Progress in Computing Applications

DLINE JOURNALS

Print ISSN: 2278 – 6465
Online ISSN: 2278 – 6473

PCA 2025: 14 (2)
https://doi.org/10.6025/pca/2025/14/2/69-76

Context-Aware and Resilient System Architecture for Autonomous Vehicles

Tobias Kain
Volkswagen AG, Wolfsburg. Germany

Philipp Mundhenk
Autonomous Intelligent Driving GmbH, München. Germany

Julian-Steffen Müller
Volkswagen AG, Wolfsburg. Germany

Hans Tompits
Technische Universität Wien. Austria

Maximilian Wesche
Volkswagen AG, Wolfsburg. Germany

Hendrik Decke
Volkswagen AG, Wolfsburg. Germany

ABSTRACT

This paper presents a novel, three-layered system architecture designed to enhance the reliability and safety of autonomous vehicles by dynamically adapting to contextual changes. The architecture consists of the context layer, reconfiguration layer, and architecture layer. The context layer extracts environmental and operational data, such as weather, traffic, and user preferences, and derives requirements from these inputs. The reconfiguration layer uses these requirements to plan adaptive actions, such as selecting software applications, assigning redundancy levels, and optimizing hardware deployment. The architecture layer then implements these changes on the vehicle's computing infrastructure, ensuring operational safety through monitoring and validation.

Two illustrative use cases demonstrate how distinct driving scenarios—like premium highway rides or budget urban rides—influence application needs and redundancy levels. The application placement problem, which involves mapping applications to computing nodes while optimizing for constraints like CPU demand and safety, is tackled using techniques such as linear programming and reinforcement learning. System self-awareness is maintained through continuous monitoring, enabling swift reconfiguration in the event of failures. The proposed approach is unique in its application of full-stack context awareness to autonomous systems, drawing inspiration from strategies in aerospace fault management. Future work includes implementing a simulator to validate the architecture's real-world feasibility.

Keywords: Autonomous Vehicles, Context-Aware Architecture, Resilient System

Received: 10 January 2025, Revised 4 March 2025, Accepted 18 March 2025

1. Introduction

Currently, automobiles come equipped with a range of sophisticated driver assistance technologies that aid individuals while driving. Features that contemporary vehicles can perform include maintaining a safe distance from the car ahead, parking autonomously, and changing lanes on highways. Although these capabilities are highly dependable and rigorously tested, drivers are still required to oversee their functioning and regain control if necessary. When it comes to fully autonomous vehicles, such interventions from passengers are not anticipated. Consequently, to ensure the safety of both passengers and other road users in the event of a malfunction affecting a critical driving function, the system managing the vehicle must be designed to operate reliably in the face of failures, meaning it must autonomously manage both hardware and software malfunctions. In this document, we introduce a strategy that can swiftly restore a safe operating state following a hardware or software failure, allowing the driving task to be resumed. Since various aspects of a system's configuration rely on the context the vehicle is currently experiencing, our reconfiguration strategy is centered on system optimization activities that adapt the system to the current context. Among the multiple factors considered, our context-aware reconfiguration method enables dynamic adjustment of safety requirements to fit the existing circumstances, enhancing the overall safety of the system.

Should a failure occur that results in the system's safety level falling below a specific threshold, our method initiates an emergency stop. The context-based reconfiguration aspect of our approach is structured around a layered framework, which consists of three interrelated layers that vary in their awareness level: The highest layer, known as the context layer, extracts contextual information from the provided input. The output from the context layer is subsequently utilized as input for the reconfiguration layer, which is responsible for determining the system's configuration. Finally, the architecture layer handles the placement of applications, that is, assigning application instances to computing nodes, and it also includes mechanisms for monitoring the system's state.

The structure of this paper is as follows: Section 2 presents our overall approach for a dependable context-based system architecture intended for autonomous vehicles. Section 3 elaborates on the techniques utilized for extracting and representing context. Section 4 highlights the features and challenges associated with context-based reconfiguration. Section 5 provides an overview of the difficulties related to implementing new configurations and monitoring system alterations. The paper concludes in Section 6 with a review of related methodologies and directions for future research.

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.

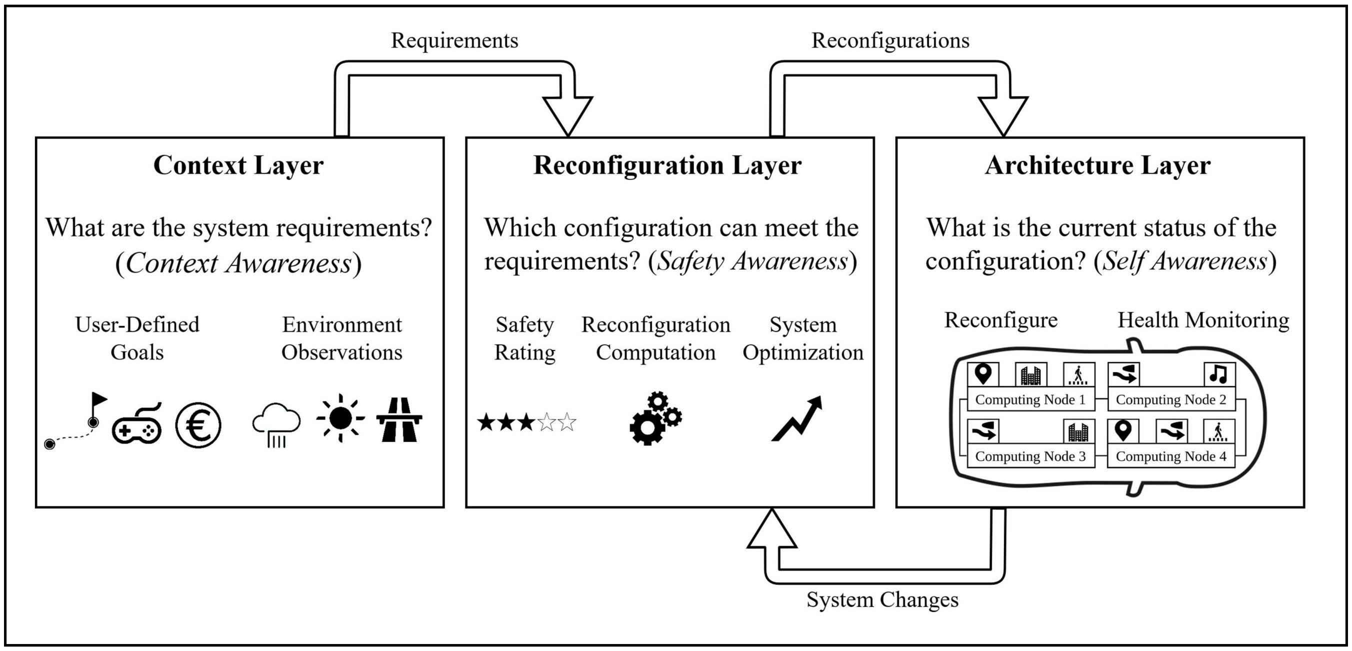


Figure 1. The three logical layers used in our approach and their relationships. The layers provide distinct levels of awareness

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.

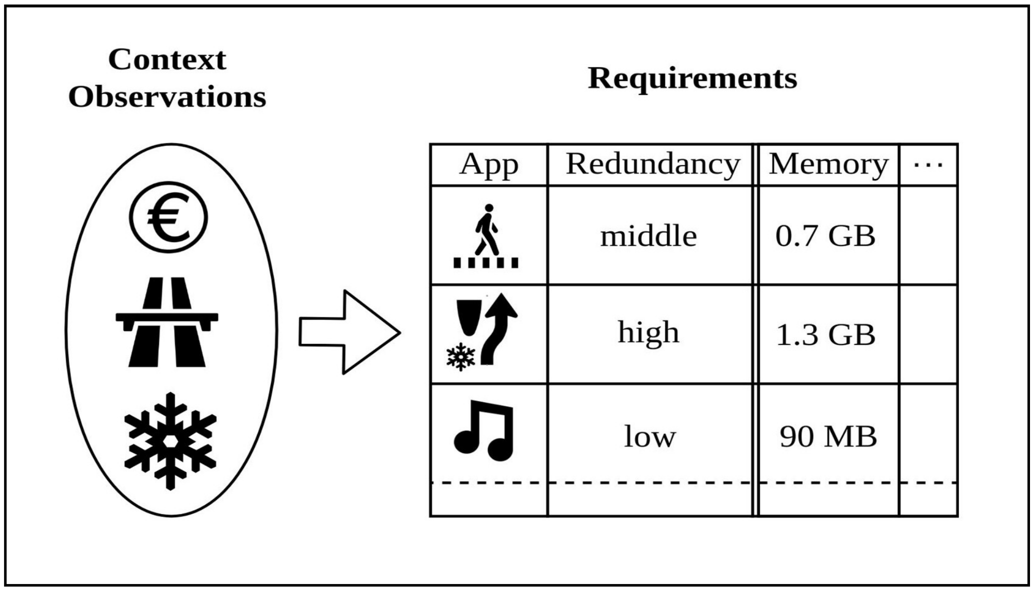
3. Context Extraction

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.

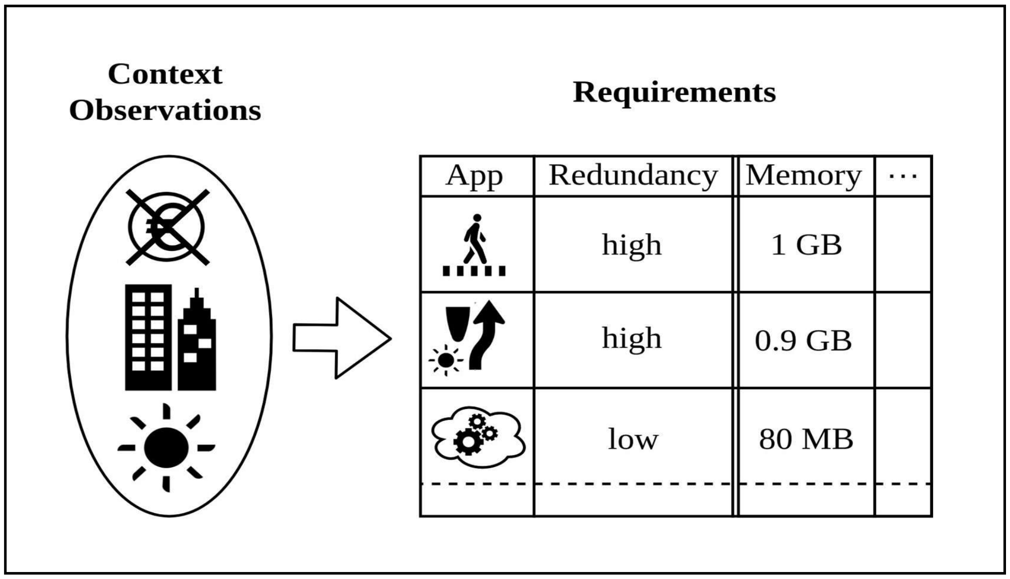
Figure 2 illustrates two use cases that show that distinct sets of context observations imply distinct sets of requirements.

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 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.



(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

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

[2] Brewka, Gerhard., Eiter, Thomas., Truszczynski, Miros³aw. (2011). Answer set programming at a glance. *Communications of the ACM, 54*(12), 92–103.

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

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

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

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

[7] Leone, Nicola., Pfeifer, Gerald., Faber, Wolfgang., Eiter, Thomas., Gottlob, Georg., Perri, Simona., Scarcello,

Francesco. (2006). The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic, 7*(3), 499–562.

[8] Li, Bo., Li, Jianxin., Huai, Jinpeng., Wo, Tianyu., Li, Qin., Zhong, Liang. (2009). 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)* (pp. 17–24).

[9] Meng, Xiaoqiao., Pappas, Vasileios., Zhang, Li. (2010). 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)* (pp. 1–9).

[10] Mori, Marco., Li, Fei., Dorn, Christoph., Inverardi, Paola., Dustdar, Schahram. (2011). 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)*, *Lecture Notes in Computer Science, 7041*, 286–301. Springer.

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

[12] Ren, Yi., Suzuki, Junichi., Vasilakos, Athanasios., Omura, Shingo., Oba, Katsuya. (2014). 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)* (pp. 1–8).

[13] SAE International. (2014). *Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems* (SAE Standard J3016).

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

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

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