Exploring Realism in Virtual Testing: Towards a Scalable Platform Using Open-Source Solutions for Automated Driving Systems

Peter Baker, Joseph Mitchell, Emil Chodowiec, Xizhe Zhang,
Siddartha Khastgir, Paul Jennings
WMG, University of Warwick, Coventry, United Kingdom, CV4 7AL
{peter.baker, joe.mitchell, emil.chodowiec, jason.zhang, s.khastgir.1, paul.jennings}@warwick.ac.uk

***Abstract*—This paper presents a novel solution to the demand of a realistic virtual test environment (VTE) for the development and safety assurance of Automated Driving Systems (ADSs). The current VTE offerings suffer limitations when it comes to creating a rich environment (from the Operational Design Domain perspective) based on the ASAM OpenDRIVE files (a formatted description of the scenery), and hence there is no automatic OpenDRIVE scenery generation available that resembles the richness required to thoroughly test an ADS in a virtual environment. Therefore, through the use of Unreal Engine, leveraging the power of esmini's roadmanager and developing on a primitive OpenDRIVE plugin, a comprehensive scenery with complex lighting, dynamic environmental conditions and photoscanned meshes can be achieved. There is then a demonstration of how this is incorporated into an end-to-end testing framework, using just one user interface to take the user from the creation and retrieval of scenarios through to the execution.**

***Index Terms*—scenario based testing, simulation, scenery generation, automated systems, safety, digital twin**

I. Introduction

Automated Driving Systems (ADSs) development is a challenging area within the automotive and wider engineering industry. It has the potential however to provide many benefits to the end user, yet the most important benefit is to attain a greater level of safety than with a human driver.

A. Scenario Based Testing

With a distance based approach to achieving AV safety, it has been proposed [1] that to prove with a 95% confidence that AVs are 20% safer than human drivers, the ADSs would need to drive "more than 11 billion miles" [1]. Therefore, rather than using a distance based approach, a method of achieving safety is through the use of scenario based testing [2] [3], which is aimed at developing trustworthy ADSs [4], as well as verifying and certifying their safety [5].

A scenario represents any situation that might occur on the road, it contains both the temporal and spatial aspects of the situation. A scenario can be categorised into three main parts, the scenery, the environmental conditions, and the dynamic elements. The scenery part alone can be described using the ASAM OpenDRIVE [6], utilising an XML format. Alternatively, the scenery is also part of the ASAM OpenSCENARIO DSL [7], WMG SDL 2 [8], BSI Flex 1889 [9], stiEF [10]. The dynamic and environmental conditions are contained within the ASAM OpenSCENARIO [11], which focuses on describing how the ego and the other actors in the scenario interact with each other. They are also contained within the other scenario formats mentioned above (OpenSCENARIO DSL, WMG SDL 2, BSI Flex 1889, stiEF etc). Through the use of scenarios there is the ability to thoroughly and iteratively test a system in either fully virtual [12], XiL (e.g., hardware in the loop) [13], or real world environments, with the goal of identifying the failure conditions of the system [14]. Through the use of virtual testing, such testing can be performed in a low risk and scalable manner (e.g., setting environmental conditions). It also has the added benefit that it is a lot less costly than real world testing. In order to virtually test the system, there is the need to mimic real world scenery and environmental conditions that the ADS may experience. Therefore the use of realistic digital twin environments for testing vehicles is paramount to achieving this goal of scenario based testing.

B. ODD

Using scenarios in isolation cannot guarantee that the right scenarios, or enough scenarios have been used during the safety assurance process. Hence, scenario-based testing needs to be underpinned by the operational design domain (ODD) of the system. ODD defines the operating conditions under which the system is designed to function safely, it is a design specification of the system. During a scenario based testing process, individual scenarios need to fully explore the performance of the system within its ODD boundary. Furthermore, it is key to test the system's risk mitigation mechanism whenever it goes beyond its ODD boundary or shows signs of doing so.

To enable the common understanding of ODD specifications and to be able to share or exchange ODDs, there is a need for a common taxonomy and format with which the industry and academia can reference. The BSI PAS 1883 (2020) [15] is one of the early standards that proposed a set of common ODD taxonomy concepts, together with their definitions and

quantifiable measures. The ASAM OpenXOntology [16] and the ASAM OpenLABEL [17] then utilised the PAS 1883 to establish a common domain ontology for use cases such as database organisation and scenario tagging. With the recent release of the ISO 34503 [18] on ODD taxonomy, the PAS 1883 has been superseded. At a high level, the ODD covers the scenery, the environmental conditions, and the dynamic elements (subject to the vehicle maximum designed speed, and dynamic agent types).

While ODD is fundamental to the safety assurance of Automated Driving Systems (ADSs), to cover the scope defined in a scenario, the ODD needs to be combined with behaviour. Using a set of ODD elements and behaviours of a system, users can start to construct individual scenarios which explore the boundary set out by the ODD.

C. Open Simulation Interface (OSI)

In order to simulate scenarios, the complete simulation framework needs to operate within a co-simulation environment, which comprises of multiple subsystems. To enable efficient interaction, OSI [19] provides standardised interfaces and message definitions for integration between different systems, subsystems, and simulation environments. Depending on the subsystems used, the OSI ground truth data can be used for the planning aspect, or raw sensor data can also be used when the sensing layer is involved. Currently the OSI Performance & Packaging WP is investigating the most efficient method for the exchange of raw sensor data.

In the context of scenario-based testing, a reliable communication interface is crucial to guarantee the accuracy of simulation results. Executing scenarios for a specific ODD, whilst utilising OSI, allows for a comprehensive and scalable development and testing of ADSs.

II. Related Work

A. Current Virtual Testing Platforms

For conversion of an OpenDRIVE description into a equivalently represented VTE, there is the need for two aspects to come together. Firstly, using a tool that has the capability for the realistic rendering needed in a VTE. Secondly, converting from an OpenDRIVE description into the 3D space. There are a few methods in existence, which will be detailed below:

An ‘open-source autonomous driving simulator’ called CARLA [20], which is based on a modified build of Unreal Engine (an open-source game engine), has attempted to do a conversion of an OpenDRIVE file (.xodr) into a tangible 3D World. However it is quite primitive in terms of the outcome, and the latest documents detailing the OpenDRIVE generation [21] show that the mesh generation for the roads has issues to do with smoothness that need mitigation. It also does not represent the other ODD objects that are present within the OpenDRIVE file.

esmini is another simulator with the capability to render the OpenDRIVE files, it is “a basic OpenSCENARIO player” [22]. It can successfully generate scenery and handle OpenSCENARIO and OpenDRIVE files, however its limitations come in terms of its graphical realism, majorly due to the intended purpose for which the simulator was developed, i.e. visually simplistic simulation. As it is a lightweight OpenSCENARIO player, it means that when trying to capture certain scene qualities, such as tree geometries, light reflections and light refractions from an object, esmini is unable to provide the level of detail needed to test an autonomous system adequately.

A third relevant work is an OpenDRIVE plugin [23] for Unreal Engine, which allows the user to import the .xodr file into the Unreal Engine editor and then generate the road layout through using esmini’s roadmanager library. However this mesh is not tangible but more of an overlay, and is translucent in appearance. Even though there are tools to sculpt the landscape and repeat objects, they need to be done manually after generating the road layout, so the pipeline is not fully automatic in its nature.

As has been discussed, each of the above methods have both their limitations and benefits, however none of the methods achieve both of the aspects outlined for OpenX conversion into a photo-realistic VTE. However, by developing on the OpenDRIVE plugin for Unreal Engine, the first aspect can be achieved as Unreal Engine can be used for realism. As this OpenDRIVE plugin uses the esmini roadmanager library, through more development, the second aspect can be achieved in order to convert all OpenDRIVE information into 3D coordinates.

III. Approach

A. Scenery Generation

With the initial OpenDRIVE plugin, some aspects to resolve were that roads did not have any realistic materials or physical properties applied to them, the junctions consisted of lines and directional arrows rather than 3D meshes, and the curved lanes that were produced had a broken mesh due to having short straight sections joined together, which resulted in the start and end of meshes not aligning (as shown in Figure 1), compared to the goal of a smooth and continuous mesh.

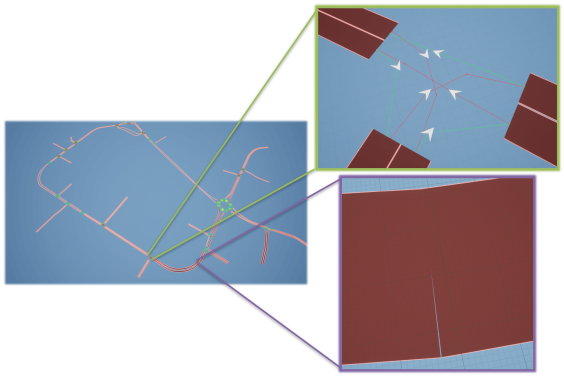

Fig. 1. OpenDRIVE Plugin’s Initial Auto Generated Road Network

Therefore, the roadmanager library was used to get the number of lanes in each road, retrieving information on each lane, such as its length, width, tangent, and S value coordinates

from OpenDRIVE. The lane position and lane width from the OpenDRIVE description were sampled dynamically based on the curvature of the road (higher sample rate at greater road curvatures) along with the tangent information in order to create spline points for each lane, which a spline would fit. The tangent information was used to allow each spline point to guide the curvature of the spline according to the OpenDRIVE specifications. This resulted in successfully correcting the broken meshes from the initial plugin. After this, physical properties of the lanes were allocated and a road surface material was applied based on the lane type specified in the OpenDRIVE file.

At junctions, there were overlapping lanes, which results in a graphical glitch, known as Z-fighting, where two or more meshes share the same 3D space. Hence, Unreal Engine will attempt to render all meshes at the same time, resulting in flickering occurring. Therefore, the textures for the road network needed to be world dependant instead of individual object dependant. By aligning the material's texture tiling (the repeating pattern of the material) to the world space instead of to individual objects, then the multiple overlapping meshes' materials will be perfectly aligned with one another.

After achieving roads and junctions being automatically generated from any selected OpenDRIVE file, it was important to be able to create a landscape that would enclose the whole of the road network, hence the maximum and minimum coordinates of the network were calculated and then a landscape mesh was created around the network, as seen in Figure 2.

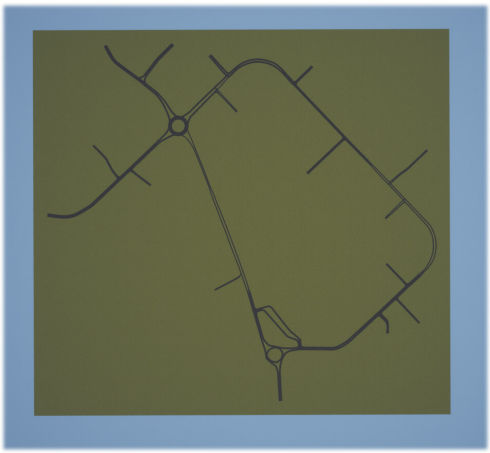

Fig. 2. Top Down View of a Road Network and Landscape Automatically Generated

One of the benefits of using Unreal Engine is that it is an Open Source game engine. As such, there is an ability to integrate many different aspects together, whether through plugins and content packs from the Unreal Engine marketplace or through adding custom logic by writing code directly in C++ and blueprints, which is Unreal Engine's visual scripting language. For creating road markings specified along the road, an Unreal Engine content pack was incorporated into the project, which contained the meshes and materials for road markings. However, the logic for how to generate them automatically based off an OpenDRIVE file still needed to be determined. There are two types of road markings, solid-line and broken-line, which require different methods for how they are generated in a 3D environment. Solid lines were created using stretched meshes whilst broken lines were procedurally placed, using the OpenDRIVE defined spacing for each road mark.

The OpenDRIVE standard [6] also defines object spawning and the automatic spawning was achieved by extracting the positions and rotations of all the objects through the use of the roadmanager and then converting them to the Unreal Engine coordinate and rotation system, and finally spawning them in the 3D environment. In a similar manner, road signs can also be spawned into the environment. As well as individual object spawning, repeated object spawning is an option in an OpenDRIVE description in order to 'avoid lengthy XML code' [24]. In order to place repeating objects along a certain path, they were procedurally placed in a similar manner to the logic of the broken road markings. Figure 3 shows the combination of a curved road, with both broken and solid line road marks, as well as traffic cones as repeating objects.

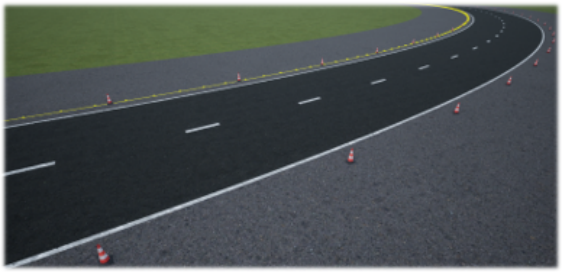

Fig. 3. Scenery Generation showing Repeating Objects along the curvature of the road, along with Solid and Broken Line Roadmarks

B. Achieving a Greater Sense of Realism

A main objective of the project was to be able to create a more realistic scenery compared to the current methods available. Through this, the integration of influences to the environment, such as the date, time and weather is essential, in order to meet ODD requirements. For example, when comparing 21st December at 12pm to 21st June at 12pm, the sun will be in a vastly different position. Hence the date and time information are a major influence in a scenario execution's accuracy, as this can influence how the sensors on the ADS perceive the environment. However, the only available tool that can both generate the scenery and also run the scenario is esmini. This tool doesn't take into account the date or time at all, and simply displays a blue background to the scenario. Hence the lighting system is very simplistic.

Unreal Engine's lighting system, is very comprehensive when using a global illumination system called 'Lumen' [25], available in Unreal Engine 5. Compared to a default Ray Tracing Global Illumination (seen on Unreal Engine 4 for

example), which isn't "reliable or performant" [25] on real-time high detail systems, Lumen is able to render "diffuse interreflection with infinite bounces and indirect specular reflections in large, detailed environments at scales ranging from millimeters to kilometers" [25], enabling "high-end visualizations" [26].

Photo-scanned meshes can be used within the Lumen lighting system in order to add to the realism of a digital twin. There is a large scalability of photo-realistic (photo-scanned) assets, due to a selection of readily available photo-scanned assets. However users can also import 3D assets that they have created themselves. Hence, by combining a photo-scanning system with the simulator, there is the ability to include custom assets of any object in the real world and bring it straight into the digital twin environment.

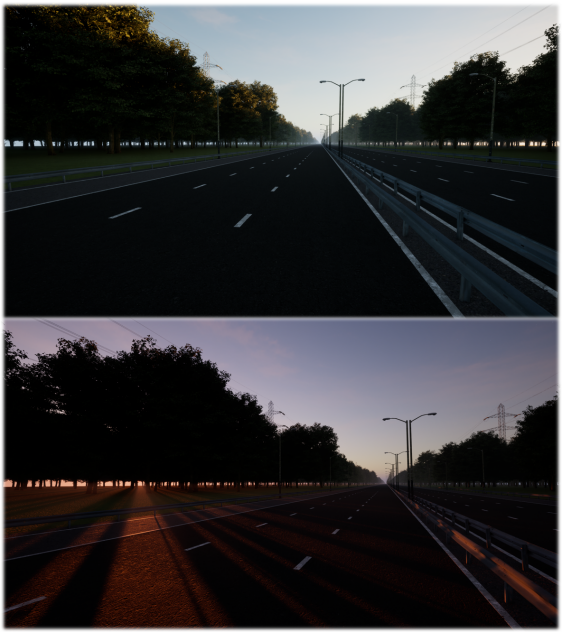

Fig. 4. Sunrise and Sunset in Real-time with Lumen Lighting

However, on top of this, the 3D assets inside the digital twin will also need to be viewed in different environmental conditions and still have realism. By integrating and modifying an environmental conditions plugin, a greater realism can be achieved for the simulator platform when compared with other OpenDRIVE and OpenSCENARIO simulators available. This includes being able to set the exact date (including the time, day, month and year) for a scenario and have a realistic real-time representation of the sun or moon position, as seen in Figure 4. Having high-fidelity assets that replicate real world assets ensures that the sensors on the vehicle can perceive the real world environment in a similar manner to the virtual environment.

As well as this, the weather can be modified in real-time, such as adjusting the volume of clouds, rain, snow and fog; there are other weather features that can be used as well, such as thunder and lightning, rainbows and sandstorms. These parameters can be adjusted individually, or in any

Fig. 5. Rain and Snow in 3D Environment with Dynamic Materials

combination, so that a user could have both snow and rain in the environment simultaneously if desired. By linking the material logic to the weather functions, the materials can dynamically react to the weather, such as by accumulating snow or by getting wet from rain, as seen in Figure 5.

C. Optimisation

Road generation was initially done in sequential order, however it was possible to do this in parallel through the use of multi-threading, which would reduce the generation time needed to produce the scenery. In unreal engine, there is a game thread, which runs gameplay logic, such as rendering and spawning actors. Then there are background threads, which can be used for other tasks, such as calculating values. In the road generation function, some code required this game thread, such as the spawning of the meshes for the roads, but for code which does not depend on the game thread, a mechanism can be used to automatically delegate non-game-thread-essential code to background threads. Therefore, all of this logic calculating how and where to place the road network is done in background threads and hence roads can be generated in parallel. In one test (consisting of a scene including a 500 metre straight road with 20 to 30 objects and road signs), the road generation time was 4.18 seconds without multithreading and 0.04 seconds with multithreading, over 100 times faster. When stress-testing the time required to generate a scenery from an OpenDRIVE file, it was discovered to take under 7 seconds to generate a 10km three lane motorway populated with over 14000 trees and more than 1000 lampposts.

With the photo-scanned meshes and the need to use a range of different objects within the environment, an efficient strategy of how to organise and create an easily expandable

object list was necessary. By using a custom-built library which maps object names to mesh references, an object's OpenDRIVE name could be input as a string variable and then the desired object could be selected, which meant adding a new object and having it link and be generated from an OpenDRIVE file would now only take a few minutes.

D. Integration of End-to-End Framework

As well as having automated scenery generation for an OpenDRIVE file, it was important to be able to retrieve or create a scenario (OpenDRIVE and OpenSCENARIO file), and then be able to execute the scenario, all from within the same tool. In Figure 6, the overarching framework is shown and the Visualisation block (Unreal Engine) is designed to retrieve scenarios through the Scenario Engine and then generate the scenery (.xodr) or execute the scenario (.xosc). This can be achieved through the creation and use of a user interface (UI), as seen in Figure 7.

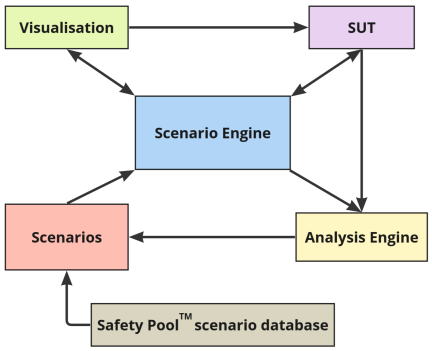

Fig. 6. Simulation Framework for Execution and Analysis of Scenarios

Fig. 7. Simulation GUI which shows the Retrieve, Create (Simple) and Create (Advanced) buttons for acquiring an OpenDRIVE/OpenSCENARIO file.

By separating the game into levels, the UI could display a loading screen ('loading screen level') whilst the scenery is generating in the background and then show the 'scenery level' once it has finished loading. Once the scenery level had loaded, the user can then 'pause' the simulator, which overlays the UI on top of the scenery world whilst they select their next scenery or scenario to load.

Safety Pool^{TM} Scenario Database (SPSD) [27], which is the 'largest public store of scenarios for testing automated vehicles', was used for the retrieval of scenarios and was accessed through the use of its API. With the user entering their SPSD API key and desired scenario collection (test suite number), a C++ function called a python function that accessed the API and then the scenarios from a test suite could be retrieved, being available to use immediately, all from within the UI (see Figure 7).

For the creation of scenarios, the UI was developed to integrate a simple to use tool called 'Safety Pool Studio' [28] within the UI, which allows users with even little experience to design scenarios by use of a graphical interface. Another more advanced tool for the creation of scenarios via an application was also able to be integrated into the UI and then newly formed scenarios and sceneries could be executed directly after designing them. With the expandability and integration capabilities of this simulation platform, even more tools could be introduced and accessed through the UI.

The next part of the end-to-end framework to integrate was the dynamic aspect, namely the "complex, synchronized maneuvers that involve multiple entities like vehicles, pedestrians and other traffic participants" [11], as seen in Figure 8.

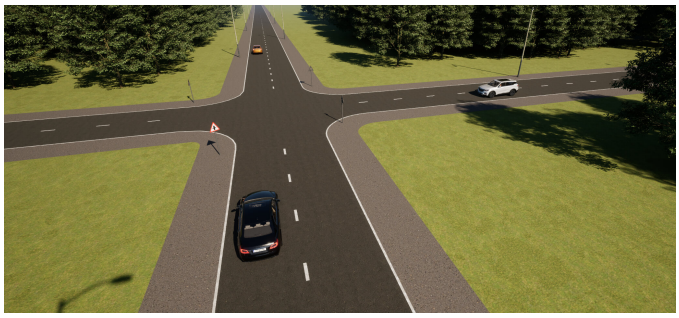

Fig. 8. Execution of a Crossroad Scenario containing Three Vehicles

Some work prior to this project had been done on the dynamic elements and the process is to call a python function from C++ which then sends positional data from esmini to Unreal Engine via a communication layer. This is then interpreted by an Unreal Engine blueprint to position and move the actors in the scenario according to the OpenSCENARIO file. As well as executing the OpenSCENARIO in it's default mode, there was also another option which allowed the scenario to run with the use of an analysis engine. This used a bayesian optimisation algorithm to tweak the dynamic parameters of a scenario and analyse the system.

IV. Conclusion

A. Summary of Work Achieved

In this paper, a unique solution to the need for an iterative realistic digital twin environment for ADS testing was proposed and demonstrated. This solution was built on the platform of Unreal Engine and contains runtime scenery generation, including photoscanned meshes for objects, a

complex lighting setup and the setting of the time of day, date, location and weather. These environmental conditions are able to also be updated in real-time. The simulator benefits from photoscanned meshes and is easily extendable. The platform also has the ability to connect with the world's largest scenario database, SPSD [27], to retrieve scenarios, as well as integrating scenario creation tools for users to make custom scenarios. The user can then generate either a scenery or execute a scenario (with or without an analysis engine), so it achieves an end-to-end framework.

B. Recommendations Towards the Industry

To have a coherent workflow from the ODD, to the scenarios, to the executing environment, to the simulation data, it is essential for the wider industry to use a common taxonomy with standardised terms and definitions within the field, otherwise interpretation and translation will need to be put in place to enable an end-to-end scenario based testing process.

C. Limitations and Future Work

With the current simulation platform, it is important to compare the digital twin and the real world to determine the accuracy of the digital twin, therefore future work will need to be performed to validate the digital twin. Another limitation is that the landscape used for the scenarios is completely flat, which doesn't mimic real world landscapes, thus future exploration into dynamic landscapes will enhance the realism. There needs to be future work undertaken into the integration of sensors into Unreal Engine, as this aspect also needs to mimic real world sensors, however the potential of this sensor work is large as the complex meshes and lighting system will influence the sensors in a more realistic way. Looking into the longer term future, the goal would be to be able to interchange any ADS and ODD with the platform in order to verify and validate the system's safety.

ACKNOWLEDGEMENT

The work presented in this paper has been supported by UKRI Future Leaders Fellowship (Grant MR/S035176/1), Department of Transport, UK, and Transport Canada (T8080-220112). The authors would like to thank the WMG center of HVM Catapult, and WMG, University of Warwick, UK for providing the necessary infrastructure for conducting this study. For the purpose of open access, the author(s) has applied a Creative Commons Attribution (CC BY) license to any Accepted Manuscript version arising. No new data was created in this study.

REFERENCES

[1] 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, no. December, pp. 182–193, 2016.

[2] X. Zhang, S. Khastgir, H. Asgari, and P. Jennings, "Test Framework for Automatic Test Case Generation and Execution Aimed at Developing Trustworthy AVs from Both Verifiability and Certifiability Aspects," in *2021 IEEE ITSC*, 2021, pp. 312–319.

[3] E. Esenturk, D. Turley, A. Wallace, S. Khastgir, and P. Jennings, "A data mining approach for traffic accidents, pattern extraction and test scenario generation for autonomous vehicles," in *International Journal of Transportation Science and Technology*, 2023, pp. 955–972.

[4] S. Khastgir, S. Birrell, G. Dhadyalla, and P. Jennings, "Effect of knowledge of automation capability on trust and workload in an automated vehicle: a driving simulator study," in *Advances in Human Aspects of Transportation: Proceedings of the AHFE 2018 International Conference on Human Factors in Transportation, July 21-25, 2018, Loews Sapphire Falls Resort at Universal Studios, Orlando, Florida, USA 9*, 2019, pp. 410–420.

[5] S. Khastgir, H. Sivencrona, G. Dhadyalla, P. Billing, S. Birrell, and P. Jennings, "Introducing ASIL inspired dynamic tactical safety decision framework for automated vehicles," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 1–6.

[6] ASAM e.V., "OpenDRIVE V1.8.0," 2023. [Online]. Available: https://www.asam.net/standards/detail/opendrive/

[7] ——, "OpenScenario DSL V2.1.0," 2024. [Online]. Available: https://www.asam.net/standards/detail/openscenario-dsl/

[8] X. Zhang, S. Khastgir, and P. Jennings, "Scenario Description Language for Automated Driving Systems: A Two Level Abstraction Approach," in *2020 IEEE SMC*, 2020, pp. 973–980.

[9] BSI, "BSI Flex 1889 - Natural language description for abstract scenarios for automated driving systems - Specification," 2022.

[10] F. Bock, C. Sippl, A. Heinzz, C. Lauerz, and R. German, "Advantageous usage of textual domain-specific languages for scenario-driven development of automated driving functions," in *2019 IEEE SysCon*, 2019, pp. 1–8.

[11] ASAM e.V., "ASAM OpenSCENARIO XML," 2022. [Online]. Available: https://www.asam.net/standards/detail/openscenario-xml/

[12] A. Wallace, S. Khastgir, X. Zhang, S. Brewerton, B. Anctil, P. Burns, D. Charlebois, and P. Jennings, "Validating Simulation Environments for Automated Driving Systems Using 3D Object Comparison Metric," in *2022 IEEE IV*. IEEE, 2022, pp. 860–866.

[13] ASAM e.V., "ASAM XIL." [Online]. Available: https://www.asam.net/standards/detail/xil

[14] S. Khastgir, S. Birrell, G. Dhadyalla, and P. Jennings, "The Science of Testing: An Automotive Perspective," in *SAE Technical Paper 2018-01-1070*. SAE International, 2018.

[15] BSI, "PAS 1883 Operational Design Domain (ODD) taxonomy for an automated driving system (ADS) – Specification," 2020.

[16] ASAM e.V., "ASAM OpenXOntology." [Online]. Available: https://www.asam.net/standards/asam-openxontology

[17] ——, "OpenLABEL V1.0.0," 2021. [Online]. Available: https://www.asam.net/standards/detail/openlabel/

[18] ISO, "ISO 34503:2023 - Road Vehicles - Test scenarios for automated driving systems - Specification for operational design domain," 2023.

[19] ASAM e.V., "ASAM OSI." [Online]. Available: https://www.asam.net/standards/detail/osi

[20] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[21] CARLA, "OpenDRIVE - CARLA Simulator." [Online]. Available: https://carla.readthedocs.io/en/latest/adv_opendrive/

[22] E. Knabe, "Environment Simulator Minimalistic (esmini)." [Online]. Available: https://github.com/esmini/esmini

[23] Bertrand Richard, "OpenDrive Plugin for Unreal Engine." [Online]. Available: https://github.com/brifsttar/OpenDRIVE

[24] ASAM e.V., "ASAM OpenDrive Repeating Objects." [Online]. Available: https://publications.pages.asam.net/standards/ASAM_OpenDRIVE/ASAM_OpenDRIVE_Specification/latest/specification/13_objects/13_02_repeating_objects.html

[25] Unreal Engine, "Lumen Global Illumination and Reflections." [Online]. Available: https://docs.unrealengine.com/5.3/en-US/lumen-global-illumination-and-reflections-in-unreal-engine

[26] D. Wright and K. Narkowicz, "Unreal Engine 5 goes all-in on dynamic global illumination with Lumen." [Online]. Available: https://www.unrealengine.com/en-US/tech-blog/unreal-engine-5-goes-all-in-on-dynamic-global-illumination-with-lumen

[27] "Safety Pool^{TM} Scenario Database." [Online]. Available: https://safetypooldb.ai

[28] WMG, "Safety Pool Studio." [Online]. Available: https://safetypoolstudio.ai/