Variant-aware Reconfiguration of Automotive Virtual Test Environments

Luca Seidel[1], Houssem Guissouma[1], Andreas Schmid[2], Eric Sax[1]

(1) Karlsruhe Institute of Technology, Kaiserstrasse 12 76131 Karlsruhe Germany, {luca.seidel, housemm.guissouma, eric.sax}@kit.edu
(2) Vector Informatik GmbH, Holderäckerstraße 36 70499 Stuttgart Germany, {andreas.schmid}@vector.com

Abstract - *Modern vehicles offer a multitude of features, adapting to individual preferences. Addressing the challenge of efficiently testing a wide range of automotive system variants within a virtual test environment is crucial in the face of the growing complexity of these vehicles. With numerous configuration options and frequent software updates, there's a pressing need for a standardized approach to variant-aware reconfiguration. In response to this challenge, we propose a delta-based reconfiguration of a virtual test environment with modular components. This reconfiguration process allows a dynamic and efficient adaption based on selected features of the Module under Test (MuT). Leveraging feature models, we define relationships between features and their impact on the MuT and test environment configuration. Our paper introduces a concept for variant-aware reconfiguration. This concept utilizes a 150 % model, representing a superset of a module containing all necessary assets to derive any variant. Additionally, we employ delta analysis to reduce reconfiguration efforts by identifying only the necessary adjustments based on changes in selected features.*

Keywords: *Virtual Test Environment, Driving Simulation, Variant Management, Reconfiguration, Automotive*

Introduction

Automotive systems can be configured in a wide range of variants to adapt to different customer needs or product classes (Becker, Weber, and Wierczoch, 2008). Looking at the German car market alone, one can see that the variety of models has already grown to well over 3000 in the last decade. This is without taking into account the numerous configuration options for the end customer's equipment (Braun, 2018). The recent large-scale introduction of Over the Air (OTA) updates continuously expands this variant space by adding new variants of software functions after production (Schindewolf, Guissouma, and Sax, 2021). Considering not only software variants but the entire buildable space of a modern vehicle, the number of possible variants increases up to 10^{100} (Zengler, 2023).
To integrate a new module version, such as enhanced Advanced Driver Assistance Systems (ADAS) functionality, into a vehicle variant space, compatibility testing is essential. To achieve seamless and efficient test coverage over the life-cycle of a vehicle, the virtual test environment must include or interact with variant management (Guissouma, et al., 2019). Modeling the vehicle variants in the test environment makes it possible to evaluate the effects of changes to individual components on the entire affected variant space. Reconfiguration of the test environment is crucial to handle the variant space effectively. While Original Equipment Manufacturers (OEMs) typically maintain databases of variants and versions of assets in their system, there is no standardized method for deriving and implementing the test environment for a specific variant. In addition to the environmental simulation consisting of sensors and actuators, the test environment must derive all necessary assets such as libraries, source code, configuration files, or functional components to validate the functionality of the system incorporating the Module under Test (MuT) (Isermann and Scha, 1999). Therefore, establishing a systematic approach to variant aware (VA) reconfiguration within the virtual test environment is crucial for reducing testing efforts and ensuring seamless integration.

Research question

How can a virtual test environment of an automotive system, including a simulation of the driving environment, be reconfigured efficiently for a wide range of system variants?

To answer this research question, the contribution of this paper can be summarized in the following three points:

1. Introducing an architectural model for a virtual test environment with variable elements.
2. Building on 1., we define a method for systematic test environment (re-)configuration of a MuT based on a set of selected features.
3. To evaluate the method from 2., we present results from a proof-of-concept implementation for a realistic Software in the Loop (SiL) environment from the automotive domain.

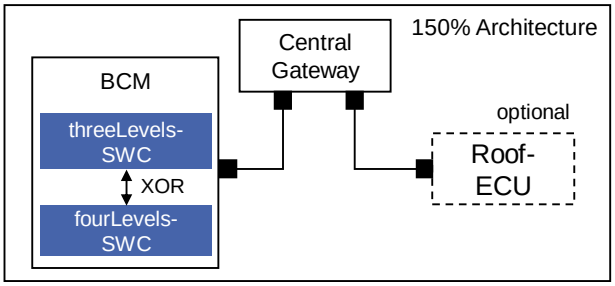

Figure 1: 150% architecture model of a variable windshield wiper system

State of the Art

Software-in-the-Loop in Automotive

The software used in Eletronic Control Units (ECUs) can be split into different abstraction layers. While the lower layers, which are close to the hardware, follow standardized specifications, the behavioral logic is implemented in specifically developed modules at the application level (*AUTOSAR Layered Software Architecture* 2022). These modules form the core of a vehicle's software, which is why a function and integration test at the application level is recommended. During development, such software modules undergo an iterative development process, which leads to a high degree of variability. The connection of these modules to be tested to the target environment and the target hardware therefore involves significant effort. This effort can be avoided by simulating the runtime environment. SiL enables scalable testability throughout the development process. The simulation of the runtime environment emulates the necessary interfaces and their interactions with the MuT. Communication-abstracting software architectures such as AUTOSAR or Robot Operation System 2 (ROS2) (Macenski, et al., 2022) contribute to reducing the effort required for software interfaces to a manageable level (Sax, 2008). Virtual test environments can be used to run SiL tests. The degree of virtualization varies. Common virtualization tools can virtualize sensors and actuators as well as the environment and execute pre- and post-processing steps (Kang, Yin, and Berger, 2019). As can be seen in section ,,Architecture of the Virtual Test Environment", we understand a virtual test environment to be the virtualization of the vehicle environment and its interface.

Feature Modeling in AUTOSAR

Nowadays, it is common for Electric/Electronic (E/E) engineering to be based on a product line approach, where a product line consists of a set of reusable artifacts and rules on how we can use a subset of these artifacts to construct a product. In turn, the product under development in the E/E engineering process is not just a single vehicle variant, but a whole vehicle family. That is why, as of version four, the AUTOSAR standard supports variant handling in shape of a feature model in the problem space and concepts to define variants in the solution space. Decoupling the solution space and problem space allows the vehicle manufacturer to develop the feature model and the AUTOSAR model with one toolset while their suppliers are free to enhance the given models with their preferred toolset.
In the solution space many elements of the Software, Communication and Network layers of the AUTOSAR model can contain a variation point (*AUTOSAR Generic Structure Template* 2023). Variation points contain the conditions under which this element is part of a certain variant. If an element does not contain any variation point, then it is part of every possible variant. The constraints used in variation points are based on system constants. Thus, we define a specific variant by the values, which we assign to the system constants. To derive a specific variant, we must assign all system constants a value, and evaluate all variation points. These system constants often find their way into the implementation in the form of precompile statements. Furthermore, AUTOSAR supports binding of values to system constants at various times (System Design Time, Code Generation Time, Pre-Compile Time, Link Time). In this way, we can exchange variants, variation points and system constants via AUTOSAR-ECU-Extract files, which do not only describe a single variant, but a set of variants which the supplier can enhance and implement.
In the problem space, AUTOSAR supports the optional use of a feature model (*AUTOSAR Feature Model Exchange Format* 2023) that is based on the FODA (Feature Oriented Domain Analysis) approach (Kang, et al., 1990). This feature model allows us to describe relationships of several types between features. For features, that share the same parent feature, these are: mandatory (all must be used), alternative (requires XOR), multiple (requires OR) and optional. For modelling relationships between any two features of a given feature model, there exist several pre-defined types (for example requires and excludes), as well as the option to define additional proprietary relationship types. Finally, the AUTOSAR standard uses a feature map to connect a valid feature selection in the problem space to a variant in the solution space. Input for a feature map is a valid feature selection and output is a set of system constant and value pairs which can be used to derive a variant as described before.

150 % Architecture Model

In combination with feature models (s. section "Feature Modeling in AUTOSAR"), 150 % architecture models are used to efficiently develop and maintain the variants in the solution space of the product line engineering process. Such a model is nothing else than a redundancy-free superset of all common and variable architectural elements in the product line (Hummell and Hause, 2015). Its main purpose is to derive specific system variants based on mapping relations to the used variability model, e.g. feature model.
By selecting a valid feature combination, the corresponding 100 % architecture can be derived as a subset of the 150 % architecture model. Additionally, it is possible to generate families of similar products, also called 120% models, based on partial feature selections (Vector Informatik GmbH, 2024). One common way to configure the components constituting the 150% model in the software domain is using pre-processor annotations in the variable code base (Apel, et al., 2013).

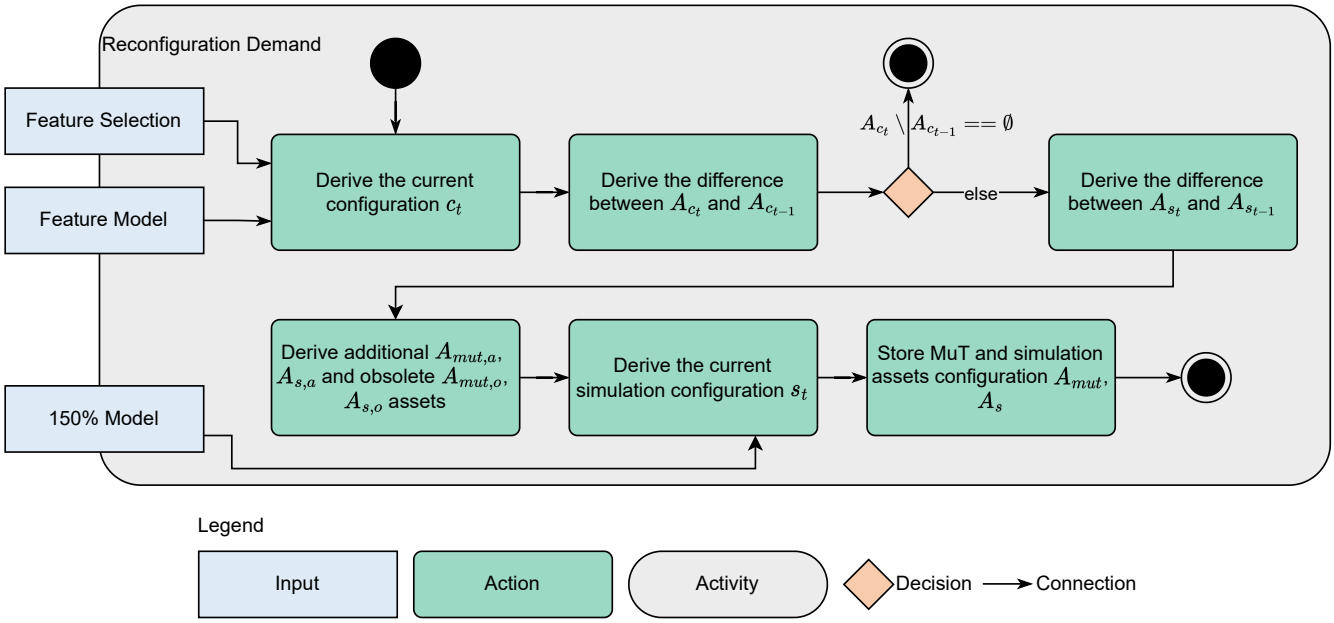

Figure 2: Determine the reconfiguration demand based on a feature selection

Figure 1 shows an example of a simple 150% architecture of a windshield wiper system. It consists of two ECUs: a mandatory Body Control Module (`BCM`), which exists in all valid variants, and an optional `Roof-ECU`, which shall be included only in convertible car variants. The `BCM` can be configured in two different variants by choosing one of the alternative software components `threeLevels-SWC` or `fourLevels-SWC`.

Methodology

Taxonomy

For the understanding of the concept of this paper, the following definitions are chosen:

- **Feature:** a characteristic or end-user-visible behavior of a system (Apel, et al., 2013).
- **Component:** “functionally or logically distinct part of a system. A component can be hardware or software and can be subdivided into other components” (ISO, 2017).
- **Module:** a set of system elements, with a physical or notional boundary and is detachable from the system. It can be a subsystem or component, as long as it can be detached and developed separately from the rest of the system (Efatmaneshnik, Shoval, and Qiao, 2020). A module may have a specific set of features, which may differ in different variants of the module.
- **Asset:** an “item, such as design, specifications, source code, documentation, test suites, or manual procedures, that has been designed for use in multiple contexts“ (ISO, 2017).

Architecture of the Virtual Test Environment

The proposed concept leverages a modular architecture as shown in Figure 3, which is designed to dynamically adapt the virtual test environment based on

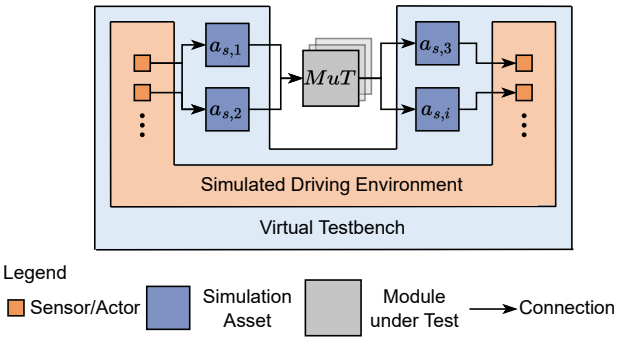

Figure 3: Modular architecture of a virtual testbench design

the selected features and their dependencies on the current MuT. The environment and vehicle simulation embedded in the *Simulated Driving Environment* is responsible for the input and output of sensors and actuators. It thus not only represents the connection to the sensors and actuators but also to the entire environment in which the vehicle or the MuT operates. This part can be generated by a driving simulator as described in (Kang, Yin, and Berger, 2019) and is executed on a dedicated server. A service-orientated architecture ensures seamless communication with the virtual testbench.
The *Virtual Testbench* emulates all necessary system components such as perceptions components, pre- and postprocessing components, controllers, configurations, etc. for the operation of the MuT. These system components, which are necessary but do not belong to the MuT, are subsequently referred to as assets. Such assets are executed on a dedicated computing unit, which may or may not be the same as the MuT computing unit. The separation of the test environment into the driving environment and MuT-dependent system components makes it possible to reconfigure and modularise the test environ-

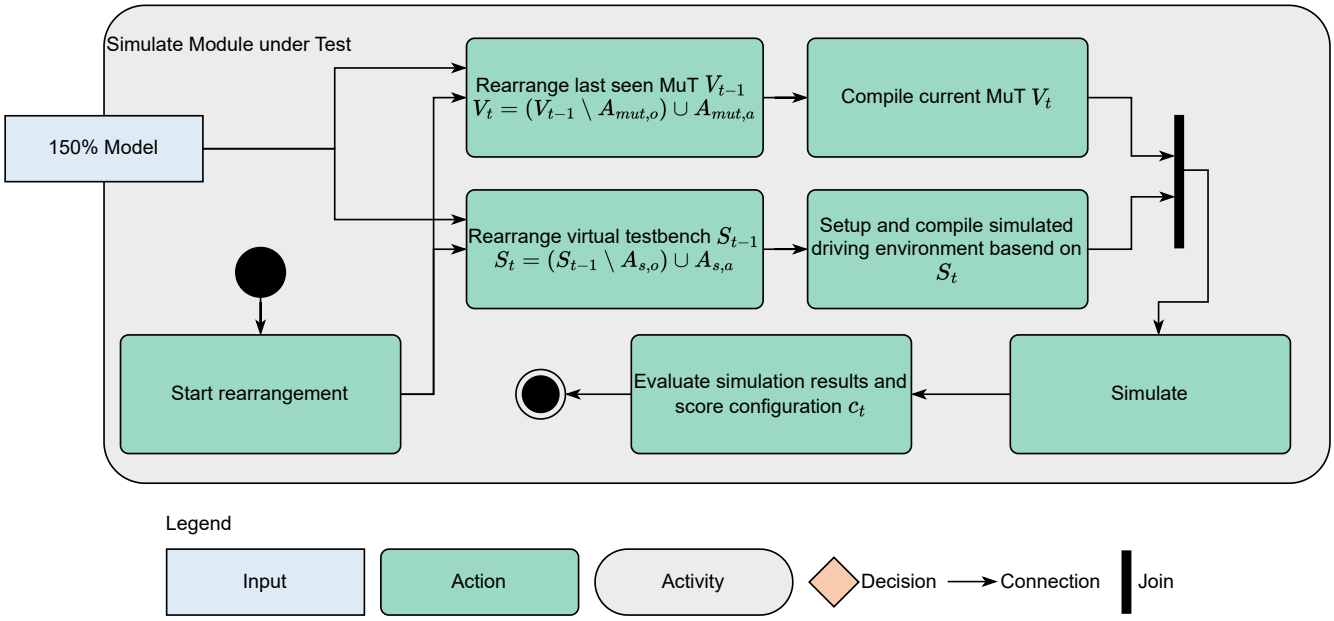

Figure 4: Rearrangement, Rebuild and Simulation virtual testbench

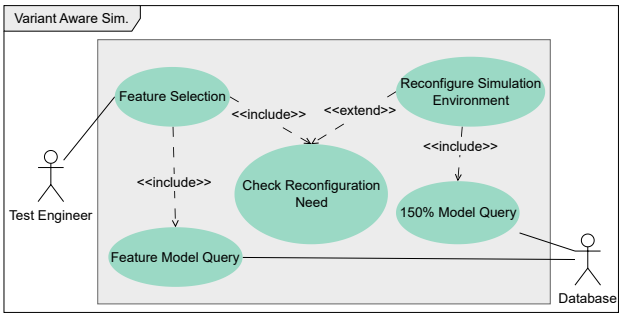

Figure 5: Interaction with the virtual testbench

ment independently of the vehicle and environment simulation software used.

Concept for variant-aware (Re-)configuration

The reconfiguration process outlined in Figure 2 and Figure 4 can be divided into two main phases and four phases in total:

1. Reconfiguration
   a. Reconfiguration Demand (RD)
   b. Rearrangment (RA)
   c. Rebuild (RB)
2. Simulation (SIM)

It fits seamlessly into the typical interaction scenario shown in Figure 5. A test engineer initiates the interaction by selecting certain features based on a product line description (see section “State of the Art“), e.g. testing new variants. A concrete, buildable variant is derived from this selection with the help of the feature model. Before the process is continued, it is checked whether the simulation environment needs to be reconfigured to fulfill the current MuT, so that unnecessary configurations are avoided. The entire simulation process takes place in four different phases. Firstly, the RD is determined, as shown in fig 2. This first phase sets the stage for the subsequent adjustments, ensuring alignment with the selected features and the current test bench configuration c_t, which reflects the current iteration. This configuration provides a set of necessary assets A_{c_t}, which includes both the MuT assets A_{mut} and the dependent simulation assets A_s of the testbench. By comparing the current set of assets with those from the previous test run, additional MuT assets A_{mut} and simulation assets A_s can be identified. The simulation environment configuration s_t can be derived and created using the asset properties available in the 150 % model. The first phase is completed by storing the assets-, MuT-, and simulation metainformation. The next three phases can be described as RA, RB and SIM and are shown in Figure 4. The RA adapts the current workspace to the number of assets determined according to eq. 1 for the MuT

$$V_t = (V_{t-1} / A_{mut,\ o}) \ \cup \ A_{mut,\ a} \tag{1}$$

where $A_{mut,o}$ denotes the obsolete assets and $A_{mut,a}$ denotes the additional assets. Similarly, the testbench is rearranged as eq. 2:

$$S_t = (S_{t-1} / A_{s,\ o}) \ \cup \ A_{s,\ a} \tag{2}$$

Depending on the metainformation stored in phase RD, this means that the corresponding assets are loaded from a database, cloud storage, local storage, etc. into the simulation workspace of the corresponding computing unit. Or removed from it. Once all the necessary assets are available, you can move on to the next phase, the RB. The newly added assets are compiled and integrated into the simulation. Once this process is complete, the actual simulation starts, referred to below as phase SIM. Once the simulation is complete, the simulation results are available to the test engineer, and the configuration c_t is saved for the next run.

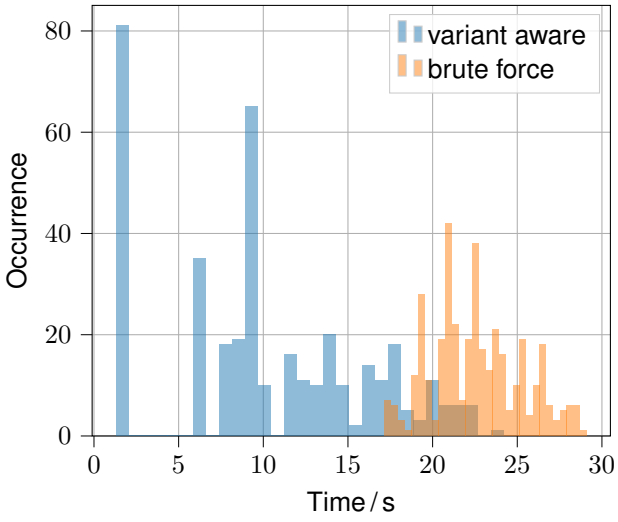

Figure 6: Histogram of the duration of the rearrangement and rebuild phases

Results and discussion

In our case study, we analyze the effectiveness of the VA reconfiguration approach using a proof of concept implementation. The feature model shown in Figure 7 serves as the basis for our study. The MuT is a monitoring component that observes vehicle dynamic states based on vehicle speed and engine rpm data. Ignoring horizontal dependencies, the feature model according to Apel, et al., 2013 shows 378 valid variants of how the MuT and its components can be configured. The features that belong to the branch of *sensors* and *actuators* fall into the area of the simulated driving environment and are simulated with the use of the CARLA simulation (Dosovitskiy, et al., 2017). This variant space can be configured directly using the simulation configuration s_t. For this purpose, a configuration file is generated in the RB phase, which is read in by our simulated driving environment adapter for the CARLA simulator and processed at the start of the SIM phase. The configuration file consists of all necessary sensors, and actors configured according to the current selection. Besides it provides a scenario definition which is later carried out in the simulation phase to test the MuT. The CARLA adapter provides a ROS2 communication interface and instantiates for each sensor and actor publisher and subscriber. The leaves of the *preprocessing* branch are parts of the virtual testbench and are necessary to operate the MuT. Those assets are deployed on a central processing unit that shares the same local network as the simulated driving environment. The components *pp1* and *pp2* contained therein form the variants *pp1.1* to *pp2.3* using an annotation-based approach (Apel, et al., 2013). Certain variants of these assets can be generated by deriving a header file and associated attributes for preprocessor directives.
The assets included in the virtual testbench and the source code of the MuT are located in an online repository, from which assets are loaded as required in the RA phase. Cloning and version checkout are performed by the help of git. To ensure dynamic communication between the simulated driving environment, the simulation assets, and the MuT, we utilize a service-oriented approach using ROS2. This abstraction allows us to offer services with which the MuT can interact, regardless of the exact variants. To demonstrate the advantage of a VA approach, all 378 variants were reconfigured and simulated in a random order. In addition to our proposed approach, the reconfiguration was carried out using a brute force (BF) methodology. Therefore the virtual testbench is prepared and simulated for each variant regardless of the past. The average times required for the individual phases with standard deviation can be seen in tab. 1.

Table 1: Time consumed by the different configuration phases

Method	RD / ms	RA / s	RB / s	SIM / s
BF	0.30	1.97 ± 0.28	20.62 ± 2.65	12.45 ± 8.95
VA	0.30	0.01 ± 0.18	9.96 ± 6.05	12.42 ± 1.42

The RB and SIM phases, in particular, are responsible for the overall duration of the process. The configuration we use does not change the actual scenario of the simulation, only the inputs and outputs and vehicle-specific properties such as the model of the car or the sensor type are changed, so it is to be expected that the simulation time between the VA and BF approach is quite similar. Only statistical deviations in the server communication or execution cycles of the simulation environment lead to deviating execution times of the SIM phase. It is clear to see, though, that the VA approach requires considerably lower RA and RB times. This is emphasized in Figure 6. Since a large part of the variant space is mapped in the reconfiguration of the simulated driving environment, many simulation runs can be carried out without significant recompilation. This leads to the accumulation of low rearrangement and rebuild times in the left-hand side of the plot. Only in the case of a few unfavorable sequences of variants does the VA approach require comparably long rearrangement and rebuild times as the BF approach. As Figure 6 shows, there are a few configurations that have long RA and RB times despite the VA approach. This can be explained by an unsuitable sequence for selecting the variants. This indicates that the approach can be enhanced even further. The current process provides metainformation about the variant to be created first. This property can be utilized, and if determined, can be collected across the variant space. A suitable estimation of the RA and RB time for variants and an appropriate cost function could be used to determine a reconfiguration and simulation sequence to cover the variant space cost-effectively.

Conclusions

In this paper, an approach for variant aware reconfiguration of SiL environments in the automotive context was presented. The continuously growing number of vehicle variants, driven by a multitude of configurable features and software updates, requires a systematic approach to efficiently test the large variant space along the lifecycle. We introduces a variant aware modular test environment that can reconfigure itself according to the demands of the current Module under Test using a 150 % model that serves as a superset of all assets occurring in the variant space. These demands can be derived based on the presented feature model. With the help of the proposed

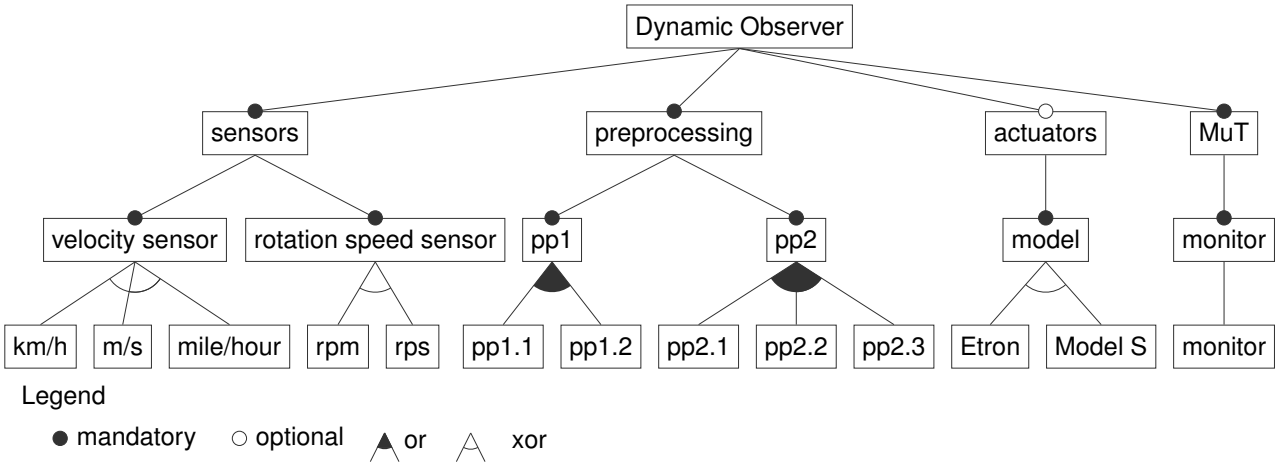

Figure 7: Feature model of the system used in the case study.

delta analysis of different feature selections, the reconfiguration effort can be significantly reduced compared to a brute force approach, as the case study with 378 variants has shown. In particular, it can be shown that most of the configuration process can be ascribed to the rearrangement of the necessary and unnecessary assets and the compilation process, for which the proposed delta approach provides a suitable remedy.
Nevertheless, it can be seen that an unsuitable sequence of feature selections can lead to considerable reconfiguration effort. This fact suggests that the sequence of testing many feature selections should be optimized by estimating the reconfiguration effort beforehand. Besides investigating the optimization of the variant selection order, we plan, in future work, to integrate the introduced approach into a CI/CD pipeline for agile analysis and validation of automotive software updates.

Acknowledgments

This publication is based on the research project SofDCar (19S21002), which is funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK).

References

Apel, S., Batory, D., Kästner, C., and Saake, G., 2013. Software Product Lines. In: *Feature-Oriented Software Product Lines*. Springer, pp. 3–15, 52–53, 251.

AUTOSAR Feature Model Exchange Format Nov. 2023. 606. Part of Standard Release R23-11 https://www.autosar.org/fileadmin/standards/R23-11/FO/AUTOSAR_FO_TPS_FeatureModelExchangeFormat.pdf. AUTOSAR.

AUTOSAR Generic Structure Template Nov. 2023. 202. Part of Standard Release R23-11 https://www.autosar.org/fileadmin/standards/R23-11/FO/AUTOSAR_FO_TPS_GenericStructureTemplate.pdf. AUTOSAR.

AUTOSAR Layered Software Architecture Nov. 2022. 53. Part of Standard Release R22-11 https://www.autosar.org/fileadmin/standards/R22-11/CP/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf. AUTOSAR.

Becker, M., Weber, M., and Wierczoch, T., 2008. Varianten in der Automobilelektronikentwicklung, pp. 252–260.

Braun, L., 2018. Modellbasierte Design-Space-Exploration nichtfunktionaler Auslegungskriterien des Fahrzeugenergiebordnetzes. de. https://doi.org/10.5445/IR/1000081298.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V., 2017. *CARLA: An Open Urban Driving Simulator*. https://doi.org/10.48550/ARXIV.1711.03938.

Efatmaneshnik, M., Shoval, S., and Qiao, L., May 2020. A Standard Description of the Terms Module and Modularity for Systems Engineering. *IEEE Transactions on Engineering Management*, 67(2), pp. 365–375. ISSN: 1558-0040. https://doi.org/10.1109/tem.2018.2878589.

Guissouma, H., Lauber, A., Mkadem, A., and Sax, E., 2019. Virtual Test Environment for Efficient Verification of Software Updates for Variant-Rich Automotive Systems. In: *2019 IEEE International Systems Conference (SysCon)*, pp. 1–8. https://doi.org/10.1109/SYSCON.2019.8836898.

Hummell, J. and Hause, M., 2015. Model-based Product Line Engineering - enabling product families with variants. In: *2015 IEEE Aerospace Conference*, pp. 1–8. https://doi.org/10.1109/AERO.2015.7119108.

Isermann, R and Scha, J, 1999. Hardware-in-the-Loop Simulation for the Design and Testing of Engine-Control Systems. *Control Engineering Practice*.

ISO, Sept. 2017. *Systems and Software Engineering - Vocabulary*. Genf, Schweiz.

Kang, K., Cohen, S., Hess, J., Novak, W., and Peterson, A., 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Tech. rep. CMU/SEI-90-TR-021. https://insights.sei.cmu.edu/library/feature-oriented-domain-analysis-foda-feasibility-study/.

Kang, Y., Yin, H., and Berger, C., 2019. Test Your Self-Driving Algorithm: An Overview of Publicly Available Driving Datasets and Virtual Testing Environments. *IEEE Transactions on Intelligent Vehicles*, 4(2), pp. 171–185. https://doi.org/10.1109/TIV.2018.2886678.

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W., 2022. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66). ISSN: 2470-9476. https://doi.org/10.1126/scirobotics.abm6074.

Sax, E. ed., 2008. *Automatisiertes Testen Eingebetteter Systeme in der Automobilindustrie*. Hanser eLibrary. München: Hanser. ISBN: 9783446416352.

Schindewolf, M., Guissouma, H., and Sax, E., 2021. Analysis and Modeling of Future Electric/Electronic Architectures for Modular Vehicles Concepts. In: *21. Internationales Stuttgarter Symposium*. Springer Fachmedien Wiesbaden, pp. 32–46. ISBN: 9783658335212. https://doi.org/10.1007/978-3-658-33521-2_3.

Vector Informatik GmbH, 2024. *PREEvision | Product Lines and Variants*. https://www.vector.com/int/en/products/products-a-z/software/preevision/product-lines-variants/. [Accessed Apr. 19, 2024].

Zengler, C., 2023. *Solving Configuration Problems with LogicNG*. Tech. rep. https://sofdcar.de/wp-content/uploads/2023/08/LogicNG-Whitepaper.pdf. BooleWorks GmbH.