Collection of Requirements and Model-based Approach for Scenario Description

Thilo Braun[1], Lennart Ries[1], Franziska Körtke[2], Lara Turner[2], Stefan Otten[1] and Eric Sax[1]
[1]*FZI Research Center for Information Technology, Karlsruhe, Germany*
[2]*ZF Friedrichshafen AG, Friedrichshafen, Germany*

Keywords: AD, ADAS, Traffic Scenarios, Automotive Testing, Scenario-based Testing, Resimulation, Scenario Description Language.

Abstract: As the level of automation and variety of Advanced Driver Assistance Systems (ADAS) and Automated Driving (AD) increases, new challenges for Verification and Validation (V&V) methods emerge. This applies especially in urban areas due to the combination of many different environmental elements, participant types, and interactions between the participants. Scenario-based testing and resimulation of recorded data are promising approaches to tackle these new challenges. An elementary component of these methods is the scenario description, which serves as a connection between different working steps in the V&V workflow. This heterogeneous usage of the scenarios during the development and validation process leads to a multitude of different, sometimes contradictory, demands on the scenario description. Nevertheless, a uniform description is desirable for easy exchange and automation. The contribution of this paper is twofold: Firstly, the described versatile field of demands is systematically broken down to requirements for the scenario description languages. This step is essential to ensure broad applicability. Secondly, this paper introduces a holistic scenario description language that is usable for generation, extraction from real-world test drives and execution of the scenarios enabling an automated workflow and increased traceability between generated, extracted and resimulated scenarios. The description and uses a model based approach and has been exemplarily tested for manually created scenarios and automatic resimulation of real-word test drives.

1 INTRODUCTION

The development of Automated Driving (AD) is progressing fast over the last years. The goal of higher automation in traffic is increased safety, as assistance systems support the driver in critical situations, e.g. by automatically initializing an emergency braking maneuver, and with increased comfort, e.g. by driving on a highway autonomously.

The degree of automation is defined by SAE International in five levels, from ”no automation” on Level 0 to ”full automation” on Level 5 (International, 2018). To proceed from the current prototype status of AD to approved products, an accepted validation process is necessary. Current validation methods like requirement-based testing or real-world endurance runs are not able to validate the Level 3+ functions, because the statistically necessary kilometers are not reachable (Wachenfeld and Winner, 2016). Furthermore the validity of a distance-based method is not sufficient in complex environments, because it is not ensured that rare corner situations are considered (Schuldt, 2017).

Scenario-based testing is considered as a promising method to support the validation of AD (Neurohr et al., 2020). With its ability to produce critical situations through a synthetic scenario description it enables focusing on the most relevant scenarios and therefore achieves a high coverage fast and efficiently. The scenarios for scenario-based testing can be manually generated from domain experts, extracted from real-world test drives for resimulation in a data-driven approach or generated from computer algorithms. During the test process, users like test and simulation engineers, require different description languages and formats to describe the scenarios depending on their specific task. As a consequence, the scenarios have to be transformed when processing and exchanging them, what is usually a manual process leading to increased effort and high error rate. A holistic and formalized Scenario Description Language (SDL) can avoid these problems but has to take into account the diverse views and needs of all users.

Braun, T., Ries, L., Körtke, F., Turner, L., Otten, S. and Sax, E.
Collection of Requirements and Model-based Approach for Scenario Description.
DOI: 10.5220/0010478706340645
In *Proceedings of the 7th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2021)*, pages 634-645
ISBN: 978-989-758-513-5

To enable a structured design of such a Domain Specific Language (DSL), the authors of this paper collect these views and needs and brake it down to requirements. Based on these, we propose a DSL for broad application which supports both the virtualization of recorded real-world data and the manually generation of scenarios in the second part of the paper. This enables an automated workflow from recorded data to resimulated scenarios, while allowing to additionally integrate handcrafted scenarios seamlessly.

This paper is structured as follows: Chapter 2 introduces different validation methods for AD and embeds the proposed approach. Chapter 3 describes, among others, the usage, requirements and state of the art for scenario description languages. Chapter 4 gives common maneuver categories and sets to describe the behavior in an abstract way. Chapter 5 explains our concept including its technical implementation. Chapter 6 shows the tools for the extraction of scenarios from recorded real-world testing as well as manual generation and editing of scenarios. Chapter 7 exemplarily demonstrates the complete workflow for resimulating a scenario. Chapter 8 contains the conclusion and future work.

2 VALIDATION METHODS FOR AD

Requirement-based testing has been the most popular method to validate AD in the past. It uses specified test cases including pass/fail criteria, that are derived from analysing the system requirements for the System under Test (SuT).

The requirement-based testing is usually combined with endurance runs, which make safety statements by means of statistical evaluation of successfully driven kilometers. To reduce the cost, the test drives can also been simulated (King et al., 2019). As stated before, the combination of specified test-cases with endurance runs is not sufficient for higher levels of automation. Therefore, new methods are under research, e.g. in the projects PEGASUS (Pegasus, 2019) and VV-Methods (VVM, 2020).

Formal test approaches such as Responsible Sensitive Safety argument with exact mathematical models and white-box testing (Shalev-Shwartz et al., 2017). However, their application and implementation is not straight forward and, thus, scope of further research (Koopman et al., 2019).

For reasons of reproducibility, recorded test drives can be replayed, which is an approach frequently used in industry. While a replay of recorded data is sufficient for open-loop systems, systems that interact with the environment require a closed-loop reactive simulation (Bach et al., 2017) (Pfeffer et al., 2019).

In scenario-based testing, a catalog of relevant traffic scenarios is generated and used for the test-coverage estimation (Schuldt, 2017). The scenarios can be created knowledge-based or, in contrast to this, data-driven and extracted from real-world test drives (Langner et al., 2018) (Stellet et al., 2015) . The execution of the scenarios is possible on different platforms (software-in-the-loop, hardware-in-the-loop, vehicle-in-the-loop) whereby most of the tests are expected to be performed in simulation environments because of their high scalability.

In this paper, we focus on scenario-based testing. As proposed in literature (Neurohr et al., 2020), we assume that the resimulation workflow is conceptually integrated in the testing process, as shown in figure 1. In addition, we show a technical integration by providing a SDL for this workflow, enabling a fully automated workflow for resimulation. In a first step the scenarios are generated, either extracted from recorded data or manually, and stored in the scenario catalog. Then they are executed in a simulation with parameter variations to enable systematical testing of the AD function.

3 SCENARIO DESCRIPTION

A test in scenario-based testing consists of several parts: Firstly, the description of the scenario, in which the static and dynamic content is described, and secondly the description of the test case itself where, among others, the SuT with test goal and test evaluation and the test setup are described. Scenarios can be represented in different forms: While they are available in the form of sensor data for recorded test drives, scenario-based testing of AD functions requires a semantic interpretation of these data.

3.1 Usage and Requirements for Scenario Description Languages

In the following, we describe where and how scenarios are used in the test process and the role of the SDL to derive requirements for its design. A common way to generate relevant scenarios is to use the knowledge of domain experts and formulate the scenarios manually. When creating scenarios manually, the SDL defines which elements can be used and how they are related to each other.
Another source of scenarios is recorded data from real-world test drives or simulation. When extracting

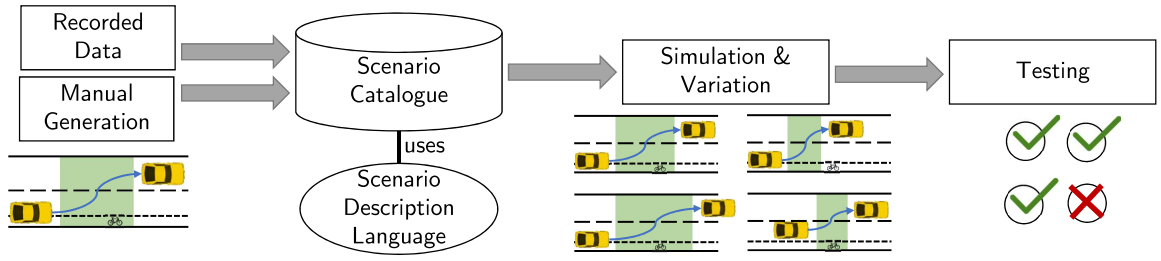


Figure 1: Workflow and usage of the scenario description language in scenario-based testing.

scenarios from recorded data, the SDL defines the abstraction level and interface for the extraction algorithm.

To save the scenarios in the catalog, the SDL defines the data format.

In the execution of the scenario within a simulation tool, the SDL has an imperative character being the instruction for the underlying behavior models.

Besides creating and executing scenarios, inspection, comparing and clustering are actions to gain further knowledge about the scenarios. Here, the SDL defines the interface to the specific tools and can support the algorithms by providing relevant information.

Based on an analysis of the previously described usages and discussions with domain experts and researchers, this section summarizes the most important requirements for the SDL.

Expressive. It should provide the ability to describe as many possible driving scenario as possible. This includes, e.g., different Operational Design Domain (ODD)s like highway or inner-urban scenarios, different participant types like cars, bikes and trucks and different types of behaviors.

Extendable. It should be possible to add new features to the language when necessary to make the language more expressive. So the language can and should evolve over time and adapt to new challenges in the testing domain.

Abstract. The SDL should be capable of describing scenarios in different abstraction levels. Higher abstraction levels are needed for comparing and interpreting scenarios, lower abstractions levels for definition of very specific driving situations. An example for a low abstraction level is the description of an objects' movement with exact trajectories, whereas a verbal description would be a high abstraction level. A higher abstraction level is necessary to condense the complex and large scenario space.

Reference on Scenery. Describing scenarios only based on trajectories will cause problems on different curve radius or junction forms, because it is not distinguishable if a curved trajectory was caused from a curved road, intersection or another maneuver. When using an abstracted description the reference on the scenery connects the behavior of the participants with the static part.

Parameter Spaces and Concrete Values. It should be possible to describe the parameters of the scenarios with concrete values and parameters spaces. This also leads to a condensation of the scenario space. This kind of abstraction is comparable to the PEGASUS levels of concrete and logical scenarios.

Comparable and Searchable. It should be possible to compare, search and filter scenarios with specific features on the basis of the SDL. This is necessary for a good handling of the scenario catalog, a categorization, clustering of the scenarios and evaluation of the covered scenario space.

Concise. Especially when generating scenarios manually, the description of a scenario should be as compact and precise as possible to avoid unnecessary long descriptions.

Executable. The language has to be defined in a way, that an interpreter can execute the scenario. The interpreter can be an algorithm of a simulation tool or a test driver.

Understandable for Humans. The content of the described scenario should be intuitively understandable for all its users, e.g. test engineers, programmers and non-programmers and also non-domain-experts.

Exchangeable and Tool-independent. It should be possible to exchange scenarios between different tools. This concerns scenario generation, editing and execution. The exchanging should be traceable.

Although many requirements have to be considered, a uniform SDL for the V&V process brings advantages: It enables easy exchange between different

tools, better traceability during the process and a common understanding between different users.

3.2 Layers of Scenarios

In order to classify the elements in road traffic, there exist a series of layered models proposed in the literature. In the following, we refer to one of the latest publications (Bock et al., 2018) introducing a total of six layers (so-called 6-layer-model). Layer 1 describes the street layer including the geometry, topology and condition of the roads. Layer 2 addresses the traffic infrastructure with, e.g., signs and markings. The focus of Layer 3 is on temporal modifications of the previous two layers describing the geometrical and topological overlay by construction sites. The movable traffic participants and objects and their behavior (interactions, maneuvers) is listed in Layer 4. Layer 5 contains the environmental conditions such as weather and daytime as well as their effects on Layer 1-4. Finally, Layer 6 is designed to summarize digital map data and information on V2X communication.

This paper focuses on Layer 4, which has a high complexity due to the variety of possible combinations of participant behaviors and maneuvers. When necessary, connections to other levels are also considered.

3.3 Types of Scenario Descriptions and Related Work

There exist different approaches to describe scenarios, targeting different use cases and user groups. This section gives an overview about existing approaches and serves as a basis for the design of our SDL.

When evaluating the concepts and designing an own implementation, the following trade-off has to be considered: Developing a language that is intuitive for humans and easy to apply on the one side or developing a machine-readable and executable language on the other side. Another important property to be taken into account is the expressiveness of the language, which specifies how many different scenarios can be described.

3.3.1 Verbal Description

Verbal languages use natural language to describe scenarios. One example is the functional description mentioned above or first drafts of scenarios written down manually. Obviously, a verbal description is very easy to understand and write for humans and has a very high expressiveness. On the other hand, it is not possible for machines to create an executable scenario of the given description. This kind of language is often used in early stages of a scenario generation, when the focus is mainly on the content and not the execution of the scenario.

3.3.2 Visual Description

Visual languages use visual representations of the scenario, e.g., snapshots in bird view perspective. They are often an alternative or addition to other language types like verbal or programming languages. Similar to the verbal language, the visual representation is easy to understand for humans, but not readable for machines. It is a good method to get a quick overview about the scenario and share it, although it is hard to write down all the necessary details.

3.3.3 Formalized Verbal and Visual Description

To overcome the lack of readability for machines, some efforts were done to formalize verbal and visual languages using a defined vocabulary and rules (Damm et al., 2018) (Menzel et al., 2018) (Kohlhaas et al., 2014). The executability is obtained at the expense of reduced expressiveness and a more complicated generation process, because the person generating the scenario has to learn the defined rules and the vocabulary.

3.3.4 Programming Language and Domain Specific Language

The need to execute the described scenarios in simulation tools suggests the use of programming languages. Here a universal programming language (e.g., python, C++) or a DSL can be used. DSLs are designed for a specific domain or use case. They can define an own syntax (external DSL) or use the syntax of an universal programming language as a basis (internal DSL). Examples for scenario DSL are the concept of OpenSCENARIO 2.0 (ASAM, 2020b) and the M-SDL (Foretellix, 2020). For programming languages, the requirement of machine-readability is obviously satisfied, but they can normally only be used and understood by experts. Therefore, programmers and simulation experts are the main target group of this kind of languages. The expressiveness and usability of the DSL depends on the design. Comparing to an universal programming language it is easier to learn because it is tailored to a domain and has a reduced set of commands.

3.3.5 Model-based Scenario Description Language

In the model-based scenario description, a technical metamodel is created that maps all relevant entities and relationships (Wuellner et al., 2019). This can be a UML class model like in OpenSCENARIO 1.0 (ASAM, 2020a), an Ecore model (Bach et al., 2016) (Jafer et al., 2018) or an ontology (Geyer et al., 2013). Each scenario is an instance of this metamodel. Using a graphical representation of the domain as metamodel, this approach is easy to understand. The formalized description also assures machine readability. If the metamodel is directly linked to a software, a change in the metamodel effects a change in the software, assuring that conceptional changes, normally done in the metamodel, and implementation do not diverge. This makes the model-based approach easy to maintain.
Previous mentioned descriptions focus on a single sub task of the validation workflow, whereas this work provides a holistic approach enabling automation and traceability throughout the whole resimulation process.

4 MANEUVERS

In order to understand which part of the driving task is described by maneuvers, the driving task is divided into several levels. (Lange, 2018) compares existing models for the driving task and summarizes them in four levels: Navigation level, Maneuver level, Trajectory level and Stabilization level.

The maneuver level converts the mission plan generated by the navigation level into suitable maneuvers and is responsible for the behavioral decision. The trajectory level converts the selected maneuvers to concrete trajectories.

A sequence of (driving) maneuvers, which are used instead of trajectories, has proven to be a useful abstraction for the dynamic behavior of traffic participants (ASAM, 2020a) (ASAM, 2020b) (Pfeffer et al., 2019) (Hartjen et al., 2019). Compared to a description with trajectories, maneuvers are much more concise and intuitive. By defining maneuver-specific parameters the expressiveness and variability of this description is increased.

To structure the maneuvers and describe their properties, a categorization of these maneuvers is often done. This section describes commonly used categories:

Direction. Frequently, a classification is made into the movement directions longitudinal and lateral. The classification is particularly intuitive for vehicle steering because lateral maneuvers are controlled by the steering system and longitudinal maneuvers by the drive train. The category can be extended by routing maneuvers (ASAM, 2020a).

Interaction. (Firl and Tran, 2011) creates a maneuver category, indicating whether an interaction with other participants is part of the maneuver or not.

Triggered and Self-terminating. (Schreiber, 2012) categorizes based on the trigger for starting the maneuver. If a maneuver is actively started with a trigger, e.g., a lane change, this is an explicit maneuver. A lane keeping maneuver following the lane change without trigger is an implicit maneuver because it starts automatically when the lane change is finished. Closely related to this category is whether a maneuver terminates itself (e.g., lane change) or whether it is without defined end (e.g., lane keeping).

Atomic and Composed. The division into atomic and composed maneuvers is often found (Bagschik et al., 2018) (ASAM, 2020b). Atomic maneuvers are a basic class of maneuvers that can be combined to composed maneuvers. The rules of combination range from quite simple sequencing (ASAM, 2020a), maneuver graphs (Bach et al., 2016) or methods of a DSL (Foretellix, 2020), where the border between maneuvers and scenarios vanishes.

5 CONCEPT FOR A SCENARIO DESCRIPTION

The concept aims to create a semantic meaningful abstraction to describe the behavior of dynamic participants and use it for recorded and manually generated scenarios and for their execution as shown in Figure 1. It is desirable to have only one SDL for all processing steps to avoid converting the scenarios. For recorded scenarios, the abstraction should contain all necessary information for their analysis and resimulation without using the recorded time series data. For manually generated scenarios, the abstraction should be as concise as possible to enable faster specification. Furthermore it should be possible to modify the scenarios manually. The abstraction should be both intuitive for humans and easy to use for machines. The mentioned requirements should be fulfilled for a variety of ODDs, including the ambitious domain of urban traffic. In this chapter, we explain the concept and

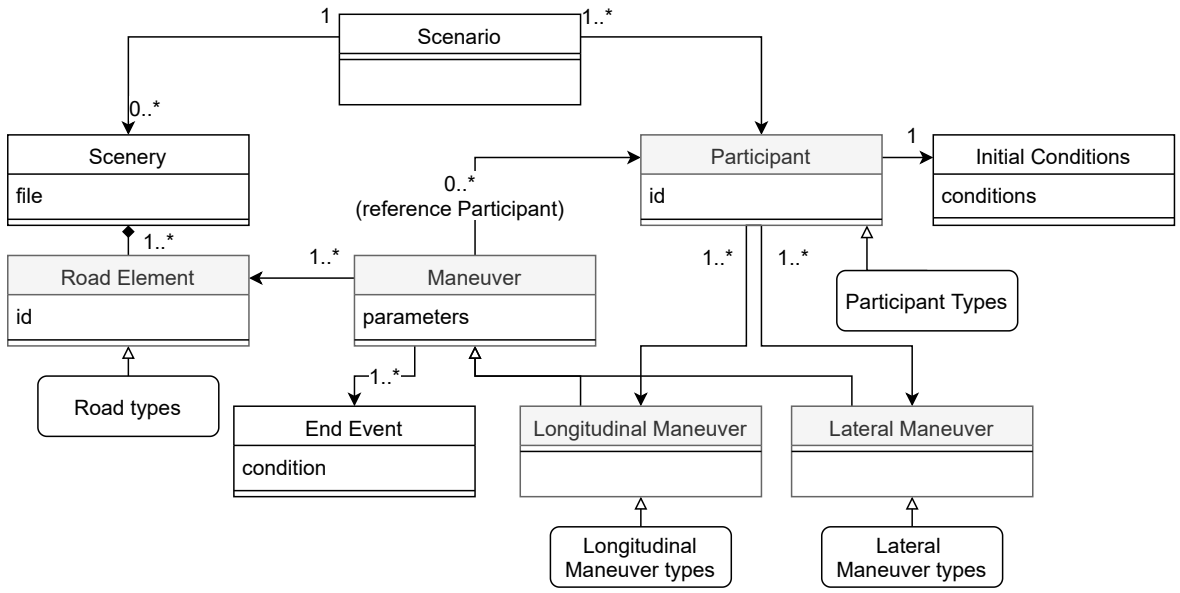


Figure 2: Elements and relationships of a scenario. Grey classes are abstract classes that cannot be instantiated. For reasons of space some inherited classes are not shown in the figure but the types are indicated with boxes with curved edges. The numbers on the connections define the cardinality.

give motivation for our design decisions, abstractions and technical representation.

5.1 Core Elements

Figure 2 shows the core elements of a scenario and their relations in form of a UML-model.
Classes with gray background are abstract classes, meaning that only an inherited class can be instantiated. So general properties of this class can be designed, but the instance is always of a specific type.

The *Scenario* consists of a static part, the *Scenery*, and a dynamic part, the *Participants*. The *Scenery* consists of several *Road Elements*. For the structure of the *Road Elements* and the *Road Types* we use a subset of the OpenDRIVE format, therefore it can be automatically imported from a given OpenDRIVE file. Each *Road Element* has a unique id.

The *Participants* have *Initial Conditions* that define the state at the beginning of the scenario (e.g. position on the road, heading, velocity). *Participant Types* are car, truck, pedestrian, bicycle and motorbike.

The dynamic behavior of the participants is described with *Maneuvers*. Each *Participant* has a list of *Lateral Maneuvers* and *Longitudinal Maneuvers* that are sequentially executed during the scenario. Figure 4 shows our concept for the graphical representation. The *Lateral* and *Longitudinal Maneuver* classes inherit from the base *Maneuver* class. Each *Maneuver* has parameters as attributes that allow to vary the behavior, e.g. velocity of a *Longitudinal Maneuver*, distance of a *Stop Maneuver* or the lateral offset on a lane of a *Lateral Maneuver*. *Maneuvers* refer also the *Road Elements* on which they are performed.

To define when to end the *Maneuver*, each *Maneuver* contains at least one *End Event*. An *End Event* occurs if the specified conditions are fulfilled, e.g. if a certain point on the road is reached or the distance to another *Participant* is below a specific value. Furthermore a *Maneuver* can have reference *Participants*, if interaction with other participants is required. This could be the preceding vehicle in a *Follow maneuver*. Examples for *Longitudinal Maneuvers Types* are *Cruise Free*, *Approach* and *Follow* (another *Participant*). Examples for *Lateral Maneuver Types* are *Hold Lane*, *Cross Street* and *Turn*.

Turn maneuvers do not have relation to the specific junction or exit, but are described as *Turn left* or *Turn right*. So it is possible to execute and compare scenarios with the same dynamic behavior on different junctions.

Figure 3 shows exemplarily a longitudinal maneuver list with the parameters of the maneuver. When creating scenarios, a rule checker is used to detect unrealistic maneuvers. It detects infeasible combinations of types, e.g., only pedestrians and bikes can use a sidewalk or perform a *Cross Street* maneuver on a crosswalk. It detects infeasible parameters of the *Maneuver*, e.g., *Follow* maneuvers

with a negative distance to the preceding *Participant*.

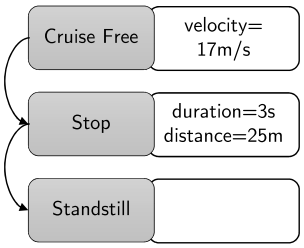


Figure 3: Example of a longitudinal maneuver list with corresponding parameters.

5.2 Used Maneuver Categories from Chapter 4

To describe the maneuvers in an urban environment, the categories *direction* and *interaction* with other participants are used.
The behavior of a participant is always described with a combination of one lateral and one longitudinal maneuver to decouple the description of steering and velocity.

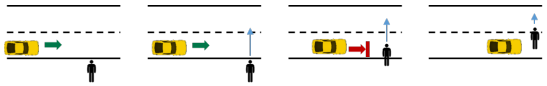
The interaction category is necessary to describe maneuvers with a reference participant. An interactive maneuver has at least one reference participant, for a non-interactive maneuver the reference participants list is empty.

We treat all maneuvers as *explicit* in our implementation, because the start of a maneuver in our concept is always marked by the end of the previous maneuver. Generally, adding the right end events to the maneuvers is complex, but for *self-terminating* maneuvers it is trivial. Therefore, we use this category to automatically add end events when possible. Until now, we treat all maneuvers as *atomic*, but it is possible to integrate *composed* maneuvers in the concept to make the description more concise.

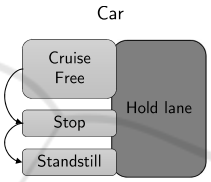
5.3 Abstraction of Evolution Over Time

The information on the progression of the scenario is mainly described by the end events, that define both the end of the current maneuver and the start of the next maneuver. We do not use an act-based description as (Bach et al., 2016) introduced for highway scenarios. Even if this is an elegant way to define simultaneous maneuvers, the number of acts increases significantly when more participants appear and the maneuvers change more often. This effect makes the act-based description not usable for an urban traffic scenario. To define simultaneous maneuvers, the conditions for the end events have to be carefully chosen.

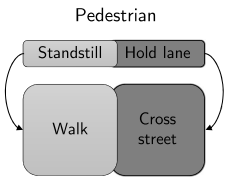
When extracting scenarios from real-world data, participants appear or disappear during the scenario runtime since they exceed the maximum sensor range or due to occlusion. We defined a special "None-Maneuver" for this time span. If the None-Maneuver is at the start of the scenario, the participants start to perform the remaining maneuvers after the end event for this None-Maneuver. As default we use the road position of the ego car as end event for the first None-Maneuver. If the None-Maneuver is at the end of the scenario, the participant disappears during this maneuver.



(a) Visualization of scenario and maneuvers



(b) Maneuvers and end events of the car



(c) Maneuvers and end events of the pedestrian

Figure 4: Example of an abstracted scenario with lateral and longitudinal maneuver lists.

5.4 Example of an Abstracted Scenario

Figure 4 shows an example of an abstracted scenario. There are two participants, a car driving on a straight road and a pedestrian that wants to cross the road. The pedestrian should start crossing the road causing the car to brake. This scenario could be used for testing an emergency brake assist.

The sequence of graphics in Figure 4a visualize the scenarios evolution over time.

Figure 4b shows the maneuvers and end events of the car. The arrows indicate the end events. The first end event occurs when the car detects the pedestrian (non self-terminating maneuver). Because Stop is a self-terminating maneuver, the maneuver ends when the velocity of the car is zero. The Standstill maneuver ends after a number of predefined seconds and at this time the scenario finishes.

Figure 4c shows the maneuvers and end events of the pedestrian. He starts crossing the street when the distance to the car falls below a certain threshold thus challenging the car.

For testing purposes the parameters of the maneu-

vers, e.g. cruising velocity of the car, and the parameters of the end events, e.g. distance for starting the Cross Street maneuver, can be varied to expand the test coverage.

The visual representation of our concept is shown in figure 4b and 4c. Light gray blocks show the longitudinal maneuvers, dark gray the lateral maneuvers, the arrows indicate the end events. This representation is intuitively understandable and enables the execution of the scenario.

5.5 Technical Representation as Metamodel

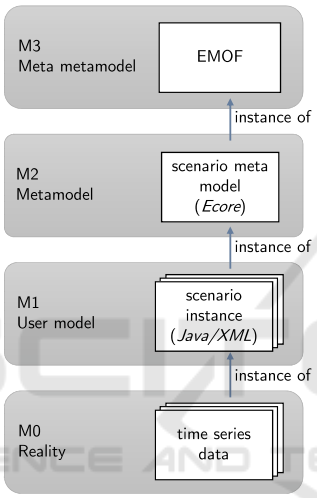


Figure 5: Model levels in model-driven engineering, their equivalents in the traffic scenario domain and used formats (italic).

We applied a metamodeling approach to design our scenario description language. Figure 5 shows the general concept of metamodels (OCUP2 ExaminationTeam, 2017), their equivalents in the domain for traffic scenarios and the formats we used for our concept. Level M3 is a meta metamodel that defines the language for specifying metamodels, meaning entities like classes and relationships. Level M2 is the domain specific metamodel, that uses M3 to describe the structure of scenarios. Here the different entities of the scenario and their relationships are defined. Level M1 describes the abstracted scenarios as instances of the metamodel in Level M2. A instance of Level M1 is a specific scenario in our SDL. Level M0 is the real-world without any abstraction, so in our case the trajectories of the scenario.

We created an Ecore metamodel using the Eclipse Model Framework (EMF) (Foundation, 2020) for level M2. The class diagram in Figure 2 shows the core of our metamodel. The metamodel is designed as Data Transfer Object (DTO) and uses object-oriented programming. The DTO enables the scenarios to be transferred easily between tools and programming languages (e.g. different tools for simulation, creating and editing scenarios, scenario extraction and workflow management).

The object-oriented programming encapsulates the information and fits the domain because the participants are physical objects. Inheritance is used to generalize specific characteristics for e.g. different participants or maneuver types. Each new participant type or maneuver type is a new class and inherits from the corresponding superclass.

A scenario can be instantiated either as Java-instance, the native format of EMF, or as XML-instance, which is suited for exchange between programs. It is possible to transfer the instances automatically form one format into the other.

5.6 Comparison of Concept with Requirements

This section evaluates the concept, mainly the conceptional abstraction to maneuvers and the metamodel as technical representations, against the requirements of chapter 3.1.

Expressive. The expressiveness of the scenario description is strongly connected to the used maneuver set and maneuver parameters. With a well-defined maneuver catalog the desired scenarios can be described adequately. To increase expressiveness new maneuvers, sub-maneuvers or maneuver parameters can be added.

Extendable. Object-oriented programming and inheritance for maneuvers, participants and road types are methods that are easily extendable. The direct link between domain description and tools in the metamodel concept ensures the transfer from the concept level to the technical level.

Abstract. The concept abstracts the scenario from concrete time series data to semantic meaningful maneuvers.

Reference to Scenery. The concept uses a subset of the OpenDRIVE standard to create a reference on the scenery. By using the standard no further definitions in the scenery description are necessary and existing files can be used seamlessly. For execution the specification of the direction of the turn maneuvers allows, e.g., to run the same scenario on different

junctions.

Parameter Spaces and Concrete Values. So far, parameter spaces are not used in the concept. For parameter variation an external tool is still necessary. This topic can be focused in future work.

Comparable and Searchable. The abstraction to maneuvers allows to compare, search or filter scenarios based on maneuvers. The scenario catalog can be searched for scenarios with specific maneuvers or scenery elements. Furthermore, scenarios can be defined as equal, when their participants have the same maneuver lists.

Concise. Maneuvers are a concise way to describe the behavior of the participants. Using the subset of OpenDRIVE reduces the effort for the scenery description. Adding standard end events to self-terminating maneuvers makes the description more concise.

Executable. By the definition of exactly one lateral and exactly one longitudinal maneuver the behavior of the participants is described in an unambiguous and executable way. Also the end events define unambiguously when to proceed to the next maneuver.

Understandable for Humans. The abstractions from time series to maneuvers and visual representation of the maneuver lists shown in figure 4 makes the scenarios intuitively readable and understandable. The graphical representation as metamodel is also a intuitive technical format.

Exchangeable and Tool-independent. Using the data transfer object design pattern and the export option as XML-file makes it easy to exchange the scenarios between tools.

In summary, the abstraction from time series to maneuvers fulfills the requirements of the conceptional level, the use of a metamodel meets the technical requirements.

6 TOOLS FOR SCENARIO GENERATION

6.1 Scenario Extraction

The target of the scenario extraction is the generation of scenario instances from recorded data. Output format is the scenario description language introduced in Chapter 5. The data can be recorded by a driving vehicle, static sensors in the infrastructure or drones. Also existing datasets with recorded trajectories can be used. Similarly, the concept applies to simulation data.

In a first step, data preprocessing is necessary to compensate noise in the recorded data and exploit knowledge about the scenery. This includes the assignment of the participants to the next participant-type specific valid road element and lane and the classification of irrelevant participants (e.g. when they are present a very short). The scenery is represented by an OpenDRIVE file.

In the next step the performed lateral and longitudinal maneuvers are classified. After classifying the maneuvers, the corresponding parameters and end events are extracted. Finally, all extracted data is restructured to fit the our SDL and the scenario is saved in the scenario catalog.
We use rule-based algorithms for all extraction steps.

Figure 6 shows an example of an Approach-and-Follow scenario in a curve. Using the road elements of the scenery the relative distance along the lane is calculated. Compared with the euclidean distance this method provides reasonable results even in curves. The Approach maneuver changes into a Follow maneuver when the ego vehicle and preceding vehicle have the same velocity.

6.2 Manual Generation and Editing

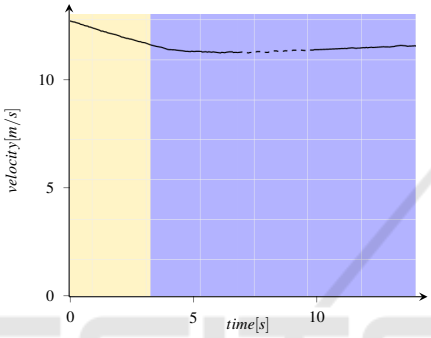
As stated before, it is possible to edit and create scenarios manually. So expert knowledge can be added to the extracted scenarios or detected false classifications of the scenario extraction algorithm can be corrected. Also the parameters of the maneuvers can be edited.

Figure 7 shows an example how to edit a junction scenario. The left side shows the scenery and initial positions of the participants. The right side shows the editable elements like participants and maneuvers. When selecting an element an interactive mask with all editable elements shows up. The same dialogue can be used to build up scenarios from scratch. The program was implemented by Zukunft Mobility GmbH, a company of ZF Friedrichshafen AG.

7 ILLUSTRATIVE EXAMPLE

For illustration of our approach we executed the complete workflow automatically for a recorded real-world scenario on a junction. Figure 8a shows the trajectories of the participants embedded in a satellite

(a) Trajectory of the ego and preceding vehicle. The black lines show the start and end of the scenario, the black boxes show the initial positions of the vehicles. The yellow color indicates the trajectory of an Approach maneuver, the blue color indicates the trajectory of a Follow maneuver. The trajectories of the two vehicles overlap.



(b) Detected maneuvers and velocity of the ego vehicle. The velocity decreases (Approach maneuver, yellow) until it is equal to the preceding car, that is driving with a constant velocity (Follow maneuver, blue). The dashed line indicates a lateral Cross-Junction maneuver.

Figure 6: Detection of maneuvers in an Approach-and-Follow scenario in a curve. Yellow indicates the Approach maneuver, blue indicates the Follow maneuver.

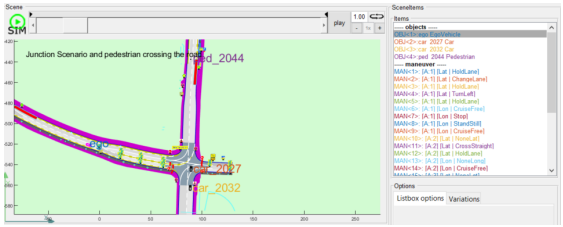


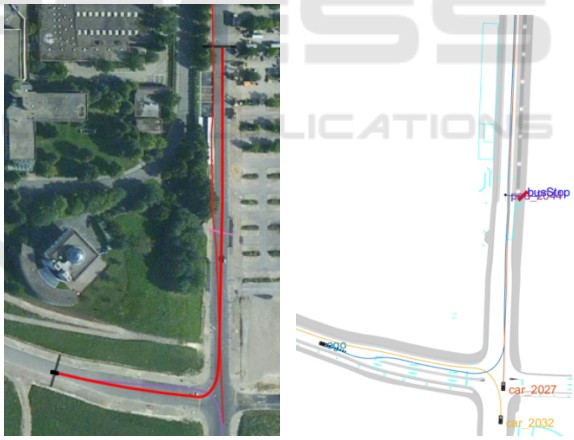
Figure 7: GUI for manual editing a scenario implemented by Zukunft Mobility GmbH (ZKM).

picture of the scenery. The scenario contains 3 cars (including the ego car that recorded the data) driving over a junction and one pedestrian crossing the street. The ego car turns left on the junctions and has to stop to give way to another car. The pedestrian is crossing the street, but not influencing other participants. The scenery was known and described in the OpenDRIVE format.

The scenario was extracted with the scenario extraction algorithm, saved as instance of the metamodel and then executed by a simulation tool. The simulation tool only takes the abstract scenario description and scenery as input, but has no knowledge about the recorded time series like velocity or trajectories. Please notice that the workflow is completely automated.

Figure 8b shows the resimulated trajectories of the scenario, that are very similar to the original ones. Figure 9 displays the recorded and resimulated velocity of the ego vehicle. In the recorded speed diagram the classified maneuvers Stop (red), Standstill (gray) and Cruise Free (green) are marked. In the resimulated velocity a similar speed profile is generated. In addition, the gear shifts produced by the simulation driver are visible as small peaks.

With this example scenarios we demonstrate how a fully automated resimulation of a scenario can be performed with our concept. The behavior is successfully abstracted and used as input for a simulation that generates comparable results. The concept also supports more complex scenario, but need a more powerful scenario extraction algorithm or manual supoort.



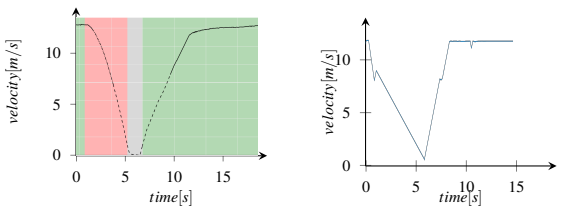
(a) Recorded trajectories

(b) Snapshot of resimulated trajectories in the ZKM simulation tool

Figure 8: Resimulation: Trajectories of a scenario on a junction. Black boxes indicate the initial positions of the vehicles, colored lines their trajectories.

8 CONCLUSION AND FUTURE WORK

We motivated this paper by the necessity for new testing methods for AD due to the rising automation level



(a) Recorded velocity of ego vehicle with detected maneuvers.

(b) Resimulated velocity of ego vehicle in the ZKM simulation tool

Figure 9: Resimulation: Velocity of ego vehicle.

and the variety of new ODDs. The scenario description was identified as an elemental part for the approaches scenario-based testing and automatic resimulation. This paper collects and structures requirements for the scenario description as basis for its design. Its shows the integration of the resimulation of recorded test drives in scenario-based testing using scenario extraction and propose an abstract and holistic model-based scenario description language. The design as tool-independent metamodel ensures an extendable, maintainable and traceable use of scenarios between different working steps, tools and execution platforms. It is designed to fit the needs of urban traffic and is easily extendable to other ODDs. Furthermore it is possible to modify and add scenarios manually to include expert knowledge. Due to the completely automated workflow for generation of scenarios from recorded real data and resimulation of these, our approach offers a scalable option to build up a scenario catalog for testing of AD. The approach was implemented and exemplarily tested on recorded real-world driving data in urban traffic.

Future work will focus on the integration of parameter distributions in the description language to be able to describe logical scenarios. A condensation of the scenarios to scenario clusters is a necessary step to define the coverage of the scenario catalog. It is planned to use the method on more different data sources and with more simulation tools. The usage of mixed datasets of recorded and resimulated scenarios as input for AI-based methods is another interesting research topic.

ACKNOWLEDGMENT

We thank Katrin Lotto (ZF) for her help in creating the concept of the SDL. We thank Markus Lemmer (FZI) for his help in implementing the concept.We thank Marco Alt (ZKM) for his help in implementing the scenario description in their simulation tool. We thank Philipp Rigoll (FZI) for his help in implementing the visualization tool for extracted scenarios.

REFERENCES

ASAM (2020a). OpenSCENARIO 1.0 User Guide. https://www.asam.net/index.php?eID=dumpFile&t=f&f=3496&token=df4fdaf41a8463e585495001cc3db3298b57d426. Accessed: 2020-10-16.

ASAM (2020b). OpenSCENARIO 2.0 Concept Paper. https://www.asam.net/index.php?eID=dumpFile&t=f&f=3460&token=14e7c7fab9c9b75118bb4939c725738fa0521fe9. Accessed: 2020-10-16.

Bach, J., Holzäpfel, M., Otten, S., and Sax, E. (2017). Reactive-replay approach for verification and validation of closed-loop control systems in early development. Technical report, SAE Technical Paper.

Bach, J., Otten, S., and Sax, E. (2016). Model based scenario specification for development and test of automated driving functions. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 1149–1155. IEEE.

Bagschik, G., Menzel, T., and Maurer, M. (2018). Ontology based scene creation for the development of automated vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1813–1820. IEEE.

Bock, J., Krajewski, R., Eckstein, L., Klimke, J., Sauerbier, J., and Zlocki, A. (2018). Data basis for scenario-based validation of HAD on highways. In *27th Aachen colloquium automobile and engine technology*.

Damm, W., Kemper, S., Möhlmann, E., Peikenkamp, T., and Rakow, A. (2018). Using traffic sequence charts for the development of havs. In *European Congress on Embedded Real Time Software and Systems 2018*, 9th European Congress on Embedded Real Time Software and Systems (ERTS 2018).

Firl, J. and Tran, Q. (2011). Probabilistic Maneuver Prediction in Traffic Scenarios. In *ECMR*, pages 89–94.

Foretellix (2020). M-SDL Specification. Accessed: 2020-08-146.

Foundation, E. (2020). Eclipse modeling framework (emf). https://www.eclipse.org/modeling/emf. Accessed: 2020-10-16.

Geyer, S., Baltzer, M., Franz, B., Hakuli, S., Kauer, M., Kienle, M., Meier, S., Weißgerber, T., Bengler, K., Bruder, R., et al. (2013). Concept and development of a unified ontology for generating test and use-case catalogues for assisted and automated vehicle guidance. *IET Intelligent Transport Systems*, 8(3):183–189.

Hartjen, L., Schuldt, F., and Friedrich, B. (2019). Semantic classification of pedestrian traffic scenarios for the validation of automated driving. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3696–3701. IEEE.

International, S. (2018). Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. https://www.sae.org/standards/content/j3016_201806/. Accessed: 2020-10-16.

Jafer, S., Chhaya, B., Zeigler, B. P., and Durak, U. (2018). SES and Ecore for Ontology-based Scenario Modeling in Aviation Scenario Definition Language (ASDL). *International Journal of Aviation, Aeronautics, and Aerospace*, 5(5):4.

King, C., Ries, L., Kober, C., Wohlfahrt, C., and Sax, E. (2019). Automated function assessment in driving scenarios. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, pages 414–419. IEEE.

Kohlhaas, R., Bittner, T., Schamm, T., and Zöllner, M. (2014). Semantic state space for high-level maneuver planning in structured traffic scenes. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1060–1065. IEEE.

Koopman, P., Osyk, B., and Weast, J. (2019). Autonomous vehicles meet the physical world: RSS, variability, uncertainty, and proving safety. In *International Conference on Computer Safety, Reliability, and Security*, pages 245–253. Springer.

Lange, A. (2018). *Gestaltung der Fahrdynamik beim Fahrstreifenwechselmanöver als Rückmeldung für den Fahrer beim automatisierten Fahren*. PhD thesis, Technische Universität München.

Langner, J., Bach, J., Ries, L., Otten, S., Holzäpfel, M., and Sax, E. (2018). Estimating the uniqueness of test scenarios derived from recorded real-world-driving-data using autoencoders. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1860–1866.

Menzel, T., Bagschik, G., Isensee, L., Schomburg, A., and Maurer, M. (2018). Detailing a Keyword Based Scenario Description for Execution in a Simulation Environment Using the Example of Scenarios on German Highways. In *Workshop Fahrerassistenzsysteme und automatisiertes Fahren*, volume 12, pages 15–26.

Neurohr, C., Westhofen, L., Henning, T., de Graaff, T., Möhlmann, E., and Böde, E. (2020). Fundamental considerations around scenario-based testing for automated driving. *arXiv preprint arXiv:2005.04045*.

OCUP2 ExaminationTeam (2017). Meta-Modeling and the OMG Meta Object Facility (MOF). https://www.omg.org/ocup-2/documents/Meta-ModelingAndtheMOF.pdf. Accessed: 2020-10-16.

Pegasus (2019). PEGASUS Abschlussveranstaltung. https://www.pegasusprojekt.de/de/pegasus-abschlussveranstaltung.

Pfeffer, R., Sax, E., and Schmidt, S. (2019). Realdatenbasierte simulationsgestützte absicherung hochautomatisierter fahrfunktionen. *ATZelektronik*, 14(11):24–29.

Schreiber, M. (2012). *Konzeptionierung und Evaluierung eines Ansatzes zu einer manöverbasierten Fahrzeugführung im Nutzungskontext Autobahnfahrten*. PhD thesis, Technische Universität Darmstadt.

Schuldt, F. (2017). *Ein Beitrag für den methodischen Test von automatisierten Fahrfunktionen mit Hilfe von virtuellen Umgebungen* . PhD thesis, TU Braunschweig.

Shalev-Shwartz, S., Shammah, S., and Shashua, A. (2017). On a formal model of safe and scalable self-driving cars. *CoRR*, abs/1708.06374.

Stellet, J. E., Zofka, M. R., Schumacher, J., Schamm, T., Niewels, F., and Zöllner, J. M. (2015). Testing of advanced driver assistance towards automated driving: A survey and taxonomy on existing approaches and open questions. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 1455–1462.

VVM (2020). V&V Methoden. https://vvm.vdali.de/. Accessed: 2020-10-16.

Wachenfeld, W. and Winner, H. (2016). The release of autonomous vehicles. In *Autonomous driving*, pages 425–449. Springer.

Wuellner, T., Feuerstack, S., and Hahn, A. (2019). Clustering environmental conditions of historical accident data to efficiently generate testing sceneries for maritime systems. In *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, volume 11842 LNCS, pages 349–362. Springer.