The Case for an Open Integrating Operating System, Middleware Specification Development Approach

Prof. Dr.-Ing. habil. Alois Knoll, Technical University of Munich

Abstract
Beyond a doubt, Software-Defined Vehicles (SDVs) will be one of the most complex machines the world has ever seen, and if we look at machines for mass deployment, then assuredly the most complex. This level of intricateness enables a plethora of degrees of freedom and a range of opportunities beyond the current limits of our imaginations. However, laying the groundwork for these rich possibilities must be done now, and those who miss the train will relinquish the chance to play a defining role in the upcoming age of mobility.

Experience teaches us that “software is eating the world” and, as a consequence, the computer industry will gobble up any other industry that can be software dominated. But even within the computer industry, victory will be had by the fastest and by those who can readily transfer their products to other domains, that is, keeping in mind the rapid development in processor, communications and storage technologies and predicting the optimal time-window for the next step. A classic example is the capitalization shift from Intel to NVIDIA – nobody would have believed even 10 years ago that computer graphics engines developed for computer gaming would dominate the AI market the way they do today.

Apart from capital supply, as always in the software business, the key factor is “developers, developers, developers”. It is therefore unrealistic to believe that the traditional automotive OEMs or suppliers will get a foothold in the development of SDVs based on proprietary systems only. Proprietary approaches will never attract even the minimum workforce needed for such a major undertaking.

Successful examples of open software projects are IBM’s significant investment in Linux around 2000 and Google’s supply of methods and development tools for distributed systems and AI (e.g., Kubernetes and TensorFlow). Therefore, the key to getting into the driver’s seat of SDV development, is to start a collection of generously sponsored open-source projects that are based on rigid interface specifications, together with automotive-grade reference implementations of OS and middleware. Moreover, it is crucial to extend the potential application range...
beyond “just automotive”. This will provide the needed impact (given the comparatively low volume in automobile production) and make it attractive for developers to see their work deployed in large numbers. So, for scaling and for increasing attractiveness to developers, it is important to recognize that all future markets for digital devices will profit from the high standards in the automotive industry, and cheaper computing power will make it possible to deploy such software even on the cheapest devices in large quantities.

We believe that only a multi-faceted, cross-domain, transparent and open approach that attracts and rewards developers (working on all levels of the software stack) worldwide is key to ensuring that the European auto industry gains a unique, competitive edge. In our talk, we will present our work in two steps:

- Briefly summarize and categorize the current approaches to architecture and OS design in the auto industry along with some examples.
- Introduce our joint initiative “Automotive Software Interfaces & Middleware Initiative” (ASIMI: https://asimi.eu/), which is investigating the general attitude in the industry towards future developments and which will support the process towards collaborative and open sourced projects. We shall describe the challenges and the approach from a technological and organizational standpoint.

1. Introduction

In January 2021 [1] Wolfgang Reitzle, Continental’s Supervisory Board Chairman, warned BMW, Daimler and Volkswagen against “going it alone” when it comes to developing Operating Systems for their cars. “It is a historic mistake for each manufacturer to try to develop their own software platform” and he emphasized that the German automotive industry still has a chance against Tesla, but only with a combined, joint effort. In a few years, there would be only three or four operating systems for cars worldwide, and he pleaded for a “standardized, European platform.” This view is shared by other suppliers.

However, the German car manufacturers, by contrast, believe that they each must be the one and only master of their own Operating System (OS) because it will be “core technology” and they are afraid of relinquishing control over it. Says Markus Duesmann, CTO of VW [1]: “We must now first manage to create a uniform operating system for our own twelve group brands” and then, one might “reconsider partnerships after 2024”. Frank Weber, CTO of BMW, seconds: “software belongs at the heart of the company,” and it would have to be closely linked to the development of the hardware.
It is easy to significantly underestimate the complexity and associated cost of developing, maintaining and constantly updating an OS. Moreover, the extensive experience gained over decades of development by many computer companies and the open-source community, along with the intimate knowledge about the processor internals necessary to achieve the performance of today’s systems, is overwhelming and cannot be copied within a short time frame. Linux was released in 1991, Windows in 1985, QNX in 1982 and UNIX in 1973. Alternatively, one can use (parts of) systems and code available to everyone, but then the question becomes even more pertinent: why should partnerships be ruled out straight from the beginning?

Most importantly, in the mid-term, it will be vital for any OEM worldwide to tap into the huge potential of developers, who will not be interested in platforms that are unique to just one OEM – no matter how many cars they produce. The number of these vehicles will always be considerably lower than the number of smartphones, let alone other interesting IoT devices that are sold in an exponentially growing market of all kinds of gadgets. It will be infinitely easier to engage the highest quality developers if an international standardized platform is developed—rather than a fraction of developers optimizing their skills on a platform that might not be here tomorrow.

There also seems to be some confusion about the terms used by many in the field. Clearly, it is software that defines the car of the future and ergo, is the source of almost all innovation. The operating system is part of this universe of software, and it typically sits on top of a hardware abstraction layer, i.e., it accesses the different hardware components in a unified way and provides certain services to the application programs, e.g., for multitasking, communication between tasks, for file services, etc. The middleware is a collection of programs “between” the operating system and the application programs that uses the services provided by the operating system, and then groups and enhances them to simplify complex application development for programmers by making available more powerful routines available than the operating system provides directly. The original meaning of middleware was “the enabling of communication and management of data”, but today there are middlewares for many other purposes, e.g., for providing physics simulations and graphics for computer games, security layers, neural computation, etc.

Please note that the middleware and the software layers on top of it are the function differentiators, and the OS by itself is only a precondition for making the hardware resources work. Just as no one cares about the type of microprocessor that is in their smartphone, the same applies here: for market players, the kind of OS inside is irrelevant.

Modern software ecosystems containing these software components are dominated by their interfaces, enabling modular integration of current and future software. Effective interfacing has
a major impact on maintenance and extensibility and is a precondition for creating the potential for independent software developments. The value attributed by the computer industry to interfaces can be seen by looking at the copyright infringement case between Oracle and Google over the application programming interfaces (APIs) that Google had used to develop Android: the dispute dragged on over several instances and the amount in dispute claimed by Oracle amounted to 9 billion dollars. After more than ten years in court, it was finally decided in April 2021 that Google was using the interfaces lawfully (although there was some mention of “fair use” in the Supreme Court’s ruling). However, this means that no one will be able to claim a hard copyright to APIs in general and that “programmers can continue to use APIs in their projects just as they have for decades before”. [4]

In the computer industry, publishing and carefully documenting interfaces has had a long tradition, starting with peripherals already in the sixties. Quite a few companies made devices that adhered to the quasi-standards defined by IBM, ranging from printer connectors by way of disk drives to complete CPUs. Components from other vendors were called “plug-compatible systems”. The seventies then saw a shift to software (which at the time was even given away by large companies), which reached its zenith in the publication of the source code of all the firmware shipped with the first IBM PC. Without this, it would have been much more difficult for third-party suppliers to build the first “PC clones”.

Obviously, there are lots of examples of large software systems that were built in cooperation between an open-source community and large companies. The most prominent example may be Linux, which was massively boosted with a $1bn investment by IBM and then made into a multi-billion service business by support companies like Suse, RedHat, etc. These companies today also participate in Linux’ continued development – along with the tool suite GNU, which may be less prominent but is of central importance; and also along with the some 20,000 programmers that have been working on the Linux Kernel directly, since around the year 2000. While this investment is often forgotten, another major move by IBM was the development of the Eclipse framework. Before it was released to the public domain through an industry consortium in 2001 and then handed over to a foundation, IBM had already invested on the order of $40m into its codebase.

In the high-availability, high reliability area, there are systems like Android of course, but with “Open RAN” there is also a recent large effort to break away from proprietary software systems running the radio access network of mobile telephone networks. Not to mention the routers and switches we all have in our homes, which are highly commoditized as far as the hardware and their constantly (remotely) updated and upgraded software are parts of a large “Software-Defined Network”. To a large extent, they are also using Linux and, for example, open file systems,
e.g., FAT, which was formerly a proprietary Microsoft standard now used in many IoT devices. Since in all these systems we see parts (or variants) of Linux, we can clearly state that an Operating System today is typically considered common property. This makes it possible to create Value-Added functions inside a device very easily and at very low cost, but by itself creates no added value (through “individualisation”) at all.

Clearly, there are different types of software development processes and people that work under them. There is, e.g., a big cultural difference between developers working in avionics and those programming computer games. However, there have been quite a few inroads from the “consumer oriented” software makers into the more safety-oriented areas, whereas the opposite has hardly been reported.

In our context it is important to note that almost all tools used in modern software engineering are open-source and/or support the participation of many distributed developers in one project.

It is also important to note that in the area of security-critical and cryptographic software it is instrumental that both the algorithms and the code are completely open – so that many eyes can scrutinize them and find errors. We also need to realize that the leading software companies (Google, Microsoft, Amazon, …) put large stacks of their software and development environment in the open-source domain in order to make as many developers as possible use them – and attract them to their ecosystem. This has been extraordinarily successful in the past, and it could serve as a perfect example how the automotive industry can transform itself into a Software-Defined sector of the economy, which produces commodity hardware, but creates highly profitable “software-defined vehicles”, offers “software-defined services” and integrates itself into all other aspects of the life in a digitalized society.

2. Where is the springboard, what needs to be done?

How can we support the move to such a scenario, in the context of a highly complex automotive sector that has optimized itself for decades towards building sophisticated hardware? As mentioned above, the key to such a transformation is the thoughtful preparation and implementation of an ecosystem in software – ideally modelled on the proven multi-tier system of automotive suppliers with specialised software-suppliers being built up and integrated into the ecosystem. Clearly, this can only happen if every player, from big to small, contributes to such an ecosystem – but it will certainly fail if the scarce resources (both human and financial) of European OEMs are scattered across the continent with various “silo” activities, none of which builds up critical mass against the software giants that will run one attack after the other for gaining market access.
There is this fear that vendor lock-in could result from an adoption of one of the traditional OSs, i.e., if an OEM licenses an OS instead of developing one of its own, then this would create a long-term dependency on that OS’s supplier. Ironically, this was exactly the same argument for why immense effort was invested in creating proprietary entertainment and GPS-based navigation systems – with the effect that none of these gained much acceptance and most users today prefer Apple- or Google-based devices and services in their cars. This story may well repeat itself if proprietary OSs are developed that none of the competitors will use. No single automotive OEM will be able to maintain their own code in the long run, and at the same time no other OEM licensee will help share the burden of maintaining it. In the end, the OS used will be “Android Automotive” and/or a similar system created primarily for the Chinese market, and Google and Apple will rapidly learn how to provide and integrate all the automotive functions into that OS and every OEM will have to use their software (exactly as we have seen happen in the mobile phone business). In other words: paradoxically, exactly what was to be prevented will inevitably happen.

So, we should move away from the fascination of the term “Operating System” that one needs to have control over and instead establish an ecosystem of development tools and automotive specific middlewares. Instead, we should draw from the rich experience that European OEMs have collected over decades in designing specific functions for their cars and take this rich body of knowledge forward and distil it into middleware. This middleware will first have to be defined by its interfaces, i.e., the interaction with the vehicle hardware, the OS and other modules that are part of that middleware (or connect to other middleware modules and/or other middlewares for other domains).

We therefore suggest not concentrating primarily on the OS development, but to prepare for the time when the customer-centric functions become the key differentiator in SDVs. We will outline this approach in the next chapter. Before doing so, however, we will briefly introduce our “Automotive Software Interfaces and Middleware Initiative” ASIMI.

3. The ASIMI Approach

The current E/E architectures require a drastic overhaul in order to keep pace with rapidly changing needs. Limited software components reusage, safety concerns, and integration complexity presuppose high development costs and long time-to-market. These factors, among many others, increase the immediate need for an extensive, well-established middleware platform.

For this reason, we started ASIMI, the Automotive Software Interfaces and Middleware Initiative, which advocates for a joint discussion and development of an integrated and up-to-date platform.
to address modern industrial challenges and promotes collaboration in pursuit of standardized development practices.

As part of ASIMI, we have conducted a poll [5], in which an extensive questionnaire was circulated to the automotive industry stakeholders in order to assess their vision of the ongoing domain transformations and the related challenges. The study focused on the following key aspects:

- Trends shaping automotive hardware and software architectures
- Impact of these trends on business
- Benefits that architecture trends bring to the market players
- Challenges caused by new architectures
- Standardization importance
- Expected standardization drivers
- Role of the survey participant in the future transformations

The survey results provide a holistic overview of current software trends in the automotive sector and their significance for major industry players. The findings show that Over-the-Air (OTA) updates, architecture, and strong software orientation were significant trends shaping the current automotive development.

These trends obviously will require a mindset shift as the software-based approaches completely redefine the current situation, namely the established relationships between the OEMs, suppliers, and other participating market players. As a logical consequence of these rapid changes, the “expert” workforce shortage and absence of clearly defined software standards were listed as major challenges (Fig. 1).
Fig. 1: Challenges as reported by participants.

In particular, the survey results highlight the increasing concerns about the lack of industry standards – issues such as efficiency, integration, and collaboration are frequently mentioned to explain the urgent need for common ground in the automotive domain via long overdue, much-needed software transformation. Standardization (Fig. 2) and cooperation are considered key ingredients, which would allow for fruitful competition between the current marker leaders. To encourage smoother cooperation, additional barriers must be overcome, e.g., the widespread contempt for using common ecosystems and sharing software platforms among the OEMs, supplies and manufacturers.

![Relevance of Standards](image)

Fig. 2: Opinion regarding the need for software standards.

In order to keep pace with rapid market changes and technological advancements, the stakeholders must be able to design and implement their solutions fast. Therefore, the need for easily updatable, modular, and maintainable software products arises. Our poll showed that industry is expected to serve as a driving force of standardization initiatives, and that a joint effort with academia is vitally important in order to identify commonly accepted solutions that can be transferred to the automotive domain. Fig. 3 depicts major benefits from a shift in the mindset of the community.
ASIMI offers a unique opportunity for joining the discussion and shaping a possible performant unified middleware platform, which would implement system-wide, essential, and common base functions in a service-oriented manner.

In our opinion this middleware platform should be specified solely through its interfaces and associated semantics, giving them central importance in future software development. These interface descriptions form the base for the middleware platform concepts and open architectures. They support, for example, safety and security relevant functions, lifecycle and resource management, as well as third party software compatibility. In addition, they support successful eco systems, which enable the commercialization of new functions with low costs and fast time-to-market.

4. Steps towards an integrated interfaces and middleware ecosystem
An isolated view of the OS, without considering its very close coupling with the vehicle hardware and the middleware(s) sitting on top of it, is not practical. Moreover, the engineering system including multiple, individual middlewares, must be very tightly interwoven with the rest of the environment. Ideally, there will no longer be a distinction between development time and run time.

We briefly recall some concepts of the RACE project [6], where many of these concepts were conceived and implemented in prototypes, including three cars of different sizes and types.
Fig. 4, taken from [6], shows the generic architecture developed in RACE, i.e., a very abstract view of the setup of controllers, the run-time environment running on centralized hardware, and the IO expanders and interfaces.

![Generic RACE Architecture Diagram](image)

Fig. 4: Overview of the generic RACE architecture.

An integral part is the engineering system, which provides all the software tools for programming the components in a certifiable way. The goal here is to abstract as much as possible from the hardware and to provide a “single system illusion” to the programmer.

A specific instance of architecture implemented based on the generic architecture in Fig. 4 is shown in Fig. 5: it illustrates the implementation of a concrete car with two central Duplex Control Computers (DCCs). The general structure supports the construction of “perception–cognition–action” architecture.
Fig. 5: One specific implementation of the RACE architecture, see [6]. Top: Logical view of an implementation with number of Duplex Control Computers x still definable. Bottom: Physical view with $x = 2$ central DCCs. Inner glass-fibre ring for DCC to DCC communication, outer ring for sensor and actuator communications.

The RACE project started around 2010 and ended in 2015, it was far ahead of its time, and industry was neither receptive nor foresightful enough to take a serious look at it. The lessons learned from it, however, can be summarised as follows:

1. The major challenge of automotive OEMs is the complexity of traditional E/E architecture conflicting with the required faster innovation cycles in the age of digitalization. This also pertains to the latest “zonal” architectures, especially if they are not being massively supported by tailor-made development environments.
2. An integrated, sophisticated runtime environment and middleware-based approach like RACE has the potential to reduce the number of controllers, minimize the heterogeneity of network technologies, and offer several generic services that need to be covered during application development.

3. The integrated engineering environment – if designed in the right way – can cut development time by orders of magnitude. It can also increase the level of safety and security to the best-in-class, and it can be easily shared with the open-source community for joint developments. One could think of a development “thread” that would allow independent developers to produce certifiable code.

Finally, this approach using generic hardware and a high-integration platform lends itself to business developments along several lines:

- The run time environment can be the core offering, including a configuration tool.
- The full system is offered as a modular toolbox to be used (i.e., integrated and conFig.d) by OEMs with focus on safety up to ASIL-D, fail-operational modes and the necessary security mechanisms; it includes an operating system from a preferred supplier, secure communication system, and hardware development support.
- The application software development is delegated to a partner network. Third parties can integrate their functionality easily.

5. Recommendations for the transformation towards total software orientation

As we have learned from the past, direct cooperations between automotive OEMs are difficult and tend to fail. Direct sharing of code and/or data, no matter how close to the hardware, is not desirable. The end result is (as briefly mentioned at the beginning) that all efforts must be duplicated, processes used are not compatible, maintenance cost will be extremely high and cannot be shared, product cycles will never be able to compete with computer and communication industry’s speed. We think, however, that a change of attitude in the industry towards future developments which will support the process towards open-sourced projects, is an essential precondition for the survival of the traditional players in the industry.

This attitude change will not come overnight, although first signs are becoming visible [7]. Therefore, we propose joining forces at another level, i.e., to develop generic middlewares and to establish domain-oriented development processes that can attract large communities of programmers and involve them at an operational level. It is important to realize that the traditional players have a much higher basic cost block to shoulder (given their long heritage and legacy systems) than newcomers like Tesla, Nio, XPeng, etc., but on the other hand, they can still
define the standards – if they do not wait too long. The new players, with their information processing background, already have such processes in place, and it is quite likely that they can roll them out rather quickly.

The middleware definition should be only through very strict interface definitions, without any exceptions. One can take inspiration from the current status and the long development of ROS, the well-known robotics middleware, which has turned robotics into a much more software-oriented business than it was before ROS appeared. Roboticists can concentrate on problem solving rather than on device driver writing, as they did in the past.

The ecosystem we propose will not come into being without an investment from all players in the industry (see the IBM Eclipse investment mentioned above). While an organizational framework that is appropriate would have to be discussed, one can also learn from the computer industry how to move thing forward via a combined, joint effort. The computer industry realized early on those economies of scale can only be achieved if consortia are formed at the right point in time, investments into the development of base components including software are made, and competitors find the right level at which to compete – and at which to cooperate. Successful examples are the Compact Disk, Bluetooth, PCI-express, Linux, Android, and many, many more.

A classic counter example is the video recorder standards war in the late seventies (and to some extent the DVD wars in the nineties), which none of the competitors could benefit from, and ultimately forced the European producers out of that market and consumer electronics altogether.

For the automotive industry to become competitive with the computer and software industry, it is mandatory to achieve the same speed as the latter. This can only happen if there is a much closer cooperation between research organisations, universities, and industry, namely in the area of computer science. This would make it possible to set up a continuous know-how and innovation transfer pipeline like what is very successfully being practiced in the USA and in China. The traditional players should interface much more closely with research, and pick the brains of the brightest young talent, in order to get the latest results from software engineering, machine learning, cryptography/security, tool design, etc. This transfer is crucial for the future of an industry that will be totally software-defined and software-based (with respect to the products, the product development systems and the production facilities).

Summarising, it is our opinion that the industry players in Europe with their dependence on foreign markets and dependence on the US AI giants would be well advised to consider establishing:
a programme for developing a comprehensive set of standards that define interfaces for middlewares, which would pave the way for new architectures, but at the same time allow for brand-specific individualisation,

- a plan for a “phase-wise” progression of the development environment including all processes and standards necessary – for example, modelled on the strategy IBM adopted for the Eclipse framework,

- an initiative for deploying the results of that programme and nurture an ecosystem that attracts global talent; a promising way could be the setting up of an independent foundation, such as Eclipse or the foundation “RISC-V international” on the hardware side. Clearly, this can only work if enough freedom and manoeuvrability away from the constraints of day-to-day series development constraints is given to the people that are tasked with solving this challenge. To successfully go through the technological transformation, the automotive industry needs to focus on collaboration as a key factor to prosper at the increasingly competitive market.
Acknowledgement

The contributions of the ASIMI group and the very lively and fruitful discussions are gratefully acknowledged by the author.


