# Research on an Open-Source Software Platform for Autonomous Driving Systems

**Lukas Bulwahn, Tilmann Ochs and Daniel Wagner**
BMW Car IT GmbH
Petuelring 116, 80809 Munich, Germany
*firstname.lastname*@bmw-carit.de

October 9, 2013

**Abstract**

The next larger step in automotive development will be towards autonomously driving cars. Autonomous driving will be a highly complex and safety-related function in future vehicles, and current software platforms are not adequate for this function. We present our ongoing research on an open-source software platform for autonomous driving software systems.

From our experience with use cases of existing driver assistance systems and research projects, we derive requirements on the software platform. We motivate the advantages of open-source development over proprietary development and provide arguments for an open-source qualified (open-proof) software platform. Furthermore, we propose an architecture for the software platform based on the Linux operating system and other open-source software. This architecture fulfills the requirements and makes development and qualification of such systems efficient.

## 1 Introduction

In the sixties, car manufacturers built first electronic driver assistance systems, such as anti-lock braking systems, into cars. With the continuous development of micro-electronics and computer technology, those systems became more and more sophisticated.

The imminent next step in this development is the advent of *partially automated driving systems*, which control the cars automatically in certain traffic situations. However, even while these systems control the cars, the human drivers remain responsible for monitoring those systems and taking control if necessary.

Current long-term research, in contrast, is targeting *autonomous driving systems*, which are able to handle any possible traffic and emergency situation safely without relying on human supervision. These systems have two seemingly contradicting characteristics: first, they must be verifiably highly reliable, because malfunctions could endanger passengers and other traffic participants; and second, they consist of complex and computing-intensive application software components.

These two characteristics make autonomous driving systems belong to a new class of safety-related systems, because "keeping it stupid simple" is limited by the inherent complexity of reaching a safe state in the case of malfunctions while driving autonomously. Given the current and future road infrastructure, reaching a safe state means pulling over a car with situation-dependent speed to a safe stationary state on every road in every condition in the event of a single safety-related system malfunction. Due to our initial experience with autonomous driving systems, we assume that this inherent complexity will require hardware and software platforms that support more complex application software and provide a much higher computational throughput than the current rather simple, highly reliable, automotive software systems, such as airbag control systems.

Such an autonomous driving system is built up from a network of redundant electronic control units that are connected to the vehicle's sensors and vehicle's actuators through a highly reliable, adequately reactive vehicle network.

In this paper, we assume that such a network can be provided and we consider only the requirements and properties of a single electronic control unit in an autonomous driving system. A single electronic control unit embodies the functionalities for autonomous driving that are mainly implemented in software, such as high-level recognition of traffic situations, prediction of other traffic participant's behavior, planning of the vehicle's maneuver and computation of the desired trajectory.

In this paper, we further focus on the software architecture for this electronic control unit. The techniques and implementation of the main cognitive software functions for autonomous driving is not of interest here; only the requirements they set on the software architecture are important.

The software architecture can be decomposed into two layers: The upper *application software layer* contains the main cognitive software functions for autonomous driving. The lower *software platform layer* provides basic services, e.g., communication between the software functions and abstraction of the concrete computing hardware. Throughout this paper, the term "platform" refers to this software platform layer and shall be read as "software platform".

In the following sections, we present ongoing research on a software platform that is based on open-source software[1]. First, we discuss the characteristic requirements of a platform for autonomous driving (§2). Second, we present arguments for the advantages of an open-source development of safety-related software in this context and the prospected impact on the style of development (§3). Third, we sketch the platform architecture based on an open-source Linux operating system (§4). At the end, we present further related work and summarize our findings.

## 2 Platform Requirements

Future autonomous driving software will implement algorithms for computer vision and artificial intelligence beyond the current state of the art. Currently, all parties engaged in autonomous driving work on prototypes to address the various challenges in this field, but no one has finalized a software product development for the general public.

As definite requirements for the platform consolidate only during the product development, a definite specification of the platform for autonomous driving cannot be presented. To derive an initial set of requirements for a platform despite the ongoing product development, we extrapolated resource demands and use cases from existing systems and research projects [11]. This initial set of requirements indicates that the software is complex, computing intensive, safety related, time sensitive, volatile and automotive specific. We elaborate on the requirements concerning these six aspects:

**Complexity.** Software in research projects for autonomous driving and robotics have tens of thousands lines of code and make use of complex and dynamic data structures, such as graphs or sparse matrices. To support the software development, the platform shall provide

- integration of loosely coupled application software components to minimize covert interference,
- a rich interface definition language to specify application software component interfaces unambiguously,
- high-level programming languages to reduce code complexity, and
- mainstream software engineering methods and tools to leverage experiences and developments from the general IT industry.

**Computing Performance.** Research projects for autonomous driving and robotics use modern personal computers to its full capacity. This indicates memory consumption in the range of Gigabytes, CPU consumption in the range of GFLOPS and heavy use of hardware acceleration. Hence, the platform shall provide

- a hardware-independent programming interface for acceleration hardware to port application software components easily,
- symmetric multiprocessing to leverage component-level parallelism on mainstream processors, and
- performance-optimized processor architectures to maximize computational throughput.

[1]We use "open source" as shorthand for "free/libre and open source" with the same meaning as in the acronym FLOSS.

**Safety.** Autonomous driving is a safety-related function, as malfunctions may lead to collisions and cause harm to passengers and other traffic participants. To assure safe operation, a safety concept is defined on the vehicle level and decomposed to derive safety requirements for all relevant vehicle components. Following the automotive standard for development of safety-related systems, ISO 26262 [16], vehicle components are classified with four integrity levels, from lowest Automotive Safety Integrity Level (ASIL) A to highest level ASIL D, depending on the consequences of failures.

The platform will host application software components with different integrity levels and must ensure that those with lower integrity level must not interfere with high-integrity components. Based on our personal communication with various hardware vendors, we assume that a platform will at most provide integrity levels up to ASIL B due to lack of high-integrity processing hardware with the required performance. Higher integrity levels will therefore be realized by vehicle-level redundancy. To execute safety-related software functions, the platform shall support

- execution of application software components with integrity levels up to ASIL B,
- coexistence of application software components with different integrity level,
- failure detection of hardware and platform software components and silent shutdown to support vehicle-level redundancy,
- misbehavior detection of application software components and restart or shutdown of affected components to increase robustness,
- control-flow monitoring in application software components and misbehavior detection,
- data-flow monitoring between application software components and misbehavior detection, and
- standardized software and error propagation models to support automated safety analyses.

**Time Sensitivity.** Autonomous driving systems have real-time requirements, as outdated environment data or lagging maneuver planning can lead to oscillating dynamic behavior and collisions. From our experience, we assume that a sporadically occurring maximum jitter below 100 microseconds in the application components does not lead to a hazardous event. Recent work of the Open Source Automation Development Lab's (OSADL) [4] shows that interrupt latency below 100 microseconds can be achieved on mainstream processing hardware. Hence, the software platform shall provide

- deterministic timing behavior with maximal jitter of 100 microseconds to ensure replicable behavior,
- real-time scheduling to meet timing requirements of application software, and
- timing behavior monitoring of application software components and detection of timing violations with 100 microseconds tolerance.

**Volatility.** Research in computer vision and artificial intelligence will evolve rapidly in the foreseeable future, and several research projects currently develop infrastructure for autonomous driving [2, 3]. We assume that car manufacturers have to offer software updates through a remote connection to provide the latest functionality and integration of locally available infrastructure services. Hence, the platform shall support

- agile development practices for software components to mitigate low concept maturity,
- remote update of application software components to maintain software continuously,
- a modular safety concept to integrate independently developed application software components, and
- remote addition and removal of application software components to provide extensibility.

**Automotive Specifics.** Autonomous driving systems are integrated into vehicle networks, are managed by standardized diagnostic functions, and participate in vehicle state and energy management. Hence, the platform shall support

- standardized diagnostic and vehicle management functions,
- integration of vendor-specific diagnosis and vehicle management functions, and
- automotive-specific communication protocols over Ethernet, CAN or FlexRay.

## 3 Open-Proof Development

Prominent free/libre open-source software (FLOSS) projects, e.g., the Linux kernel project [23] or the Apache HTTP server project [6], employ stringent development processes [7, 18] and deliver software of very high quality [14], but they do not fulfill the requirements of current safety standards, such as the ISO 26262 standard for software in automotive systems, or the EN 50128 standard [15] for software in railway systems. These standards impose strict demands on project management, developer qualification, risk management, requirements management, quality assurance and documentation. Therefore, open-source software can not be used in safety-related systems without further activities.

Nevertheless, it is reasonable to incorporate open-source software in such systems rather than re-implementing such general-applicable commodity software. Currently, there are three endeavors in this direction, which we call *open-proof projects*.[2]

As a first endeavor to allow developers to create open-source software for safety-related systems, Wheeler [24] collects open-source verification tools and links to existing open-proof projects. It is a prerequisite to develop open-proof software, based on this supposition: To formally prove the correctness of the software, the developers also deliver formal specifications and all required artefacts to allow everybody to check the correctness of the software using the available open-source verification tools.

The second endeavor also devises an open-source development process that conforms to the safety standards. To develop the software kernel for the European train control system, a highly complex and safety-related software, Hase initiated the cooperative project OpenETCS [5, 10]. Under the umbrella of a publicly-funded research project, a consortium of European rail operators and equipment manufacturers is formed that fulfills the organizational and technical requirements of the EN 50128 safety standard. The organizational requirements are fulfilled by dedicated governance rules that restrict contributions to contracted partners and put obligations on the staff working on behalf of those companies. The development process is designed to fulfill the safety standard and to support the massively distributed development team. The project also ensures that the technical documents and artefacts can be edited and inspected using open-source tools.

The third endeavor addresses the open-source qualification and certification of existing open-source software for safety-related systems.

The SIL2LinuxMP project [20] plans to certify the base components, i.e., boot loader, root filesystem, Linux kernel and C library bindings, of an embedded GNU/Linux real-time operating systems compliant to Safety Integrity Level 2 (SIL2) according to safety standard IEC 61508 [13], which roughly corresponds to ISO 26262's integrity level ASIL B.

In this paper, we propose to compose platform for complex safety-related application software from existing open-source software. Furthermore, we propose to collaborate on the open-source platform and safety-related activities. The core of our concept is a stripped configuration of the GNU/Linux operating system. In our opinion, the development of this platform in an open-proof approach has several advantages compared to proprietary solutions:

- *Lower cost:* Cost for development and qualification of this complex platform can be shared.
- *Higher quality:* One widely-used implementation matures more quickly, as experience from multiple use cases can be considered.
- *Higher confidence:* Risk classification and effectivity of safety measures are publicly assessed and rated by all partners and potentially also by academic institutions. This accelerates the process to define an accepted state of the art.
- *Higher agility:* Innovative car manufacturers, application software developers and integrators who need additional platform capabilities can collaborate on new features, implement in a dedicated branch and use them in their product development, before the changes are integrated in the main branch.

However, to develop the platform in an open-proof manner, the companies in the automotive software industry must take some organizational actions: First, the open-proof platform development must be organized by a consortium of the interested partners and users, similar to the existing AUTOSAR consortium [1]. Second, the contributing partners must disclose the development and safety processes for the platform and make them compatible to each other. Third, the partners must develop and sign-off a safety concept and qualification goals for this platform. Fourth, to allow independent checking of the verification steps, the partners must provide the verification tools that were developed for the use of this project as open-source software. And fifth, to mature quickly during the development, users of this

[2]The term *open proof* has been coined by David A. Wheeler and Klaus-Ruediger Hase.

platform must detect, isolate and report bugs and shortcomings.

Even after the platform's development has finalized, the car manufacturers can benefit from the collaborative open-proof approach. Once the platform is deployed in production vehicles, the car manufacturers must monitor the platform's behavior in the field. A cooperative monitoring of the common platform has several advantages for the partners: As the platform is running on numerous cars of multiple car manufacturers, flaws in the widely-used system are detected more quickly in the field. Due to legal obligations, all car manufacturers must react if severe issues are detected. Hence, an immediate public reporting of severe issues avoids negligence-related claims and an instantaneous response is proved by public review of the taken actions in the platform software repository. Therefore, companies that report and resolve many issues are recognized as actively rising public safety for autonomous driving.

## 4 Linux Platform Architecture

In this section, we sketch our proposed architecture for the platform for autonomous driving and derive requirements for the Linux operating system.

We presume a mainstream computing hardware that is well supported by Linux, qualified compliant to ISO 26262's ASIL B and for which sufficient real-time capabilities could be demonstrated, e.g., by the OSADL testing lab [4]. There are several systems on chip (SoC) available from several silicon vendors that fulfill those requirements.

We further assume that the hardware offers the key features, symmetric multi-processing, hardware acceleration, e.g. through GPGPUs, Ethernet, CAN or FlexRay communication, and flash-based mass storage. We explicitly exclude the availability and use of application-specific I/O hardware, e.g., GPIO or analog pins, from the platform's hardware design as it increases platform complexity and reduces its reusability.

Figure 1 depicts the different components of the platform running on an electronic control unit. The platform is structured in five parts:

**Application Software Components.** The desired functionality of an electronic control unit is implemented in the application software, which is decomposed into application software components, following the high-cohesion and loose-coupling principle. To achieve loose coupling, application software components are mapped to separate Linux processes and their temporal isolation is realized using a real-time scheduler. Communication between the software components is realized using Linux's efficient interprocess communication mechanisms.

**AUTOSAR Interfaces.** To allow reuse of application software components and assist commissioning in industrial cooperations, all component interfaces are defined employing the AUTOSAR software component template. AUTOSAR provides an established standard metamodel, templates and tools, and a functioning consortium for further extensions. All properties required for the integration, such as memory usage, and scheduling, are described with current AUTOSAR specification templates. Furthermore, the components' behavioral characteristics, such as timing behavior, data and control flow, can be specified with AUTOSAR means that are currently under development and on which the future safety concepts and analyses will rest upon.

**Middleware.** The middleware interconnects application software components and allows them to access platform, communication and system services.

In existing automotive ECUs, this middleware is defined by specifications employing the AUTOSAR metamodel, i.e., the software composition and system template defines all interconnections that are realized by a statically generated middleware software, called the AUTOSAR runtime environment.

According to the requirements in §2, the platform for autonomous driving shall dynamically integrate, remove and update application software components. This requirement cannot be fulfilled with a statically generated software, but needs a discovery mechanism that dynamically establishes communication links between software components at start-up. However, this discovery mechanism can be configured using extracts of the specification with the current AUTOSAR metamodel. There are two open-source implementations that could fulfill the main requirements:

First, the Robot Operating System (ROS), a framework for robotics applications, provides a dynamic middleware with publisher/subscriber communication and a remote-procedure-call mechanism. It was initially developed in academia, but recently also industrial users are engaged to prepare ROS for use in products. It does not provide guaranteed timing behavior and dependable communication yet, but its community plans to address this in the future.

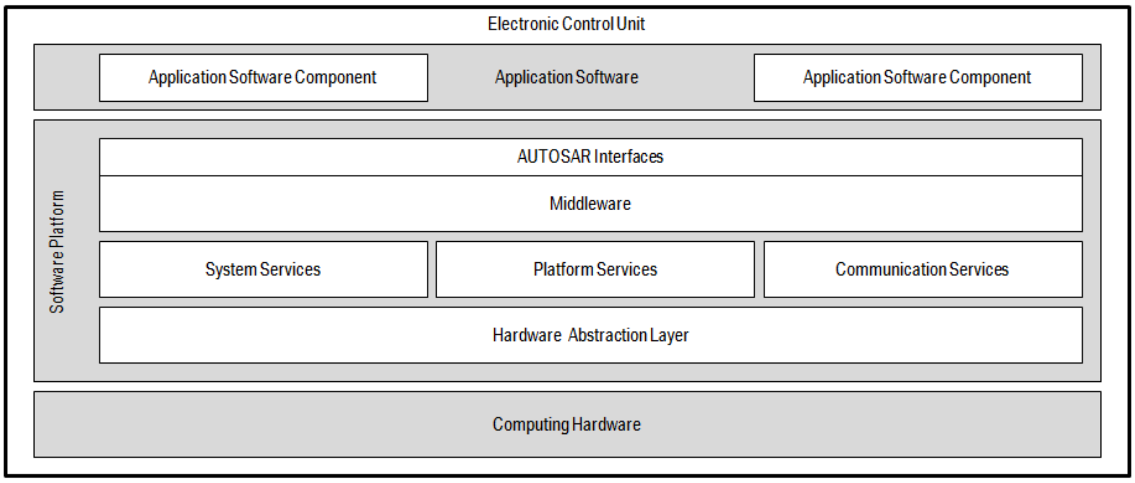

Figure 1: Software platform architecture

Second, OpenDDS is an open-source implementation of the Data Distribution Service for Real-Time Systems (DDS). DDS is a standard for interoperable data exchanges and focuses on scalability, guaranteed timing behavior, dependability and high performance. It is future work to evaluate if the OpenDDS implementation fits the needs for a middleware of the platform.

To employ those frameworks in the platform, they must be augmented by control software that configures the middleware with the provided AUTOSAR descriptions.

**Platform Services.** Platform services are local services that are required to execute application and platform software components. Supported services are CPU scheduling, process isolation, flash storage access, or platform health monitoring. CPU scheduling and process isolation is supported in Linux employing the cgroups mechanisms [17]. Frequent writing on the flash storage due to continuous software updates is supported by the Unsorted Block Image File System (UBIFS) [12]. The open-source boot loader U-Boot [8] is well suited for starting up the platform and updating the boot image.

**Communication Services.** The communication services realize communication with vehicle networks using Ethernet, CAN or FlexRay links. They also support automotive-specific features, such as the ISO 15765-2 transport protocol and a specific network management. The CAN communication is supported by the CAN network driver in Linux kernel [9]. Currently also a Flexray network driver [22] is emerging.

**System Services.** System services provide functionality to participate in vehicle management functions, such as vehicle-level power and mode management. They must be provided by any electronic control unit in a car and are therefore placed in the platform. System services often follow a master-slave or client-server pattern, where the client or slave component is integrated into the common platform. Common system services are diagnosis, coding, logging and tracing, calibration, and debugging.

**Hardware Abstraction Layer.** The hardware abstraction layer consists of device drivers for all the hardware components used by the platform. The required device drivers for all hardware components used by the platform are well supported by Linux.

## 5 Related Work

From our personal exchange with colleagues of various companies in the automotive industry, we are aware of research and development of hardware and software for advanced driving assistant systems. However so far, we have not seen any publications describing the architecture of a possible platform.

The benefits of open-source development from a technical view are motivated and described in “The Cathedral and the Bazaar” [21]. The benefits from a business point of view are further described in a short primer at opensource.org [19]. These benefits also apply to an *open-proof development* in general, and to the described platform.

## 6 Conclusion

In this paper, we showed the requirements on a future software platform for autonomous driving, the benefits of an open-source development and qualification of the software platform, and a platform architecture based on Linux and other open-source software. In future work, we want to increase evidence for the platform requirements and evaluate the quality of the discussed open-source software in the platform. However, our main ambition is to increase momentum for a collaborative open-proof development and establish an open, common and extensible high-quality platform before others are engaged in closed-source, vendor-specific and limited platforms.

**Acknowledgements.** We thank Christoph Ainhauser, Moritz Blume, Raphael Fonte Boa Trinidade and Thomas Schutzmeier for suggesting several textual improvements. We express our gratitude to Simon Fürst, Harald Heinecke, Oliver Nölle, Michael Rudorfer, Tillmann Schumm and Reinhard Stolle for their full support.

## References

[1] AUTOSAR. `http://www.autosar.org/`.

[2] Communication Network Vehicle Road Global Extension. `http://www.converge-online.de`.

[3] Ko-FAS research initiative. `http://www.kofas.de`.

[4] Open Source Automation Development Lab. `http://www.osadl.org`.

[5] openETCS: European Train Control System. `http://www.openetcs.org`.

[6] The Apache HTTP Server Project. `http://httpd.apache.org/`.

[7] Jonathan Corbet. *How the development process works*. The Linux Foundation, 2011.

[8] Wolfgang Denk and Detlev Zundel. The DENX U-Boot and Linux guide for canyonlands. `http://www.denx.de/wiki/DULG/Manual`.

[9] Oliver Hartkopp. The CAN networking subsystem of the Linux kernel. *Proceedings of the 13th international CAN conference*, 2012.

[10] Klaus-Rüdiger Hase. Open proof for railway safety software: A potential way-out of vendor lock-in advancing to standardization, transparency, and software security. *13th Real-Time Linux Workshop*, 2011.

[11] Stefan Holder, Markus Hörwick, and Hariolf Gentner. Funktionsübergreifende Szeneninterpretation zur Vernetzung von Fahrerassistenzsystemen. *Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel (AAET 2012)*, 2012.

[12] Adrian Hunter and Artem Bityutskiy. UBIFS – new flash file system. `http://lwn.net/Articles/275706/`.

[13] IEC. IEC 61508-1 ed2.0: Functional safety of electrical/electronic/programmable electronic safety-related systems, 2010.

[14] Coverity Inc. The 2012 Coverity Scan open source report. 2013.

[15] ISO. EN 50128:2011 railway applications: Communication, signalling and processing systems – software for railway control and protection systems, 2011.

[16] ISO. ISO/FDIS 26262 road vehicles – functional safety, part 1 – 10, 2011.

[17] Paul Menage. Resource control and isolation: Adding generic process containers to the Linux kernel. *9th Linux Symposium*, 2007.

[18] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering Methodology*, 11(3):309–346, 2002.

[19] Open Source Initiative. Open source case for business. `http://opensource.org/advocacy/case_for_business.php`.

[20] OSADL. The OSADL project: SIL2LinuxMP. `http://www.osadl.org/SIL2LinuxMP.sil2-linux-project.0.html`.

[21] Eric S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* O'Reilly, 2001.

[22] Benedikt Spranger. Add FlexRay support. `https://lwn.net/Articles/563254/`, 2013.

[23] The Linux Kernel Organization. The Linux Kernel Archives. `http://www.kernel.org`.

[24] David A. Wheeler and Alan Dunn. Open Proofs. `http://www.openproofs.org`.