GeoScenario: An Open DSL for Autonomous Driving Scenario Representation

Rodrigo Queiroz[1], Thorsten Berger[2], Krzysztof Czarnecki[3]

***Abstract*— Automated Driving Systems (ADS) require extensive evaluation to assure acceptable levels of safety before they can operate in real-world traffic. Although many tools are available to perform such tests in simulation, the lack of a language to formally design test scenarios that cover the complexity of road traffic situations hinders the reproducibility of tests and impairs the exchangeability between tools. We propose GeoScenario as a Domain-Specific Language (DSL) for scenario representation to substantiate test cases in simulation. By adopting GeoScenario on the simulation infrastructure of a self-driving car project, we use the language in practice to test an autonomy stack in simulation. The language was built on top of the well-known Open Street Map standard, and designed to be simple and extensible.**

I. INTRODUCTION

Developing automated driving systems (ADS) with increasing levels of automation requires extensively and rigorously evaluating them before releasing them to customers. As the level of automation increases [24], more driving tasks are transferred from the human driver to the ADS, which has to deal with real-world traffic and all of its disturbances, interacting with human-controlled vehicles, pedestrians, and other traffic agents. Hence, testing ADS should consider their interactions with other agents in realistic traffic conditions to ensure safety and conformity to the applicable traffic legislation.

ADS testing in real-world traffic is extremely expensive and risky. Before deploying the system to a real car, engineers typically rely on simulation tools to test it under many different traffic situations until it reaches acceptable levels of safety. Researchers and engineers can manually design test scenarios based on expert knowledge and on common traffic situations the ADS must be able to cope with. Another approach is to reproduce or augment situations collected from traffic data [30], [22] and crash databases [20]. Figure 1 shows a typical pre-crash scenario based on National Highway Traffic Safety Administration (NHTSA) traffic crash data [20], [21]. This scenario is ranked as the most frequent crash scenario (20%) and is typically characterized by front-to-rear impacts between vehicles. As the trailing vehicle, Ego must stop in time to avoid the imminent collision.

Testing ADS capabilities, such as rear crash avoidance, requires simulation tools to execute the relevant scenarios, but also a language to formally represent them. Many simulators focus on sensors and vehicle dynamics (e.g., VREP [23] or Microsoft's AirSim [28]), but do not provide tools to simulate scenarios with traffic environment. Other tools, such as Carla [13], include scenarios based on free-roaming (traffic agents randomly following roads) and need to be programmed to allow controlled traffic scenarios. When tools provide features to simulate controlled scenarios, such as Virtual Test Drive (VTD) [6], they are typically based on a language exclusive to their own simulation environment. In summary, to design and run test scenarios for self-driving cars, engineers need to learn tool-specific languages or program simulated traffic from scratch. Migrating scenarios between different simulation tools requires extra effort and impairs comparisons between different driving systems.

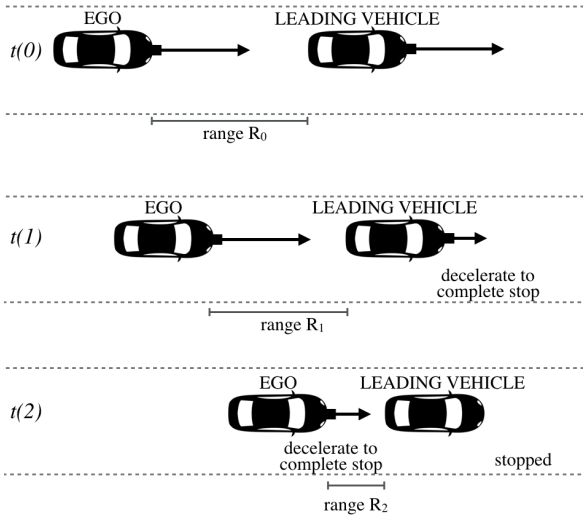

Fig. 1. Testing scenario based on typical rear end pre-crash scenario from NHTSA studies [20], [21]. At $t(0)$, both vehicles are following the same lane while Ego is keeping a safe Range and TTC (Time to Collision). At $t(1)$, the Leading vehicle changes its behavior, starting to decelerate until a complete stop. At $(t2)$, Ego (the following vehicle) needs to stop in order to avoid a collision.

Since typical scenarios for testing self-driving cars aim at reproducing realistic traffic situations, they are similar by definition and must be able to offer the same set of core features. A well-designed, tool-independent domain-specific language (DSL) that is expressive enough to cover these features has the potential to help researchers and engineers to engineer tool-independent test cases, migrate scenarios between different tools, and to evaluate their systems under alternative testing environments.

[1]Faculty of Electrical & Computer Engineering, University of Waterloo, Canada rqueiroz@gsd.uwaterloo.ca

[2]Department of Computer Science and Engineering, Chalmers | University of Gothenburg, Sweden thorsten.berger@chalmers.se

[3]Faculty of Electrical & Computer Engineering,University of Waterloo, Canada kczarnec@gsd.uwaterloo.ca

In this paper, we propose GeoScenario as a DSL for scenario representation and evaluation. We identify relevant elements that compose typical test cases and need to be formally defined and executed in simulation testing. Additionally, we provide a tool-set to easily design and validate scenarios using our DSL. We hope that our language is adopted by the community, and that we can create a shared database of tool-independent test scenarios for self-driving vehicles. We apply the DSL on the simulation infrastructure of the Autonomoose Project, demonstrating its applicability in practice.

II. Background and Related Work

The term *scenario* is not used consistently in the literature [17], [14], [29], [15]. Although its usage varies depending on the discipline, the main components are similar: actors, background information on actors and assumptions about the environment, goals, actions and events [15]. In the remainder, we rely on Ulbrich et al. [29] who analyzed the term scenario (and other related terms) across multiple disciplines and propose a consistent definition based on requirements for testing automated vehicles:

"*A scenario describes the temporal development between several scenes in a sequence of scenes. Every scenario starts with an initial scene. Actions & events as well as goals & values may be specified to characterize this temporal development in a scenario. Other than a scene, a scenario spans a certain amount of time.*" [29]

Figure 2 illustrates this temporal development from the Initial Scene. From a single initial scene, a scenario can evolve through alternative paths leading to different scenes. Each path is by definition a single individual scenario. A scene can be interpreted as a snapshot of the environment, and is composed by the scenery (stationary elements), dynamic elements (elements that have the ability to move, or whose state changes withing the scene), actors, and observer self-representation (attributes and states).

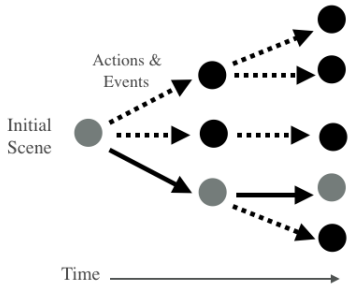

Fig. 2. A Scenario (solid line) as a temporal sequence of actions&events (edges), and scenes (nodes). Adapted from [29]

The main component of the scenery is the road network. It contains all topological information about the roads and their semantics, including lanes, road markings, traffic signs, traffic lights, crosswalks, intersections, and all relevant details that could affect the traffic.

A. Tool-Specific Languages

Many tools and studies create their own language to define scenarios. Although they do not attempt to propose an open and tool-independent language, they can potentially be used as a reference to an open format.

CommonRoad [9] proposes a composable benchmarks structure for motion planning based on three main components: vehicle model, cost functions, and scenario. Although this work does not attempt to propose a DSL for scenario description, it creates its own file format for this task. It also provides a collection of scenarios partially recorded from real traffic (from NGSIM [22]) and partially hand-crafted. Similar to GeoScenario, this work also relies on Lanelets [11] for representing the road network. The planning problem is formulated with the Ego vehicle having an initial state and several goal regions (point, shape or a specific lanelet), and all other vehicles having detailed trajectory data. Apart from trajectories, there are no advanced tools to orchestrate a scenario based on key locations or metric conditions. The format fits the benchmark goals, but is currently not expressive enough to be applicable to other simulation applications and benchmarks aiming to cover more complex traffic situations.

Simulation tools often include a language to describe scenarios. However, these languages are limited to their simulation environments, making it hard or impossible to be translated or interpreted across environments. Examples include SUMO (Simulation of Urban Mobility) [10], and Carla [13] with the recent inclusion of the optional module Scenario Runner (scenarios are controlled by scripts using API calls).

B. Open Languages

In the context of scenario representation, a map format can be used to express the features of the Road Network. Open Street Map (OSM) [2] is a well-known collaborative project to create and publish free maps using an open XML format. However, OSM and other general map standards do not contain detailed information about the road topology at the lane level. Therefore, they are not suitable to be used in a scenario language as is. Lanelets [11] is an open extension of the OSM format specifically to support Road Network representation for automated vehicles. By definition, *lanelets* are *"atomic, interconnected, drivable road segments geometrically represented by their left and right bounds"* [11]. The bounds are encoded by an array of OSM nodes forming a polyline. Together, they compose the *lanelet map*. With lanes represented by road-segments with precise boundaries, Lanelets can be used to compose the Road Network of a scenario.

OpenScenario [4] is an emerging open file format for the description of dynamic contents in driving simulation applications. The project is managed by the Association for Standardization of Automation and Measuring Systems (ASAM) and is currently in its early stages. The project plans to cover dynamic content of simulators, such as driver behavior, traffic, weather, environmental events and other features. The static content is supported by another format, OpenDRIVE [3].

While OpenScenario and our language, GeoScenario, have similar goals, they differ in their structure, and level of abstrac-

tion. OpenScenario can be closer to a logical level [19]. For example, a scenario can be expressed with high-level maneuvers (e.g., a lane change or overtaking), even if the specific trajectories and locations are not explicit. This brings the challenge of running a scenario as a concrete executable test without additional specification. OpenScenario also describes maneuvers that should be performed by the ADS during the scenario (Ego vehicle), making it hard to be reproduced and even applicable to different systems. Both specifications, OpenScenario and OpenDRIVE, are open, but there are no freely available libraries or tools to interpret and process the data.

III. Designing a driving scenario language

In this section we discuss essential requirements for a well-designed driving scenario language for testing automated vehicles, and how we address them in GeoScenario. We identify key elements that compose a scenario, discuss the main scenario design approaches they must support, and the basic principles we follow to make the language practical.

A. Supporting Test Case Development

The ISO 26262 standard for functional safety [16] provides a framework based on the V-model as a reference to guide all development phases of safety-critical electric/electronic vehicle systems. According to the standard, scenarios are used to support the development process, from requirements to the test phase by supporting test cases.

When supporting test cases, scenarios such as car-following, lane-changing, and overtaking maneuver, can be created using different approaches. One approach is for experts to manually design scenarios based on functional requirements and designs and hazard analysis. A complementary approach is to reproduce or augment situations collected from traffic data. In the recent years, a huge progress has been made in collecting and publishing naturalistic driving data, which can be used for scenario development. Examples of publicly available datasets include IVBSS [26], SPMD [30], and NGSIM [22]. For instance, scenarios in CommonRoad [9] were extracted from NGSIM data. A combination of the two mentioned methods is also used, by designing manual scenarios and extracting primitives from naturalistic databases. Instead of reproducing strict trajectory data, behavioral characteristics of maneuvers are extracted, such as the distribution of time to collision, range, and speed while performing lane-change maneuvers from all scenarios in the database. These primitives can be sampled and combined to generate new scenarios [32]. Finally, a scenario can also be systematically generated. Apart from the sampling method on naturalistic or manually defined distributions, scenarios can be created to achieve specific goals (e.g., lead the system to explore a certain behavior such as an emergency maneuver, or find a critical situation leading to a crash). For example, Abdessalem et al. [7] use evolutionary optimization methods combined with neural networks to find critical scenarios (e.g., a crash scenario). The generation starts from designing the input space, and new generations are created based on how the system performs under simulation. Similar approaches are also used to test autonomous parking system [12], and to guide the search-based generation of tests faster towards critical test scenarios [8]. A scenario language should be able to support all mentioned approaches. It must be simple and human readable, yet be able to represent precise trajectories collected from traffic data, support input space exploration from methods generating scenarios, and also support unknown stochastic behaviour for sampling methods. We built GeoScenario to support all these approaches.

B. Levels of Abstraction

Scenarios can be described at several levels of detail and expressed using formal, informal or semi-formal notations [15]. Menzel et al. [19] propose three levels of abstraction for scenarios along the development process of the ISO 26262 standard [16]: (i) scenarios described as a high-level abstraction in the concept phase (functional scenarios), (ii) scenarios with parameter ranges of the state space in the development phase (logical scenarios), and (iii) scenarios with concrete parameter values in the test execution phase (concrete scenarios). We decided to tackle the concrete level by designing a language to express scenarios as a base for the test phase, using concrete state values, and assuring reproducibility.

C. Scenario Orchestration

When dynamic elements in a scenario follow pre-defined paths, we assume a deterministic evolution from the initial scene. However, when the ADS is responsible for the Ego's driving mission, a scenario can evolve to alternative scenes, and its execution becomes nondeterministic. Scenarios described in CommonRoad [9] are clear examples of this challenge. Dynamic elements are defined as time-discrete states of a trajectory containing position and orientation over time. After slicing NGSIM data in different scenarios, one vehicle is selected to represent the Ego while the remaining vehicles are selected as dynamic elements with their original trajectories. The challenge arises the moment Ego starts to perform differently from the original vehicle (by different route, velocity or maneuver). The scenario then evolves to a different situation. This is a natural limitation of any scenario directly reproduced from traffic data. Consequently, a model for scenarios must be able to orchestrate the evolution between scenes with a flexible language for Actions & Events.

A different approach is to describe intelligent dynamic agents making decisions, behaving like human drivers and pedestrians, and reacting to every other traffic agent (including the Ego). However, this brings the challenge of modeling complete and realistic behavior of traffic agents and makes reproducible scenarios hard to achieve. We designed GeoScenario to provide ways of reproducing trajectory data, but also created mechanisms to orchestrate its evolution under different conditions (time and space), allowing engineers to carefully craft scenarios that explore controlled situations. Focused dynamic models of agent maneuvers can be integrated as a future extension.

D. Basic principles

We designed GeoScenario using the following basic principles. (i) *Reuse*: Leverage existing open formats to build a new language on top of well-known and used structures. With this approach, existing tools can be reused to support our new language with only minor adjustments. (ii) *Simplicity*: The language is simple enough to be human readable when simple scenarios are modeled. Tools are encouraged to support complex scenarios. (iii) *Coverage*: It is able to express the main components of a scenario. (iv) *Extensibility*: It can be easily extended with new features and specializations of its standard components. (v) *System independence*: It supports test cases for different ADS designs, operating on different levels of automation. (vi) *Tool independence*: It can be interpreted and executed by alternative simulation and test environments. (vii) *Executability*: It can express concrete scenarios that can run in simulation without an additional language.

IV. GeoScenario Architecture

GeoScenario was developed to express a scenario in a formal language, following the requirements discussed on Section III. The format is XML-based and built on top of the OSM standard. The main components include: Ego start position and goals, a road network, agents (vehicles and pedestrians), paths, and triggers & actions. Additional elements are omitted for simplicity, but they are available in our full specification. Figure 3 illustrates a sample scenario with the main components in place. Figure 4 shows a meta-model (a.k.a., syntax model) of our components. In the next section we will describe how all those elements work and interact.

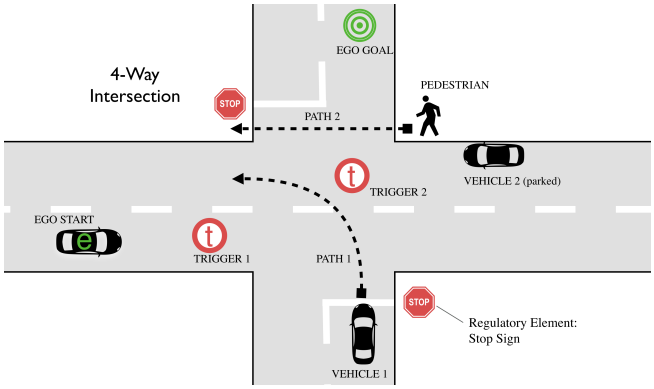

Fig. 3. Overview of the main GeoScenario components in a 4-way intersection scenario.

A. GeoScenario Basics

All GeoScenario elements are based on two OSM primitive types: *node* and *way*. **Nodes** are the core elements of GeoScenario, representing a specific point on earth's surface. Each node comprises an ID number and a pair of coordinates (latitude and longitude). Nodes are used to define standalone point features (e.g., a *vehicle* or a *pedestrian*), but also to compose the shape of other elements (e.g., a path). **Way** is a ordered list of nodes defining a polyline. Ways are used to define linear features such as *paths* and boundaries of areas

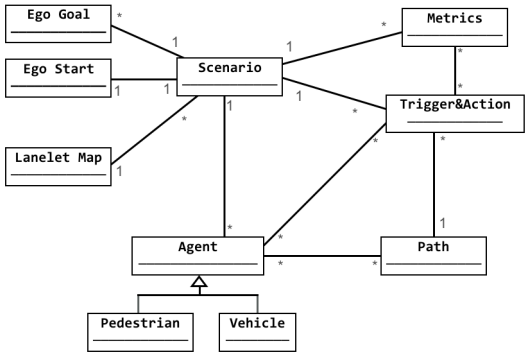

Fig. 4. GeoScenario meta-model (class diagram notation from UML).

(solid polygons that represent an obstacle on the road, or a named area for dynamic element placement). To define areas, the way's first and last node must be the same (closed way).

All elements (nodes and ways) can have tags describing attributes of an element with the pair of text fields *k* (key) and *v* (value). We use a tag *gs* to define an element's role in the scenario, that is, the element's function in the GeoScenario model (e.g., $gs = vehicle$). Elements without a *gs* tag do not have a specific role in the scenario, but can be used to compose other elements. For example, nodes composing a path do not have a *gs* tag. An element cannot have two tags with the same key, and they must be unique.

All elements with a role must also contain the tag '*name*' (with a few exceptions). The name is a unique string that identifies one element in a scenario. This tag is used to derive relations between elements. Nodes have coordinates in the WGS84 coordinate frame (as part of the OSM standard). There is a fixed dictionary of tags documented in our GeoScenario specification, but we will highlight the main properties per element in this paper. Listing 1 shows an example showcasing a vehicle node and its basic components.

Listing 1. GeoScenario element example

```
<node id='1' lat='43.5094' lon='-80.5367'>
   <tag k='gs' v='vehicle'/>
   <tag k='name' v='leading_vehicle'/>
</node>
```

B. Ego and the Driving Mission

In a Scenario, Ego is the entity representing the ADS. In our language we decided not to define actions or maneuvers for the Ego. Instead, GeoScenario only specifies initial conditions and goals. During a test case execution time, the ADS is a black box system responsible for deciding the best route and maneuvers based on the traffic conditions (road network, static objects, dynamic agents on the path, etc.). We decided for this approach to allow the language to be system independent and to reflect a real world driving scenario. In practice, a driving mission is given to the driver or ADS as a global location to be reached as a long-term task. The initial condition is defined as a node representing Ego's starting position and orientation. We assume Ego always starts a scenario in a parked position.

The goal is defined as an *egogoal* node. A scenario can have multiple ordered goal locations. They represent the intermediate and final driving mission the ADS should achieve. The final goal for the driving mission task is the one with highest order number and must finish the scenario with a success state. The nodes can be used to compose a global path for the system, or create a goal point on the system's internal map. However, this is particular to the ADS configuration and is out of the scope of our model.

C. Scenery and Road Network

We use Lanelets [11] to represent the scenario road network. We decided to use Lanelets because of their compact and lightweight structure; the GeoScenario format follows a similar spirit itself. The road network is stored in a separate XML file to make replacements easy. However, a scenario can only be interpreted within the context of the road network. Consequently, a GeoScenario must always be distributed with its associated road network file.

To represent stationary obstacles that are not part of the road network, but block or limit the drivable surface, we introduce *static object*. Static objects can be defined as a single node, a way, or a closed-way. A closed-way can assume arbitrary shapes, but in order to be valid, it must have the first and last node reference pointing to the same node ID. A reference to a model can be used to give the object a more defined form. We chose to keep the GeoScenario simple and flexible, and the model must be defined elsewhere.

D. Dynamic Elements

We define as *dynamic elements* all GeoScenario elements that are able to move (having kinetic energy) or are able to change their state. This is different from Geyer's [14] definition of dynamic elements, which are based on the temporal extent of the scene. In GeoScenario, a parked vehicle is also defined as a dynamic element. Dynamic elements that are able to move are called *agents*, and are separated in two types: vehicles and pedestrians. Both are represented as nodes and share similar attributes. Vehicle is defined with the tag $gs = vehicle$, and pedestrian with tag $gs = pedestrian$. The *orientation* tag is used do define an agent's initial orientation (for example, a vehicle yaw). In our model the orientation is given in degrees, with origin on East and clockwise direction. Different types of vehicles (e.g., car, truck, bus) are represented with the same type, with an optional attribute *model* specifying a vehicle model. We do not specify details of the vehicle model dynamics or 3D meshes. Therefore, testing results must take into account additional details of the simulation infrastructure running the scenario. A *speed* attribute (in km/h) is used to define a standard velocity.

In order to move, vehicles and pedestrians need to be assigned to a path. A *path* is defined as a Way element, and can be used for both vehicles and pedestrians. Paths should be interpreted as splines composed by ordered connected nodes. When a dynamic agent is assigned to a path, it will travel along the path with its standard speed.

To support more realistic kinetics with variable velocity and acceleration, or to reproduce scenarios from recorded traffic data, an agent can be assigned to a *speed profile*. When a path has a speed profile, it must contain nodes with the tag *agentspeed* to indicate the target speed in km/h for the agent once it reaches that node. The agent must always try to match the speed of the next node in its path with a constant acceleration.

With high density paths (i.e., more nodes) and a speed profile, a GeoScenario model can represent a diverse range of traffic situations, manually designed by experts, extracted from real traffic by sensors, or imported from naturalistic driving databases. As examples, Listing 2 shows a typical dynamic agent as a vehicle, and Listing 3 shows its path defined as a way. Note how the ID references to the nodes composing a path are given by the tag *nd* and must be interpreted as an ordered list.

Listing 2. Dynamic agent

```
<node id='1' lat='43.5094' lon='-80.5367'>
   <tag k='gs' v='vehicle' />
   <tag k='name' v='leading_vehicle' />
   <tag k='speed' v='30' />
   <tag k='orientation' v='45' />
   <tag k='path' v='northpath' />
   <tag k='usespeedprofile' v='yes' />
</node>
```

By default, all paths are grounded to fixed node coordinates. We introduce the tag *abstract* to define flexible paths. Abstract paths are designed on fixed coordinates, but during execution must be shifted to a new origin point based on the agent's current location. Abstract paths can be used to design dynamic maneuvers. For example, a lane change that can occur at different locations of the road network.

Listing 3. Path

```
<way id='39'>
   <tag k='gs' v='path' />
   <tag k='name' v='vwest_path' />
   <tag k='abstract' v='no' />
   <nd ref='3' />
   <nd ref='4' />
   <nd ref='5' />
   <nd ref='6' />
 </way>
```

Some properties can be described by a fixed value or by value ranges. As an example, a dynamic agent's speed can be defined by a fixed value (e.g., 30 km/h) or by a range (e.g., from 20 to 40 km/h) using the notation [20:40] for continuous values, and [20,25,30,40] for a list of arbitrary discrete values. The variable attribute notation is used for scenarios that rely on sampling and test input mutation and allows our model to represent both logical and concrete levels of scenario. Mutation of test input values is commonly used is software testing, including driving automation systems [7]. Assigned values represent boundaries, and a concrete value is selected before a scenario is executed. Stochastic behavior with probability distributions is also supported. However, since many different probability distributions can be used, (e.g., Gaussian), they must be defined elsewhere.

E. Triggers & Actions

In GeoScenario we introduce *triggers & actions* to orchestrate how a scenario evolves. The basic concept is to add trigger nodes over strategic places of the road network, and activate different actions over dynamic elements. Each triggers has *owners* and *targets*. Owners activate triggers, whereas targets execute the action (Figure 5). Owners can be the Ego itself or agents (vehicles, pedestrians). Targets can be any dynamic element whose state can change over the scenario, but can not be Ego. This rule follows our assumption of the ADS as a black box system, limited to the initial conditions and the driving mission. Actions can change an agent's state, or the scenario itself. Listing 4 shows a trigger example.

Fig. 5. GeoScenario Trigger. When the owner activates a trigger, an action is executed on the target. The trigger can be activated when the owner reaches the trigger node location, when the Scenario reaches a certain time t, or when a metric between two agents reaches a certain value x.

A trigger can be activated by three types of conditions, or by a combination of them: *(i) Time*: activated when the scenario execution reaches a given time t. A set of timed triggers allow the designer to control the scenario in chronological order with timed events. For example, at a given time $t = 10$, a pedestrian starts crossing an intersection. *(ii) Location*: activated by overlap, when the owner reaches the trigger node location. Must be placed over strategic points of the Road Network. They are especially useful when timed events can not guarantee Ego and other agents are at the right place at the right moment. For example, one can place a trigger with $Owner = Ego$, and and action for a pedestrian to start a path over a crosswalk. This trigger guarantees the walking happens at the desired distance between Ego and pedestrian. *(iii) Metric condition*: activated when a given condition based on a metric is true. This trigger allows situations where an Action needs to be performed with no specific location, but at any location after a relative condition. For example, a vehicle moving over a path on the road starts to decelerate to stop only when the distance between Ego and a vehicle is less than 100 meters. To support a condition, a GeoScenario needs to track a given metric between agents.

Listing 4. Trigger

```
<node id='4' lat='43.50909' lon='-80.53654'>
   <tag k='gs' v='trigger' />
   <tag k='activate' v='location' />
   <tag k='name' v='start_trigger' />
   <tag k='owner' v='Ego' />
   <tag k='target' v='leading_vehicle' />
   <tag k='apath' v='west_path' />
   <tag k='aspeedprofile' v='yes' />
</node>
```

A metric is also defined as an element in GeoScenario, by explicitly declaring which agents are tracked. We encourage scenarios to include references for how a metric is calculated since different approaches can be used. For example, TTC can be used by a variety of methods leading to different values [31], [27], [25].

This paper only provides an overview of our model. All details (with examples) can be found in our project's repository.[1] Additionally, the model can be easily extended with new features and attributes to support tool specific requirements (for example, to support conversion between models).

F. Tool Set

Accessible tools are important to make the model useful for engineers and adopted by the community. Since our format was developed on top of OSM primitives, we adapted it's standard map editing tool: JOSM [1]. By adding a set of custom presets and style sheets, we can now easily design and understand a GeoScenario on top of the Road Network (Lanelet layer) and other map layers (e.g., Bing Maps, ESRI maps) before its execution. Figure 6 shows a sample scenario designed in our custom tool.

The second tool is the GeoScenario Checker: a set of scripts to evaluate a Scenario's conformity with the standard. Both tools are available at the Project's website along with their usage instructions. A third tool, a complete scenario simulator will be released as part of this project, but is out of the scope of this paper.

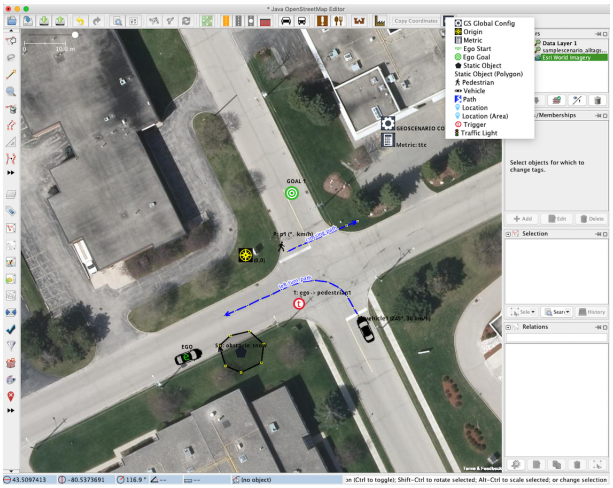

Fig. 6. JOSM adapted to design GeoScenarios

V. Application

We incorporated GeoScenario to Autonomoose Project testing infrastructure as the official data format to design and run test scenarios in simulation. In this section we describe our project and present a sample scenario, tested in simulation with our autonomy stack.

[1] https://git.uwaterloo.ca/wise-lab/geoscenario

A. The Research Platform

"Autonomoose" is the University of Waterloo self-driving research platform. The platform is a Lincoln MKZ Hybrid modified to autonomous drive-by-wire operation and a suite of lidar, cameras, inertial and vision sensors (see Figure 7). The car is equipped with computers to run a complete autonomous driving system, integrating mapping, sensor fusion, motion planning, and motion control software in a custom autonomy software stack fully developed at Waterloo as part of the research. The system was the first Canadian-built ADS to be tested on public roads in Canada in August 2018. More info is available on the project website. [2]

The autonomy stack is implemented on top of Robot Operating System (ROS) framework [5]. ROS offers an inter-process communication interface based on publish/subscribe anonymous message passing. We can explore this interface to isolate the components we want to test and use data from our simulation tools to create a realistic testing environment. We focus our test on motion planning modules to explore Ego's interactions with other traffic agents. Because the publish/subscribe system is anonymous, we can isolate Motion Planning modules by simulating data from sensors (e.g., gps, imu), and Perception modules (vehicle and pedestrian detection) through ROS topic messages. We assumed all sensors work perfectly without failure. This means our simulation environment is able to provide accurate detection of all vehicles within the range of the sensors in a map representing the Road Network.

Fig. 7. Autonomoose research platform. Lincoln MKZ Hybrid modified to autonomous drive-by-wire operation and a suite of lidar, cameras, inertial and vision sensors

B. Designing a Test Scenario

We model the most frequent crash scenario according to NHTSA (lead vehicle stopped) as illustrated in Fig. 1. We model this car-following scenario with a single Ego goal at the end of the road to define a driving mission, a dynamic agent as the leading vehicle shortly after the Ego start position following the road with constant speed over the $east\_path$. When the leading vehicle reaches a trigger, it switches to $decelerate\_path$. This path contains a speed profile, decelerating from $30km/h$ to a complete stop in a short space. Ego is the trailing vehicle and must react to avoid a collision. If a collision happens, the ADS failed and the scenario must end. Figure 8 shows how we modeled our scenario, and its execution in our simulation environment. Triggers with different conditions can be used to explore this scenario with different ranges, and different speed profiles (e.g., an aggressive deceleration profile increasing the level of difficulty).

[2] https://www.autonomoose.net/

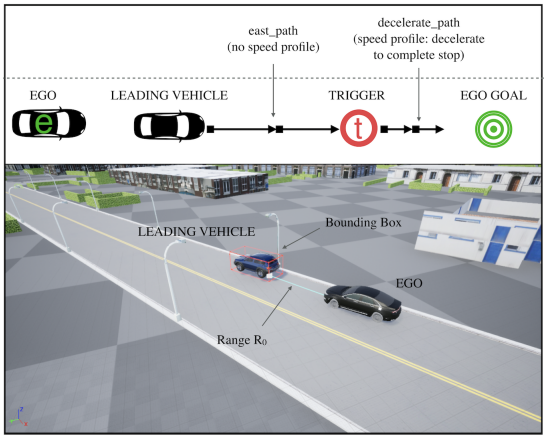

Fig. 8. Rear end pre-crash scenario modeled in GeoScenario and executed in our Simulation Environment. Both vehicles are following the same lane. When reaching the trigger, the leading vehicle switches to a decelerating path profile until a complete stop. The blue line between vehicles indicates the *range* (distance between two vehicles). The red box is the bounding box used to simulate detection and bypass perception modules.

VI. Future Work

We plan to improve GeoScenario to support testing with a focus on the perception task. For example, environment conditions that must be simulated to affect detection (e.g., rain, snow, sun light or visibility in general) can be modeled as both static and dynamic elements (weather changing scenarios). New actions performed by dynamic agents can be added to support scenarios exploring agent behaviour that goes beyond path following. For example, we can explore scenarios where vehicle lights are used for prediction before a leading vehicle starts to decelerate or turn, or simulating a distracted pedestrian talking over the phone to test the system's capabilities to predict unsafe behavior.

We are building a database of testing scenarios with a wide coverage of requirements, exploring a wide range of system capabilities along many phases of development. Scenarios will be both manually designed by engineers and extracted from traffic data. All scenarios will be openly available, and the database will be extended with new scenarios from the community.

We plan to release two additional tools to support the applicability of GeoScenario in projects from the community. The first is the scenario extractor, to generate scenarios from different naturalistic driving databases (e.g., NGSIM [22], [18], SPMD [30]) into GeoScenario files. Finally, a complete driving scenario simulator with full support for GeoScenario will be released as an open source Unreal Plugin, supporting

a series of available open source Unreal based simulation tools (e.g., AirSim [28] and Carla [13]).

VII. Conclusion

We proposed GeoScenario as a DSL for scenario representation. We identified key elements that compose typical test cases and which need to be formally declared and executed on self-driving vehicle testing. By adopting GeoScenario on the simulation infrastructure of the Autonomoose Project to validate the autonomy stack under simulation, we demonstrated its applicability in practice. The language was built on top of well-known Open Street Maps primitives, designed to be simple and easily extensible. A tool-set to easily design and validate scenarios using our DSL is publicly available and we hope it encourages the adoption by the community. With the contribution from more researchers, we plan to publish a shared database of tool-independent test scenarios for self-driving vehicles.

References

[1] Java Open Street Map Editor. https://josm.openstreetmap.de.

[2] Open Street Map (OSM). https://www.openstreetmap.org.

[3] OpenDRIVE. https://www.opendrive.com.

[4] OpenScenario. https://www.asam.net/standards/detail/openscenario.

[5] Robot Operating System (ROS). https://www.ros.org/.

[6] Virtual Test Drive (VTD). https://vires.com/vtd-vires-virtual-test-drive.

[7] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter. Testing advanced driver assistance systems using multi-objective search and neural networks. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 63–74, Sep. 2016.

[8] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. Testing vision-based control systems using learnable evolutionary algorithms. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 1016–1026. ACM, 2018.

[9] M. Althoff, M. Koschi, and S. Manzinger. Commonroad: Composable benchmarks for motion planning on roads. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 719–726, June 2017.

[10] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. Sumo - simulation of urban mobility: An overview. In *in SIMUL 2011, The Third International Conference on Advances in System Simulation*, pages 63–68, 2011.

[11] P. Bender, J. Ziegler, and C. Stiller. Lanelets: Efficient map representation for autonomous driving. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 420–425, June 2014.

[12] Oliver Bhler and Joachim Wegener. Automatic testing of an autonomous parking system using evolutionary computation. *SAE Technical Papers*, 03 2004.

[13] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[14] S. Geyer, M. Baltzer, B. Franz, S. Hakuli, M. Kauer, M. Kienle, S. Meier, T. Weissgerber, K. Bengler, R. Bruder, F. Flemisch, and H. Winner. 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, 2014.

[15] Kentaro Go and John M. Carroll. The blind men and the elephant: Views of scenario-based system design. *Interactions*, 11(6):44–53, November 2004.

[16] ISO/FDIS 26262:1994. *Road vehicles Functional safety*. ISO, Geneva, Switzerland, 2011.

[17] Matthias Jarke, X. Tung Bui, and John M. Carroll. Scenario management: An interdisciplinary approach. *Requirements Engineering*, 3(3):155–173, Mar 1998.

[18] V. G. Kovvali, V. Alexiadis, and L. Zhang. Video-based vehicle trajectory data collection. In *in Proc. of the Transportation Research Board 86th Annual Meeting*, 2007.

[19] Till Menzel, Gerrit Bagschik, and Markus Maurer. Scenarios for development, test and validation of automated vehicles. In *2018 IEEE Intelligent Vehicles Symposium, IV 2018, Changshu, Suzhou, China, June 26-30, 2018*, pages 1821–1827, 2018.

[20] W. G. Najm, John D. Smith, and Mikio Yanagisawa. Pre-Crash Scenario Topology for Crash Avoidance Research. Technical report, U.S. Department of Transportation, NHTSA, April 2007.

[21] W. G. Najm, S. Toma, and J. Brewer. Depiction of Priority Light-Vehicle Pre-Crash Scenarios for Safety Applications Based on Vehicle-to-Vehicle Communications. Technical report, U.S. Department of Transportation, NHTSA, April 2013.

[22] Vincenzo Punzo, Maria Teresa Borzacchiello, and Biagio Ciuffo. On the assessment of vehicle trajectory data accuracy and application to the next generation simulation (ngsim) program data. *Transportation Research Part C: Emerging Technologies*, 19(6):1243 – 1262, 2011.

[23] E. Rohmer, S. P. N. Singh, and M. Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326, 2013.

[24] SAE. Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems (sae j3016). Technical report, SAE International, 2014.

[25] SAE. Operational definitions of driving performance measures and statistics (sae j2944). Technical report, SAE International, 2015.

[26] J. Sayer, D. LeBlanc, S. Bogard, D. Funkhouser, S. Bao, M. L. Buonarosa, and A. Blankespoor. Integrated Vehicle- Based Safety Systems Field Operational Test Final Program Report. Technical report, The University of Michigan Transportation Research Institute (UMTRI), June 2011.

[27] Chris Schwarz. On computing time-to-collision for automation scenarios. *Transportation Research Part F: Traffic Psychology and Behaviour*, 27:283 – 294, 2014.

[28] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. *CoRR*, abs/1705.05065, 2017.

[29] S. Ulbrich, T. Menzel, A. Reschka, F. Schuldt, and M. Maurer. Defining and substantiating the terms scene, situation, and scenario for automated driving. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 982–988, Sept 2015.

[30] UMTRI. Safety Pilot Model Deployment. Technical report, The University of Michigan Transportation Research Institute (UMTRI), (Accessed: 20-Dez-2017).

[31] Richard van der horst and Jeroen Hogema. Time-to-collision and collision avoidance systems. 01 1994.

[32] Wenshuo Wang and Ding Zhao. Extracting traffic primitives directly from naturalistically logged data for self-driving applications. *IEEE Robotics and Automation Letters*, 2017.