# Towards a Reliable and Context-Based System Architecture for Autonomous Vehicles

**Tobias Kain**
Volkswagen AG, Wolfsburg, Germany
tobias.kain@volkswagen.de

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

**Julian-Steffen Müller**
Volkswagen AG, Wolfsburg, Germany
julian-steffen.mueller@volkswagen.de

**Hans Tompits**
Technische Universität Wien, Austria
tompits@kr.tuwien.ac.at

**Maximilian Wesche**
Volkswagen AG, Wolfsburg, Germany
maximilian.wesche@volkswagen.de

**Hendrik Decke**
Volkswagen AG, Wolfsburg, Germany
hendrik.decke@volkswagen.de

## Abstract

Full vehicle autonomy excludes a takeover by passengers in case a safety-critical application fails. Therefore, the system responsible for operating the autonomous vehicle has to detect and handle failures autonomously. Moreover, this system has to ensure the safety of the passengers, as well as the safety of other road users at any given time. Especially in the initial phase of autonomous vehicles, building up consumer confidence is essential. Therefore, in this regard, handling all failures by simply performing an emergency stop is not desirable. In this paper, we introduce an approach enabling a dynamic and safe reconfiguration of the autonomous driving system to handle occurring hardware and software failures. Since the requirements concerning safe reconfiguration actions are significantly affected by the current context the car is experiencing, the developed reconfiguration approach is sensitive to context changes. Our approach defines three interconnected layers, which are distinguished by their level of awareness. The top layer, referred to as the *context layer*, is responsible for observing the context. These context observations, in turn, imply a set of requirements, which constitute the input for the *reconfiguration layer*. The latter layer is required to determine reconfiguration actions, which are then executed by the *architecture layer*.

**2012 ACM Subject Classification** Computer systems organization → Reconfigurable computing

**Keywords and phrases** autonomous driving, fail-operational systems, context-based architecture, application placement, optimization, monitoring

**Digital Object Identifier** 10.4230/OASIcs.ASD.2020.1

## 1 Introduction

Nowadays, vehicles are equipped with various advanced driver assistance systems that support the driver while operating the vehicle. Actions that modern vehicles are capable of doing are, for instance, keeping the distance to a preceding vehicle, autonomous parking, or switching lanes on highways. Although these functions are highly reliable and well tested, the driver is still constrained to monitor their behavior and take over control, if required [3].

As far as fully autonomous vehicles are concerned, such takeover actions by passengers are excluded [13]. Therefore, to guarantee the safety of the passengers and other road users in case an occurring failure causes a safety-critical driving application to misbehave, the system responsible for operating the car has to be designed in a *fail-operational manner*, i.e., the system has to handle hardware and software failures autonomously.

In this paper, we present an approach capable of quickly recovering a safe system state after an occurrence of a hardware or software failure so that the driving mission can be continued. Since various parameters of a system configuration depend on the context the vehicle is currently experiencing, our reconfiguration approach is based on system optimization actions which adjust the system according to the context at hand. Among the different dimensions which are taken into account, our context-aware reconfiguration approach allows to dynamically adjust the safety requirements to the present situation, enabling an increased safety of the system. In case an occurrence of a failure causes the system safety level to drop below a certain threshold, our approach performs an emergency stop.

The context-based reconfiguration feature of our method is based on a layered architecture, defining three interconnected layers, which are distinguished by their level of awareness: The top layer, referred to as the *context layer*, extracts context information from the given input. The output of the context layer is then in turn used as the input for the layer responsible for determining the configuration, called the *reconfiguration layer*. Finally, the application placement, i.e., the assignment of application instances with computing nodes, is then taken care of by the *architecture layer*, which also implements means to monitor the system state.

The paper is organized as follows: Section 2 introduces our general approach for a reliable context-based system architecture for use in autonomous vehicles. Section 3 discusses the methods used for extracting and representing the context. Section 4 illustrates the characteristics and challenges of a context-based reconfiguration. Section 5 gives an overview of the challenges involved in applying new configuration and monitoring system changes. The paper concludes in Section 6 with a discussion on related approaches and future work.

## 2 The General Approach

Figure 1 shows the general framework of our approach for a reliable context-based system architecture, which defines three interconnected logical layers, whereby each layer comprises a set of interrelated tasks, providing distinct levels of awareness, viz. *context awareness*, *safety awareness*, and *self awareness*.

The top layer, dealing with the first type of awareness, is accordingly referred to as the *context layer*. This layer determines the current context the vehicle is in and extracts requirements affecting the actions of lower layers. Mission goals, like the target destination or the level of entertainment requested by the driver, or environment information, like the current weather situation or road and traffic conditions, are examples for parameters influencing action decisions.

The requirements determined by the context layer are used as input for the *reconfiguration layer*. This layer evaluates the received requirements and plans further actions considering the current context. These actions include, for example,

- selecting a set of applications,
- determining their redundancy and hardware segregation requirements,
- computing valid reconfiguration actions, as well as
- optimizing the entire system architecture.

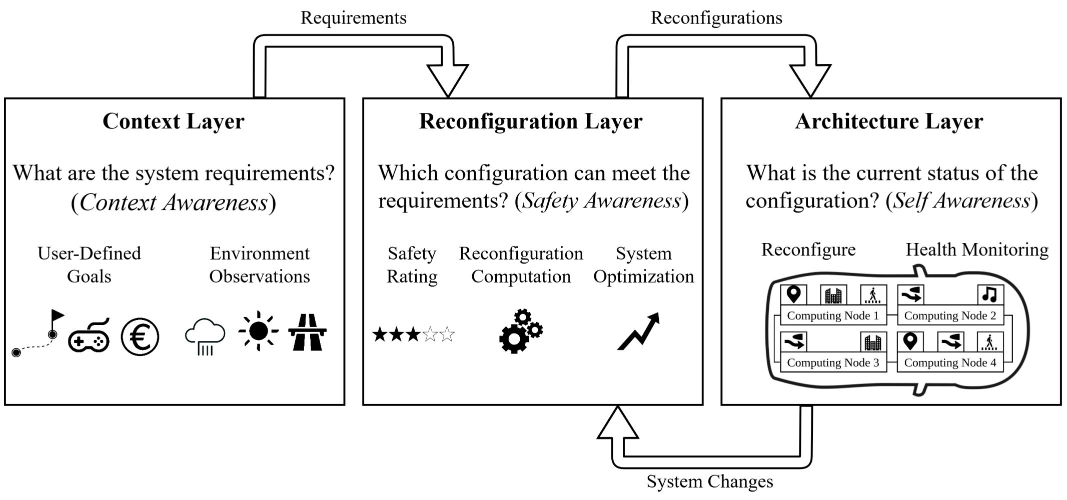


**Figure 1** The three logical layers used in our approach and their relationships. The layers provide distinct levels of awareness.

The reconfiguration measures determined by the reconfiguration layer are then executed by the *architecture layer*. This layer is responsible for distributing application instances among the available computing nodes as instructed by the layer situated above, whereby a minimum level of safety has to be preserved. Furthermore, this layer also implements monitor mechanisms that control the health of the hardware and software components the car is equipped with. In case a system change is observed, the reconfiguration layer is informed such that a new configuration is determined.

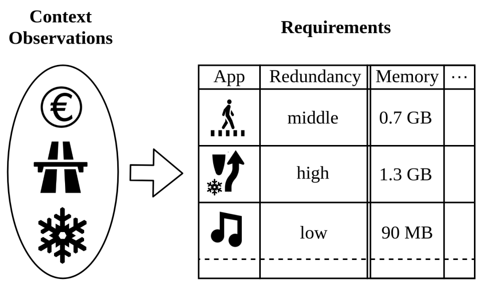
## 3 Context Extraction

Adjusting the system configuration according to the current context first requires the extraction of context observations from environmental parameters. These observations then imply a set of requirements, which are used as input for the subsequent reconfiguration actions.

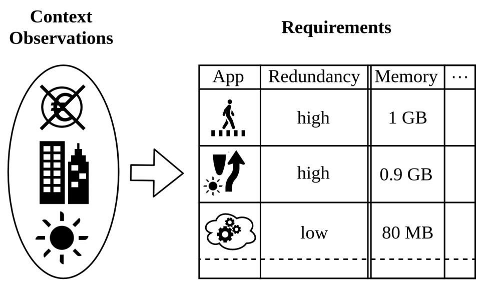
Figure 2 illustrates two use cases that show that distinct sets of context observations imply distinct sets of requirements.

In the first use case, depicted in Figure 2a, the passenger of an autonomous taxi booked a premium ride. Furthermore, we assume that the vehicle is currently driving in snowy weather on a highway. From these context observations, a set of required software applications can be implied. This set may, for example, include applications for detecting pedestrians, planning trajectories taking the rough weather conditions into account, as well as entertainment applications that are included in the ride due to a booked premium package. Moreover, from the context observations, we can imply the safety-criticality of the respective applications and thus the required level of redundancy, as well as other performance parameters.

The use case illustrated in Figure 2b, on the other hand, assumes a low-budget ride in an urban environment under good weather conditions. Consequently, the set of required applications include, for example, a pedestrian detection module. The demanded level of redundancy of this application is high, as this application is considered safety-critical in the current context since many pedestrians are present in urban environments. Note that in the first use case, the required level of redundancy of the same application is lower since



**(a)** A premium ride on a highway in wintry conditions. These context observations imply, for example, that the set of required applications include a passenger detection, a trajectory planner which takes the rough weather conditions into account, and an entertainment application.



**(b)** A low-budget ride in an urban environment under good weather conditions. These context observations imply, for example, that the set of required applications include a passenger detection, a trajectory planner which takes the good weather conditions into account, and an application for computing tasks received from cloud services.

**Figure 2** Two use cases showing the correlation between context observations and requirements. The two distinct scenarios imply a distinct set of requirements.

on highways, encountering pedestrians is unlikely. For the pedestrian detection module, a medium level of redundancy can be, for example, satisfied in case one redundant module is executed. On the other hand, the level of redundancy can be considered high if two redundant instances of this module are executed.

The discussed use cases illustrate the two main challenges of the context layer: Extracting context observations and implying requirements.

The former task, extracting context observations, necessitates perceiving environment parameters. These parameters are, for example, determined by sensors the car is equipped with, communicating with backend services, and interacting with the passengers.

The second task, implying requirements from context observations, requires methods for specifying implication rules. Therefore, the system architecture designers, as well as the application developers, have to define requirements for different contexts. A conceivable approach for representing such rules is employing *answer-set programming* [2], a declarative problem-solving approach based on logic programming for which sophisticated solver technology exists [7, 4].

## 4 Context-Based Reconfiguration

The task of determining a context-based reconfiguration, i.e., a mapping between application instances and computing nodes that respects the prevailing context, is not trivial since the placement decisions depend on various parameters. We refer to this problem as the *application placement problem.*

This problem is not only limited to our problem setting, but the placement of applications on computing nodes is indeed a well-studied topic in other fields too. In particular, research on cloud and edge computing has addressed this problem, like, e.g., approaches for optimizing properties like energy consumption [8], network traffic load [9], and resource utilization [5] have been discussed in the literature.

Generally speaking, the input of the application placement problem is a set of application instances and a set of resources, like, e.g., computing nodes, operating systems, or communication links. Furthermore, we define for each application instance and each resource, a set of parameters including, for example, performance parameters such as the minimum required memory and CPU demand, as well as safety parameters like the minimum required level of redundancy and hardware segregation. These parameters have to be specified by the system architecture designers and the application developers. The output of the application placement problem is an assignment that maps each instance to exactly one node.

In order to restrict the number of valid assignments, constraints based on the specified parameters can be defined. Depending on the specified constraints, either none, one, or multiple valid assignments exist. In case that there are different solutions, an optimization function can be defined that specifies which assignments are the most desired.

This optimization function also depends on the current context. Therefore, an approach allowing a context-based update of the optimization goal leads to configurations that are well adjusted to the current situation.

For solving the application placement problem, various optimization approaches are applicable. The options range from integer linear programming and evolutionary game theory [12] to reinforcement learning approaches [1].

## 5 Architecture Interaction

The architecture layer of our approach comprises the tasks responsible for interacting with the architecture, i.e., the application instances and computing nodes.

One main task of this layer is to apply the reconfiguration actions determined by the reconfiguration layer. The challenge thereby is to ensure a fast, safe, and organized configuration roll-out. Furthermore, it has always to be guaranteed that the reconfiguration actions do not decrease the level of safety. Therefore, safety-validation operations have to be executed prior to the configuration roll-out.

Besides applying reconfiguration actions, also monitoring the state of the computing nodes and the executed application instances is an important task.

Self-awareness requires monitoring the status of the system to maintain an operational state. Monitoring the system, in turn, depends in general on the observation of several level-specific data. Concerning safety, different levels may define different requirements for a minimum operational capability.

Since full vehicle autonomy excludes human takeover actions, classical failure tolerance is not sufficient as errors may have various causes and interference effects. Failure handling requires knowledge of cross-layer dependencies. Thus, system monitoring and self-awareness are cross-layer problems [14].

In case a failure is detected, the reconfiguration layer is notified to reconfigure the system to obtain a safe system sate. If safety-critical applications are affected by the failure, the reconfiguration layer has to ensure that lost functionality is recovered within a short time. An approach addressing this challenge, called FDIRO, standing for “fault detection, isolation, recovery, and optimization”, has been introduced in a recent paper [6], adopted from a similar method from the aerospace domain [16].

## 6 Conclusion

In this paper, we introduced a three-layered approach towards implementing a reliable and context-based system architecture in autonomous vehicles. By employing this approach, we anticipate an increase in safety, enabled by a fast and context-oriented reconfiguration in case a hardware or software failure is detected.

To the best of our knowledge, the introduced safety and context-aware configuration approach for autonomous vehicles is novel. However, in the past, efforts in the automotive research field focused on developing concepts for context-aware advanced driver assistance systems [15, 11]. Context-awareness of applications is also pursued in other research fields [10].

Since the advance of autonomous vehicles is imminent, further work concerning each layer of our approach for context-based system architecture is necessary. In our future research activities, we plan to implement a simulator to show the feasibility of our proposed architecture.

## References

1 Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *CoRR*, abs/1611.09940, 2016. `arXiv:1611.09940`.

2 Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

3 Karel A Brookhuis, Dick de Waard, and Wiel H. Janssen. Behavioural impacts of advanced driver assistance systems–An overview. *European Journal of Transport and Infrastructure Research*, 1(3), 2019.

4 Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. clasp: A conflict-driven answer set solver. In *Proceedings of 9th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR 2007*), volume 4483 of *Lecture Notes in Computer Science*, pages 260–265. Springer, 2007.

5 Fatma Ben Jemaa, Guy Pujolle, and Michel Pariente. QoS-aware VNF placement optimization in edge-central carrier cloud architecture. In *Proceedings of the 2016 IEEE Global Communications Conference* (*GLOBECOM 2016*), pages 1–7, 2016.

6 Tobias Kain, Hans Tompits, Julian-Steffen Müller, Philipp Mundhenk, Maximilan Wesche, and Hendrik Decke. FDIRO: A general approach for a fail-operational system design, 2020. Submitted draft. Abstract accepted for presentation at *30th European Safety and Reliability Conference* (*ESREL 2020*).

7 Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

8 Bo Li, Jianxin Li, Jinpeng Huai, Tianyu Wo, Qin Li, and Liang Zhong. EnaCloud: An energy-saving application live placement approach for cloud computing environments. In *Proceedings of the 2009 IEEE International Conference on Cloud Computing* (*CLOUD-II 2009*), pages 17–24, 2009.

9 Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *Proceedings of the 2010 IEEE International Conference on Computer Communications* (*INFOCOM 2010*), pages 1–9, 2010.

10 Marco Mori, Fei Li, Christoph Dorn, Paola Inverardi, and Schahram Dustdar. Leveraging state-based user preferences in context-aware reconfigurations for self-adaptive systems. In *Proceedings of the 9th International Conference on Software Engineering and Formal Methods* (*SEFM 2011*), volume 7041 of *Lecture Notes in Computer Science*, pages 286–301. Springer, 2011.

11 Andry Rakotonirainy. Design of context-aware systems for vehicles using complex system paradigms. In *Proceedings of the CONTEXT 2005 Workshop on Safety and Context*, volume 158 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

12 Yi Ren, Junichi Suzuki, Athanasios Vasilakos, Shingo Omura, and Katsuya Oba. Cielo: An evolutionary game theoretic framework for virtual machine placement in clouds. In *Proceedings of 2014 International Conference on Future Internet of Things and Cloud* (*FiCloud 2014*), pages 1–8, 2014.

**13** SAE International. Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems. *SAE Standard J3016*, 2014.

**14** Johannes Schlatow, Mischa Möstl, Rolf Ernst, Marcus Nolte, Inga Jatzkowski, Markus Maurer, Christian Herber, and Andreas Herkersdorf. Self-awareness in autonomous automotive systems. In *Proceedings of the 20th Conference & Exhibition on Design, Automation & Test in Europe* (*DATE 2017*), pages 1050–1055. European Design and Automation Association, 2017.

**15** Gereon Weiss, Florian Grigoleit, and Peter Struss. Context modeling for dynamic configuration of automotive functions. In *Proceedings of the 16th International IEEE Conference on Intelligent Transportation Systems* (*ITSC 2013*), pages 839–844, 2013.

**16** Ali Zolghadri. Advanced model-based FDIR techniques for aerospace systems: Today challenges and opportunities. *Progress in Aerospace Sciences*, 53:18–29, 2012.