# Fusion: A Safe and Secure Software Platform for Autonomous Driving

**Philipp Mundhenk**
Autonomous Intelligent Driving GmbH, München, Germany
philipp.mundhenk@aid-driving.eu

**Enrique Parodi**
Autonomous Intelligent Driving GmbH, München, Germany
enrique.parodi@aid-driving.eu

**Roland Schabenberger**
Autonomous Intelligent Driving GmbH, München, Germany
roland.schabenberger@aid-driving.eu

## Abstract

The vastly increasing amount of software in vehicles, its variability and complexity, as well as the computational requirements, especially for those built with autonomous driving in mind, require new approaches to the structure and integration of software. The traditional approaches of single-purpose embedded devices with integrated software are no longer a suitable choice. New architectures introduce general purpose compute devices, capable of high-performance computation, as well as high variability of software. Managing the increasing complexity, also at runtime, in a safe and secure manner, are open challenges. Solving these challenges is a high-complexity development and integration effort requiring design-time and runtime configuration, approaches to communication middleware, operating system configuration, such as task scheduling, monitoring, tight integration of security and safety, and, especially in the case of autonomous driving, concepts for dynamic adaption of the system to the situation, e.g., fail-operational concepts. We present Fusion, a next-generation software platform supporting the development of autonomous driving systems.

**2012 ACM Subject Classification** Computer systems organization → Embedded software

**Keywords and phrases** middleware, software platform, autonomous driving

**Digital Object Identifier** 10.4230/OASIcs.ASD.2020.2

## 1 Introduction

Traditionally, automotive software has been aligned with hardware, also known Electronic Control Units (ECUs). All communication is performed via signals, either between software components (e.g., AUTOSAR Classic), or between ECUs [2]. In more modern software architectures, especially in use for more complex systems such as vehicles with driver assistance systems, service-oriented communication is introduced (e.g., AUTOSAR Adaptive) [1]. This is supported by new means of communication, among other changes, introduced in the automotive domain, such as Automotive Ethernet[4]. These modern frameworks are typically built on top of real-time capable POSIX operating systems, using operating system processes as the atomic building blocks for applications. Communication is defined and performed between processes and devices. Inside the processes, there is limited to no support for the developer in terms of coordination (multi-threading), introspection, and communication between individual parts of the software.

Software stacks for autonomous driving are significantly larger and more complex than all existing software in vehicles, with a trend to grow larger and ever more complex over time. An example of the architecture of such a stack, including its internal and external interfaces is shown in 1. For developers to be able to focus on advancing functionality, some

2nd International Workshop on Autonomous Systems Design (ASD 2020).
Editors: Sebastian Steinhorst and Jyotirmoy V. Deshmukh; Article No. 2; pp. 2:1–2:6
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

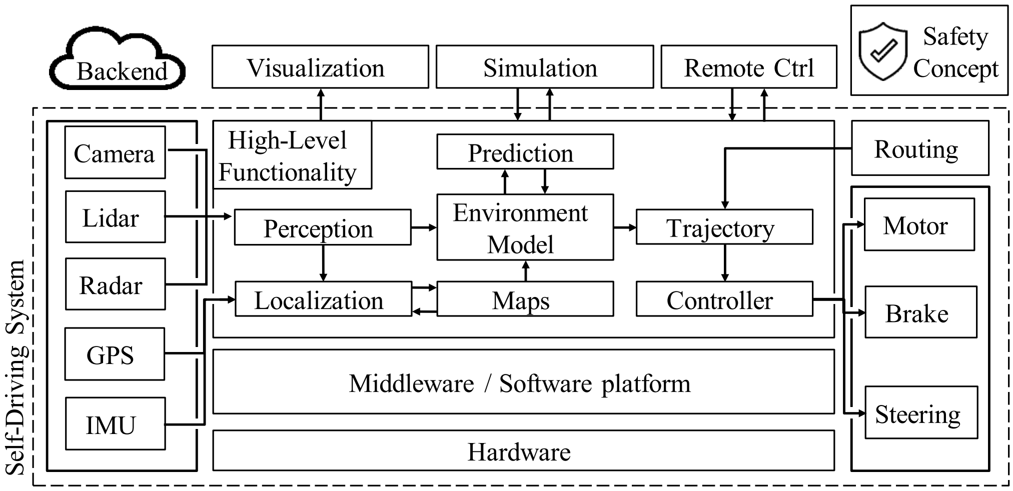

**Figure 1** High-level view of a self-driving system and its interfaces.

support is required. This is especially relevant for performance reasons. Autonomous driving software requires a large amount of performance and the hardware needs to be utilized to the maximum possible degree. Additionally, when running an autonomous vehicle without supervision, the correctness of the software is fundamental.

## 2 Fusion Software Platform

In this paper, we propose the Fusion Software Platform to combat the complexity and performance requirements of autonomous driving software. As shown in Figure 2, each instance of Fusion is running inside a single operating system process. Through the use of a dedicated Operating System Abstraction Layer (OSAL), support for different operating systems is possible and has been implemented in the past. Fusion instances can communicate with each other, across processes and across devices, such as processors or hypervisor partitions, by the help of a pluggable transport mechanism (e.g., using Data Distribution Service (DDS)). All communication means are abstracted by Fusion, so that all application software is independent of the used transport mechanism.

Internally, Fusion improves upon existing software platforms by using a microkernel approach, and combining the benefits of service-oriented and component-based design. The atomic building block is an *Object*. Fusion objects contain properties, methods and events, defined in a dedicated *Object Specification Language (OSL)*. Additionally, custom data structures can be defined. Properties allow the storage of data, whereas events and methods are used to connect to other objects with (method) or without (method) return path. Changes of properties can automatically result in events being fired. Timers are objects, which can emit events at configurable times. OSL is generated into the target code (e.g., C++) to be used by developers when building their functionality (see Figure 3). Developers can implement the functionality, and for this use the full set of features in their target language (e.g., C++) if they wish to do so.

In order to support concurrency, all objects are assigned to execution contexts. Thus, developers are not required to use synchronization primitives, such as mutexes, condition variables, etc. Each execution context consists of a queue and (typically) one or multiple

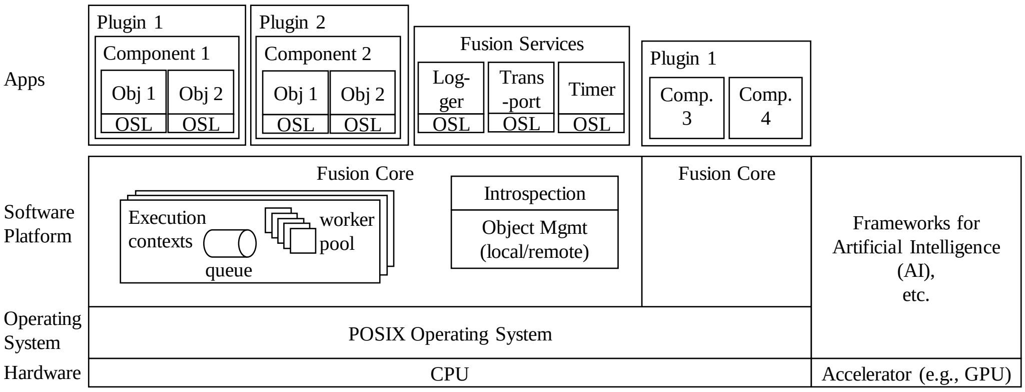

**Figure 2** An example of a software system built with Fusion: Multiple identical instances of the Fusion engine (Fusion core) run in operating system processes and host components, which are implemented through Fusion objects (*Obj*). Execution contexts are used to process messages and services for users are provided. Artificial Intelligence (AI) frameworks already abstract accelerators and are running alongside Fusion.

operating system threads (worker pool). All events and method calls are enqueued in the queue of the execution context containing the receiving object and are processed in order of arrival by the worker(s). If the execution context is located in a different Fusion engine, the pluggable transport mechanism is used. Through this mechanism, objects can be local or remote to an engine. To the application, this is transparent.

As an additional layer of organization, components are introduced. These define interfaces, timing properties, etc. Objects are used to implement components. One to multiple objects or components are packed into plugins, which are stored as libraries and loaded by the Fusion core.

Each Fusion instance is configured through a manifest. These manifests define plugins to load, execution contexts to establish, objects to instantiate, as well as their configuration, among many others. Additionally, connections are explicitly configured in manifests. Connections originate from the event of one object and connect to the method of another object. This allows the passing of events and data, e.g., the change of a property, including the new value, between objects.

In addition to the means for communication, scheduling, management, etc., Fusion also provides a set of basic services, such as distributed Execution Management, Logging, Health Monitoring, Recording, Replay, Storage, etc. These services are implemented on top of Fusion with the same means and Application Programming Interfaces (APIs) as the applications.

Fusion is currently being used as a software platform for the development of autonomous driving functionality at the Autonomous Intelligent Driving GmbH (AID).

## 3 Challenges

Developing a new software platform for complex systems, such as autonomous driving, is a highly complex endeavor. This holds especially true, as safety and real-time behavior are critical aspects of the system under development. In the following, we will outline some challenges in the areas of complexity, performance, safety, and security that such a platform is facing.

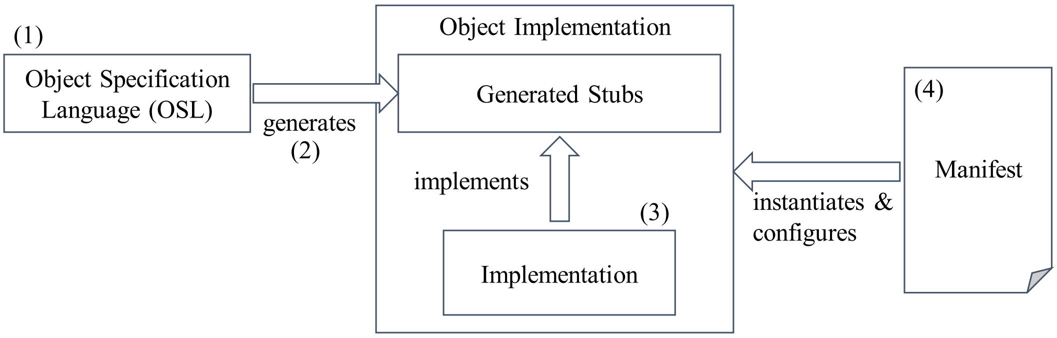

**Figure 3** Process to develop and instantiate an object: (1) Specify in Object Specification Language (OSL), (2) generate stubs, (3) implement against generated stubs, (4) instantiate, configure and connect via manifest.

## 3.1 Complexity

While the introduction of the additional layers is increasing the development speed of applications significantly, while at the same time reducing the number of errors in development, it also introduces complexity. When building a system using Fusion and applications built on top of it, we have to consider Fusion objects, Fusion components, Fusion execution contexts, Fusion plugins, Fusion engines, Hypervisor partitions, physical devices, and potentially multiple systems (e.g., in a fleet of vehicles).

Managing this complexity is not trivial. Objects and components need to be instantiated, often multiple times. The correctness of connections defined in manifests needs to be ensured. This includes syntactical correctness, as well as semantical correctness, e.g., are expected and provided update frequencies, latencies, ASIL level matching, etc. Another aspect of correctness is security: Are all communication participants authenticated and allowed to communicate? While Fusion makes it much harder for developers to create deadlocks across threads, due to the messaging approach between components, it is still possible to create deadlocks on a higher layer, through the connection of objects. Also the creation of loops is possible, leading to large increase of data processing in the system. As Fusion introduces new abstraction levels and means of communication, as well as offering new usage patterns, new approaches to finding such problems have to be developed.

In terms of scheduling, the additional abstraction layers also need to be considered. Anything being scheduled in the system needs to consider execution contexts, the operating system scheduler, potentially hypervisors, etc.

Furthermore, with safety-critical systems and specifically autonomous driving, concepts for redundancy and fail-operational scenarios are required [3]. As in a level 4 self-driving vehicle, no safety driver is available to take over in case of a system failure, the system needs to recover from failures whenever possible [5].

In normal operation, as well as in such failures, the system needs to be ensured to operate correctly. This includes handling crashes of applications, and slow-down due to system load, among many other possible concerns.

Furthermore, being the platform that all functional applications are based on, Fusion needs to support a large selection of different target systems, which in turn can benefit from the abstractions provided by Fusion (e.g., for simulation). These include, but are not limited to vehicles, hardware-in-the-loop setups, multiple generations of different processor architectures, virtualized systems, developer laptops, cloud, etc.

### 3.2 Performance

When building a software platform for autonomous driving, high performance is key. The software system needs to be able to process large amounts of data from a large number of cameras, LiDARs, radars and other sensors with high bandwidth, at low latencies. A software platform needs to keep introduced overhead minimal to avoid slowing down the system, while at the same time requiring all features and benefits listed in Section 2.

Furthermore, due to the complexity and nature of autonomous driving, hundreds of developers in an agile setup are required. Their objects and components need to be correctly orchestrated and connected in the system. Any issues need to be detected as early as possible at design time or, if not otherwise possible, at runtime.

The large amount of input from developers to the system also means a large number of changes. Almost all parameters of the system can change up to hundreds of times per day. Many of these changes might require new computations (e.g., schedule computations). To keep development speed high, all computations caused by such changes need to take a minimum amount of time. To give an order of magnitude: One hour of computation is considered very long in this context.

### 3.3 Safety

Despite the increased complexity of the system, no reduction in safety can be permitted. This is in strong contrast to many other complex automotive software systems, the most complex of which today are found in the area of infotainment, where safety-requirements are often secondary, due to isolation. On the contrary, safety and reliability of the system need to increase when moving from a level 2 self-driving vehicle to level 4, where no driver is available to take over in case of a fault. Thus, ensuring real-time behavior, determinism and integrity of the system are paramount. This holds for all components, such as hardware, operating system and also software platform. However, as Fusion adds the new layers described in Section 2, new ways for guarantees have to be found. Fusion objects might be assigned to the same components and Fusion engine. Furthermore, execution contexts can be shared, as well as address spaces, when objects and components are running in the same Fusion engine.

Thus, a trade-off between the performance requirements as listed above, and the safety requirements need to be found.

### 3.4 Security

Similar to safety, security is an important topic for any connected vehicle. The same holds for autonomous vehicles. Some data in the system needs to be specifically secured (e.g., cryptographic keys). To fulfill the requirements set up in the General Data Protection Regulation (GPDR), additional technical measures need to be taken. These requirements do not only hold for the driver of the vehicle and the passengers, but also the pedestrians around the vehicle, potentially recorded on camera. To secure communication, certain, if not all objects and components need to be authenticated, certain traffic needs to be encrypted.

All of these measures add a potentially very large performance overhead. As in safety, balancing the performance and security requirements is key.

## 4 Design-Time Verification

In the automotive domain, like in many industries extensive testing is used to determine the correctness of a system. However, for such testing approaches, the running system is required. Additionally, there is no guarantee that all cases are covered when using testing.

For complex systems like autonomous driving, runtime testing on the system level is not efficient. Instead, wherever possible, the system should be described in a model at design time, where the consistency and correctness can also be checked. This can solve some of the complexity challenges listed above. Furthermore, optimizations can be performed based on the model (e.g., assignment of software to hardware components). Fusion already offers some means to describe the system at design time through OSL and manifests.

Whenever using modeling approaches, multiple challenges exist. The key is to find the right level of abstraction. If the model is too detailed, it is difficult to handle and check. If the model is too abstract, the benefit of modeling vanishes. Additionally, the model needs to be integrated into the development process to avoid divergence of model and implementation.

With a system as complex as an autonomous driving software stack, the model easily becomes very large. The additional layers introduced by Fusion add to this challenge. Due to the high frequency of changes and the checks needing to be performed on the model, computations need to take minimal time (see also Section 3.2). Furthermore, the model needs to be used by a large number of engineers from different backgrounds (e.g., software, safety, systems, security engineering). Without presentation of the model and tools according to the developers background, the acceptance and usability will be insufficient for a high development speed.

Last, but not least, a model and its tooling face challenges when developed alongside the software (and not upfront). To increase acceptance of the approach, it is essential to maximize the benefit and prioritize parts of the model. E.g., for a first Minimum Viable Product (MVP) of a software, modeling the exact resource consumption to the last bit in memory does not add value, while this might differ when moving closer to the release of, especially a safety-critical, software.

## 5 Conclusion

In this work, we presented Fusion, a new generation of software platform, lowering the size of atomic building blocks for complex autonomous driving systems. Furthermore, we raised a number of challenges, which are relevant for Fusion, as well as other, similar systems, targeting complex software systems, such as those used in autonomous driving.

In future, the fine-grained defined building blocks of Fusion will be enhanced to increase predictability, freedom from interference, as well as real-time behavior. Initial proofs of concepts have been built and development is ongoing. Fusion is actively being used for the development of level 4 autonomous driving in AID.

## References

1 AUTOSAR Consortium. Specification of Communication Management, 2019. Document Identification No 717, Release R19-11.

2 AUTOSAR Consortium. Specification of RTE Software, 2019. Document Identification No 84, Release R19-11.

3 Philip Koopman and Michael Wagner. Challenges in autonomous vehicle testing and validation. *SAE Int. J. Trans. Safety*, 4:15–24, April 2016. doi:10.4271/2016-01-0128.

4 OPEN Alliance Special Interest Group. 100BASE-T1 System Implementation Specification, 2017. Version 1.0.

5 SAE International. Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems J3016, 2018. Release 201806.