Application of Model-Based Systems Engineering Methods in Virtual Homologation Procedures for Automated Driving Functions

Till Temmen[1], Max-Arno Meyer[1], Louis Wachtmeister[2], Mohammadsadegh Zabihi[3], Christopher Kugler[3], Sebastien Christiaens[3], Bernhard Rumpe[2], Jakob Andert[1]

[1] Teaching and Research Area Mechatronics in Mobile Propulsion (RWTH Aachen University), Forckenbeckstraße 4, 52074 Aachen, Germany
[2] Chair of Software Engineering (RWTH Aachen University), Ahornstraße 55, 52074 Aachen, Germany
[3] FEV.io GmbH, Neuenhofstraße 181, 52078 Aachen, Germany

Abstract. Further development of vehicles into software-defined products is increasingly influencing vehicle approval [1]. In particular, approval procedures are required for updates to homologation-relevant software, such as automated or autonomous driving functions. Appropriate solutions such as the "Digital Loop" showcase [2] focus primarily on virtual homologation methods. The verification in such procedures requires complete traceability from legal requirements to adjustments in the software architecture and functionality as well as to derived test cases and the tested digital twin. When applied to automated driving functions, scenario-based methods and a high degree of automation due to the scope of testing are also required.

This contribution uses the automated driving function Automated Lane Keeping System (ALKS) to show how software extensions can be formally specified using the scenario and model-based systems engineering method CUBE based on legal requirements. The scenario, requirements and architecture specification using Systems Modelling Language (SysML) can then be used for the implementation in the vehicle and in the Digital Twin as well as for homologation. In addition, the automatic derivation of machine-readable, homologation-relevant test scenarios and test cases from the specification model is shown. The test specification generated in this way can be used directly for simulation-based tests and vehicle tests. The use of the procedure in the homologation of software updates enables complete traceability with reduced specification and testing effort at the same time.

Keywords: Model-based Systems Engineering (MBSE), Scenario-based Systems Engineering (SBSE), Virtual Homologation, SysML, autonomous driving functions

A. Heintzel (Hrsg.), *Automatisiertes Fahren 2024*, Proceedings,
https://doi.org/10.1007/978-3-658-45196-7_1

[TMW+24] Temmen, Till and Meyer, Max-Arno and Wachtmeister, Louis and Zabihi, Mohammadsadegh and Kugler, Christopher and Christiaens, Sebastien and Rumpe, Bernhard and Andert, Jakob:
Application of Model-Based Systems Engineering Methods in Virtual Homologation Procedures for Automated Driving Functions.
In: Automatisiertes Fahren 2024, Heintzel, Alexander (Eds.), pp. 1-14,
DOI 10.1007/978-3-658-45196-7_1, Springer Fachmedien Wiesbaden, Frankfurt, 2024.

1 Introduction

Homologation procedures are time consuming and costly. This is due to the fact that for every new feature all tests must be reperformed to ensure that the complete system still fulfills every necessary regulation. To accelerate the process, the Digiloop project was established [2]. The idea is to virtualize and streamline the homologation process. This can be done by introducing cutting-edge engineering methods, like for example Model and Scenario based Systems Engineering [3, 4]. In this context, the CUBE method is used (cf.

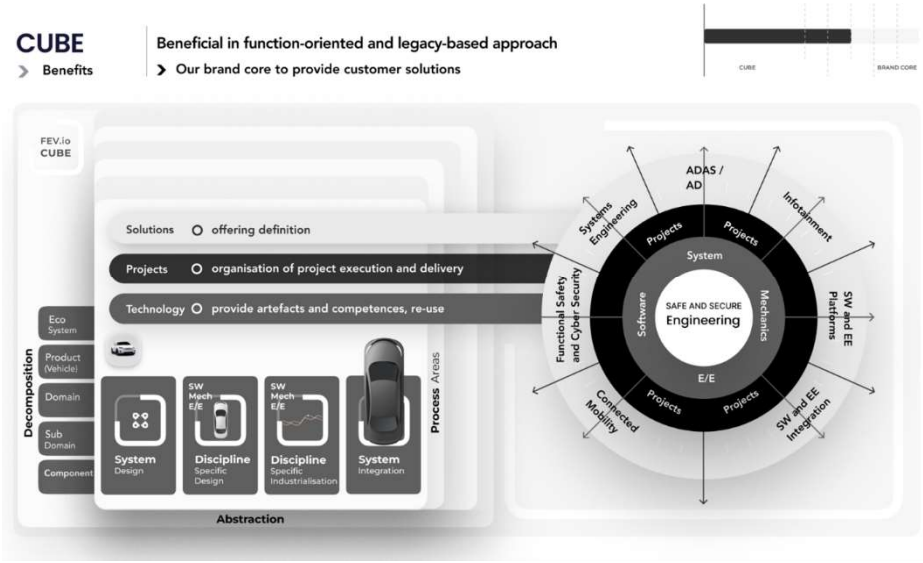


Fig. 1) that extends related methods such as SMArDt [5], or SPES [6, 7]. In the variant

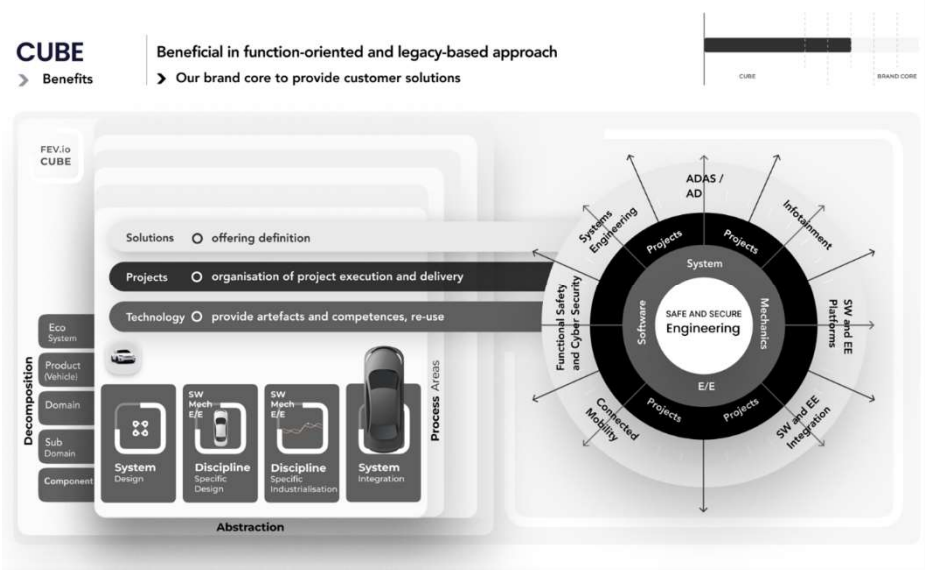


Fig. **1** presents, CUBE defines two dimensions of transforming the view onto the system – decomposition and abstraction. In the different levels of abstraction, the perspective shifts from a black-box to a white-box view. In the dimension of decomposition, the system is deconstructed into sub-systems, where different elements of the system can be allocated to features. By combining both dimensions of variation, it is possible to trace requirements from use cases to activities of a system, but also across different layers of decomposition [8, 9]. Using this approach, it is possible to identify homologation relevant changes to the system, perform analysis of relevant scenarios for retesting, and ultimately define new testcases automatically. For the definition of use cases, activities, and corresponding diagrams the Systems Modeling Language (SysML) is used [10, 11].

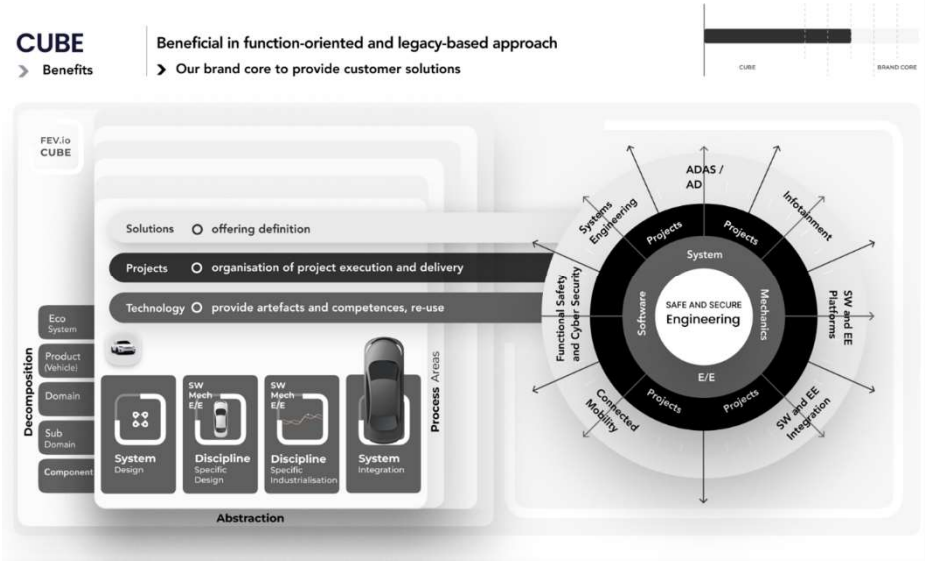


Fig. 1. Cube Methodology [9]

The procedure shall exemplary be further illustrated in the following chapters. In the example an Automated Lane Keeping System (ALKS), which has several potentials to increase the road safety [12], is the object of the homologation procedure. It is assumed that the system has already been approved for a maximum velocity of 60 km/h under UNECE R157 [13] and shall now be extended with functionalities to fulfill the regulation for a velocity of up to 130 km/h. These new requirements are an input to the first decomposition level and mark the starting point for the definition of additional use cases and scenarios. Based on the use cases, less abstract SysML activity diagrams are designed, which specify the behavior of the system based on the use cases. The behavior is split into features and the different actions are allocated to domains. This procedure is repeated for the next decomposition layer to give a more fine-granular view of the system. Various modeling tools can be used for this approach e.g., Enterprise Architect [14].

2 Use Case and Scenario Definition based on UNECE R157

The first step in the process is to formalize the regulations and add use cases for the new regulation [4], A new functionality is an automated Lane Change, which is demanded by the UNECE R157 [13]. This function can be addressed by the use case shown in

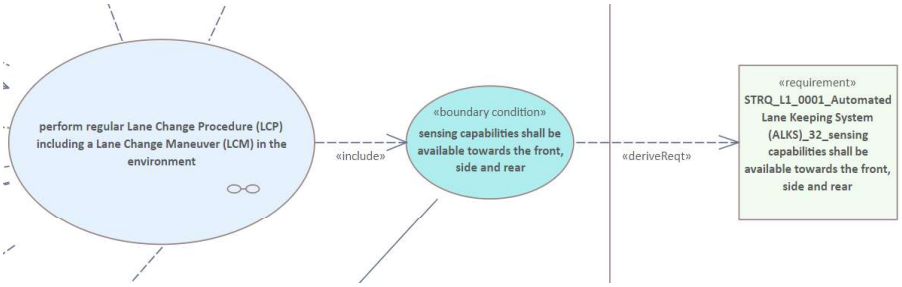


Fig. **2** - "perform regular Lange Change Procedure (LCP) including a Lane Change Maneuver (LCM)". To enable traceability to additional requirements that are not concerned with the system's functionalities and describe characteristics concerning the system performance, specific qualities, or constraints [15] the diagram in

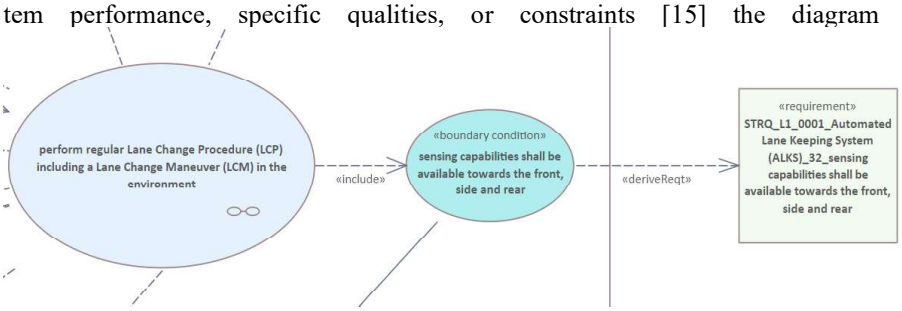


Fig. **2** marks these use cases with the stereotype boundary condition. This type of use case does not describe the behavior of the system, but rather processes or properties which must be fulfilled. For the derivation from the actual regulation, a connector with the stereotype "deriveReqt" is used. For the connection of the boundary condition use case to the lane change use case, an include relationship is used.

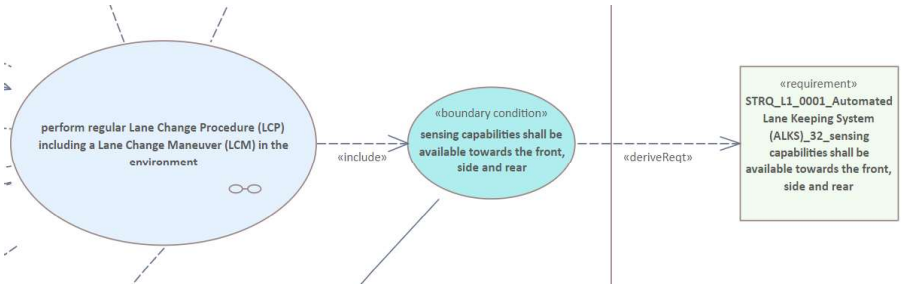


Fig. 2. Example for formalization of requirements and linking to use cases

To introduce tests that serve the purpose of validating that all use cases are addressed by the system, logical scenarios can be defined. These logical scenarios [16] describe the overarching state space, which is the ODD. The definition of scenarios is based on six layers of information, which are also used in the open file formats OpenDRIVE and OpenSCENARIO [17, 18, 19]. These formats define both the static and the dynamic environment.

After defining a logical scenario, it can be attached to certain use cases by include or extend relationships [4]. This is done by using composite diagrams (see

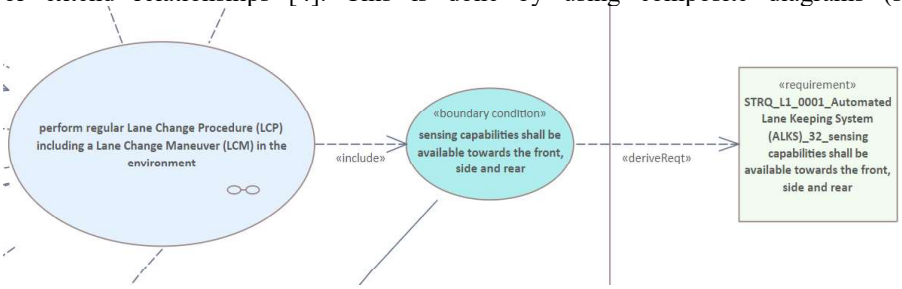


Fig. **2**). In the example the scenario "SC_0001_perform regular Lane Change Procedure (LCP) including a Lane Change Maneuver (LCM)" is included by a use case that extends the base use case of performing a Dynamic Driving Task (DDT). The new scenario only represents a part of the ODD "ODD_L1_Vehicle_0001_Automated Lane Keeping System (ALKS)". This is shown by the connector with the generalization stereotype.

The regulation demands that a lane change maneuver shall only be possible if certain conditions are fulfilled. This can be tested with a specific lane change scenario. This specialized scenario "SC_0006_R157_lane change test" is also shown in

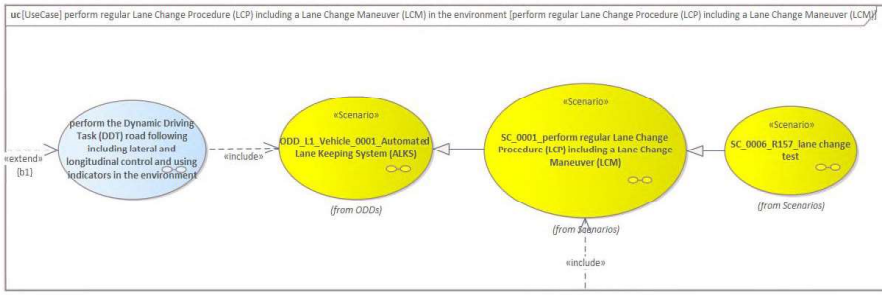


Fig. **3** with the scenario "SC_0001_perform regular Lane Change Procedure (LCP) including a Lane Change Maneuver (LCM)" being the generalizing scenario. An allocation process like this is needed to specify which scenario needs to be executed to validate the system under test.

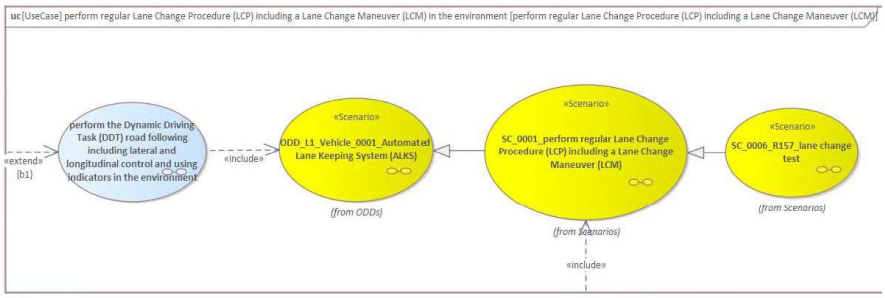


Fig. 3. Attachment of scenarios to use cases

Following the CUBE Method, SysML activity diagrams are created based on the use-cases (Stakeholder Value), which define the second level of abstraction – Operating Principle. This level describes how the different parts of the system interact. [8]

Based on these activities, the system can be further decomposed, where the different actions can serve as a starting point for defining use cases for the second layer of decomposition.

3 Delta Analysis of Operating Principle

After new functionalities are modeled, a delta analysis is performed. The purpose of the analysis is the identification of changes and measures that must be taken to ensure that new functionalities work as intended. With this analysis, the testing effort can be drastically reduced, since not all features are necessarily needed for the new functionality. In the case of the homologation of the ALKS for 130 km/h one example is the turning on of indicator lights. This base functionality of the vehicle doesn't necessarily need to be retested, if no new use cases change the functionality itself. Due to the formalized and structured way of building traceability and allocation with CUBE and the modelling in, for example, Enterprise Architect this analysis can be performed efficiently. In the given example, the Lane change functionality had to be added. With the connections modeled in the SysML use case diagrams, every chain of use cases can be analyzed across all decomposition layers. This makes it possible to identify the activities that are affected by the introduction of the new functionality, as shown in

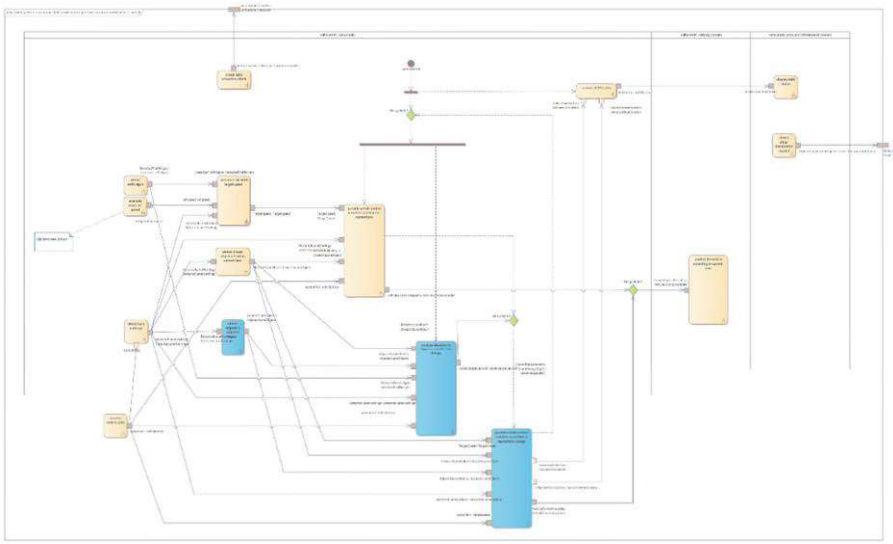


Fig. **4**, with all activities marked in blue being relevant for homologation.

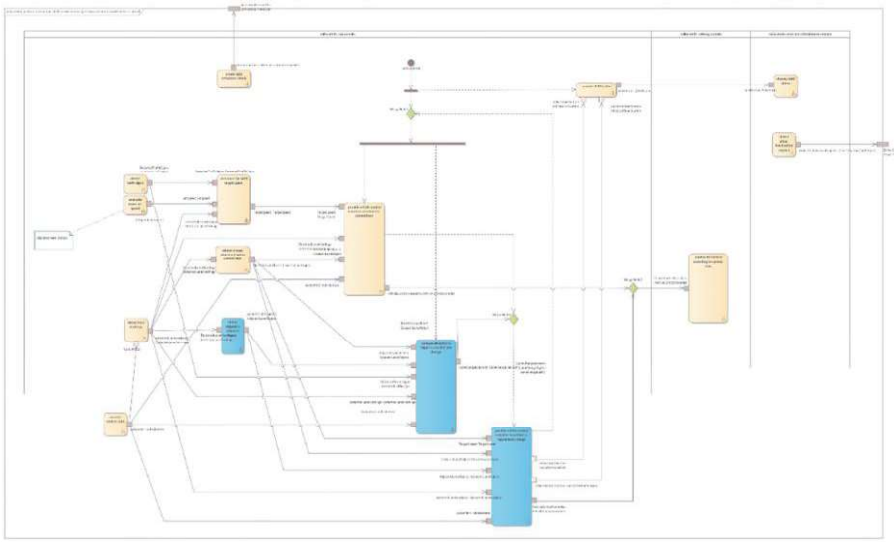


Fig. 4. Delta analysis in activity diagram

4 Testcase Generation

To automatically generate test cases, two artefacts are necessary. The first thing is the logical scenario, which represents the state space. The second artefact are SysML activity diagrams which describes the possible order of actions a system executes

during an activity or process [20, 21, 22, 23]. The process is shown in

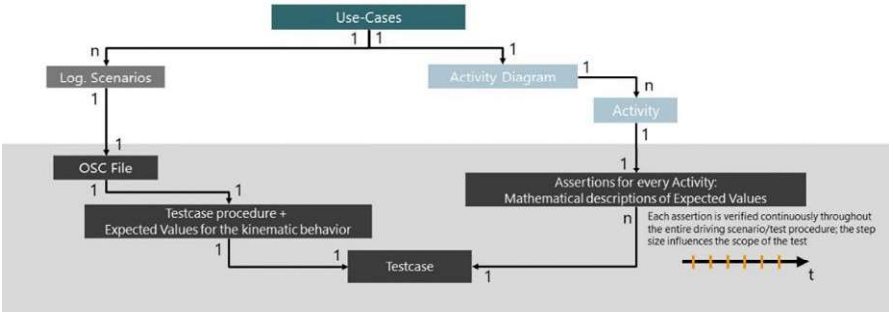


Fig. **5** [16].

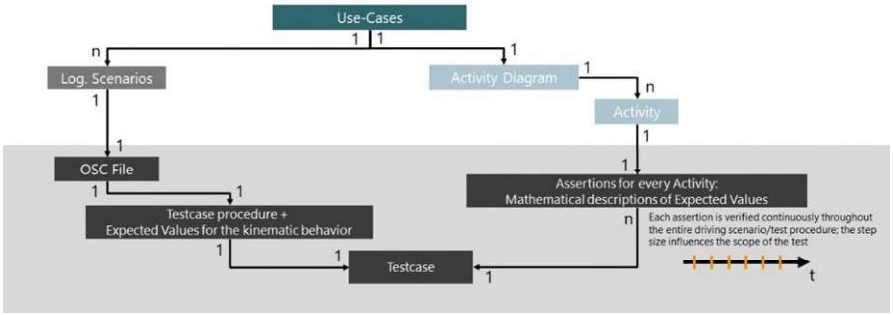


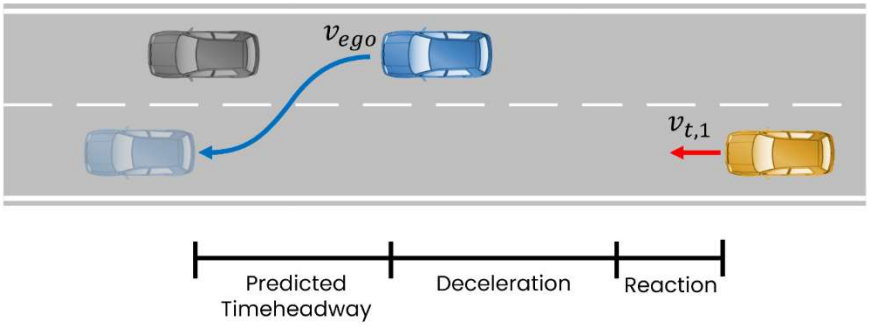
Fig. 5. Artifacts necessary for the generation of testcases [16]

In addition to the behavior of the system the Stakeholder Value holds another important part of information. Since scenarios define the temporal order of execution in a test procedure, they are closely linked to SysML activity diagrams and scenarios can be designed to trigger certain actions of the system. To validate that the system behaves as expected, it is necessary to define expected values – Key performance indicators (KPIs) - for the modeled actions. The calculation of the KPIs is directly attached to the actions [16]. Expected values can be derived from regulations or other sources.

One example for where expected values can be derived from are the functional chains within the system. These functional chains can be retrieved from the defined SysML activity diagrams by following every possible path of actions the system could take. It is important to investigate these chains to take for examples decisions into account, that need to be made to let the system perform a specific behavior. Based on these specific expected values can be defined. This holds especially true for the expected values of superordinate actions, which strongly depend on the flow of actions inside [16].

For the given example of the ALKS extension expected values are formulated in the regulation. One demand is that a lane change shall only be conducted if the maneuver would not force another vehicle to break in a way that is dangerous and could cause

accidents. To provide a way of validating this demand, a logical scenario with expected KPIs is given by the regulation. The setup of the scenario is shown in Fig. 6.



Fehler! Verweisquelle konnte nicht gefunden werden.

Fig. 6. Example of a scenario needed for homologation

The vehicle that is equipped with the ALKS function, the "ego vehicle", is approaching a slower vehicle in the same lane. Meanwhile, a faster vehicle is driving on the second lane. Multiple phases are assumed. The first phase is the reaction phase, in which the vehicle in the target lane maintains a constant velocity (here $v_{t,1}$) for 0.4 seconds. After this short delay the vehicle in the target lane decelerates with a maximum of 3 $\frac{m}{s^2}$. Meanwhile, the ego vehicle keeps its velocity v_{ego}.

It is expected by the regulation UNECE R157 [13] for an ALKS operation (at the speed of 130 $\frac{km}{h}$), that the distance between the ego vehicle and the vehicle in the target lane shall always be greater than a required time headway. The required time headway is defined as the distance that the ego vehicle would travel at a given velocity in 1 second.

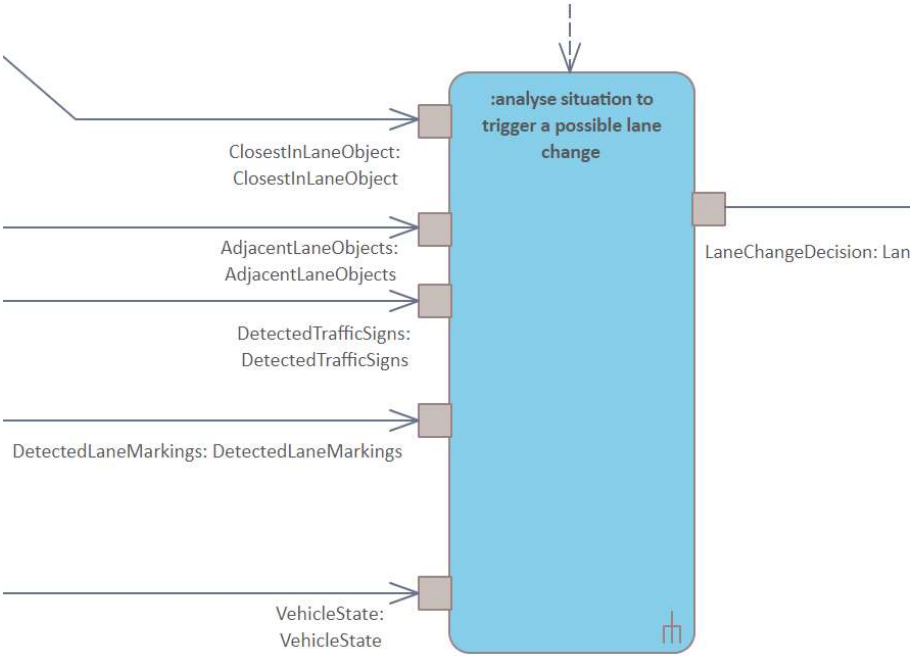


Fig. 7. Activity relevant for the given homologation scenario and attachment of expected values

In the example model, the necessary calculations are attached to the action, which analyses the environment to determine if a lane change is possible. This action was identified as relevant for homologation as part of the delta analysis (see

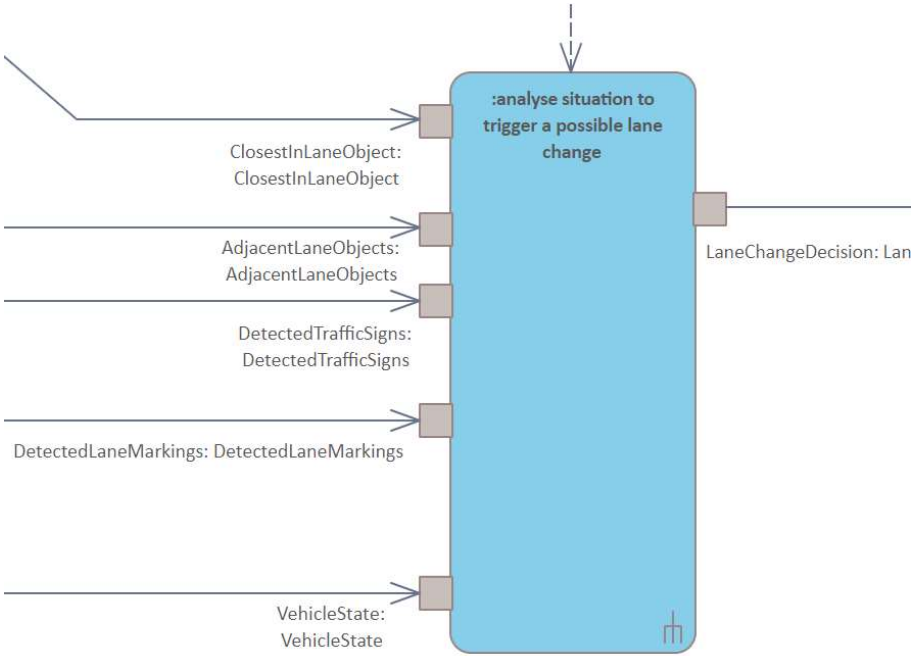


Fig. 7).

With the addition of the logical scenario, it is possible to generate testcases automatically. For this purpose, a specialized testcase format was developed, which includes all necessary data about the scenario (static and dynamic), test procedure and expected values as Pass/Fail criterions [24]. Based on this artefact, simulations can be performed to validate the system. The generated results together with all other produced artifacts can then be provided directly to the relevant partner responsible for the homologation process. In practice, this accelerates the workflow.

5 Summary

This contribution shows how it is possible to use cutting-edge methods to speed up the homologation processes. To this end, traceability is important. Because of this, the CUBE Method is used to provide traceability into the decomposition layer as well as into the abstraction levels of the system. Furthermore, to fulfill additional regulations, new use cases must be introduced. To allow traceability, the requirements are integrated into the model in a formalized way. Scenarios which define the ODD or parts of it are attached to the new use cases with composite diagrams. Based on the new use cases the second abstraction level Operating Principle can be extended. With these changes to the system, a delta analysis is performed. The result is a collection of all the changes to the system and which part of the system must be revalidated for the homologation process. This implies that the result also shows which part of the system must not be

revalidated, which reduces the test effort drastically. It can also be determined, if scenarios can be reused that were defined before the changes to the system.

Based on this analysis, test cases can be generated automatically. two artefacts are needed. A scenario and expected values (KPIs) of the system, which can be compared to the actual values while testing. To provide a way of calculating the expected values a new file format is introduced. These files can be connected to actions in the second layer of abstraction – Operating Principle.

References

1. Stokar, R. V.: Accelerated Type Approval by Validating Software Updates. ATZelectronics worldwide, 15(3), 42-45 (2020).
2. The Digital Loop, 2023, https://www.digi-loop.com/, last accessed on 2024/01/26.
3. Sippl, C., Bock, F., Lauer, C., Heinz, A., Neumayer, T., German, R.: Scenario-Based Systems Engineering: An Approach Towards Automated Driving Function Development. In: Proc. 2019 IEEE International Systems Conference (SysCon) (2019).
4. Meyer, M., Silberg, S., Granrath, C., Kugler, C., Wachtmeister, L., Rumpe, B., Christiaens, S., Andert, J.: Scenario- and Model-Based Systems Engineering Procedure for the SOTIF-Compliant Design of Automated Driving Functions (2022).
5. Drave, I., Hillemacher, S., Greifenberg, T., Kriebel, S., Kusmenko, E., Markthaler, M., Orth, P., Salman, K. S., Richenhagen, J., Rumpe, B., Schulze, C., Wenckstern, M., Wortmann, A.: SMArDT modeling for automotive software testing. In: Software: Practice and Experience 49.2, 301-328 (2019).
6. Pohl, K., Hönninger, H., Achatz, R., Broy, M.: Model-based engineering of embedded systems: The SPES 2020 methodology. Springer, Heidelberg (2012).
7. Pohl, K., Broy, M., Daembkes, H., Hönninger, H.: Advanced model-based engineering of embedded systems. Springer International Publishing (2016).
8. Granrath, C., Kugler, C., Silberg, S., Meyer, M., Orth, P., Richenhagen, J., Andert, J.: Feature-driven systems engineering procedure for standardized product-line development. Systems Engineering. 24. 10.1002/sys.21596 (2021).
9. Richenhagen, J., Granrath, C., Gehrt, J., Hake, J., Kugler, C., Mrohs, B., Ligtelijn, H., Heinen, K., Kriebel, S.: E/E industrialization with MBSE for Series Development of Software Defined Vehicles, 32nd Aachen Colloquium Sustainable Mobility 2023, Aachen (2023).
10. OMG Systems Modeling Language, Version 1.6 ed., Computer software manual (2019).
11. Kautz, O., Rumpe, B., Wachtmeister, L.: Semantic Differencing of Use Case Diagrams, Journal of Object Technology (2022).
12. Utriainen, R., Pöllänen, M., Liimatainen, H.: The safety potential of lane keeping assistance and possible actions to improve the potential. In: IEEE Transactions on Intelligent Vehicles 5.4, 556-564 (2020).
13. Addendum 156 – UN Regulation No. 157 Amendment 4 - Uniform provisions concerning the approval of vehicles with regard to Automated Lane Keeping Systems, United Nations, 2023.

14. Model Driven UML Tool, https://www.sparxsystems.de/, Sparx Systems Ltd und SparxSystems Software GmbH (2024).
15. Glinz, M.: On non-functional requirements. In: 15th IEEE international requirements engineering conference (RE 2007), IEEE (2007).
16. Kugler, C., Granrath, C., Pischinger, F., Malvankar, A., Däsch, C., Meyer, M., Andert, J.: Ausleitung von Testfällen für automatisierte Fahrfunktionen auf Basis semiformaler Systemspezifikation (2022).
17. OpenDRIVE: Concept Document, ASAM e.V., 2020: https://www.asam.net/index.php?eID=dumpFile&t=f&f=3907&token=fffa694711f0cd3cc37e61f38587b3a308e9a720, last accessed 2024/01/26.
18. OpenSCENARIO: User Guide, ASAM e.V., 2022, https://www.asam.net/index.php?eID=dumpFile&t=f&f=4092&token=d3b6a55e911b22179e3c0895fe2caae8f5492467, last accessed 2024/01/26.
19. Scholtes, M., Westhofen, L., Turner, L. R., Lotto, K., Schuldes, M., Weber, H., Wagener, N., Neurohr, C., Bollmann, M. H., Körtke, F., Hiller, J., Hoss, M., Bock, J., Eckstein, L.: 6-Layer Model for a Structured Description and Categorization of Urban Traffic and Environment," in IEEE Access, vol. 9, pp. 59131-59147 (2021).
20. Ali, J., Tanaka, J.: Implementing the dynamic behavior represented as multiple state diagrams and activity diagrams. In: Journal of Computer Science & Information Management (JCSIM) 2.1, 24-34 (2001).
21. Störrle, H.: Semantics and verification of data flow in UML 2.0 activities. In: Electronic Notes in Theoretical Computer Science 127.4, 35-52 (2005).
22. Eshuis, R.: Symbolic model checking of UML activity diagrams. In: ACM Transactions on Software Engineering and Methodology (TOSEM) 15.1, 1-38 (2006).
23. Kautz, O.: Model Analyses Based on Semantic Differencing and Automatic Model Repair. In: Aachener Informatik-Berichte, Software Engineering, Band 46 (2021).
24. Meyer, M., Christiaens, S., Amit, P.: Datenstruktur zum Testen autonom fahrender Kraftfahrzeuge, (2022).