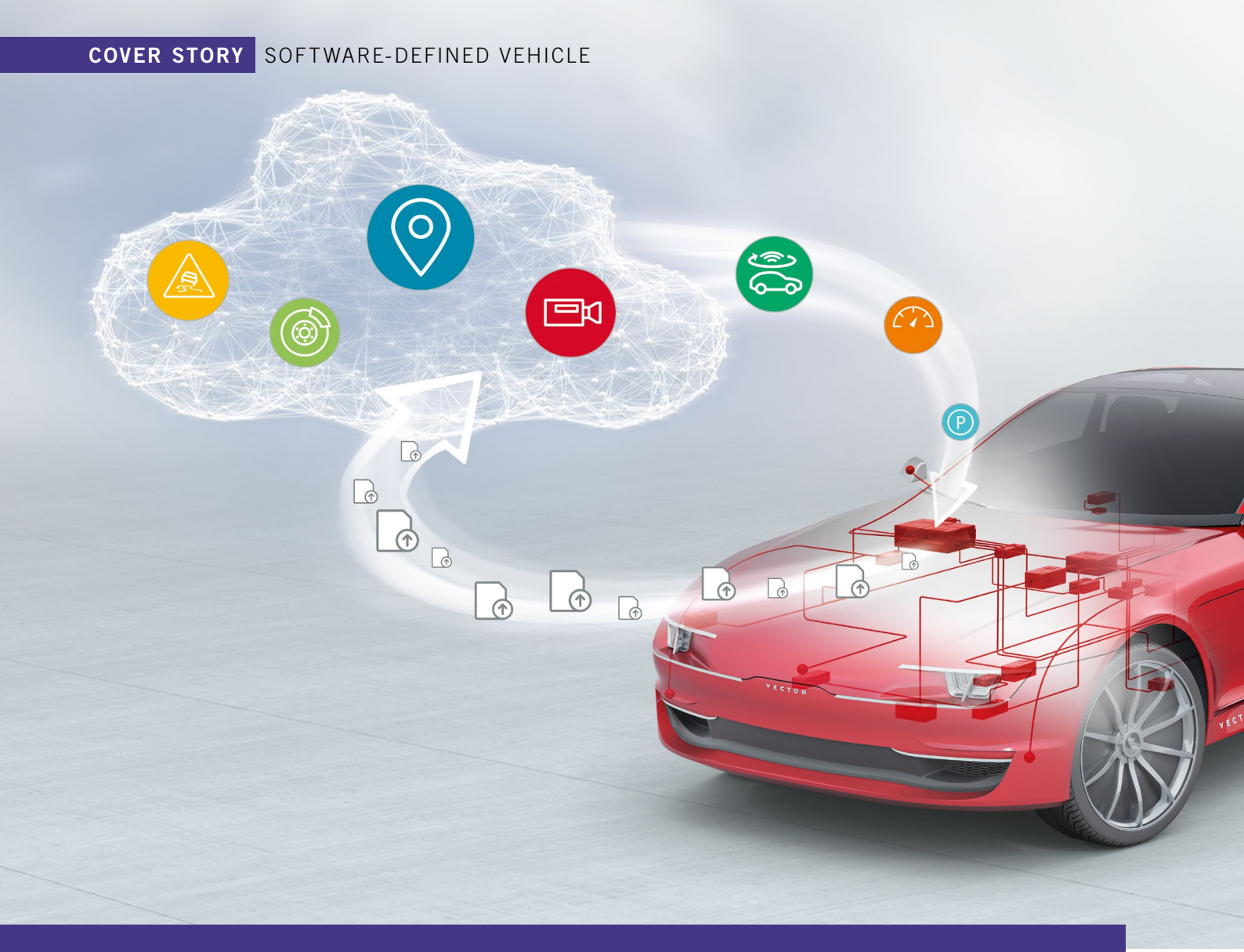

Vehicle OS: Software Platform and Ecosystem for All Vehicle Domains

Within Software-defined Vehicles, a Vehicle OS addresses the increasing challenges of software development and integration. One of these challenges is the coexistence of domain-specific software solutions that have not been developed in the context of a concrete E/E architecture but must nevertheless interact seamlessly with it. Vector Informatik shows what characterizes a holistic Vehicle OS and what role Android plays in this context.

© Vector Informatik

WRITTEN BY

Dr. Marc Weber
is Head of Solution Management for the product line Embedded Software & Systems at Vector Informatik in Stuttgart (Germany).

Andreas Raisch
leads a team in pre-development for embedded software that deals with domain-specific ecosystems and development methods in the area of Vehicle OS at Vector Informatik in Stuttgart (Germany).

Many vehicle manufacturers are undergoing a transformation process, moving away from mechanically dominated vehicles and toward Software-defined Vehicles (SdV). More and more functions that can be experienced by the customer are determined to a large extent by software and less by mechanical or mechatronic components. As a result, manufacturers have a new opportunity to equip their vehicles with improved or new functions over the entire life cycle by means of software updates, thus opening up new business areas.

In order to exploit the full potential, three prerequisites must be met for a SdV:

- The E/E architecture must support or natively provide for the decoupling of mechatronic components, computing power (hardware, HW) and software (SW). Therefore, most future vehicle generations will be based on a central/zonal architecture with three ECU categories: High-Performance Computer (HPC), zonal integration nodes and sensor/actuator ECUs, **FIGURE 1**.
- HPCs and zonal integration nodes require high-performance microprocessors and microcontrollers with appropriate automotive qualification. Such chips are already available, and their computing power increases with each generation.
- A powerful software platform with associated ecosystem, called Vehicle OS, is necessary to meet the increasing challenges of software development and integration. This is especially true for HPCs and zonal integration nodes, which often have a heterogeneous HW/SW architecture and run dozens to hundreds of SW applications.

VEHICLE OS

Since the automotive industry currently uses and interprets the term Vehicle OS – also known as Car OS and Automotive OS – inconsistently, the following definition is proposed:

A Vehicle OS is a development and operating platform for software applications and services of all vehicle domains. It consists of a base layer and a software factory and supports collaboration between companies.

- The Vehicle OS runtime environment is called base layer. Its instantiation may differ from execution platform to execution platform (microcontroller, microprocessor and backend).
- As Vehicle OS infrastructure, the software factory supports and automates the development, integration and roll-out of base layer and SW applications.
- Close and agile collaboration between vehicle manufacturers and suppliers is the key to success and is being promoted accordingly.

A Vehicle OS addresses ECUs with a high software content, above all HPCs and zonal integration nodes, as well as the associated backend. Vehicle manufacturers increasingly consider these areas as core elements of their value chain and largely assume control and responsibility for the Vehicle OS used.

BASE LAYER

There are two basic kinds of the base layer: one for in-vehicle ECUs (in-vehicle base layer) and one for the associated backend (backend base layer). The focus in the following is on the in-vehicle base layer. It consists of software modules on

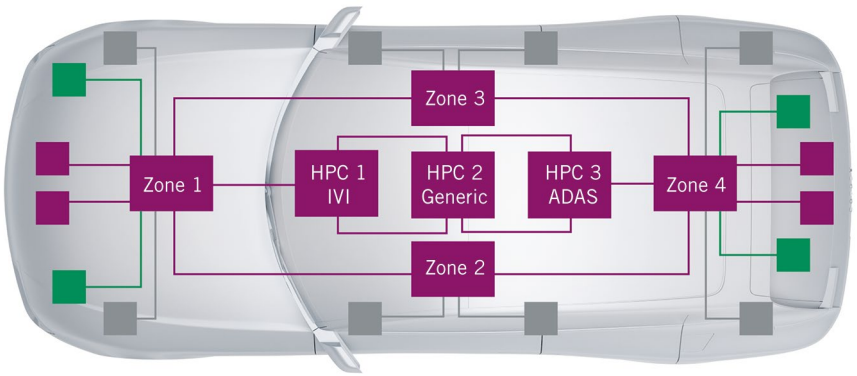

FIGURE 1 Central/zone architecture (© Vector Informatik)

several architectural levels. From HW-related infrastructure software, through operating systems (OS) and middleware solutions, to vehicle-wide defined system functions, **FIGURE 2**. This superset of software is available for the entire scope of the Vehicle OS. When instantiating the base layer on a specific ECU, only the required modules are considered.

At the operating system and middleware level, the Autosar Classic Platform is the established automotive standard for microcontroller software and the associated base layer portion is correspondingly homogeneous. For reasons of symmetry, the OS is shown as a separate component in **FIGURE 2**. The situation is different for microprocessors. Here, several POSIX-based operating systems are usually used in combination with different middleware implementations. The reasons for this are the special requirements regarding runtime environment and infrastructure software as well as different development processes in the respective vehicle domains. For this reason, specific software solutions are used in some cases, especially in the areas of in-vehicle infotainment (IVI) and ADAS/AD.

In contrast to the Autosar Classic Platform, the Autosar Adaptive Platform does not define its own operating system, instead it uses established POSIX interfaces. In addition to the ECU-internal data exchange via zero-copy mechanism and the efficient connection of communication protocols such as SOME/IP, the middleware supports further automotive use cases, such as diagnostics and network management. In its definition, special emphasis was and is placed on functional safety and cybersecurity, without neglecting the high requirements regarding data throughput. Due to these characteristics, the Autosar Adaptive Platform has established itself as middleware for ADAS/AD applications and in other vehicle domains such as body and comfort. In the infotainment domain, software solutions inspired by or derived from consumer electronics are increasingly used. Due to their origin and orientation, vehicle-specific integration is often necessary. A prominent example of this is the Android Automotive Operating System, which will be discussed later in more detail.

SOFTWARE FACTORY

SW development for HPCs and other integration ECUs is generally no longer carried out according to the classic V-model but based on agile methods such as development and operations (DevOps) and close cooperation between vehicle manufacturers and suppliers. The basis for this is feature-based development of the application software with a large number of short-lived source code branches. This makes the merging of these branches and the associated rapid verification of the source code changes made particularly important. Even in smaller ECU projects, the integration of application software and base layer is time-consuming. The effort required increases exponentially with the number of applications to be integrated, which are often developed at different times in geographically distributed development centers. A manual approach is therefore no lon-

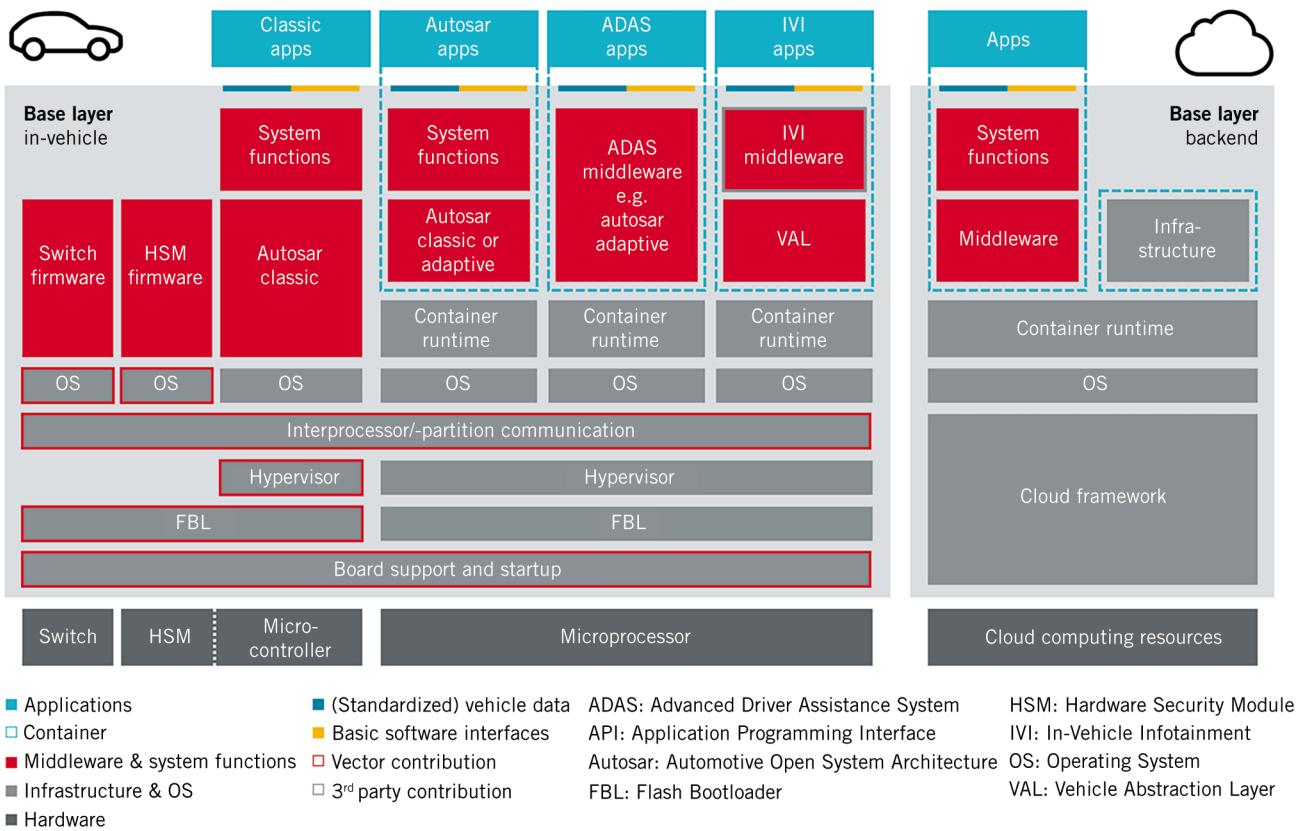

FIGURE 2 Architecture and building blocks of the base layer (© Vector Informatik)

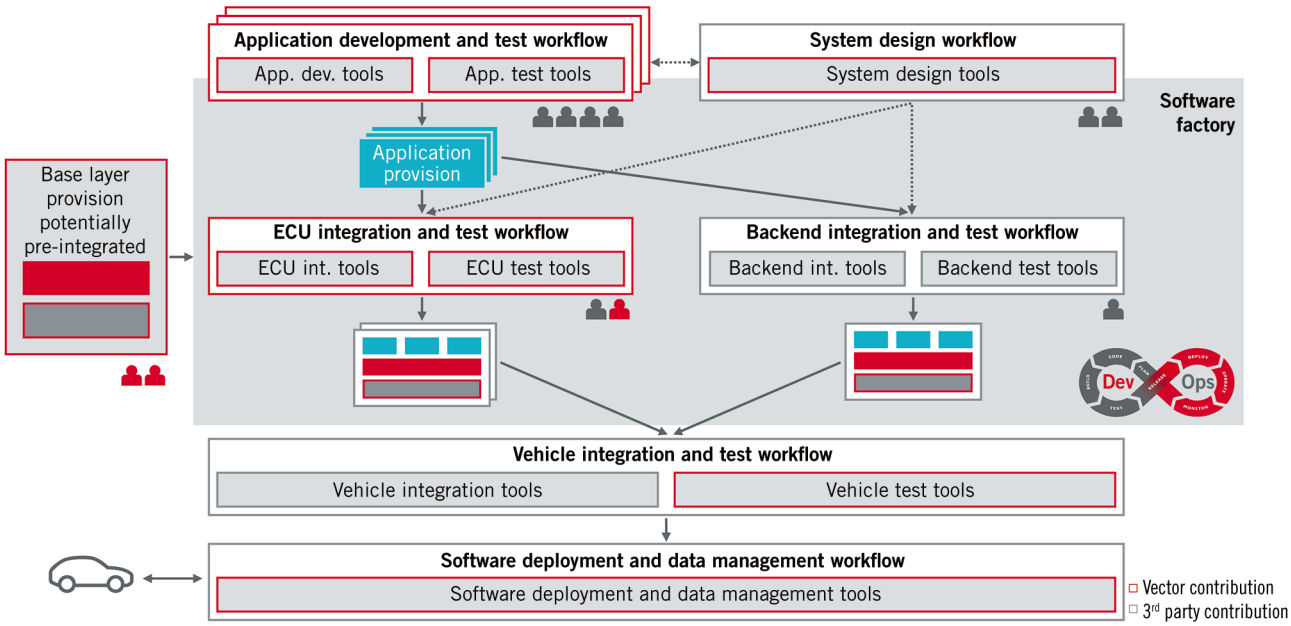

FIGURE 3 Workflows and tools of the software factory (© Vector Informatik)

ger practicable. The software factory addresses this challenge by automating the integration process as completely as possible, **FIGURE 3**. Some of the information required for this is already available in the system design, which is usually in an Autosar exchange format (ARXML). Missing integration conditions or integration instructions, such as scheduling information or other dependencies on the specific base layer configuration, can be easily added in a human-readable format.

The software factory is based on common DevOps tools, such as GitHub and GitLab, and supplements them with automotive specifics, such as automated control of configuration tools and specialized integration pipelines. In analogy to the base layer, the software factory must take into account various standards as well as existing ecosystems and interact with them in order to fully automate the integration process.

ANDROID

Android was developed as an operating system for smartphones. This class of devices is equipped with a graphical, touch-sensitive interface and has extensive audio and video functions. Smartphones can handle typical interfaces from consumer electronics and mobile communications and are also capable of dynamically adding and exchanging applications (apps). Android offers apps a standardized, largely hardware-independent and easy-to-use runtime environment as well as a suitable ecosystem with software development kit (SDK), emulator, documentation and examples. Based on this, a large, worldwide app developer community has been established. The extensible core of the solution is the Android Open-Source Project (AOSP) provided by Google.

Since the requirement profiles of IVI systems have a high degree of congruence with those of smartphones, the use of Android for this domain in the vehicle is obvious. When using AOSP, a vehicle manufacturer has the choice of developing important functions such as the map service, voice assistant and app store itself or licensing them from Google as Google Automotive Services (GAS). There are already various AOSP-based IVI systems in the field, with and without GAS.

ANDROID AUTOMOTIVE OS

Purely AOSP-based IVI systems require a greater development effort until they are ready for series production. Google has recognized this and introduced enhancements with the Android Automotive Operating System (AAOS) that significantly facilitate use in the vehicle. One example of this is the camera hardware abstraction layer, which enables the rear-view camera image to be displayed early in the boot process. Another example is the vehicle hardware abstraction layer (VHAL), which represents a vehicle property model designed for IVI apps. Properties provided include battery size and state of charge, as well as target and actual interior temperatures. Equipped with the appropriate rights, apps can change the setpoints, which enables, among other things, control of the air conditioning system via the graphical user interface. Since the IVI system is the central control element for many vehicle functions, the VHAL is usually extended on a vehicle manufacturer-specific basis and hence includes more properties than are provided by Google as standard.

The VHAL allows apps to be developed with a high degree of reusability. In its current implementation, it provides a suitable decoupling of different vehicles and their respective further development in the IVI system but requires vehicle-specific adaptation efforts when integrating AAOS into a particular ECU. The connection to the vehicle network can be established in different ways, for example via a dedicated ethernet interface, an inter partition/inter processor communication (IPC), or a mixture of these.

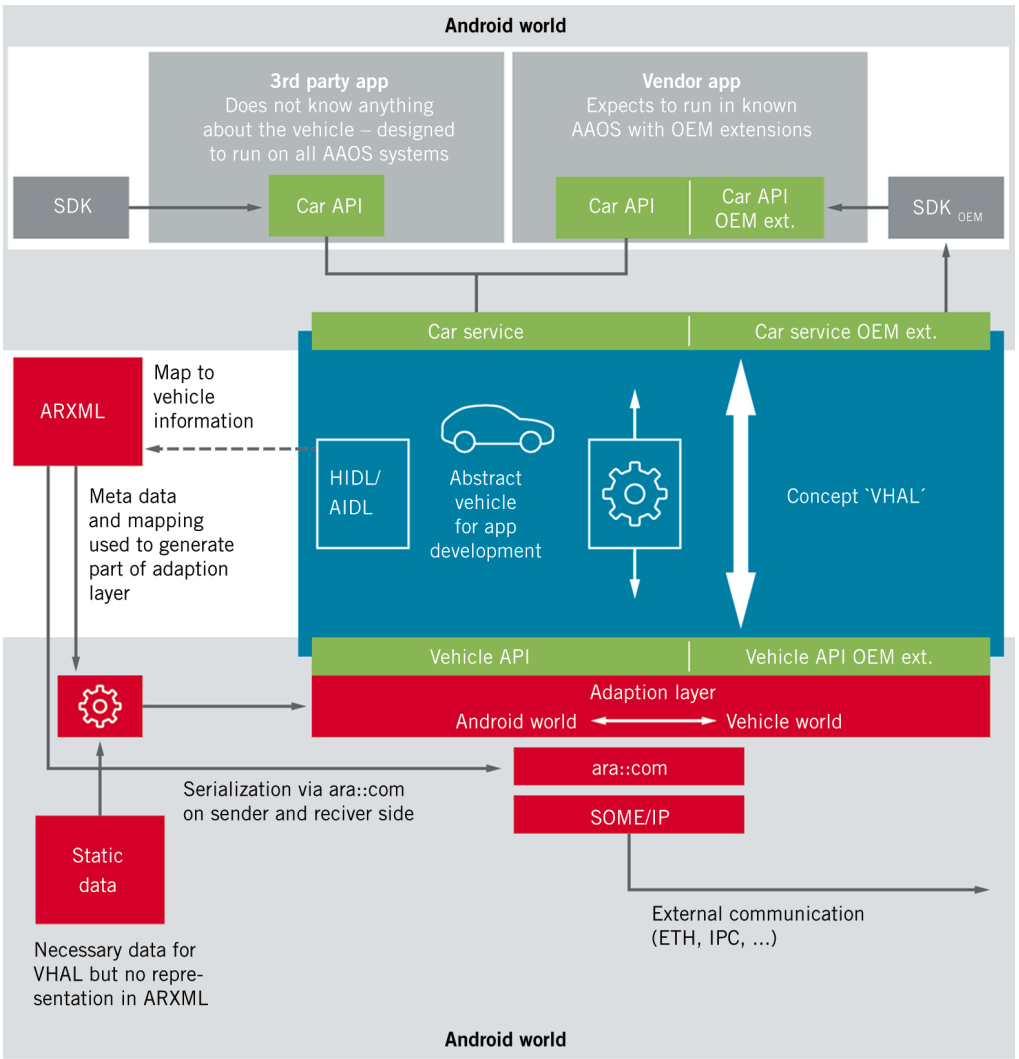

FIGURE 4 Android Automotive OS VHAL with generated adaptation layer to in-vehicle communication (© Vector Informatik GmbH)

VHAL GENERATION

Since the in-vehicle communication between ECUs is usually described according to Autosar methodology and in ARXML, this information can be used for an automated linking of the signals and services provided on the vehicle side with the corresponding VHAL properties. It should be considered here that Android apps expect VHAL-compliant behavior, but other considerations are paramount when modeling vehicle communication. Therefore, signals and services cannot necessarily be mapped one-to-one to VHAL properties. In addition, the behavior in critical operating phases must be taken into account, for example during system start-up or during a software update. An appropriate solution that translates between the communication elements modeled in ARXML and the expected VHAL behavior simplifies the initial integration. Furthermore, it helps to significantly reduce the adaptation effort for future updates of AAOS or for a changed vehicle communication, **FIGURE 4**.

CONCLUSION

As a powerful software platform with associated ecosystem, a Vehicle OS is a prerequisite for the realization of SdVs. Autosar plays an important role in both the embedded runtime environment and the associated workflows, however it does not represent a complete solution for all domains. Specific requirements in the vehicle domains require different software solutions and lead to a heterogeneous overall system. This results in new challenges for system integration, such as the connection of the Android Automotive OS to the in-vehicle communication infrastructure. In this case, however, the integration effort can be minimized by generating the VHAL based on existing Autosar system design information.

Vector is continuously expanding its Vehicle OS product portfolio with embedded software modules and tools that ensure the interoperability of different solutions and enable or simplify their integration at system level. Examples of this are the signal-/service-to-VHAL adapter for the efficient connection of AAOS to the vehicle network and the support of AAOS as an execution environment for the Autosar Adaptive Platform implementation Microsar Adaptive.

THE BEST FOR YOUR TEAM. THE WORLD'S LEADING AUTOMOTIVE MAGAZINES IN ONE PACKAGE.

YOU GET:

- Access to the online specialist articles archives
- Keyword Search in the e-magazines
- Interactive animations and editorial videos

WATCH OUR VIDEO AND GET TO LEARN MORE:
www.atz-magazine.com/automotive-package

OEMs · Suppliers · Service providers · Universities