V-model-based Refactoring of an Automated Valet Parking System

Mohamed Amine Mejri, Felix Meyer, Marius Westendorf,
Marcel Kascha, Silvia Thal, and Roman Henze*

Abstract: With Automated Valet Parking (AVP), a vehicle is guided to perform a parking use-case. AVP systems can be integrated either in the vehicle or in the infrastructure of the parking facility. Obtaining both vehicle-centered and infrastructure-based systems offers a redundant AVP solution. However, developing each system separately is time-consuming and prevents the knowledge transfer. Therefore, leveraging existing AVP systems in the development process is required. In this work, we introduce a systematic approach to develop an infrastructure-based AVP system by adapting an existing vehicle-centered system. We followed the V-model to define the development stages, which facilitates the traceability of the complete process.

Keywords: Automated Valet Parking, Development Process, System Engineering, V-model

1 Introduction

Automated Valet Parking is rapidly gaining importance in the automotive field. In the last decade, several collaborations between Original Equipment Manufacturers (OEMs) and Tier I suppliers have been announced towards the development of AVP systems[1]. The functional system architecture in AVP is derived from the general architecture in Autonomous Driving (AD), consisting mainly of vehicle's self-localization, environmental perception, motion planning and vehicle control [1]. The operational design domain in AVP shows specific characteristics, e.g. a maximum allowed driving speed of 10 km/h. To regularize these specifications and facilitate the development, the international standard ISO 23374 [2] provides a detailed definition of the AVP system framework, requirements and test scenarios. These regularizations served to accelerate the standardized development of AVP systems. Thus, AVP systems are the first certified systems in Germany fulfilling the SAE Level 4 requirements [3]. According to [2], AVP systems can be classified into two categories. The first is a vehicle-centered system, called AVP type 1. The system functions are integrated in an intelligent vehicle, which is equipped with the necessary sensors and computation units. AVP type 2 is an infrastructure-based system, where a central server guides a connected vehicle remotely by sending driving commands. Perception sensors are mounted in the parking area to obtain an environment model. AVP type 2 offers the advantage of controlling multiple vehicles simultaneously, without

*All authors are with the Institute of Automotive Engineering (IAE), TU Braunschweig, Hans-Sommer-Str. 4, 38106 Braunschweig, Germany (e-mail: mohamed.mejri@tu-braunschweig.de).

[1]https://www.bosch-presse.de/pressportal/de/en/bosch-and-apcoa-to-provide-automated-valet-parking-technology-in-parking-garages-across-germany-250496.html

increasing the costs, since the vehicles do not have to be equipped with neither additional environment sensors nor a computational intensive Electronic Control Unit (ECU).
The development of advanced systems, such as AVP, has pushed the automotive industry to intersect with the IT field. Despite the fruitful combination of the two sectors, there is a significant difference in their development processes. As highlighted in [4], developing a vehicle's system requires between three and five years, which is longer than the development process of typical IT systems, e.g., smartphones. These extended development cycles increase the criticality of issues that arise in the later development stages. As a result, a systematic development process is required. This was statistically analyzed in [5] through a comparison of 20 case studies of product successes and failures in the automotive market. The study indicated that the criticality of the development process surpasses other important key factors, such as safety and fuel economy. Since AVP systems can be implemented in two different types, an efficient AVP development process could involve leveraging one AVP type in the development process of the other. In this paper, we present a system engineering guideline for designing, developing, and testing an AVP type 2 system, referring to an existing Type 1 system as a baseline.

2 Methodology

Modularity and scalability are the major challenges targeted in system development. A modular system is capable of being divided into subsystems, which facilitates the maintenance and integration of the different system components. A scalable system shows a flexibility for further extension to adapt to increased complexity and changes in the use-cases. Both aspects must be addressed starting from the early stages of the development process, and not during the implementation stages [6]. For this purpose, one of the appropriate development approaches is the V-model .It facilitates traceability and refinement during the development stages [7]. The aim of this work is to transform an AVP type 1 system into a type 2 system, which necessitates the modularity and scalability of both systems. To achieve this, we revisit the V-model as defined in ISO 26262 [8] as a guideline for defining the system development process. In this section, we investigate each block of the V-model, as depicted in Figure 1, to define the development steps.

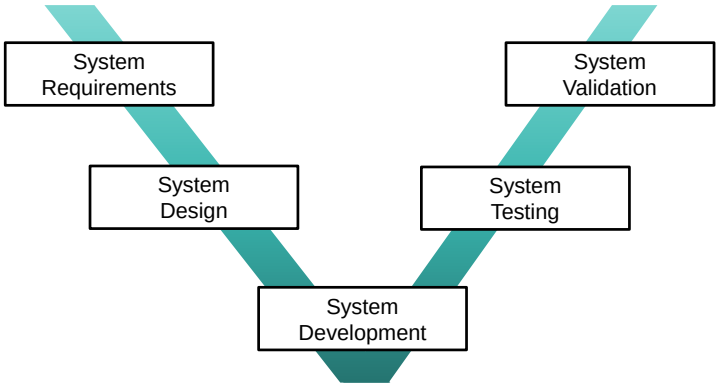


Figure 1: V-model of product development [8].

2.1 System Requirements

The first step involves defining the requirements for an AVP type 2 system. These are outlined in the ISO 23374 standard [2], which also includes driving test scenarios for safety assessment. In addition, the German Association of the Automotive Industry (VDA) has introduced a technical guideline for the development of AVP type 2 systems [9]. This guideline summarizes recommendations from OEMs and smart parking providers involved in the development of AVP systems. According to [2], an AVP type 2 system consists of four sub-systems, which are classified under management and operational interfaces, as shown in Figure 2. The first sub-system is the vehicle backend, an external service of the vehicle. The vehicle backend forms with the system backend the management interface, handling organizational tasks within the AVP mission, such as sending and approval of requests or the allocation of parking lots. The remaining two sub-systems consist of the vehicle on-board and the system operation. The vehicle on-board can be considered as a set of ECUs responsible for vehicle control, while the system operation is the central computation unit within the infrastructure, controlling the vehicle remotely. Both interfaces interact logically through messages. An AVP mission is the result of a certain sequence of these messages. To ensure a safe driving mission, these messages must conform to specific standards, defined as safety and functional requirements.

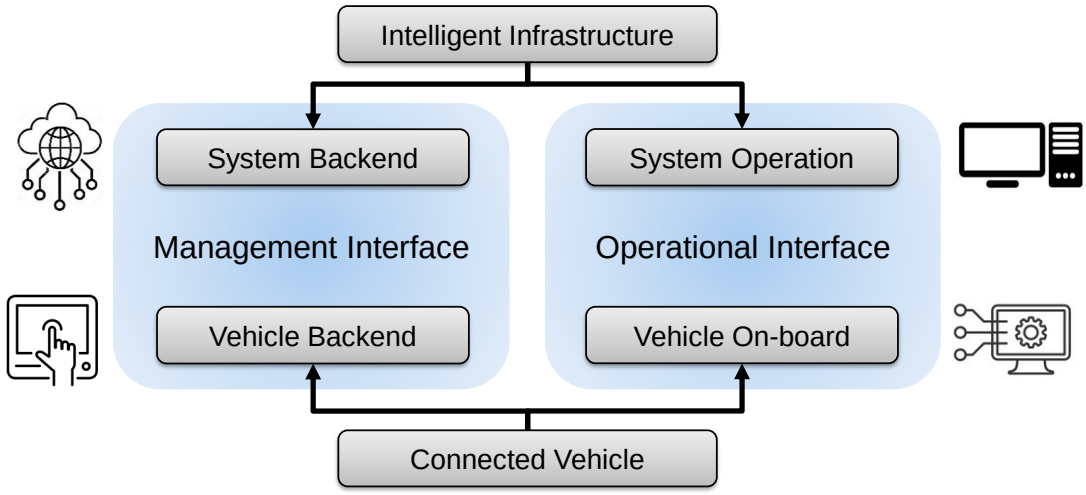


Figure 2: The configuration of an AVP type 2 system [9].

2.1.1 Safety requirements

The safety requirements are realized through safety messages within the operational interface. These messages serve to supervise and ensure the redundancy of the communication network. The safety messages run in a cycle including mainly a driving permission from the server and a vehicle feedback. An uninterrupted transmission of these messages during the driving mission is crucial, and any disruption would lead the mission to be aborted. The specified safety requirements are not included in AVP type 1 systems. This must be considered in the transfer between both systems.

2.1.2 Functional requirements

The functional requirements are designed to standardize the control of the vehicle by the server, specifying a set of messages that must be supported by all vehicles with an AVP type 2 system. These messages encompass both high-level driving maneuvers, such as braking, and low-level data, including the vehicle's state or the desired trajectory. The validity of these messages is established, with specified expiration time intervals to ensure safe driving. The functional requirements are fulfilled as well in the AVP type 1 system. This facilitates the scalability of the functions to the targeted AVP type 2 system.

2.2 System Design

As defined in [10], the system design is a preliminary definition of the functional system components, along with the derivation of the interactions and interfaces between these components. The keynote in system design lies in the specification of the term "system", which generally represents the vehicle in the case of AD [11]. AD systems, including AVP type 1, mostly share a standard system design consisting of data retrieval, environmental perception, driving mission planning, and vehicle control. In contrast, the type 2 system does not maintain the defined sequential design but adopts a distributed design instead. In this case, we distinguish between two systems: the vehicle and the infrastructure. To derive the design of our AVP type 2 system, we start first with revisiting the design of the baseline system.

2.2.1 AVP Baseline

We refer in this work to an existing AVP type 1 system developed by [12] as a baseline for designing our AVP type 2 system. As illustrated in Figure 3, the system is based on the standard AD design and implemented on the institute's own experimental vehicle TEASY 3, a Volkswagen Passat B8 that is equipped with LiDAR Sensors perceiving the surrounding environment of the vehicle [13]. The motion planning has a modular design [14], which facilitates the adaptation of the system to our use-case. The motion control module is based on model predictive control (MPC) [15]. The work was demonstrated in different real-world environments[2], which reflects the potential of scalability of the system to AVP type 2.

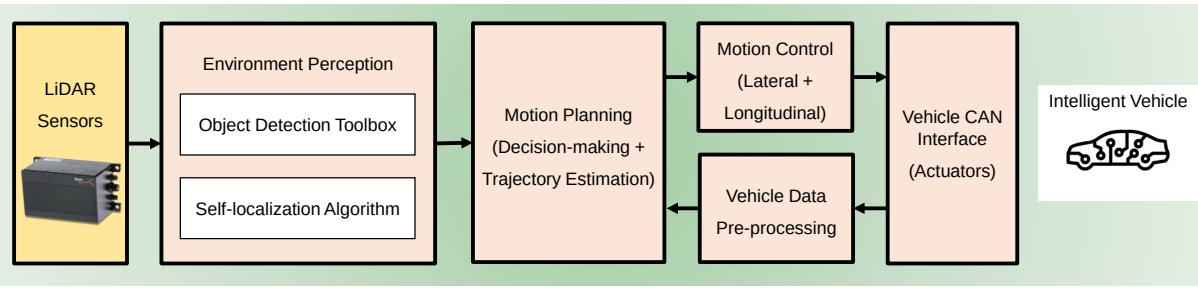


Figure 3: The design of AVP Type 1 system baseline[12].

[2]https://die-region.de/wirtschaft-forschung/innovation/autonomes-parken-im-forschungsparkhaus-braunschweig/

2.2.2 Model Refactoring

The AVP type 1 system should be transformed into type 2 without applying fundamental modification on the system components. This was obtained by applying model refactoring, a software engineering technique that involves redistributing subsystems across the system hierarchy, which enhances modularity [16]. By applying model refactoring on the design in Figure 3, we redistribute the functionalities between a connected vehicle and an intelligent infrastructure, consisting of a computation server, as shown in Figure 4. In case of AVP type 2, a vehicle's self-localization is not required. The vehicle position can be directly estimated by the object detection from the infrastructure sensors. Low-budget surveillance cameras replace here the LiDAR sensors, which decreases the total costs of the system. Both subsystems are extended with a communication interface. To ensure maintaining a consistent information flow between the modules as in the baseline model, we add a signal adapter module to each subsystem. The block manages sending, receiving, and forwarding the data signals in the appropriate format.

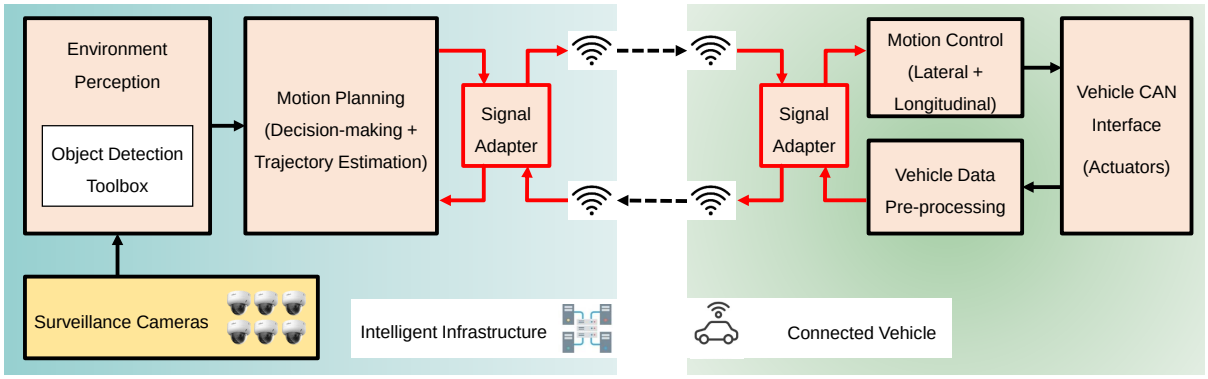


Figure 4: Design of an AVP type 2 system after applying a model refactoring.

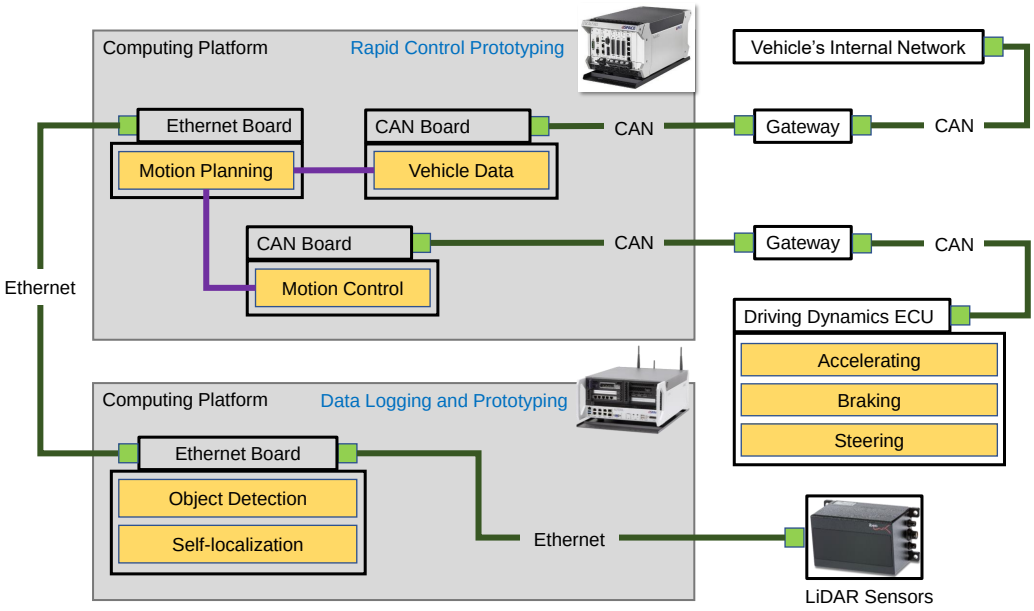
2.3 System Development

The following phase in the V-model involves extracting a system architecture based on the previously presented design. This process begins by identifying the essential hardware components. Then, a methodology for the software implementation of the system functionalities is defined. The resulting architecture offers a comprehensive system description, facilitating not only the implementation but also the verification and validation in the next stages [17].

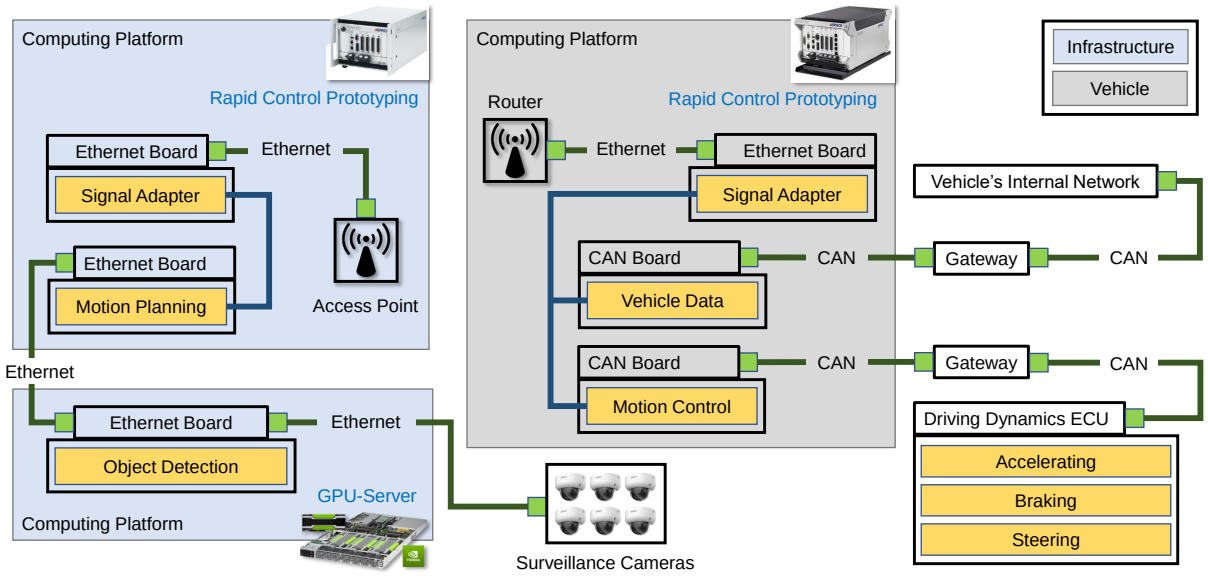
2.3.1 Hardware Architecture

To implement the proposed model refactoring, a comprehensive hardware architecture of the baseline system is required. As shown in Figure 5a, the model functions are implemented on two computing platforms. The first is the rapid control prototyping platform, dSPACE SCALEXIO AutoBox, equipped with Ethernet as well as Controller Area Network (CAN) boards. This enables direct communication with the vehicle's dynamics ECUs and internal network to retrieve vehicle data. The second computation unit, dSPACE AUTERA, is dedicated to the object detection and self-localization algorithms.

It is connected via Ethernet to the LiDAR sensors. For the AVP type 2 system, we retained the same vehicular hardware and refactored the model functionalities, as depicted in Figure 5b. The signal adapter was associated with the Ethernet board of the AutoBox, thereby facilitating the sending and receiving of messages by the router. On the server-side, a dSPACE SCALEXIO LabBox is used to run the motion planning functionalities. An access point was connected to its Ethernet board, establishing the communication with the vehicle's router. For the environmental perception, the surveillance cameras are powered and connected via Ethernet to a second computation platform, consisting of a GPU-based server used for running the object detection algorithms.



(a) Hardware architecture of the AVP type 1 system.



(b) Hardware architecture of the AVP type 2 system.

Figure 5: Refactoring of the hardware architecture between both AVP types.

2.3.2 Software Architecture

The baseline functions in the AVP type 1 system are implemented in an object-oriented manner, which facilitates redistribution and adaptation. The standardization of the messages between the vehicle and the server is obtained by the signal adapter class. Another class, consisting of a User Datagram Protocol (UDP) socket, is required to fulfill the designed AVP type 2 model. The Unified Modeling Language (UML) class diagram in Figure 6 illustrates the interactions between the different system modules. All attributes conform to the datatypes defined in the requirements of [2] and [9].
The vehicle's software is initially implemented in MATLAB and the architecture is modeled using Simulink [18]. Subsequently, the complete software package is compiled into C code and integrated into the dSPACE platform. The same process is replicated on the server for the modules associated with the dSPACE LabBox. For the object detection, we utilize a commercial software that runs on a separate GPU-server.

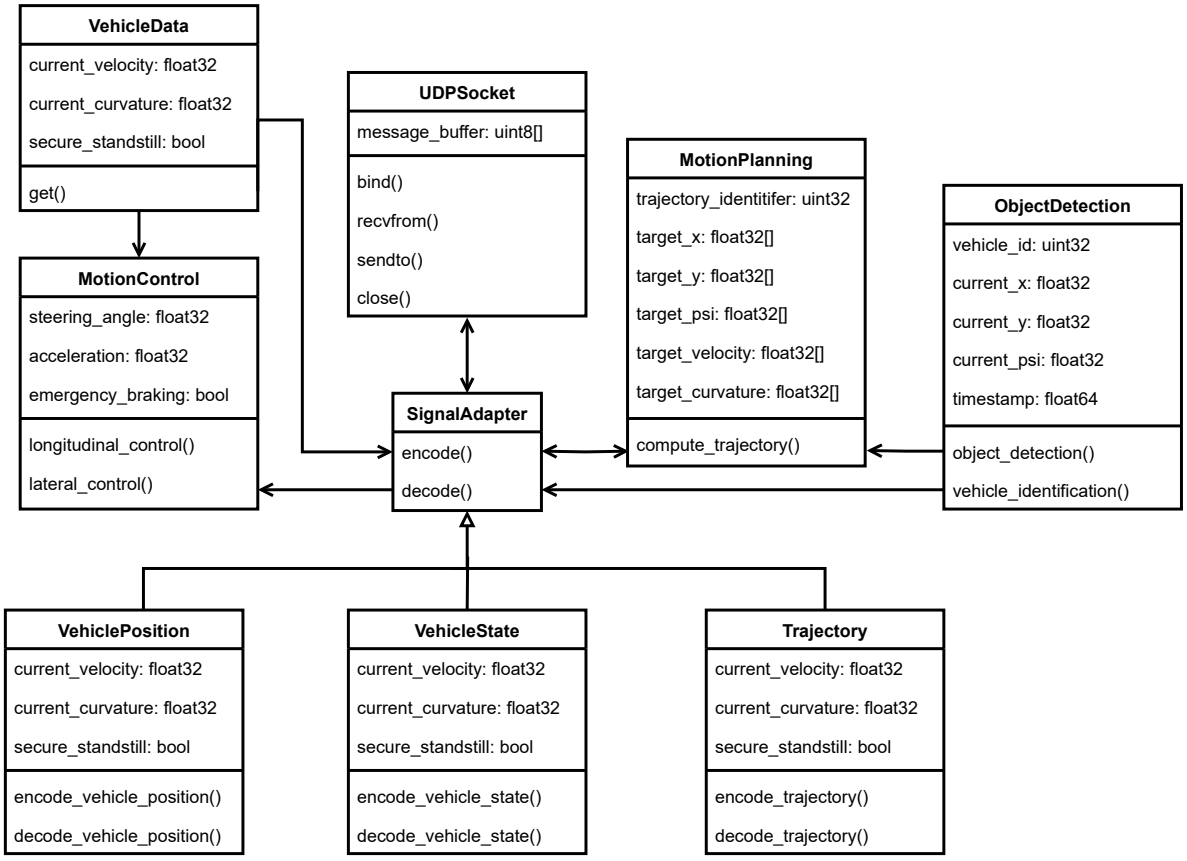


Figure 6: UML class diagram of the software modules.

2.4 System Testing

Once the system has been developed, we proceed to the testing phase. In this work, we followed the X-in-the-Loop testing approach introduced by [19]. The test assessments are defined in three steps:

- **Model-in-the-Loop (MiL):** The goal here is to run the system in a simulated AVP environment to ensure maintaining the functions refactored from the AVP type 1 system. For this purpose, we used the IPG CarMaker simulator, which has the advantage of an accurate and realistic simulation of the vehicle's dynamics characteristics. The driving environment is simulated referring to the ASAM OpenDRIVE[3] standard to ensure a reliable description of the driven roads. We integrate the vehicle control module in the simulation. The motion planning modules are integrated in a second computer, connected to the simulation computer via a local network. The perception data are retrieved directly from the simulation. Since MiL tests are not running under realistic communication conditions, the safety requirements are not considered during these tests. Finally, we define the test driving scenarios as described in [2].
- **Software-in-the-Loop (SiL):** The described AVP simulation setup is run in a closed loop. Hence, the software modules and the compatibility between the classes are tested. After successful compliance of the software, we can ensure maintaining the same logic of the prior AVP type 1 model.
- **Hardware-in-the-Loop (HiL):** The aim of these tests is to ensure the integration of the software modules in the target hardware. We used the dSPACE LabBox here as a test platform for both the vehicle and server. The LabBox also offers an interface with the CarMaker simulator for the closed-loop simulation, which allows running the simulation on a real-time system. The communication hardware is also integrated in the testing platform, which helps to evaluate also the redundancy of the network.

2.5 System Validation

The last step in the V-model consists of the validation of the resulting system. For this purpose, we implemented the AVP type 2 system on two different test vehicles[4]. The first vehicle is TEASY 3, the experimental vehicle used to demonstrate the AVP Type 1 system, the baseline of this work. The second test vehicle is the autonomous shuttle RAION, a testing platform used in the development of AD functions for urban areas [20]. Both vehicles are equipped with the required vehicle hardware. We extended the hardware setup with the communication interface. In addition, we defined a set of vehicle parameters for each vehicle, to ensure calculating an dynamically feasible trajectory for each vehicle by the server. A comparison of the trajectories generated by both AVP systems for the same parking scenario showed a consistency between the driven paths.

3 Conclusion

In this work, we introduce a systematic approach for transferring an Automated Valet Parking system from an automated vehicle to an intelligent infrastructure. Following the V-model, we adapted the original system by applying a model refactoring. The resulting system was tested and validated on different test vehicles, which reflects consistency and modularity. This shows the importance of system engineering in the acceleration of the development process without hindering the knowledge transfer.

[3]https://www.asam.net/standards/detail/opendrive/

[4]https://www.tu-braunschweig.de/iffzg/forschung/ausstattung/experimentalfahrzeuge

Acknowledgement

This work is funded by the German Federal Ministry of Education and Research (BMBF) as a part of the research project *Resiliente Kommunikationssysteme fuer sichere und flexible Produktionssysteme (RePro)*[5].

References

[1] R. Matthaei and M. Maurer, Autonomous driving - a top-down approach. *at-Automatisierungstechnik*. Vol. 63, no. 3, pp. 155–167, 2015.

[2] ISO/DIS 23374-1. Intelligent transport systems - Automated valet parking systems (AVPS), Part 1: System framework, requirements for automated driving, and communication interface. 2022.

[3] SAE International. Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. SAE Standard J3016_202104. 2021.

[4] H. Stoll, D. Grimm, M. Schindewolf, M. Brodatzki and E. Sax, Dynamic reconfiguration of automotive architectures using a novel plug-and-play approach. *2021 IEEE Intelligent Vehicles Symposium Workshops (IV Workshops)*. pp. 70-75, 2021.

[5] E. S. Hanawalt and W. B. Rouse, Car wars: Factors underlying the success or failure of new car programs. *Systems Engineering*. Vol. 13, pp. 389-404, Wiley Online Library, 2010.

[6] P. Obergfell, S. Kugele, C. Segler, A. Knoll and E. Sax, Continuous software engineering of innovative automotive functions: An industrial perspective. *2019 IEEE International Conference On Software Architecture Companion (ICSA-C)*. pp. 127-128, 2019.

[7] B. Liu, H. Zhang and S. Zhu, An incremental V-model process for automotive development. *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*. pp. 225-232, 2016.

[8] ISO/DIS 26262-2. Road vehicles - Functional safety - Part 4: Product development: system level. 2009.

[9] German Association of the Automotive Industry. Automated Valet Parking Systems - Requirements for automated valet parking systems. 2023.

[10] A. Kossiakoff, W. N. Sweet, S. J. Seymour and S. M. Biemer, *Systems engineering principles and practice*. Vol. 83. John Wiley & Sons, 2011.

[11] J. Bach, S. Otten and E. Sax, A taxonomy and systematic approach for automotive system architectures-from functional chains to functional networks. *International Conference On Vehicle Technology And Intelligent Transport Systems*. Vol. 2, pp. 90-101, 2017.

[5] https://www.forschung-it-sicherheit-kommunikationssysteme.de/projekte/repro

[12] M. Kascha, Entwicklung einer Level-4-Funktion fur das autonome Fahren *PhD Thesis, TU Braunschweig*. Shaker Verlag, 2025. ISBN: 978-3-8440-9916-4.

[13] M. Kascha and R. Henze, Requirement Analysis of Lidar Sensor Setups for Self-Localization in Automated Valet Parking. *2022 22nd International Conference On Control, Automation And Systems (ICCAS)*. pp. 1474-1480, 2022.

[14] M. Kascha and R. Henze, Modular Decision Making Framework for Level 4 Applications in Automated Driving. *IEEE 29th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*. pp. 1-6, 2023.

[15] M. Kascha, S. Gierke, M. Frede and R. Henze, Modular Survey and Real Implementation of Lateral Controllers for Automated Driving. *2023 IEEE 11th International Conference on Systems and Control (ICSC)*. pp. 796-803, 2023.

[16] T. Mens and T. Tourwé, A survey of software refactoring. *IEEE Transactions on software engineering*. Vol. 30, pp. 126-139, 2004.

[17] P. Obergfell, S. Kugele, and E. Sax, Model-based resource analysis and synthesis of service-oriented automotive software architectures. *2019 ACM/IEEE 22nd International Conference On Model Driven Engineering Languages And Systems (MODELS)*. pp. 128-138, 2019.

[18] MATLAB, version 9.6.0 (R2019a). *The MathWorks Inc.* Natick, Massachusetts, 2010.

[19] F. Reisgys, J. Plaum, A. Schwarzhaupt and E. Sax, Scenario-based x-in-the-loop test for development of driving automation. *14. Workshop Fahrerassistenzsysteme und automatisiertes Fahren*. Uni-DAS eV, 2022.

[20] L. Everding, I. Aslam, C. Raulf, O. Aviv Yarom, J. Fritz, S. Jacobitz, T. Hegerhorst, C. Pethe, T. Şahin, J. Iatropoulos, Jannes, T. Vietor, A. Rausch, X. Liu-Henke and R. Henze, Dynamically Configurable Autonomous Vehicles for Urban Cargo Transportation. *Towards the New Normal in Mobility: Technische und betriebswirtschaftliche Aspekte*. pp. 851-869, 2023.