EPSAAV: An Extensible Platform for Safety Analysis of Autonomous Vehicles

Joelle Abou Faysal, Nour Zalmai, Ankica Barisic, Frédéric Mallet

EPSAAV: An Extensible Platform for Safety Analysis of Autonomous Vehicles

Joelle Abou Faysal[1], Nour Zalmai[2], Ankica Barisic[3], and Frederic Mallet[4]

[1] Renault Software Labs (RSL), Université Cote d'Azur, Cnrs, Inria, I3S
Sophia Antipolis, France
joelle.abou-faysal@etu.univ-cotedazur.fr
[2] Renault Software Labs (RSL)
Sophia Antipolis, France
nour.zalmai@renault.com
[3] Université Cote d'Azur, Cnrs, Inria, I3S
Sophia Antipolis, France
ankica.barisic@univ-cotedazur.fr
[4] Université Cote d'Azur, Cnrs, Inria, I3S
Sophia Antipolis, France
frederic.mallet@univ-cotedazur.fr

Abstract. In this paper, a novel model related to the safety of autonomous vehicles (AVs) is presented. A simulation platform is designed to analyze the environment and the trajectory of AVs within a given Operational Design Domain (ODD). This platform relies on model-based systems and includes the environment model, safety rules and their priorities, and execution scenarios. The goal is to create a simulation environment that enables safety experts to detect rule breaches by analyzing problems at run-time using generated monitors. Therefore, this platform will help to reevaluate the existing rules in two ways: either by reconsidering rule priorities or by proposing new rules to be integrated into the existing safety model. The validation and verification of the generated rules will follow a process based on the history of the executed scenarios. All the aforementioned work is carried out by using the GEMOC initiative tool to coordinate models using logical time.

Keywords: Autonomous cars · Safety · Model development and verification · Testing and simulating · Formal methods · Rule-Based Planner.

1 Introduction

With the advent of autonomous vehicles (AVs), engineers have witnessed several deaths and accidents that raise troubling questions about the safety of such vehicles and the current limitations of the technology. Automobile manufacturers are facing an increasing demand for reliable handling of these autonomous vehicles. It is, therefore, crucial to provide concrete evidence to measure how safe AVs are. Safety is associated with guarantees that certain conditions must be met when contingencies arise. If not, the behavior must be adjusted accordingly. In

dynamic environments, real-time safety checks of the planned trajectories and the driver are sometimes lacking to ensure that the trajectories obtained from trip planning are safe. Nowadays, 90% of car accidents are caused by human errors, such as poor judgment due to a failure of human perception or the driver's lack of attention [1].
It is important to show that the autonomous vehicle makes better decisions than a human driver under all circumstances. One proposal to assess operational safety is to test-drive AVs in real traffic and observe their behavior. As logical as this may sound, it is not efficient because it poses significant risks to the environment. We can look at the Uber accident [2] where the driver was doing mileage tests in Arizona. Several things went wrong while the driver was not paying attention to the road: there was no real-time driver safety check, and Uber did not have the resources in the vehicle to assess the driver's attention.
Mileage testing requires a lot of time and testing to ensure safety, and that doesn't make them scalable [3]. The dilemma of how many miles autonomous vehicles would need to be driven to demonstrate safety leads to an answer of 8 billion miles in 400 years with a fleet of 100 vehicles driving all the time [3]. This is somehow impossible and unachievable as several works argue that AVs cannot express safety capabilities based on road tests alone [4]-[5]. Therefore, simulation testing becomes a promising alternative [6]. Moreover, since AVs are currently tested only in limited Operational Design Domains (ODDs), they are not exposed to the wide variety of driving conditions and road user behaviors. Thus, we need rigorous and exhaustive approaches to ensure operational safety.
In the typical software development lifecycle, we try to translate needs into natural language requirements and produce code that meets them. However, it is only after the code has been produced and tested that we discover errors between what was intended and what was built. The same problems occur when testing scenarios. To avoid these problems, Model-Based System Engineering (MBSE) approaches offer a promising alternative by enabling domain models as a communication medium between engineers instead of text documents and supporting automatic code generation. Many researchers now agree that MBSE approaches are a solution for security verification and are seen as an answer to these challenges [7]. MBSE helps to deal with the increasing size and complexity of systems [8], and allows to reason on the model before deployment, Formal modeling and verification in automotive systems are essential to provide sufficient guarantees, especially in the case of dangerous and unforeseen situations.

In this paper, an MBSE approach is proposed based on a simulation environment where a safety rule monitor is specified and generated. It helps us to define types of breaches on an AV trajectory. The main goal is to develop a resilient and safe driver monitoring system that continues to operate safely as long as the assumptions about the environment are satisfied. To sum up, the proposed approach brings a light overview of the **four goals** pursued: (1) formal modeling of safety rules with their priorities alongside the assumptions and the environment on which they rely, (2) monitoring the behavior of the AV at run-time according to these rules and assumptions, (3) triggering alarms to the driver or safety en-

gineer when the behavior of the AV violates some of the safety rules or some of the assumptions and (4) checking and verifying incoherences between the rules and producing the output of the functionality blocs in the planned trajectory.

This paper is organized as follows. Section 2 discusses related work. Section 3 presents the technologies used by the approach, the illustrative real-life use case, and the Extensible Platform for Safety Analysis of Autonomous Vehicles (EPSAAV) approach to specifying the environment and the prioritized rules. It also details the approach to generate an execution policy from the behavior specification. Finally, Section 4 draws conclusions and details some possible future work.

2 Related Work

Due to the drawbacks of not having safety verification, many existing solutions have been proposed. Formal verification approaches are considered as candidates to reduce the intractability of empirical validation [9–11]. One of these approaches is called Responsibility Sensitive Safety (RSS) [5] and has largely inspired our proposal. The authors have developed a white-box verification approach for interpreting safety requirements. They have developed redundant sensing systems with a complex environment. Their interpretable, mathematical model does not guarantee that a vehicle will not be involved in an accident. The limitation of the model also lies in the fact that assumptions and driving policies are taken explicitly to guarantee the safety of the vehicle. They assume that the data is reliable and independent of environmental conditions. The reality is different and there are a lot of possible scenarios depending on the weather, sensor reliability, and many other parameters. Our EPSAAV framework allows circumventing this difficulty by providing the possibility to add measures for safety and environmental conditions. Conceptually, we view our framework as a complementary safety assessment to [5]. The modularity of our simulation framework easily allows us to include more sophisticated notions of safety, such as temporal-logical specifications or implementations of RSS distances mathematical formulas.

In [12], the authors proposed a modeling and simulation environment called STIMULUS in which they developed and tested requirements in real-time, revealing inconsistencies and ambiguities. Even though textual requirements look simple and reasonable, they contain inconsistencies and ambiguities that lead to undesirable behavior and contradictions. Our framework will help detect these inconsistencies and help the experts to improve the system. Moreover, the tool used in [12] is proprietary and requires a license. However, what the world of autonomous driving needs is an accessible, open-source standard with formal semantics. On the other hand, Measurable Scenario Description Language (M-SDL), created by Foretellix [13], is an open-source solution that unfortunately does not include support for scenario description. Despite the open release, the

modeling and simulation tools are proprietary solutions, and there is no way to specify the environmental properties (sampling rate, accuracy) to be captured.

3 Monitoring Platform Specification

Our approach provides a simulation platform that analyzes the environment and trajectory of autonomous vehicles within an operational domain design (ODD). It also aims to help safety designers and experts to verify safety breaches and problems. The information provided by hardware components represents our software architecture necessary for driving. The proposed Extensible Platform for Safety Analysis of Autonomous Vehicles (EPSAAV) is considered as a testing and verification tool to identify fault types and trigger alerts to the user. It is divided into **three parts**: (1) abstract description of EPSAAV (Fig. 4), (2) the autonomous vehicle environment and formal rules with priorities, and (3) generation of monitors to analyze rule violations and inconsistencies.

3.1 Technologies used by the approach

Limited expressiveness makes it harder to express wrong things and facilitates comprehension. Domain-Specific Languages (DSL) can solve some of these problems by raising the level of abstraction closer to the problem domain, rather than code [14]. Our framework uses GEMOC, an open-source tool based on Eclipse. This modeling environment focuses on design and validation problems in complex systems. One of them is enabling the evolution or creation of languages and models. It also integrates heterogeneous parts for different applications that work together to deliver a global service. Specification and simulation techniques aim to model and validate system design and architecture. They are combined with formal verification tools, in particular model checkers, to describe and simulate what a system should do.
The use case of this paper is intended to illustrate this. To implement our DSL, we chose GEMOC Studio [15] to combine several heterogeneous technologies. It covers all aspects of a DSL, from abstract and concrete syntax to semantic operations, as shown in Fig.1. We started with the definition of the abstract syntax and the metamodel, as shown in Fig.4. It is based on Eclipse Modeling Framework (EMF), which supports Ecore metamodel implementation. It is also interesting to note that the GEMOC framework generates an IDE with syntax checking. Once we have the final libraries, this generative approach allows us to generate various concrete syntaxes by using Xtext artifacts [16]. Once the concrete syntax was processed, we needed the operational semantics to assign behavior to each of the declarations in our DSL. To do this, we use Xtend, a programming language used to implement the execution semantics of Ecore metamodels. We are generating two types of documents: one for the integration with the internal system, and the other human-readable that describes rules specification. This technology allows us to add methods and verify the defined properties. The next step will be combining the generated code from Xtend with

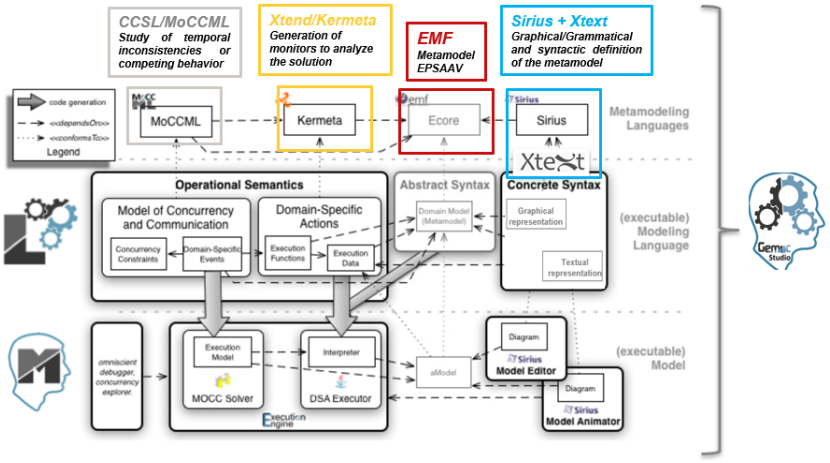

Fig. 1: Overview of the GEMOC Execution Framework with the used technologies in our approach.

CCSL that helps to study temporal inconsistencies and concurrent behavior. To visualize the behavior, Sirius is also available for the graphical display implementation. Interestingly, the GEMOC framework is open source and easily integrates with other tools. It features easy code generation and adapts when settings are changed so that it works properly as development continues. In other words, the language allows us to express the Operational Design Domain (ODD) in a common, non-ambiguous language in which we can express the scenarios that need to be handled safely by a vehicle to achieve "certification" as discussed by [17]. To create this type of system, which is too complicated to do by hand, we use these technologies to shorten the development cycles and have an effective method of reasoning on these rules. The interesting thing about this approach is that we can adopt new requirements that we meet or change the existing ones without changing the language, so it is scalable.

3.2 Illustrative real-life use case

To apply operational safety to the trajectories of autonomous cars, we will need elementary data necessary for the system that is perception. We also need to know the rules that need to be defined with their types. Therefore, our use case relies on Renault's unsafe scenarios document, which describes the raw data of the abstract scenarios, their risks, and the actions to be taken. Renault's safety requirements are based on the Safety Of The Intended Functionality standard (SOTIF— ISO/PAS 21448) [18]. This document aims to determine most of the different use cases for the system based on the verification process. The objective is to verify that Renault can handle all the identified scenarios. We then need to manage each use case in the most different conditions that help us estimate that

the other use cases are managed and validate all the other conditions regarding the scenario. Tests will be performed to verify and validate the use cases in each scenario handled by the ego vehicle, while we will focus on specific environmental conditions to show that the perceptual sensors can avoid any hazard, and to verify any scenario that was not successfully executed in the simulation. After that, we will use expert feedback from System, Fusion, and Safety teams.

Fig.2 shows a use-case scenario where a neighboring car in traffic mode enters the lane of the autonomous vehicle.

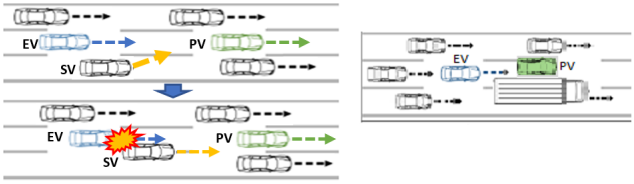

Fig. 2: Straddling Vehicle (SV) swerving over the Ego Vehicle (EV) to take over Preceding Vehicle's (PV) place.

There is a risk of collision with this adjacent vehicle. The measures to be taken are a regulation of the safe distance, i.e. a stable control, and if this lasts longer than a certain time, we perform an emergency maneuver, as we called it *emergency_operation* in our library (Fig.5a). Based on this document, we could describe these unsafe scenarios more concretely in a specification document, in terms of rules and priorities, warnings and actions, as seen below in Fig.3:

```
BEGIN GOALS
when
      traffic_jam is yes
then
    BEGIN GOALS
        when
              stable_control is stable AND
            (
                straddling_car_tracking is straddling_more_than_t7
            )
          then goal:
              executing  emergency_operation
```

Fig. 3: Formal rules specification based on the swerving case.

The notion of priority consideration in the model will allow us to determine if the rules are complete, that is if they cover all situations or not. What will also be good is to see the gaps that are missing in this document when examining inconsistencies and rule violations.

3.3 Abstract description of the Extensible Platform for Autonomous Vehicle Safety Analysis (EPSAAV)

Our metamodel in Fig.4 is composed of three levels. The first level contains the

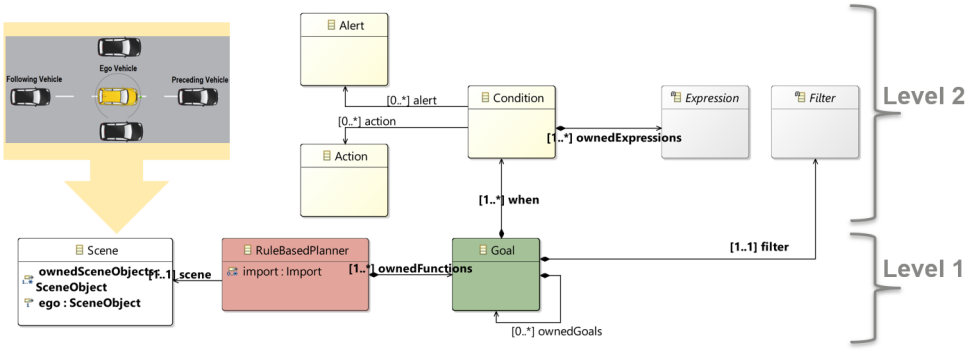

Fig. 4: Abstract description of EPSAAV metamodel using EMF technology.

Rule-Based Planner RBP, which refers to a described scene and is composed of goals. The RBP is responsible for formalizing and specifying rules. Each driving task requires object and Event Detection and Response (OEDR). It helps to identify objects around the ego car, detect events that occur nearby, and then react to them. There are three crucial parts of perception: (1) Static Objects (e.g.: road structure, traffic lights, and signs), (2) Dynamic Objects (e.g.: vehicles and pedestrians), (3) Ego Module. For this reason, we created a scene specification, which is very important to describe the maximum capacity of perceived objects around the autonomous vehicle. The scene specifies the roles of these objects and their types. The goals consist of the conditions we want to apply, which relate to actions we can associate with the motion planner, and alarms we want to trigger for the user. This is at the second level of the metamodel. These conditions are described by logical expressions. The metamodel also can filter the rules either by library type or by role or even by expression. The third level defines the libraries in which we want to describe the properties for the ego and the obstacles. Creating this metamodel gives us the right to a more concrete description of the environment and the rules described in 3.4.

3.4 Concrete description of the Autonomous vehicle's environment

In this part, a concrete description of the environment is presented. The main feature of EPSAAV is to allow the user or safety expert to create multiple libraries to specify the properties. In our use case, the main goal was to have one library for the ego and another one for the obstacles. For future work, we can

create properties related to pedestrians, for example, but for now, let us consider them as obstacles. This use case is edited to be able to insert everything that is shown in the Renault specification document. In the scene, as we said before, we specify the roles of the obstacles and their types with reference to the libraries created. We also created a library for actions and one for alarms in Fig.5.

```
ActionLibrary Actionsv1

Action "brake_moderate"
Action "brake_strong"
Action "emergency_operation"
Action "emergency_maneuver"
Action "no_action_needed"
Action "acceleration_moderate"
```

(a)

```
AlertLibrary Alertsv1

Alert "longitudinal_report"
Alert "lateral_report"
Alert "emergency_operation"
Alert "violation_safety_distance"
```

(b)

```
Scene scenev1
{
    ego ^ego
    role "ego" PropertyTypeLibrary EgoPropertyTypeLibrary
    role "pedestrian" PropertyTypeLibrary ObstaclePropertyTypeLibrary
    role "following_vehicle" PropertyTypeLibrary ObstaclePropertyTypeLibrary
    role "straddling_vehicle" PropertyTypeLibrary ObstaclePropertyTypeLibrary
    }
```

(c)

Fig. 5: Concrete description of the version 1: (a)action library *Actionsv1.actions* (b)alert library *Alertsv1.alerts* (c)scene *scenev1.scene*.

Xtext files were needed too to create semantic forms for rules with priorities. Rules without priorities just don't make sense. Take the case of these two rules, one that applies slight acceleration to escape a crash with the rear vehicle, and one that says you have to brake because a pedestrian is crossing the road. Hence the notion of priority must be included in the syntactic part. Our model gives us the option to prioritize rules in an easy concrete way as seen in Fig.6.

```
RuleBasedPlanner RBP{
    scene "scenev1"
    GOAL pedestrian_priority{
        filter SelectByRole ego
        WHEN{
            propertyType "EgoPropertyTypeLibrary.front_pedestrian_tracking" is "EgoPropertyTypeLibrary.front_pedestrian_tracking.exist"
            action "Actionsv1.brake_strong"
            alert "Alertsv1.violation_safety_distance"
        }
        GOAL rear_vehicle_safedistance{
            filter SelectByRole ego
            WHEN{
                propertyType "EgoPropertyTypeLibrary.rear_car_distance" is "EgoPropertyTypeLibrary.rear_car_distance.acc_distance"
                action "Actionsv1.acceleration_moderate"
                alert "Alertsv1.longitudinal_report"
            }
        }
    }
}
```

Fig. 6: Prioritizing rules in the RuleBasedPlanner *RBP.rbp*.

The user can add new versions of his work and follow this helpful expressive formal specification to write the rules. If something goes wrong, the Rule-based Planner is responsible for informing the driver of the problem detected. It analyzes the event of a subsystem failure, then triggers the alert to the user. The description of the violation's type gives the user a hint when a takeover is requested. There is a possibility to link this description to the trajectory planner if we take it at a higher level in the taxonomy [19]. The verification of the expressiveness of all the rules retrieved from the unsafe scenarios Renault's documents is done by creating generators described in the following part.

3.5 Generation of monitors to analyze the solution

We want to make sense of the environment and generate behaviors in it. Using Xtend technology, we created a document generator that describes all the resources and predefined libraries used, as well as the prioritized rules. This text generation has a documentary goal for a good description of our environment and the rules with their priorities. It will help safety experts to save their work whenever a change is made. We also created a code generator that will be used to check violations in an unsafe case and to test for inconsistencies between the priority of these rules and with the block functionalities for Renault. To do this, it is necessary to include in the code generator a function that adapts to the driving data and the rules, and that can access all the driving data from Fusion or the simulators. This is an ongoing work while retrieving structures of data in Renault's fusion runner. Our safety checker module adapts the rules and the input data and then returns the warnings and actions that were triggered in the Rule-Based Planner.
The next phase consists of examining the consistency of the rules. As seen in Fig.7, we need to compare the output of our safety checker module with the output of a feature block, e.g. the Autonomous Emergency Braking (AEB) block, to see whether or not there is consistency with our actions in an unsafe case in the event of braking.

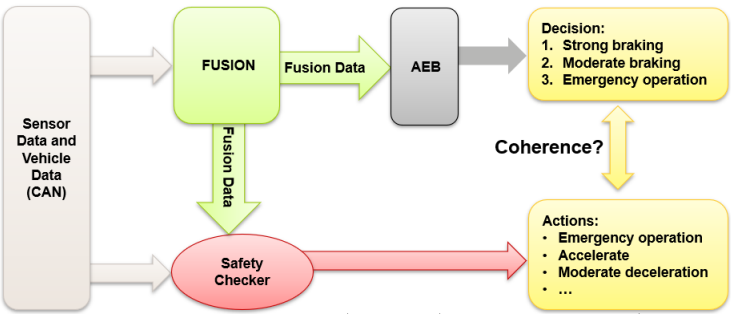

Fig. 7: Study of the consistency between the outputs of the AEB functionality block and the rules described in the safety checker module.

4 Conclusion and Future Work

In this article, we propose an MBSE approach based on a simulation environment in which we specify and generate libraries and a set of safety rules. These rules have different priorities that help us define the types of violations along a trajectory. We also generated a human-readable document and a code for the internal system integration. The main contribution of this approach is to develop a resilient and safe driver monitoring system that continues to operate safely as long as the assumptions about the environment are met. This is a general approach that could be used for multiple domains. It is a checker that triggers alerts when rules are violated and detects inconsistencies between these rules. It facilitates the description of formal safety rules for experts and gives them the power to version any modification of the environment. The approach can also modify/replace decision-making in trajectory planning. This approach will also be used to find new scenarios to complete the verification of the operational safety on the databases. This is an ongoing work where we provided concrete syntax, completed our abstract metamodel, and developed the first generator.
For future work, studying inconsistencies by scheduling analysis with CCSL based on priorities of the objectives, and adding a graphical interface using Sirius are the two following steps, along with the integration of other public simulators. Carla and Webots for automobiles show great promise as they are widely used by the community. The use of simulation and display tools at Renault is also possible, such as the running data of the fusion.

References

1. L. Vanbever. (2019) Self-driving networks: Breaking new ground in network automation. Accessed: 2020-06-05. [Online]. Available: http://univ-cotedazur.fr/en/eur/ds4h/research/forum-numerica/forum-numerica/past-sessions/laurent-vanbever
2. The Free Encyclopedia. Death of elaine herzberg. Accessed: 2021-04-30. [Online]. Available: https://en.wikipedia.org/wiki/Death_of_Elaine_Herzberg
3. N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, 2016.
4. P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15–24, 2016.
5. S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *arXiv preprint arXiv:1708.06374*, 2017.
6. X. Yan, S. Feng, H. Sun, and H. X. Liu, "Distributionally consistent simulation of naturalistic driving environment for autonomous vehicle testing," *arXiv preprint arXiv:2101.02828*, 2021.
7. J. D'Ambrosio and G. Soremekun, "Systems engineering challenges and mbse opportunities for automotive system design," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2017, pp. 2075–2080.

8. J. Duprez, "An mbse modeling approach to efficiently address complex systems and scalability," in *INCOSE International Symposium*, vol. 28, no. 1. Wiley Online Library, 2018, pp. 940–954.
9. S. A. Seshia, D. Sadigh, and S. S. Sastry, "Formal methods for semi-autonomous driving," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2015, pp. 1–5.
10. M. O'Kelly, H. Abbas, S. Gao, S. Shiraishi, S. Kato, and R. Mangharam, "Apex: Autonomous vehicle plan verification and execution," *SAE World Congress*, vol. 1, Apr 2016.
11. M. Althoff and J. M. Dolan, "Online verification of automated road vehicles using reachability analysis," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 903–918, 2014.
12. B. Jeannet and F. Gaucher, "Debugging embedded systems requirements with stimulus: an automotive case-study," in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016.
13. Foretellix. (2020) Open measurable scenario description language (M-SDL). Accessed: 2021-04-30. [Online]. Available: https://www.foretellix.com/open-language/
14. J. Gray, S. Neema, J.-P. Tolvanen, A. S. Gokhale, S. Kelly, and J. Sprinkle, "Domain-specific modeling." *Handbook of dynamic system modeling*, vol. 7, pp. 7–1, 2007.
15. B. Combemale, O. Barais, and A. Wortmann, "Language engineering with the gemoc studio," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017, pp. 189–191.
16. L. Bettini, *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd, 2016.
17. P. Koopman, R. Hierons, S. Khastgir, J. Clark, M. Fisher, R. Alexander, K. Eder, P. Thomas, G. Barrett, P. Torr *et al.*, "Certification of highly automated vehicles for use on uk roads: Creating an industry-wide framework for safety," *White Rose Research Online*, 2019.
18. P. Koopman, U. Ferrell, F. Fratrik, and M. Wagner, "A safety standard approach for fully autonomous vehicles," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2019, pp. 326–332.
19. SAE Mobilus. (2018, Jun.) Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. Accessed: 2021-04-30. [Online]. Available: https://www.sae.org/standards/