Using Traffic Sequence Charts for the Development of HAVs

W Damm, S Kemper, E Möhlmann, T Peikenkamp, A Rakow

Using Traffic Sequence Charts at the Development of HAVs*

W. Damm[1,2], S. Kemper[1], E. Möhlmann[1], T. Peikenkamp[1], A. Rakow[2]

[1] OFFIS - Institute for Information Technology, Escherweg 2, 26121 Oldenburg, Germany

[2] Carl von Ossietzky University of Oldenburg, 26111 Oldenburg, Germany

Regular Paper

Keywords: highly autonomous driving, formal specification languages, testing, type acceptance

1 Introduction

Within the german automotive industry there is a convergence on proposing catalogs of scenarios as a key element for acceptance testing of highly autonomous vehicles (HAV). This approach is currently taken in the german PEGASUS project and the ECSEL project ENABLE-S_3, and will be matured in follow up projects currently under evaluation.

A traffic scenario catalog lists possible traffic evolutions. In this paper, (i) we illustrate how a scenario catalog captured via Traffic Sequence Charts (TSCs) can advantageously accompany the development process of HAVs and (ii) we give an overview of the main features of TSCs. TSCs are a new formal specification language [4, 3], especially tailored for the specification of scenario catalogs.

In particular in the early and late phases of the development, the TSC scenario catalog will have a high impact. In the early phase it can serve as means to structure the application domain of the HAV. There it can be used to distinguish relevant use cases and critical scenarios. When finally the developed system is tested, the TSC scenario catalog specifies the test cases, in which the vehicle under test is examined to show the expected behavior. Such behavior can for instance be, avoiding a near crash situation with a vehicle ahead by changing to the opposite lane if the lane is known to be "sufficiently free". A successful acceptance test against such a scenario catalog thus demonstrates, that the vehicle's perception system (including sensors, sensor fusion, object identification algorithms, possibly including information gained from car2x messages) is able to

- identify all objects in the scenario,
- assess statements about the future evolution of the perceived internal representation of the prevailing traffic situation such as "the opposite lane is sufficiently free", and
- make, based on its perception of the car's dynamic capabilities and prevailing road and weather conditions, informed judgments about the higher safety margin of carrying out a lane change to avoid an obstacle ahead, as compared to, say, a maximum breaking maneuver and staying in the same lane.

This paper proposes a visual, formal specification language for capturing scenarios, and thus addresses key industrial needs for supporting scenario-driven specification and scenario-based acceptance testing for HAVs:

1. How can we at all capture the infinite number of possible traffic situations of an HAV in a finite catalog with finitely represented scenarios?
2. How can we at the same time determine unambiguously whether a particular evolution of traffic situations is covered by a scenario?
3. How can we, thus, decide, whether the reaction of the vehicle-under-test in a given concrete traffic situation is compliant to the scenario catalog (and thus derive test suites for acceptance testing from scenario catalogs)?
4. How can we assess, whether all possible traffic situations are covered by scenarios?

This paper provides answers to challenges 1-3, and provides two necessarily incomplete, complementary proposals of how to deal with challenge 4.

We propose to use TSCs for scenario catalog specification. TSCs are a declarative specification language, where thus every individual chart determines via its formal semantics an infinite set of evolutions of an infinite set of initial traffic situations meeting the constraints on individual traffic situations and their evolution specified in a TSC. Each additional chart thus adds further constraints. TSCs observe entities defined in an ontology of the categories of all types of artifacts which must be observable in real traffic situations currently developed within the OpenSCENARIO[1] initiative of OEMs and Tier1 suppliers. A formal dense time world model implements the entities and types. It thereby captures all relevant physical phenomena of the real world. These phenomena include

- environmental conditions, such as status of road surface, aquaplaning, lighting conditions, and
- weather conditions, both influencing the perception

*This work has been partially conducted within the ENABLE-S3 project that has received funding from the Ecsel Joint Undertaking under grant agreement no 692455. This joint undertaking receives support from the European Union's Horizon 2020 Research and Innovation Programme and Austria, Denmark, Germany, Finland, Czech Republic, Italy, Spain, Portugal, Poland, Ireland, Belgium, France, Netherlands, United Kingdom, Slovakia, and Norway.

[1] http://www.openscenario.org/

and

- maneuvering capabilities of a HAV.

Formally, TSCs visualize first-order real-time temporal logic formulas [4] that refer to the world model. TSCs are tailored to be intuitively comprehensible by visualizing scenarios. Additionally, TSCs have a formal semantics. They are hence tool independent and unambiguously machine-interpretable.

The design of this language benefits from our extensive experience on designing visual specification languages such as Live Sequence Charts and Timing Diagrams, and the underlying technology for automatic generation of observers, test cases, and animations from such visual specification languages.

Outline In the next section, we present an introductory example in order to give an impression of how a TSC specification looks like. In Section 3 we provide a brief overview of the TSC formalism and basic notions. Then in Section 4 we outline how scenario catalogs can be used for the development of HAVs. Section 5 illustrates on a running example how TSCs can be advantageously used to define scenario catalogs and accompany the development process. Before we give our conclusions in Section 7, we use Section 6 to discuss the already achieved and also future work related to TSCs.

2 Introductory Example

A TSC specification consists of (i) a formal world model (for instance specified via probabilistic hybrid automata) defining entities, (ii) a visual specification of scenarios or rules and (iii) a symbol dictionary that defines the link between the visual symbols and the entities of the world model. In this section we give an impression of the visual specification by sketching the development of a collision avoidance maneuver. As initial situation we consider a car that drives on a two-lane highway on the right lane and approaches an obstacle, for instance a non-moving object (e.g. a construction site) or a slowly moving object (e.g. vehicle).

We first structure the space of possible scenarios arising at that situation. There are two basic scenarios: either the car stays on the right lane and collides with the obstacle, or it changes lane and avoids the collision. Figure 1 shows a TSC that models a collision scenario. TSCs are to be read from left to right. So, Figure 1 consists of a header (explained later) followed by three *snapshots* (*sn*s). Sn_1 (the black frame with gray hatching) is the empty snapshot and specifies that we allow anything to happen before sn_2. Sn_2 specifies our initial situation. The car is on the right lane, distance $\leq d_1$ away from the obstacle (the black rectangle).[2] The *distance arrow* $\mapsto$ is used to specify bounds on distances between objects. The third snapshot describes a collision between the car and the obstacle. The hatching on the lanes denotes that—for now—we do not constrain whether there are other objects ("don't care"). Figure 2 specifies the collision-avoidance scenario. Again, the sequence of sn_1, sn_2 expresses that *eventually* sn_2 is reached –the car is $\leq d_1$ away from the obstacle. Before the car gets closer to the obstacle than d_2, it starts changing lane (cf. sn_3). The dashed *somewhere-box* surrounding the car indicates that the car may be anywhere within the box. The whole process of changing onto the left lane is, hence, covered by sn_3. Sn_4 describes that the car has arrived on the left lane and drives past the obstacle. Finally, the last snapshot describes that the car has passed the obstacle. Note that we require snapshots (of a sequence) to contiguously hold during a trajectory. Hence, the somewhere-boxes at sn_3 and sn_4 are an important mean to write succinct specifications.

The headers in Figure 1 and Figure 2 declare that both TSCs are to be understood existentially (quantification mode = exists). That is, we specify that the scenarios of Figure 1 and Figure 2 exist. Existential TSCs allow cataloging observations.

While Figure 1 and Figure 2 describe observations, the TSC in Figure 3 specifies behavior of `ego`, the car under design at a collision avoidance maneuver.

It specifies that if `ego` gets into the situation of sn_1 –`ego` is closer than d_1 to an obstacle– and if the left lane *will be* free for a time duration greater than t (cf. sn_2), then `ego` changes to the left lane and drives past the obstacle. We now introduce the syntactical elements used at Figure 3 step by step. The TSC uses a premise-consequence chart to express "if `ego` [...], then `ego` changes lane [...]". The dashed hexagon contains the premise. Right of it follows the consequence. Our premise consists of two parts: It specifies the initial situation via sn_1 (so the premise expresses "if `ego` is closer than d_1 to the obstacle") and via sn_2 the future (which adds to the premise "and if there will be no car within a distance of d_4 behind `ego` up to d_5 in front of `ego`"). We use the *nowhere-box*, a black frame with diagonal lines, to denote that we rule out the existence of cars within the box. The dimensions of the nowhere-box are specified via the distance arrows anchoring at `ego`'s and the box's borders. The hour glass on top of sn_2 specifies that the left lane will be free for a time duration greater than t. The consequence (sn_3 to sn_5 of Figure 3) is like sn_3 to sn_5 of Figure 2, but with the additional annotation of a ⊜ bar at its top. This annotation specifies how consequence (sn_3 to sn_5) and the future (sn_2 abbreviated by ⊜) synchronize. `ego` has to perform the lane change while the left lane is guaranteed to be free. Thus, the future snapshot sn_2 is concurrent to sn_3 and ends some time during sn_4.

[2] d_1 denotes a predicate specifying the emergency braking distance while, e.g., d_2 at Figure 2 denotes the distance a car needs to get around the obstacle. They depend on the current state of the involved cars and the environment. Due to the lack of space, we refrain from spelling out the distance predicates d_i, $1 \leq i \leq 5$, in this example.

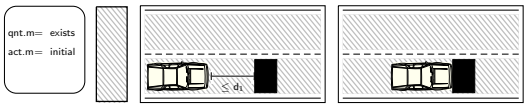

Figure 1: The car collides with the obstacle.

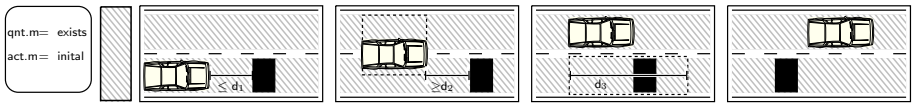

Figure 2: The car avoids the collision with the obstacle.

As the activation mode of the TSC of Figure 3 is always and the quantification mode is all, all trajectories have to satisfy the TSC and if at any time the premise matches ("ego is close to an obstacle and the left lane will be free"), the consequence has to hold ("ego changes to the left lane"). The TSC of Figure 3 specifies a very abstract lane change rule –chosen here for simplicity and ease of the example. A TSC for a concrete implementation will rephrase the future part of the premise of Figure 3 ("there will be free") in terms of sensor readings and on-board prediction so that a sufficiently free corridor is guaranteed.

3 TSCs in A Nutshell

In the following, we briefly introduce the TSC formalism. The interested reader can find the formal semantics of TSC in [3, 4].

Traffic Sequence Charts (TSCs) are a formalism to unambiguously express families of trajectories, respectively trajectory segments, in an intuitive, scenario-based way. To this end they refer to world model and restrict its trajectories by combining two formalisms synergetically: Snapshot Charts (SCs) [3, 4] focus on the visual specification of constraints on the continuous evolution at a scenario, while Live Sequence Charts (LSCs) [2] support the visual specification of communications between the entities. A TSC consists of a header, SC and optionally an LSC part (all TSCs of Section 2 consist only of SCs with headers). A TSC specification consists of a collection of TSCs that have to hold conjunctively.

Snapshot Charts (SCs) can be recursively built of snapshots (see below) composed by concatenation, choice, concurrency and negation (cf. Figure 4). A trajectory segment is contained in the concatenation of two SCs SC_1 and SC_2 if it can be split into two subsegments that are contained in the denotations of SC_1 and SC_2, respectively. Similarly, choice, concurrency and negation are defined. The resulting snapshot graphs can additionally be annotated with timing constraints. Timing constraints can be used (i) to specify the time duration spent at a snapshot subgraph as well as (ii) to synchronize concurrent developments of a scenario. The SC syntax also provides a pattern to express implications, the premise-consequence SC (an example is given at Figure 3). The premise is surrounded by a dashed hexagon and consists of two parts, the *past* and the *future*. Both past and future are specified via snapshot graphs. Right of the premise follows a snapshot graph specifying the consequence. Intuitively, such an SC is satisfied by a trajectory, if the consequence holds in all situations where the past has been observed and the future will occur.

We can translate an SC to a first-order multi-sorted real-time formula, by composing the snapshot formulas according to the graph's structure and annotations.

Snapshots The leaves in the hierarchy of SCs are given by *snapshots*. A *snapshot* describes a traffic situation. It specifies invariant properties that constrain the (otherwise unconstrained) traffic situation by requiring absence or presence of objects of the underlying world model and constraining their states and relations among them. Formally, a snapshot is equivalent to a first-order predicate on (i) the traffic participants (cars, bikes, pedestrians, ...) and (predicates on) their attributes (like position, speed, acceleration, mass, size, color, ...), (ii) the traffic infrastructure elements (roads, lanes, traffic signs, ...) and (predicates on) their attributes, (iii) the relationships among these entities (like distance, visibility to each other, relative speed of two cars, same platoon membership, friction between road and car (wheels), ...).

LSCs are a planned and optional part of a TSC specification. An LSC is used to represent communication protocols between the traffic participants. We refer the reader to [2] here.

World Model An SC (temporal formula) is interpreted wrt. a world model WM, that is a formal model of the "real world". The world model defines classes of objects (cars, bikes, ...) with a set of attributes (position, size, velocity, ...) and the dynamics of moving objects. The world model allows a possibly unbounded number of objects each belonging to one of finitely many classes of, e.g., (probabilistic) hybrid automata. Nevertheless, the world model might reflect for instance physical constraints on the number of objects.

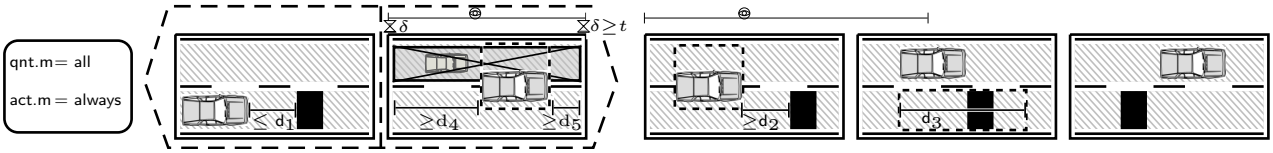

Figure 3: The rule "Change lane to avoid collision, if next lane is free."

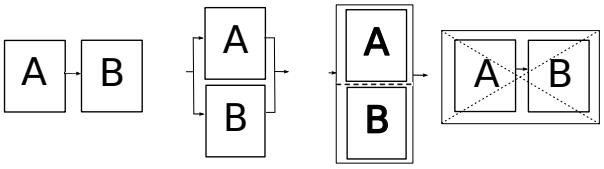

Figure 4: Composing snapshots: (a) Concatenation: First A holds then B (b) Choice: A holds or B holds (c) Concurrency: A holds and B holds (d) Negation: It does not hold, that first A holds and then B holds.

TSCs restrict the overall possible behavior of the world model (to the behavior modeled at the scenarios). The symbols and predicate annotations used in the snapshots refer to objects and their states within the world model. A TSC specification thus defines the set of trajectories that arise from the world model and satisfy the predicates defined via SCs. Having a formal world model provides the basis for a wide range of automated analysis techniques, such as model-checking (e.g. to analytically answer, whether a specified scenario is possible within the world model) and test-case generation (e.g. simulate trajectories for each scenario).

(Spatial) View In the previous section, we have seen an example of an SC describing a scenario via a *spatial view*. For collision freedom spatial dimensions certainly have to be reflected. Hence, at our example all object symbols refer to objects with a spatial dimension. The spatial view encodes predicates that relate the borders of the objects and hence encode relative placement and containment of objects.

In general, TSCs allow visualizing distinct views of a scenario concurrently. Up to now, we mainly concentrated on the spatial view.

Symbol Dictionary The link between the visual symbols used in the snapshots and the world model is defined in a symbol dictionary. It declares the symbols and links the symbols to object(classe)s of the world model. The symbol dictionary also defines the meaning of symbol modifications, i.e., which object features are represented by a symbol modification (e.g., it declares that the modified car symbol,  , represents a car that indicates to its left and not to its right). To link symbol positions to object positions, for each symbol at least one anchor (a distinguished position) is declared at the symbol dictionary and mapped onto an anchor of a corresponding world model object.

Conclusion To summarize, the TSC formalism is a mean to formally specify a system, requirements or scenario catalogs. TSC specifications are conceptually divided into a reference world model and the visual specification of constraints on the world model (cf. Figure 5), their link is defined via the symbol dictionary.

4 Scenario Catalogs for Developing HAVs

In this section, we sketch an exemplary development process of HAVs using scenario catalogs (and following the V-model). In the next section, we will illustrate how in particular TSCs are apt to accompany such a development process.

Within the German Automotive industry there is a convergence on proposing catalogs of scenarios as a key element for acceptance testing of HAVs. Scenarios are derived from (i) collected real world data such as, e.g., traffic accident data bases, field operational tests (FOTs) or natural driving studies (NDSs) or (ii) along an analytical process structuring the design space, e.g., for risk analysis or by use cases, during the development process of an HAV.

Building a Scenario Catalog In the sequel, we describe how a scenario catalog is built following mainly [13, 12]. To this end we outline the steps to build a scenario catalog: scenario screening, clustering, identification of entities, world model definition and the definition abstract scenarios.

The first step of building a scenario catalog is screening data from risk analysis, accident data bases (like GIDAS (German In-Depth Accident Study)) and virtual and real long term testing etc. The German PEGASUS identifies a list of characteristics of critical situations (like time to collision). For each of these, concrete scenarios will be produced either by simulation or by field tests, so that they get represented as entries of the scenario catalog.

At a clustering step, then scenario clusters are built of the concrete scenarios that are similar wrt context and evolution. Relevant parameters, influencing factors and the characteristics of critical situations (like time to collision) are identified at this step.

After that, the concrete scenarios will be abstracted to obtain entries for the catalog and a world model is defined that consists of the relevant objects and that reflects relevant real world phenomena. Based on this world model, abstract scenarios are derived for the

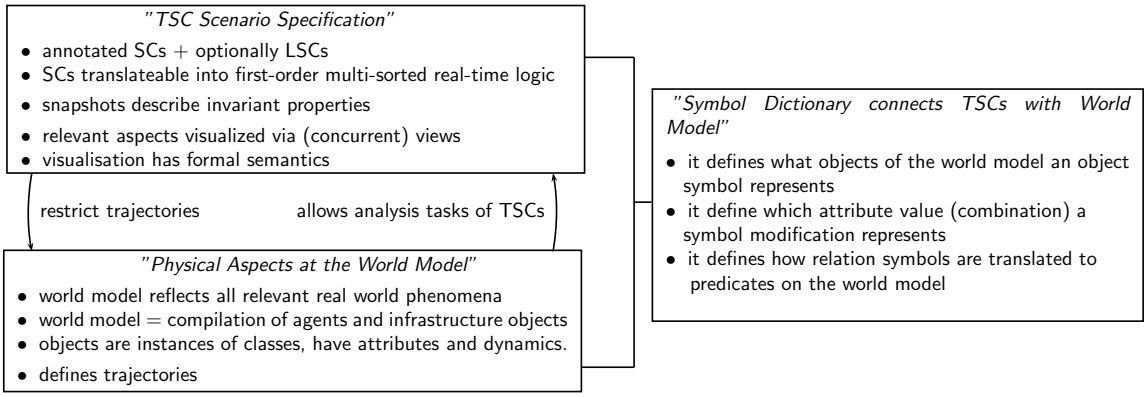

Figure 5: TSCs in a nutshell

collected concrete scenario(cluster)s.

At this step the right level of abstraction has to be ensured. Given we generalize a concrete scenario s_c to an abstract scenario s_g, we will have to investigate whether all concretizations of s_g still reflect the relevant criteria of s_c. If for instance only some concretization of an abstracted critical scenario lead to a critical situation while other concretization do not, the scenario specification has to be refined yielding more determinate abstract scenarios that precisely characterize critical situations.

The resulting scenario catalog consists of selected scenarios of the world model that capture the core of the scenario space. As well, they cover peculiar, relevant and critical scenarios.

We envision to this end, that a completion procedure is performed on the scenarios in the catalog. This procedure checks for completeness of cataloged scenarios. The completion procedure is subject of current research efforts and is currently in the process of being investigated deeply. The basic idea is to abstract from a concrete scenario representing a scenario cluster (e.g., critical scenarios). To abstract from the concrete scenario, constraints are weakened (e.g., instead of a certain number of pedestrians, the number has to be within a certain interval). The level of abstraction has to be chosen so that the resulting abstract scenario still can only be refined to realistic concrete scenarios.

We then check whether all evolutions of the world model are covered by the scenario catalog, which is a formal analysis task.

It remains to check that the information within the world model is represented sufficiently and correctly. The question of determining an appropriate world model perimeter is an important research direction, which has been approached in [1].

We envision that such a scenario catalog is maintained by a public accredited trust center as described in [11, 12] and agreed upon by manufacturers as to provide the relevant scenarios for type acceptance.

Scenarios at the Development Process At Figure 6 we illustrate the design process for a safety concept following the V-model and using such a catalog of scenarios.

As a first step of the development process the system context is analyzed and a world model reflecting the relevant phenomena and entities is defined (WM). At this step, the existing scenario catalog –built in a step referred to as (Sc)– is taken into account. The steps (WM+Sc) provide the basis to define homology criteria and use cases for the system under design.

According to our development process, functional and non-functional requirements (Rq) of the system under design are defined and then refined until they are concrete enough for implementation (Rq+Im). The requirement specification is strongly influenced by the scenario space, that reflects the necessities, risks and optimization potentials of the system within the targeted context.

Part of the design is a safety concept to combat the identified risks. A risk identification process is started already with the identification of use cases and accompanies the development process.

In the phase (VA), verification and analysis, analysis methods such as virtual testing or model checking, are triggered on different levels of the abstract system and finally on the realized system. The goal of (VA) is to ensure that the realized system fulfills the requirements of (Rq+Im). The realized system is tested component-wise with increasing system complexity, and finally acceptance is tested.

At verification and analysis, the scenario catalog provides the system context and targeted uses cases, based on which test runs can be generated.

Conclusion Scenarios catalogs allow collecting use cases, critical situations and peculiarities that can drive the development process. They help to structure the design space early at the development process and help to identify system borders. Next they provide a reference test environment that supports virtual tests already at abstract system specifications up to the definition of concrete tests for acceptance testing.

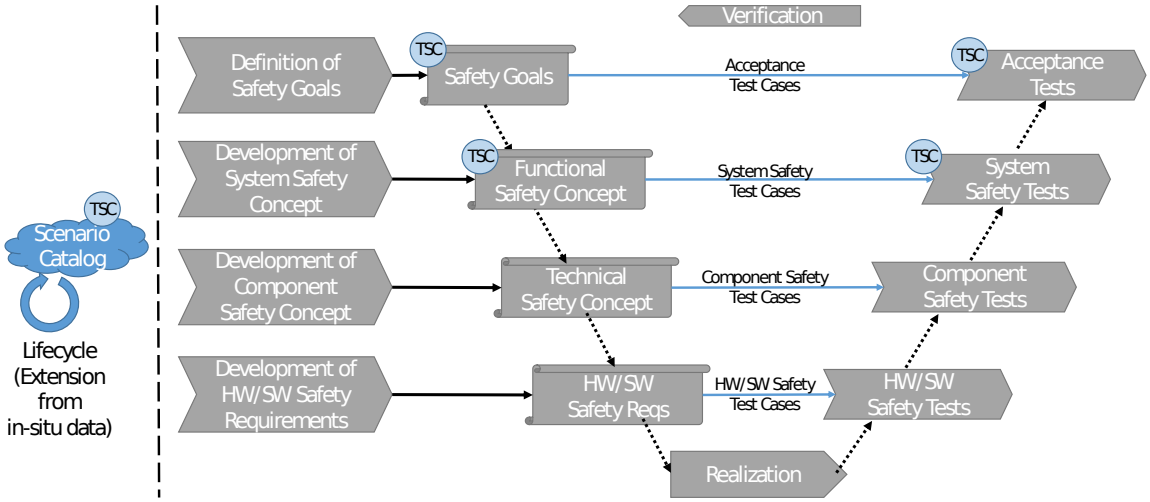

Figure 6: The V-Model for a Safety Concept. TSCs are used at the early and late stages of development.

5 TSCs for Developing HAVs

As outlined in the previous section, scenarios provide early in the design process information based on which requirements of the autonomous driving function (ADF) can be derived, and furthermore they accompany the design process by providing a reference for testing and validation. Along the development process scenarios will be considered at different levels of detail and underlie a refinement process that is influenced by the characteristics of the implemented system and the analysis at hand.

TSCs are a formalism that allows to formally represent scenarios at different levels of abstraction. TSCs can, hence, accompany the development process of HAVs, supporting refinement, maintenance and analysis of scenario catalogs during the development of HAVs by formal methods.

They allow to specify a scenario (bundle) as a sequence of snapshots. A TSC specification of scenarios is hence visual and yet formal. The scenario specification refers to a formal world model, which is an important product/artifact of building a scenario catalog [13]. TSCs distinguish between the world model definition and the specification of the scenarios (WM+Sc).

Furthermore, TSCs can be used to specify requirements (Rq). These can successively be made more detailed up to rules that capture the preconditions of implementable specifications (Im). So TSCs allow to accompany activities at the left arm of the V-model, from requirements towards implementation.

Further, TSCs formally capture scenarios and allow describing abstract as well as concrete scenarios. A TSC scenario catalog can easily be used for simulation and testing wrt the requirements. So also at the right arm of the V—verification and analysis (VA)— TSCs are nicely applicable. TSCs are especially well suited for virtual tests.

In the following, we give simple yet representative TSC examples to illustrate how TSCs can advantageously be used. TSCs are a powerful formalism, because it combines the world model formalism with the first-order temporal logic of the SCs. Their key feature is to provide a mean to specify formally yet visually. For this reason, we see their application especially at the early phases of the development process and at the late where experts of different domains cooperate.

(WM+Sc) According to the previous section, the reference scenarios are defined wrt a world model WM that identifies the relevant objects and real world phenomena. The TSC formalism is based on such a WM. For TSCs WM is supposed to overapproximate every possible—and relevant—behavior of the real world. That means everything that is relevant and possible within the real world has to be possible at WM as well.

Example 1. *Let us for a toy example consider a simple world model that defines the attributes and dynamics of roads, lanes and cars. We model these objects for instance via hybrid automata classes. An automata instance of class* `Car` *defines for example the steering and acceleration capabilities of a car. The control of the car is not subject of the world model. To overapproximate every possible behavior, a car hence behaves non-deterministically. Here we leave it to SCs to restrict the behavior appropriately and thereby making assumptions on the nominal and exceptional behavior explicit.* □

The list of reference scenarios can be expressed via existential TSCs. Note, that a single TSC usually represents infinitely many trajectories of WM, that is all trajectories that satisfy the requirements characterizing the scenario variation.

Example 2. *Consider for instance a single snapshot as in Figure 7 to see how a TSC represents an infinite number of concrete scenarios.*

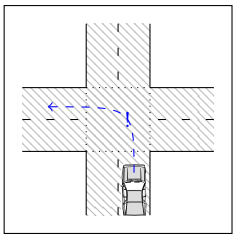

Figure 7: Car with the intent to turn left at the intersection

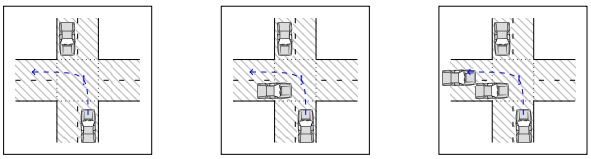

Figure 8: Exemplary instances of the initial scenario

For a TSC specification a symbol dictionary formally documents the (used) formal meaning of symbols. For our toy example we use  *to represent* `Car` *objects of the world model. We represent the intended path a car wants to drive via a blue dashed arrow. We represent a lane with no adjacent lane below but with an adjacent lane above via*  *Its norm traffic direction is such that the dashed line is to the left hand side of the vehicle.*

At Figure 7 a car is at one of the four lanes of the intersection and intends to turn left. Via this snapshot we specify that the car is in front of the crossing and that the considered crossings have two roads with two lanes, but we do not, e.g., specify how far the car is away from the crossing or how fast it is. Further we do not constrain the number or positions of other traffic participants. Every compilation of other traffic participants, that is possible at the world model, satisfies this snapshot.

The snapshot hence represents infinitely many situations within the space of possibilities that is spanned by the different attribute values of the car that wants to do a left turn and by the possible other traffic participants. □

Having a formal definition of the scenarios and a formal definition of the world model allows to formally check whether the scenario is actually possible within the world model.

Moreover, using TSCs to specify a scenario catalog, allows to describe those scenarios detailedly that are considered to be relevant and represents others more abstractly. We can hence cover all behaviors of the world model via scenarios.

Example 3. *At Figure 9 we captured all scenarios where the car eventually successfully performs a left turn maneuver. Again* eventually *is expressed by the* `True` *snapshot, sn_1 of the SC. At our initial situation the car is in front of the intersection and intends to turn left (sn_2), then the car drives onto the intersection (sn_3), turns left and finally arrives at the intended destination (cf. sn_5). Similarly, Figure 10 captures a scenario starting at our initial situation, but then every behavior is allowed that does not accord not the left turn maneuver of Figure 9. At Figure 10 we hence described a scenario where the car* does not *eventually successfully performs the left turn.* □

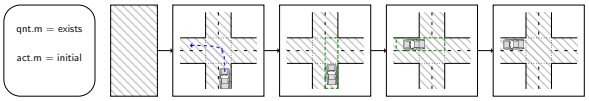

Figure 9: Successful Left Turn

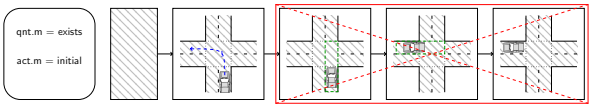

Figure 10: Failed Left Turn

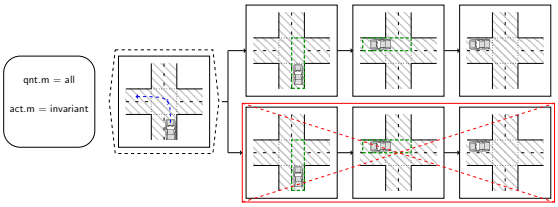

Figure 11: Fully covered left-turn scenario

We advocate to maintain scenario catalogs that completely cover all WM behavior. TSCs can express that a set of existential scenarios covers completely the world model. To this end, we can for instance use a universal TSC that disjunctively lists the scenarios that together cover the world model behavior.

Example 4. *At Figure 11 we captured in a universal invariant TSC rule with the premise consequence pattern that Figure 9 and Figure 10 cover the complete scenario space at the initial situation of Figure 7. The SC in Figure 11 specifies that whenever we are at the initial situation, then the SC of Figure 9 or the SC of Figure 10 is satisfied. So, a scenario catalog with the TSCs of Figure 9, Figure 10 and Figure 11 expresses that there are a successful and a failed left turn and that from the initial situation, these two are the only possibilities.* □

(Rq+Im) TSCs not only allow to list scenarios, they can also be used to capture requirements (Rq). These requirements can be specified (i) in a future dependent and abstract way, which seems natural for a requirement early in the design process as well as (ii) in a more detailed way, up to pure "past implies a future consequence" rules.

Example 5. *At Figure 3 we have already presented an example how TSCs can be used to specify requirements. The TSC specifies (R1): "the system is required to change lane, if it meets an obstacle at its lane and the next lane is free". (R1) is quite intuitively saying "change lane, if possible". It requests the car to change lane, if the car* will be able to change *onto the next lane and* no other car *will be there* closer than the safety distance. *The requirement (R1) is overly strong, since the car cannot foresee the future, but has to predict based on the currently available information, whether next lane will be free. TSCs allow specifying abstract requirements like (R1). A more detailed version of "change lane, if possible" is given at Figure 12.*

Figure 12 specifies that `ego` *is required to change lane if there is no other car at the next lane within distance d_6 to its rear and within distance d_7 ahead. d_6 and d_7 abbreviate functions on the* `ego` *car and the car to the back or respectively front that takes the relative speed into account to conservatively determine a distance great enough to guarantee that the obstacle avoidance maneuver can be performed safely, whatever other cars may do. Note that hence d_6 and d_7 will usually be greater than safety distances d_4 and d_5.* □

Requirements can successively be made more detailedly. While at (Rq), TSCs of the "past and future imply a future consequence" pattern seem most appropriate to capture the precondition, TSCs of the pattern "past implies a future consequence" are expected to be used to specify requirements close to implementation (Im), where the precondition refers to information available at the present.

Hence, from (Rq) to (Im) the future (oracle) is replaced by collected information (perceivable via sensors) that allow to predict that the formerly oracled future is guaranteed. This step of identifying sufficient sensable information is vital for the implementation of the system in hardware. Since both, the initial oracle pattern and the implementation-close "past implies the future" pattern are formal and relate to the same world model, we can reason about coverage and criticality.

Example 6. *At example 5, (R1) is specified via a TSC "past and future imply a future consequence" rule, whereas (R2) is specified via a "the past implies a future consequence" rule. Applying formal methods on the TSC specifications allows to analyze the relation of (R1) and (R2). For instance, "Are there trajectories for which a lane change is required by (R1) but not by (R2)?", "Is a lane change performed according to (R2), that is not possible according to (R1)" are questions that can be analyzed with the help of model checking.* □

(VA) Given the system has already been implemented—fully or in parts—, simulation and testing will be performed to verify that the realized system fulfills the specification.

TSCs conceptually separate the world model and the specification of controller. The constraints on the controller under design are captured via SCs. For a virtual test, a play out [10] of environment behavior is basically determined by the world model. At a test run it has to be monitored whether the implemented controller satisfies the SCs.

Given a FOT according to a scenario is performed, the observed behavior is of discrete and continuous nature (e.g., the discrete controller and continuous car dynamics). Formally, we ask whether the observed behavior is possible within the world model and satisfies the SCs. This is not decidable in general. Nevertheless, since the observations will only approximate the reality (due to limited observation accuracies and resources for storing), we rather ask whether the sequence of observations is robustly (due to the jitter in accuracies) possible. To this end we plan to employ the results of [7, 8, 5, 19] to analytically answer this questions in a broad range of settings.

6 Achievements and Next Steps

In the following, we summarize the main features of TSCs and outline the next steps of the development of the TSC formalism for developing HAVs.

6.1 Benefits of Using TSCs

To summarize, TSCs are a specification formalism that allows us to accompany the development process of HAVs from the specification of a scenario catalog, over the requirement specification, to testing and validation. Since TSCs have a formal semantics, they open the door to application of formal methods.

A TSCs specification is conceptually divided into a world model specification and SCs that capture scenarios and requirements. The world model of TSCs can be seen as the test environment at the analysis and validation phase. Since the testing of ADFs cannot be done retrospectively [13] but has to accompany the development process, TSCs are, hence, in particular suited as a formalism.

Moreover, TSCs are a visual specification formalism allowing specifications at different levels of abstraction. The formalism allows the introduction of customized sets of symbols. Although we focused here only on the spatial view, different aspects of a behavior can be visualized at specialized views. TSCs aim to accelerate the discussions among experts of different domains by providing an intuitive, visual and yet formal specification language.

Using TSCs to specify a scenario catalog, we can (a) list required scenarios as existential TSCs and also (2) express that a set of scenarios covers all possible evolutions via universal TSCs. Whereas (1), *listing scenarios*, is the minimum requirement for a formalism to specify scenario catalogs, (2), *expressing complete coverage*, is a distinguishing features of TSCs. A scenario catalog that covers all possible evolutions of the

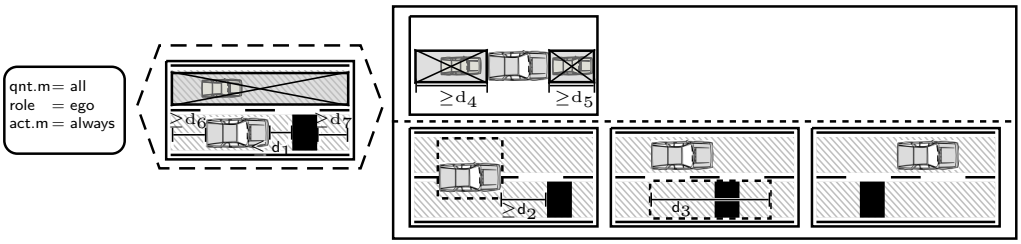

Figure 12: Rule: Change lane to avoid collision, if next lane is free.

world model—the relevant more detailedly, the others more abstractly— helps not to miss scenarios.

We see TSCs mostly at the early stages of the V-model and at the end. In particular, they are suited to support scenario based testing. Certainly testing inspects the system only in a finite number of test runs, while for HAVs usually infinitely many evolutions are possible. This is in contrast to methods like model checking, which are used for instance to verify hardware. Model checking exhaustively examines the system's functionality. For HAVs there are two main problems: (WG) the gap between model and real world, and (CC) the complexity of the verification task. Even if the formal model of a system is verified to be correct, the question remains whether the system is correctly modeled. HAVs, due to the continuous dynamics, are usually too complex (and often undecidable) for practical use of model checking of the complete system. Thus testing is irreplaceable at the system level, even if formal methods prove parts of the system to be correct.

Scenario catalogs allow us to focus our attention on relevant and critical scenarios. Especially TSC catalogs with complete coverage can hence ensure that the test efforts are invested where necessary and allow to relieve the test effort for other scenarios.

6.2 Next Steps

In Section 4 and Section 5 we sketched how TSCs may accompany the development of HAVs. We mentioned that one major benefit of TSCs is the application of formal methods. Conceptually, TSCs allow rich specifications at the SCs and the world model and are, hence, in general undecidable [14]. So we cannot expect that a model checker will terminate in all cases applied to check for instance whether a scenario catalog covers all world model behavior. Instead of the full model, simplified models can be examined, which often provides valuable hints regarding the full model's properties. As real world systems with limited sensor precision also allow to use a notion of robust satisfaction, we might profit from the results of [7, 8, 5, 19] on decidability for robust satisfaction. In the future we will look into efficient automatic methods for checking consistency and completeness relative to a given world model.

A TSC specification is based on a world model. The dynamic models of other traffic participants, reflecting observed behaviors in real traffic situations, calls for a formalism like probabilistic hybrid automata. The current TSC formalism can be based on such world models, but does not yet allow to specify probability properties on the set of alternative evolutions, as for instance "the left turn has to be successful in at least 99, 9% of the situations". We plan to extend the TSC formalism to this point. In [9], we have already shown that a preliminary version of TSCs is suitable for testing and statistical inference over the satisfaction of safety requirements in complex traffic scenarios. The TSC formalism is based on a world model. Although, using TSCs can help to detect inconsistencies within the world model, TSCs are not a tool to build a world model. The question of how a suitable world model can be obtained, poses an important future research challenge, which is subject of future projects.

We claim that another major benefit of TSCs is succinctness and intuitiveness of the resulting specifications. The intuitiveness of TSCs depends on a careful and contemplative design of symbols. For instance we expect that the symbol  is usually attributed with properties like, "fits onto a lane", "can accelerate to at most 300km/h". So representing a general vehicle, a van, a big jeep, a racing car or truck would disguise problematic situations. Since TSCs are based on a formal semantics such misconceptions can be detected by simulation runs at the specification level. Accomplishing an intuitive visualization is a major research challenge. We plan to conduct experimental evaluation of the intuitiveness of TSCs and more over we will work with traffic psychologists for revisiting the language design, in parallel to testing the approach in non-trivial sample use cases.

All symbols used in an SC—except the pins, somewhere- and nowhere boxes—refer to world model objects. So the symbol set is flexibly adaptable to the user's needs. The link between symbols and world model is defined in the symbol dictionary. Note that, in order to use TSCs to specify the scenario catalog of a public accredited trust center, as outlined at page 5, a consent on the symbol dictionary and the world model allows each company to adopt the visual symbols to its own needs without changing the semantics, e.g., a car of the world model can be represented by

 or  .

6.3 TSCs and Scenario Languages

The importance of studying scenarios for the development process has long been recognized—not only in the transportation domain.

For instance, there are commercial tools that allow defining virtual scenarios for testing. Also in academia, scenario languages for driving simulations have been discussed, e.g., [17, 20, 24]. These languages aim to orchestrate activities to define a traffic evolution within a virtual world.

In contrast to the above scenario languages for driving simulations, TSCs do not aim to define a single traffic evolution but to specify scenario bundles, visualizing the constraints defining the scenario (bundle). A TSC scenario usually represents infinitely many concrete traffic evolutions (cf. Example 2) due to its declarative nature. Usually only the play-out [10] of a TSC represents a single traffic evolution. Being able to specify bundles of scenarios, TSCs can be used to structure the scenario space. Moreover, they allow expressing, e.g., complete coverage of all possible scenarios of the world model (cf. Example 3).

Recently, the merits of a tool independent definition of scenarios got into the focus of academia [16, 6] and industry. A tool independent approach is made by OpenSCENARIO [23]. OpenSCENARIO [23] "is an open file format for the description of dynamic contents in driving simulation applications" [23]. Together with OpenDRIVE [21] and OpenCRG [22] they form a complementary set of exchange formats. These formats aim at becoming a standard. There the static road networks are basically described as graphs of lanes labeled with geometric shapes. Dynamic content is described as a storyboard with trigger-action pairs.

In contrast to OpenSCENARIO, TSCs are based on a formal semantics, consequently they certainly are tool independent and unambiguously define scenario (bundles).

Furthermore, TSCs—other than the above formalisms—*visualizes* scenario *specifications*. In this respect, TSCs are close to Multi Lane Spatial Logic (MLSL). MLSL is a spatial interval logic that was introduced in [15] to simplify reasoning about safety of road traffic by abstracting from the car dynamics. TSCs like MLSL have a formal visual semantics to represent (abstract) traffic evolutions. In contrast to MLSL, TSCs do not determine the level of abstraction, but leave it open to the specification of the world model and the symbol dictionary.

6.4 Tools for TSCs

TSCs can be nicely used to automatically generate simulations like for LSCs [10]. A simulation is a concretisation of an abstract scenario of the scenario catalog where concrete valuations have been assigned to the quantified variables. Also TSCs can be used to generate requirement monitors, e.g., for functional (safety) requirements. Requirement monitors, when attached to a simulation, can be used to check the satisfaction of requirements. Such a procedure has already been successfully established using timed-automata in the case of LSCs [18].

Additionally, we envision tool support for inspecting and editing TSCs. This will allow engineers to make use of TSCs during the development and refinement of e.g. a safety concept and to use the visualization during inspection of possible violations.

An advantage of TSCs in this context is that the visualization of virtual runs can be close to the visualization of the scenario specification. We expect that this feature eases debugging.

Moreover, representing concrete scenarios in the OpenSCENARIO format allows us benefit from any OpenSCENARIO-based tool such as simulators.

We have already developed an internal prototypical editor for TSCs. We envision, that TSCs can nicely be used within an integrated development environment. There different visualizations can be switched on and off and formal analysis techniques can be triggered and their results are visualized in terms of TSCs.

7 Conclusion

In this paper, we have introduced TSCs and illustrated their usability. TSCs are a visual specification language based on a formal semantics targeting to describe the system in scenarios. As a consequence TSCs support testing based on scenarios well but also enable the use of formal methods to analyze TSC specifications. Especially at the early and late phases, the visualization of the formal specification provides an appealing mean to involve the various stakeholders of different backgrounds.

References

[1] W. Damm and B. Finkbeiner. Does it pay to extend the perimeter of a world model? In *FM 2011: Formal Methods: 17th International Symposium on Formal Methods, Limerick, Ireland, June 20-24, 2011. Proceedings*, pages 12–26, 2011.

[2] W. Damm and D. Harel. LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[3] W. Damm, S. Kemper, E. Möhlmann, T. Peikenkamp, and A. Rakow. Traffic sequence charts - from visualization to semantics. Reports of SFB/TR 14 AVACS 117, SFB/TR 14 AVACS, 10 2017.

[4] W. Damm, E. Möhlmann, T. Peikenkamp, and A. Rakow. A formal semantics for traffic sequence charts. In *Festschrift in honor of Edmund A. Lee*, 2017.

[5] W. Damm, G. Pinto, and S. Ratschan. Guaranteed termination in the verification of ltl properties of non-linear robust discrete time hybrid systems. In *Automated Technology for Verification and Analysis: Third International Symposium, ATVA 2005, Taipei,*

Taiwan, October 4-7, 2005. Proceedings, pages 99–113. Springer, 2005.

[6] M. C. S. Filho and J. J. P. C. Rodrigues. Human readable scenario specification for automated creation of simulations on cloudsim. In *Internet of Vehicles – Technologies and Services: First International Conference, IOV, Beijing, China, September 1-3, 2014. Proceedings*, pages 345–356, 2014.

[7] M. Fränzle. Analysis of hybrid systems: An ounce of realism can save an infinity of states. In *Computer Science Logic: 13th International Workshop, CSL'99 8th Annual Conference of the EACSL Madrid, Spain, September 20–25, 1999 Proceedings*, pages 126–139, 1999.

[8] M. Fränzle. What will be eventually true of polynomial hybrid automata? In *Theoretical Aspects of Computer Software: 4th International Symposium, TACS 2001 Sendai, Japan, October 29–31, 2001 Proceedings*, pages 340–359, 2001.

[9] S. Gerwinn, E. Möhlmann, and A. Sieper. Statistical model checking for scenario-based verification of adas. In *Control Strategies for Advanced Driver Assistance Systems and Autonomous Driving Functions*, 2017. to appear.

[10] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine*. Springer, 2003.

[11] P. Heidel and W. Damm. Highly Automated Systems: Test, Safety, and Development Processes, management summary. http://www.safetrans-de.org/en. accessed on 20.10.2017.

[12] P. Heidel and W. Damm. Hochautomatisierte Systeme: Testen, Safety und Entwicklungsprozesse, Roadmap. http://www.safetrans-de.org/de/Aktivitaeten/Roadmapping.php. accessed on 16.11.2017.

[13] T. Helmer, K. Kompaß, L. Wang, T. Kühbeck, and R. Kates. Safety performance assessment of assisted and automated driving in traffic: Simulation as knowledge synthesis. In *Automated Driving: Safer and More Efficient Future Driving*, pages 473–494. Springer, 2017.

[14] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.

[15] M. Hilscher, S. Linker, E.-R. Olderog, and A. P. Ravn. An abstract model for proving safety of multi-lane traffic manoeuvres. In *Formal Methods and Software Engineering, ICFEM 2011, Durham, UK, Proceedings*, volume 6991 of *Lecture Notes in Computer Science*, pages 404–419. Springer, 2011.

[16] S. Jafer, B. Chhaya, U. Durak, and T. Gerlach. Formal scenario definition language for aviation: Aircraft landing case study. In *AIAA Modeling and Simulation Technologies Conference*, 2016.

[17] J. Kearney, P. Willemsen, S. Donikian, and F. Devillers. Scenario languages for driving simulation. In *Driving Simulation Conference, DSC'99*, pages 377–393, 1999. [Online]. Available: www.cs.uiowa.edu/ kearney/pubs/dsc99_kearney.pdf.

[18] S. Li, S. Balaguer, A. David, K. G. Larsen, B. Nielsen, and S. Pusinskas. Scenario-based verification of real-time systems using uppaal. *Formal Methods in System Design*, 37(2):200–264, Dec 2010.

[19] S. Ratschan. Safety verification of non-linear hybrid systems is quasi-decidable. *Formal Methods in System Design*, 44(1):71–90, 2014.

[20] P. Suresh and R. Mourant. A tile manager for deploying scenarios in virtual driving environments. In *Driving Simulation Conference*, pages 21–29, 2005.

[21] VIRES Simulationstechnologie GmbH. OpenDRIVE, 2015. http://www.opendrive.org, accessed: 2017-11-10.

[22] VIRES Simulationstechnologie GmbH. OpenCRG, 2016. http://www.opencrg.org, accessed: 2017-11-10.

[23] VIRES Simulationstechnologie GmbH. OpenSCENARIO - bringing content to the road. 2nd OpenSCENARIO Meeting, June 29th, 2016. http://www.openscenario.org, accessed: 2017-11-10.

[24] I. H. C. Wassink, E. M. A. G. van Dijk, J. Zwiers, A. Nijholt, J. Kuipers, and A. O. Brugman. *Bringing Hollywood to the Driving School: Dynamic Scenario Generation in Simulations and Games*, pages 288–292. Springer, 2005.