From Simulation to Operation: Using Design Time Artifacts to Ensure the Safety of Advanced Driving Assistance Systems at Runtime

Malte Mauritz, Falk Howar, and Andreas Rausch

Institute for Applied Software Systems Engineering,
Technical University Clausthal.
D-38678 Clausthal-Zellerfeld, Germany
firstname.lastname@tu-clausthal.de

Abstract. Advanced driver assistance systems and (semi-)autonomous mobility systems will arguably be the biggest disruption of our everyday life in the next couple of years. The development of such systems comes with legal and technical challenges: Product liability regulations impose high standards on manufacturers regarding the safe operation of advanced driver assistance systems (ADAS). In the Automotive domain, sufficient safety has yet to be proven through extensive and expensive testing. As a consequence, car manufacturers try to move testing effort from the road to simulation. It is not clear, however, how results obtained from simulations transfer to the road. In this paper, we present an approach for leveraging simulation results during road tests. Our approach utilizes runtime monitors that are generated from specifications, test scenarios, and simulated components. These monitors can be used during road tests and during operation for identifying untested situations and for checking functional correctness of an ADAS.

1 Introduction

Driver assistance systems and (semi-)autonomous mobility systems are gaining more importance in mobility carriers, such as vehicles, aircraft or rail- transport systems. Guided by the vision of "zero accidents" (cf. [3]) regulatory authorities require such systems to meet highest standards for ensuring road safety. The product and producer liability (e.g., in Germany: ProdHaftG §1, BGB §823 I, BGB §433) oblige manufacturers of ADAS to ensure that ADAS safely operate in their highly dynamic environments and to eliminate harm for drivers, vehicles, and any persons or objects in their environments.

However, today's conventional engineering methods are not adequate for providing such guarantees: Common vehicle field tests are too expensive and cannot proof the system's dependability. They require too many miles to be driven in order to show that a system is sufficiently safe. One way of reducing the cost of quality assurance is the application of structured testing as well as transferring a significant portion of the testing effort to simulations. It is not clear, however,

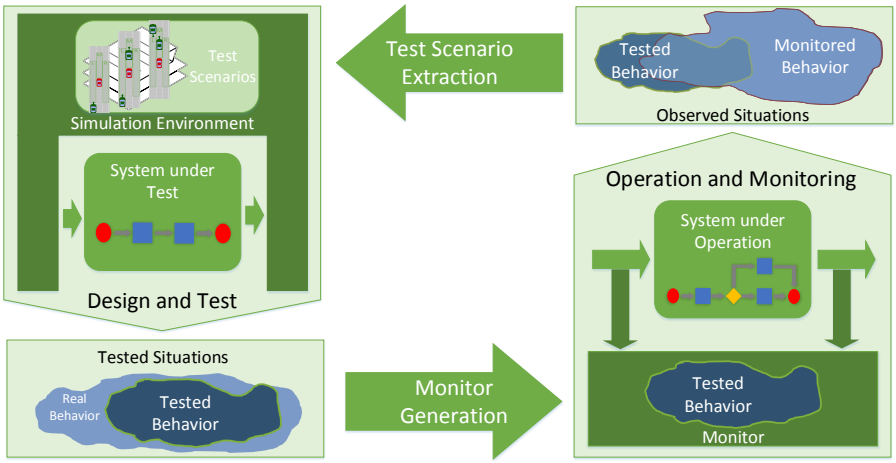


Fig. 1: Overview of the DADAS approach.

how results obtained from simulations transfer to the road. Therefore, methods have to be developed that enable transferring results from simulations to the real world.

The DADAS[1] project [9] aims at developing a sound combination of design time testing and runtime monitoring that will enable the transfer of safety guarantees obtained in simulations to the road. Figure 1 shows an overview of the envisioned approach: In a first step, the ADAS is tested for a set of scenarios in a simulation environment (cf. left side of Fig. 1). Results from the simulation, i.e., the tested behavior and situations, as well as simulation components are used for the generation of runtime monitors. The generated monitors are then used to verify that the system operates within the tested limits during (limited) road tests (cf. right side of Fig. 1). Any untested behavior of the system is recorded in order to extract new test scenarios for further simulation-based testing. This establishes a feedback loop for the iterative enhancement of ADAS.

In this paper, we focus on the generation of monitors from design time artifacts. We present a novel method for generating runtime monitors from specifications, test scenarios, and simulation components. The monitors can be used during development or operation to collect information about untested situations and unsafe behavior of an ADAS. Our main contribution is a conceptual component-based architecture for these monitors along with methods for generating the individual monitoring components from design time artifacts.

Related Work. In the automotive domain, runtime monitoring is primarily used for the correctness and reliability of controllers for physical components. The field of diagnostics mainly uses supervision, fault detection and fault management techniques based on physical parameters and mathematical models of

[1] DADAS is an acronym for Dependable Advanced Driver Assistance Systems

the system's physical behavior to detect and resolve deviations and failures in the component's behavior (cf. [7], [10], [8]). The monitoring of software components in general and the monitoring of the requirements for functional correctness only starts to gain traction in the automotive domain, one example being the standardized "E-Gas" Monitoring Concept.

In academia, a wide range of runtime monitoring approaches for the evaluation system properties exist. They can be categorized in different fields, based on the objectives and property formalization. In the field of requirements monitoring aims to observe a system's runtime behavior in order to detect its deviations from its requirements specification. The assumptions about requirements are formalized in special languages, e.g. FLEA (cf. [5]), or as goal driven specifications (cf. [4], [13]). For timed properties, variations of linear time logic are often used. Events of the system are monitored to evaluate the properties based on the current values of the considered system parameters (cf. [1]). The authors of [6] use the functional safety standard ISO 26262 for the definition of monitored properties for automotive embedded systems.

In general, testing and test coverage is not considered within runtime monitoring fields, but some published works address the combination of testing and runtime monitoring. In [12], the author propose to continuously monitor the achievement of test obligations. After eliminating covered probes for several test runs, leaving only untested program code monitored, the changes of the test coverage can be observed at runtime. The authors of [11] use a technology named software tomography in order to enable the continuous, minimally intrusive analysis and test of software in the field by remotely monitoring and maintaining multiple distributed instances. In [2], the authors use runtime monitors as test oracles in order to identify compromising test cases of system safety properties of a air-traffic control system. None of theses approach addresses the extraction of new test cases for the system and its environmental situations from previously untested system behavior and situations.

Outline. The remainder of this paper is organized as follows. In Section 2, we describe the conceptual architecture and elements of our envisaged runtime monitoring solution. Section 3 provides a case study based on the simulation of an lane change autopilot. We summarize our approach and point out future work in Sec. 4.

2 Generating Safety Monitors for ADAS

We are interested in discovering if an ADAS is operated within conditions that were tested to be safe in simulations. Untested operation conditions can be used as a basis for deriving further test scenarios. We are interested in identifying these conditions and in creating test scenarios from them. Our approach to this challenge is a combination of simulation at design time and monitoring at runtime.

In this section we present a novel conceptual architecture for monitors and a method for generating these monitors. In particular, we establish an interface

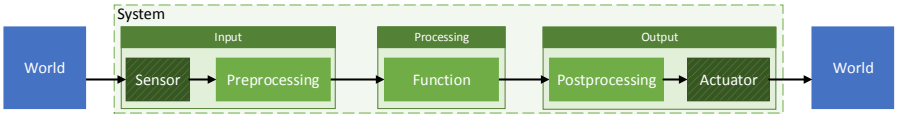


Fig. 2: The input-processing-output architecture of ADAS.

between design time simulation and runtime monitoring that can be used to generate monitors from tested driving scenarios and help to identify new driving scenarios in data recorded from monitors.

Monitoring a complex ADAS. The general architecture of an ADAS can be described by the Input-Processing-Output (IPO) pattern and is shown in Fig. 2. The *sensors* of the vehicle acquire data from the environment. The data is then *preprocessed* in order to generate a single consistent interpretation of the current environmental situation (e.g., the identified objects on surrounding lanes). The *main function* of the ADAS analyzes this situation and determines necessary measures (e.g., changing lanes or adjusting speed) and sends corresponding commands to the output components. The commands are *post-processed* into set values for actuators (e.g., brakes, gearbox, or engine).

We are mainly interested in monitoring the main function of the ADAS to ensure that it operates within tested limits, i.e., for inputs known to be safe from tests. In order to transfer results obtained in simulations to the road, we have to monitor the preprocessing as well: We have to ensure that sensor inputs are preprocessed equivalently by components in the simulation and at runtime. We thus use a combination of two monitors: one for the preprocessing components and one for the main function.

The purpose of our monitors is to gradually ensure that an ADAS is tested to be safe in all relevant conditions. In case of untested conditions, we report these conditions for further testing. In this work we do not consider the application of appropriate measures during operation, e.g., disabling the ADAS, which is a highly complex issue in its own right. Moreover, we do not address the problem of guaranteeing functional safety of the complete system. This would require monitoring of all components and also a strategy for reaching a safe state in case of failure.

Generating monitors from design time artifacts. One of the main challenges in the DADAS scenario is finding a strategy for exchanging information between simulations and operation of an ADAS. In order for the proposed approach to become feasible in practice, we need to generalize from the data that is passed over the interfaces of the ADAS' components. If all our monitors would work at the level of physical signals, the monitors would almost never be able to identify a situation as tested. At the same time, we have to be very careful when generalizing. Otherwise we may identify conditions as safe wrongfully.

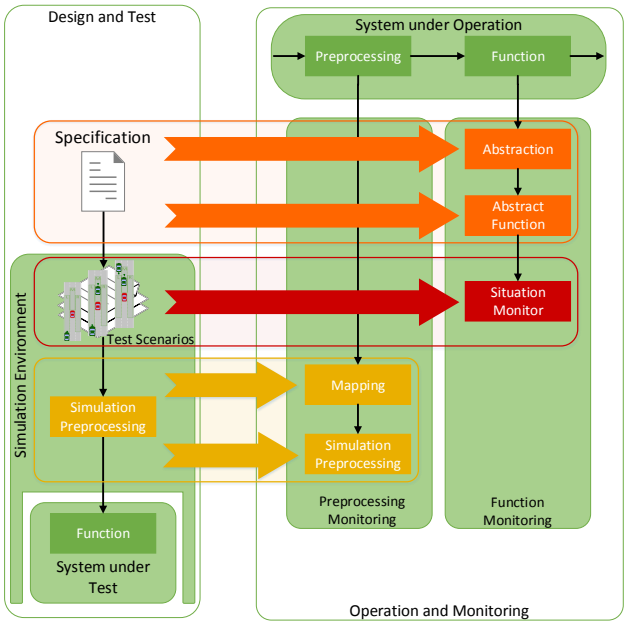


Fig. 3: Generating monitoring components from specifications, test scenarios, and components of a simulation environment.

A sensible level of abstraction can usually be found in the specification of an ADAS (e.g., . . . on the lane to the right of the ego vehicle. . .). We can use this level of abstraction as a basis for passing information from design time testing to operation and vice versa for two reasons.

- The main function of the ADAS is developed against this specification. It should thus implement the specification and we can assume that it is safe to abstract from concrete data values to the level of the specification. In order to not solely rely on this assumption, we can additionally generate an abstract function from the specification and monitor the conformance of the main function of the ADAS to the specification at runtime.
- The specification will often be the source for generating test scenarios and for measuring test coverage (of the specification). If we record situations at this level of abstraction during operation, we can directly feed those situations back into testing.

We have established that we generate two different monitors: one for the preprocessing and one for the main function. We have also discussed how these monitors are generated from design time artifacts. Figure 3 shows an overview of how monitoring components are generated from design time artifacts, in particular specifications, test scenarios, and simulation components. We describe the architecture of the resulting monitors in the remainder of the section.

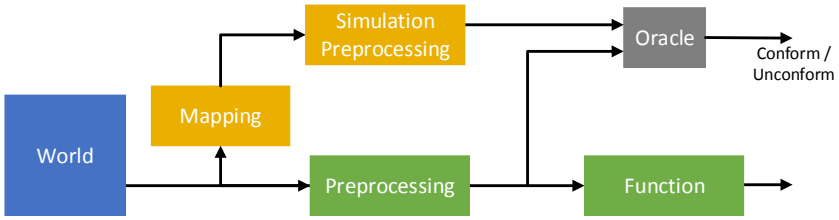


Fig. 4: Monitoring the preprocessing of environmental data.

Monitoring the preprocessing components. We use components from the simulation to validate the preprocessing of the ADAS. We check the equivalence of the two sets of preprocessing components in order to ensure that tested scenarios transfer from simulation to the road. Please note that this is not a general verification of the data fusion performed by these components.

Figure 4 shows the architecture of the monitor for the preprocessing of an ADAS. The bottom line of components are part of the original ADAS' processing chain. In this setup, the real sensor data of the ADAS is mapped to the format of the input expected by preprocessing components in the simulation environment. We expect the two sets of preprocessing components to be almost identical, i.e., we assume that the simulation includes realistic preprocessing code and (simulated) sensor data. The results of the preprocessing of the ADAS is then compared to the results of the preprocessing from the simulation environment. In case the results differ, we cannot assume to be in a tested situation.

Monitoring the main function of the ADAS. Figure 5 shows the components we use for monitoring the main function of the ADAS. The bottom line of components are part of the original ADAS' processing chain. We use components that abstract the input and output of the main function to the level of the specification. These abstract inputs and outputs are then used as a basis for monitoring functional correctness and for logging encountered unverified situations.

The monitors verify correctness by comparing the main function of the ADAS to the abstract function that is derived from the specification. The function is safe if the (abstracted) output of the main function is covered by the output of the abstract function, i.e., if the concrete output is allowed by the specification.

During development, the situation monitor logs the abstracted inputs of the main function, e.g. the situations, for which we are interested in generating more or better test scenarios during development. During operation, the monitor is equipped with a database of situations that were tested to be safe in simulations. In case the monitor observes an untested situation, no guarantees about the ADAS' safety can be made.

We have evaluated this concept for an architecture in a small case study successfully. The results from the evaluation are presented in the next section.

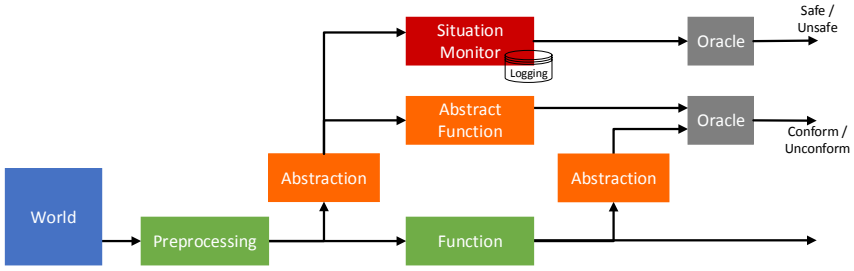


Fig. 5: Monitoring the main function of an ADAS.

3 Preliminary Evaluation: The Lane Change Autopilot

In order to evaluate our monitoring concept for the main functionality of ADAS (cf. Section 2), we implemented a basic simulation environment and a simplified lane change autopilot according to a given specification. From the specification we also derived the components for the abstractions and the abstract function (cf. Fig. 5). We then used these components to test the feedback loop envisioned by the DADAS project: First, we simulated test scenarios and recorded safe situations. For this initial evaluation, we did not use a prototype vehicle (This will be done in a later step). Instead, we simulated random driving scenarios in order to find untested situations and functional errors. The remainder of this Section covers the setup and results of this preliminary evaluation.

Specification. We wrote a simple specification for the autopilot as a basis for the case study and restricted ourselves to overtaking on multi-lane roads. An existing specification of the behavior of a dead spot warning system served as guidance and example. As shown in Fig. 6, the specification assumes three road lanes and defines one zone to be observed on each lane.

For the two neighboring lanes left and right of the ego vehicle, these zones cover the space that would be needed for a safe change to these lanes. The zone in front of the ego vehicle is used for triggering lane changes in our lane change autopilot. Our specification just requires that the autopilot does not overtake vehicles in its lane on their right side (this is illegal on German highways): If an object is observed in the zone in front of the vehicle, the autopilot may not initiate a lane change to its right lane.

Implementation. We have implemented two versions of the lane change function. One version is compliant with our specification while the other one is not. Both versions take a list of objects around the ego vehicle as input and process the target lane for the ego vehicle as output. We also implemented an abstract function in order to check the functional correctness of both versions at runtime.

While the architecture of our implementation closely resembles the components present in Fig. 2, we have abstracted from some of the details inside the single components. Our sensors, e.g., work with world objects instead with

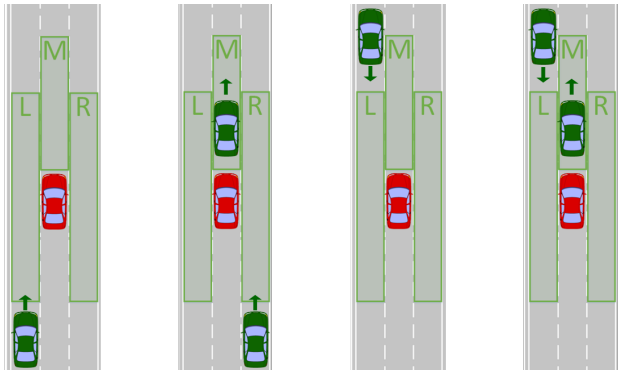


Fig. 6: Scenarios for the Advanced Lane Changing Assistant

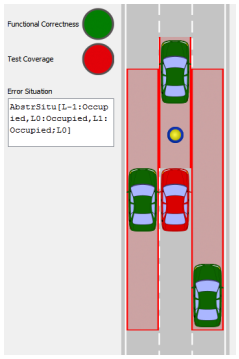
mileages of light waves and we have not implemented the execution of the lane change itself. On the other hand, since we were primarily interested in evaluating the architectural pattern presented in Sec. 2, we have implemented all monitoring components shown in Fig. 5.

Figure 7 shows the visualization of our simulation environment. The main component overlays the current abstract situation with the objects around the ego vehicle. Zones are colored red when they are occupied by a vehicle. The target lanes of the concrete and the abstract function are shown as blue (yellow respectively) points in front of the ego vehicle on the three lanes. While the yellow points mark the possible target lanes of the abstract function, does the blue point symbolize the concrete target lane of the ADAS.

Experiments. We have conducted a series of experiments during which we first trained both versions of the lane change autopilot with a given test set of test scenarios. In a second step, we used the trained situation monitors while simulating random driving scenarios.

Figure 6 shows an excerpt of our initial set of test scenarios at design time used to train the situation monitors. In all displayed test scenarios, a vehicle enters the zone required for the lane change from behind or the front. In addition, some test scenarios place a vehicle in front of the ego vehicle in order to initiate a lane change. Figure 7b displays the set of situations that were recorded during testing by the situation monitor. Situations are pairs of zones on lanes (occupied or unoccupied) and possible target lanes.

Results. In our experiments, the functional correctness was checked correctly at the level of the abstract function. We were also able to train the situation monitor and then discover new (abstract) situations at runtime. These new situations could then be used as the basis for the iterative improvement of the test coverage by additional test scenarios. With increasing number of iterations, the most common situations had been tested and the test coverage of the ADAS reached a



(a) Visualization

```
----------Knowledge----------
Lane Occupation (LEFT |MID|RIGHT)| TargetPoints (LEFT | MID |RIGHT)
[Occupation: L(-1):Free,L(0):Free,L(1):Free|L(-1),L(0),L(1)]
[Occupation: L(-1):Blocked,L(0):Free,L(1):Free|L(0),L(1)]
[Occupation: L(-1):Free,L(0):Free,L(1):Blocked|L(-1),L(0)]
[Occupation: L(-1):Free,L(0):Blocked,L(1):Free|L(-1),L(0)]
[Occupation: L(-1):Blocked,L(0):Blocked,L(1):Free|L(0)]
[Occupation: L(-1):Free,L(0):Blocked,L(1):Blocked|L(-1),L(0)]
--------------------
```

(b) Tested Situations Database

Fig. 7: Implementation of the function's runtime monitoring

stable level, when the monitor only recognized exceptional situations as untested situations.

These initial results indicate the feasibility of our approach for the runtime monitoring of ADAS. We are currently working on a refined version of this case study, using the prototype of an autopilot with automatic lane change. The refined version will be implemented in ADTF[2] and simulated in VTD[3] - two common tools for the development of ADAS. In the refined case study, we are also evaluating how one can generate monitors for the preprocessing components of an ADAS using components from the simulation.

4 Conclusion

The DADAS approach addresses safety and correctness of advanced driver assistance systems. It consists of two parts - the verification of the ADAS in simulations for a finite set of defined test scenarios at design time and the monitoring of the ADAS at runtime. The runtime monitoring ensures that the system and its environment remain within the behavior and situations which have been verified in simulations at design time. We have presented an conceptional framework for the runtime monitoring based on (i) decomposing the monitoring problem and on (ii) using and simulation components and artifacts for defining monitors: In the presented approach the monitoring task is split into monitoring of the actual main function and monitoring the preprocessing components. We have presented a conceptual architecture for the necessary monitors and have discussed how the necessary components can be derived from design time artifacts. We have performed an initial evaluation of our approach using a small case study.

We are currently performing a second, larger case study on an automated lane changing autopilot. In this bigger case study, we have derived abstractions

[2] ADTF is an acronym for Automotive Data and Time-Triggered Framework
[3] VTD is an acronym for Virtual Test Drive

and the abstract function successfully in a structured fashion. As a next step, we will try to automate the generation of monitoring components by using model-based methods and evaluate their performance. We are currently investigating how to use components of the development and simulation environment (ADTF and VTD) for the equivalence check of the preprocessing components. The main goal of this second case study is to deploy our framework in a car eventually and to test the transfer from simulation to operation in the field.

References

1. Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *TOSEM*, 2011.
2. Marko Dimjašević and Dimitra Giannakopoulou. Test-case generation for runtime analysis and vice versa: verification of aircraft separation assurance. In *ISSTA*, pages 282–292. ACM, 2015.
3. EC–European Commission. Roadmap to a single European transport area - Towards a competitive and resource efficient transport system. *White Paper (COM (2011) 144)*, 2011.
4. M.S. Feather, S. Fickas, a. Van Lamsweerde, and C. Ponsard. Reconciling system requirements and runtime behavior. *9th Int. Workshop on Software Specification and Design*, 1998.
5. Stephen Fickas and Martin S Feather. Requirements monitoring in dynamic environments. In *IEEE 2nd Int. Symposium on Requirements Engineering*, pages 140–147. IEEE, 1995.
6. Donal Heffernan, Ciaran MacNamee, and Padraig Fogarty. Runtime verification monitoring for automotive embedded systems using the iso 26262 functional safety standard as a guide for the definition of the monitored properties. *Software, IET*, 8(5):193–203, 2014.
7. Rolf Isermann. *Fault-Diagnosis Applications*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
8. R. Marino, S. Scalzi, P. Tomei, and C.M. Verrelli. Fault-tolerant cruise control of electric vehicles with induction motors. *Control Engineering Practice*, 21(6):860–869, 2013.
9. Malte Mauritz, Andreas Rausch, and Ina Schaefer. Dependable ADAS by Combining Design Time Testing and Runtime Monitoring. In *FORMS/FORMAT 2014, 10th Int. Symp. on Formal Methods*, pages 28–37, 2014.
10. Javad Mohammadpour, Matthew Franchek, and Karolos Grigoriadis. A survey on diagnostics methods for automotive engines. *Int. Journal of Engine Research*, 2011.
11. Alessandro Orso, Donglin Liang, Mary Jean Harrold, and Richard Lipton. Gamma System: Continuous Evolution of Software after Deployment. *ACM SIGSOFT Software Engineering Notes*, 27(4):65, 2002.
12. C. Pavlopoulou and M. Young. Residual test coverage monitoring. *ICSE*, 1999.
13. Yiqiao Wang, Sheila A McIlraith, Yijun Yu, and John Mylopoulos. An automated approach to monitoring and diagnosing requirements. In *ASE*, pages 293–302. ACM, 2007.