energies

MDPI

Article

Software Framework for Testing of Automated Driving Systems in the Traffic Environment of Vissim

Demin Nalic [1,*], Aleksa Pandurevic [1], Arno Eichberger [1], Martin Fellendorf [2] and Branko Rogic [3]

[1] Institute of Automotive Engineering, TU Graz, 8010 Graz, Austria; pandurevic@tugraz.at (A.P.); arno.eichberger@tugraz.at (A.E.)
[2] Institute of Highway Engineering and Transport Planning, TU Graz, 8010 Graz, Austria; martin.fellendorf@tugraz.at
[3] MAGNA Steyr Fahrzeugtechnik AG Co. & KG, 8045 Graz, Austria; branko.rogic@magna.com
* Correspondence: demin.nalic@tugraz.at

Abstract: As the complexity of automated driving systemss (ADSs) with automation levels above level 3 is rising, virtual testing for such systems is inevitable and necessary. The complexity of testing these levels lies in the modeling and calculation demands for the virtual environment, which consists of roads, traffic, static and dynamic objects, as well as the modeling of the car itself. An essential part of the safety and performance analysis of ADSs is the modeling and consideration of dynamic road traffic participants. There are multiple forms of traffic flow simulation software (TFSS), which are used to reproduce realistic traffic behavior and are integrated directly or over interfaces with vehicle simulation software environments. In this paper we focus on the TFSS from PTV Vissim in a co-simulation framework which combines Vissim and CarMaker. As it is a commonly used software in industry and research, it also provides complex driver models and interfaces to manipulate and develop customized traffic participants. Using the driver model DLL interface (DMDI) from Vissim it is possible to manipulate traffic participants or adjust driver models in a defined manner. Based on the DMDI, we extended the code and developed a framework for the manipulation and testing of ADSs in the traffic environment of Vissim. The efficiency and performance of the developed software framework are evaluated using the co-simulation framework for the testing of ADSs, which is based on Vissim and CarMaker.

Keywords: automated driving; scenario-based testing; software framework

Citation: Nalic, D.; Pandurevic, A.; Eichberger, A.; Fellendorf, M.; Rogic, B. Software Framework for Testing of Automated Driving Systems in the Traffic Environment of Vissim. *Energies* **2021**, *14*, 3135. https://doi.org/10.3390/en14113135

Academic Editor: Wiseman Yair

Received: 2 April 2021
Accepted: 21 May 2021
Published: 27 May 2021

1. Introduction

The use of TFSS in automotive engineering has significantly improved the scope of the virtual testing of ADS. It is mostly used in co-simulation with other software tools for vehicle testing and simulation. There are various co-simulation platformss (CSPs) for the testing of ADSs in complex traffic environments. Hallerbach presented in [1] a simulation-based tool-chain to identify critical scenarios using a SUMO and a vehicle dynamic software. A framework coupling SUMO with vehicle dynamic software VTD for the development of ADSss is presented in [2]. In [3], a human-driven car from SILAB interacted over an interface with SUMO traffic participants in order to evaluate human interactions and the effect of ADSs in traffic. Implementing automated driving functions in MATLAB and coupling this with Vissim, an impact analysis of ADSs is performed in [4]. The CSP used in this work is based on the co-simulation between Vissim and CarMaker and is explained in greater detail below; see [5]. Common to all these interfaces is the fact that the vehicle under test has been developed separately from a certain vehicle simulation software. The traffic is created externally and imported by means of TFSS. In this study, the vehicle being tested is referred to as an EGO vehicle.

As a form of TFSS, Vissim provides comprehensive traffic flow modeling options and the possibility to manipulate traffic participants, making it suitable for the testing

of ADSs and the generation of safety-relevant scenarios. The term "scenario" is defined in [6]. Focusing on safety-critical scenarioss (SCSs), a variety of research works have offered possible definitions and approaches to determining the criticality and safety relevance of a given scenario, for example [7–9]. The aim of generating and finding SCSs in the realistic and stochastic traffic environment of Vissim is based on the idea that the EGO vehicle drives through the traffic simulated by the TFSS and possibly encounters situations that cannot be resolved by the implemented ADS. The approach based on this idea is beneficial for the virtual testing of an EGO vehicle, since it reflects normal driving in traffic and takes in account all the effects and factors which could potentially occur during a ride. The difficulty of using Vissim or any other TFSS is that it is not guaranteed that a significant amount of SCSs for ADS testing will be generated. To provide a reference and an approximate estimation of the amount and relevance of SCSs, that will occur, distance-based approaches can be used. In [10,11], distance-based testing approaches for ADS are introduced. Based on accident data, the average distance between two accidents was statistically analyzed in order to determine how many kilometers an ADS should be tested in order to achieve the same safety level as a human driver. In both works, the number of kilometers, depending on the accident considered, lies in millions of real-world testing kilometers, which are needed to prove the safety of ADS compared to a human driver. In [12], the distance-based testing is reduced to scenario-based testing. In [12] a statistical method is introduced in order to calculate the number of scenarios required for the same evidence as the approaches presented in [10,11]. These accident rates and scenario amounts can hardly be reached via Vissim because the Vissim driver model relies on tactical driving behavior. This is due to the fact that traffic participants plan their actions with a temporal and spatial horizon; see [13]. In such trajectory planning, the neighboring vehicle is taken into account, as well as vehicles that are far in front of the EGO vehicle. This means that the vehicles in the Vissim traffic simulation have enough of a planning horizon to avoid conflict areas and conflict situations, which is comparable to human driving behavior. Due to this face, the cars collide with each other extremely rarely, and do not make unpredictable movements and maneuvers, which corresponds to the real-life situation. Nonetheless, human driving behavior is in rare cases incorrect, resulting in conflict situations and accidents. Accident statistics suggest that up to 90% of police-reported accidents are mainly caused by human drivers [14]. On the one hand, TFSS-based driving behavior correspondsto the realistic planning of real drivers. However, on the other hand, if we use, e.g., Vissim as a TFSS for the testing and validation of ADSs, as described in [5,15], this would not yield a satisfactory number of SCSs. Another recent study in [16] introduced a method based on deep reinforcement learning to train traffic participants with naturalistic driving data. The main goal of this approach is to train traffic participants in such a way that they produce SCSs and reach accident rates corresponding to those occurring in the real world. With a similar approach and objective, but using a more deterministic approach, we present a software framework for researchers using Vissim for the generation of SCSs.The main objective of this software framework is to offer an appropriate and adjustable environment for testing purposes of ADSs and, more specifically, for the generation of SCSs. The so-called driver model framework (DMF) is based on the DMDI software code provided by Vissim. This code allows Vissim users to manipulate traffic participants in a defined manner. Since the provided Vissim interface and the code lacks explanations, it is complex to use. Therefore, the DMF provides a structured C++ code with a class architecture and useful methods for accessing and setting different vehicle parameters for traffic participants. This provides an optimized environment for the testing and development of ADSs in the complex traffic environment of Vissim. As the main case study to describe the functionality of the DMF, the co-simulation between CarMaker and Vissim described in [4] was used in this study. This framework was adjusted by means of the DMF to utilize the co-simulation for the generation of SCSs. This utilization was carried out by implementing the stress testing method (STM) presented in [17]. The

DMF code itself can also be used independently of Vissim and is available to users for testing purposes and other related applications.

2. Software Description

For the DMF, we used the DMDI provided by PTV Vissim (see [18]). This C++ code contains three essential functions from Vissim: DriverModelSetValue, DriverModelGetValue and DriverModelExecuteCommand. These functions are used by Vissim to provide and retrieve the data from the DMF. The Vissim participants, which are controlled by the DMDI, are called DLL driver modelss (DDMs) hereinafter. The Vissim DriverModelExecuteCommand value is passed upon each simulation step, denoting the action to be taken, such as initialization, driver creation, driver deletion and the driver movement of each traffic participant controlled by the DMDI in the TFSS. In between the commands, multiple calls of DriverModelSetValue and DriverModelGetValue are made for each vehicle controlled by the DDM. The function DriverModelSetValue provides the DDM with the current vehicle values, which can be stored, processed and modified. In order to provide Vissim with new values, which will be used for the vehicle's movement, multiple calls of DriverModelGetValue for each traffic participant are necessary. Building on top of the provided DMDI, DriverModelSetValue and DriverModelGetValue are encapsulated into setInjectorData and getInjectorData, member functions of InjectorAbstract which is further described in the software architecture section.

In these functions, all the necessary logic for extracting, storing and updating of selected vehicles is contained. As the DMF has been created to manipulate traffic participants in the surrounding area of the EGO vehicle, one vehicle in the Vissim traffic is set to be the EGO vehicle by means of the vehicle ID, which is provided by the DMDI. This vehicle has an individual ID, which can be freely defined and set. Using the DMDI in the testing framework presented in [5], the EGO vehicle ID equals 1 and is fixed within the co-simulation between Vissim and CarMaker. For the DMF, which is described in this paper, the area of interest is the surrounding area of the EGO vehicle, which consists of other traffic participants, referred to as nearby vehicles in this work. The DMF concept is depicted in Figure 1. First, all the surrounding area has to be defined, stored and updated continuously through each Vissim simulation step. This is done with the DMF method capture, in which certain nearby vehicles are selected as target vehicles by means of user-defined rules. To manipulate the traffic participants around the EGO vehicle, an action method is defined, which activates a user-defined critical maneuver for the Vissim vehicle. This critical maneuver is performed by the traffic participant from Vissim, which is referred to as the target vehicle and is activated close to the EGO vehicle. The software implementation of the process logic of the DMF is shown in Figure 2. Each Vissim call to the setInjectorData or getInjectorData method passes multiple arguments, one of which always denotes the value which is going to be sent or achieved. After setInjectorData is executed for each value of the EGO vehicle, multiple setInjectorData calls follow, with the values of the nearby vehicles. Thus, information about EGO vehicles and detected nearby vehicles is obtained and can be processed and prepared if needed. The whole setInjectorData and getInjectorData routine is performed for each vehicle controlled by the DMF in each simulation step. The DMF provides the user with the basic methods necessary to read and manipulate the traffic participants relevant to the EGO vehicle being tested. The user has to implement two DMF methods, capture and action, in order to create a usable DLL. A description of the methods capture and action is given in the following points:

- In the capture method, it is guaranteed that nearby vehicles in relation to the EGO vehicle with current values can be queried. The user then defines the rules by which certain or all nearby vehicles are selected as target vehicles and which action is applied during defined time intervals. Duration and pauses between those intervals are also user-configurable.
- In the action method, the user defines an action which will be applied to the target vehicles. Examples of such actions can be a braking maneuver carried out by a traffic

participant which is directly in front of the EGO vehicle, or a cut-in maneuver by the traffic participant in front of the EGO vehicle.

The obvious advantage of the DMF is that the user does not care about monitoring all nearby vehicles, taking care of vehicle IDs, or observing if the nearby vehicles have current values. The tasks of the user are merely to select vehicles of interest and define the action that will be applied.

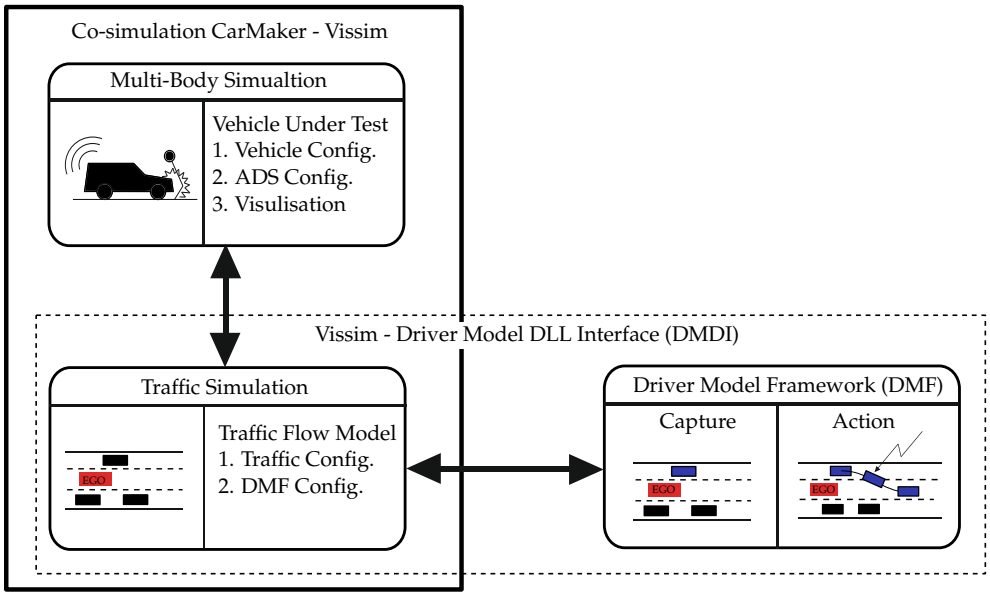

Figure 1. The co-simulation between Vissim and CarMaker of the framework concept in [5] is adapted with the DMF.

2.1. Software Architecture

The DMF is written in C++, encapsulates the provided Vissim code, and provides a simple interface for the user. The class hierarchy can be seen in Figure 3. The InjectorAbstract class comprises the majority of the DMF logic, including setInjectorData and getInjectorData methods, which are called from Vissim. They are required to be a part of InjectorAbstract, since the whole DMF logic depends on them, reading out the data, storing it, scheduling the execution of capture and action methods and passing new data to Vissim. The Injector class is a child class of the InjectorClass, which implements the pure virtual methods capture and action. This inheritance serves to keep users away from the basic Vissim DLL code of the framework and provides them with simple vehicle manipulation. The user-code is separated from the DMF logic, and it is simple for the user to write and organize. Aside from that, the user is free to extend the functionality of NearbyVehicle and EgoVehicle classes, which contain the data of the respective vehicle categories.

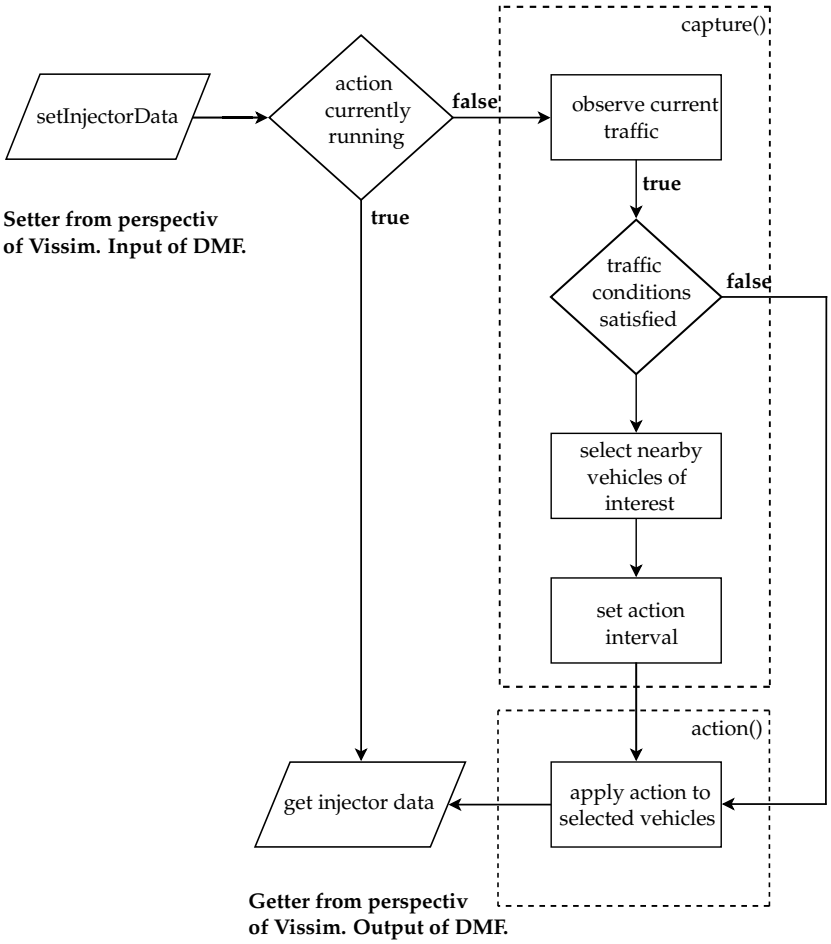

Figure 2. The process logic and the main components of the DMF.

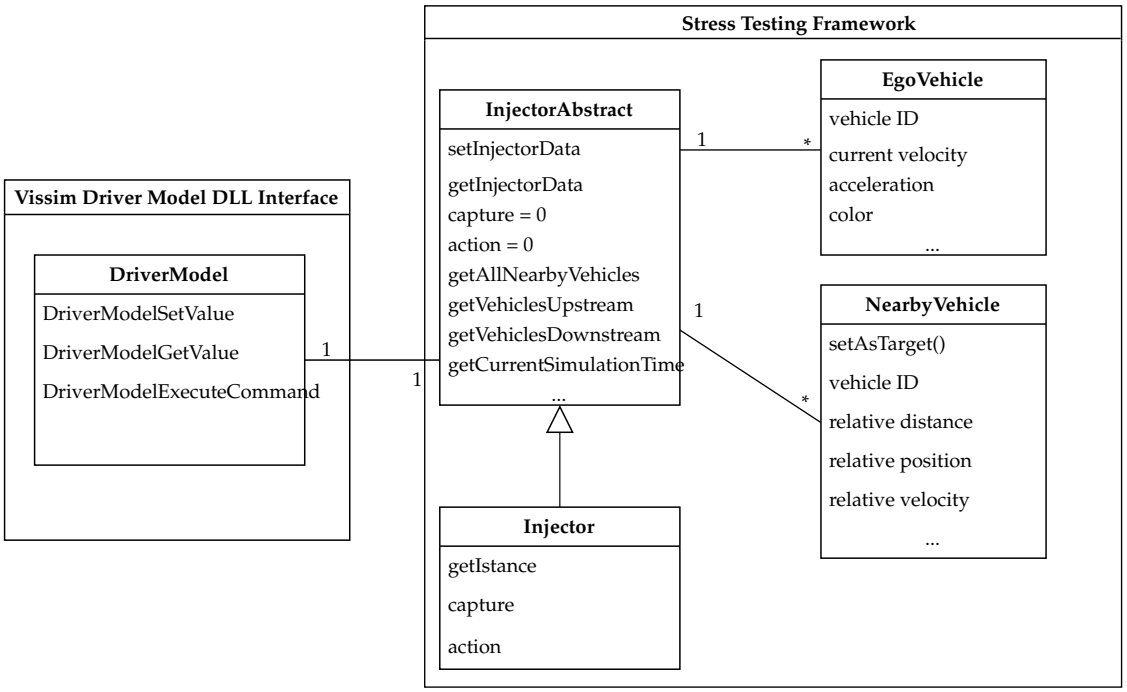

Figure 3. A simplified view of the framework class hierarchy.

2.2. Software Functionalities

As mentioned in the software description, the user has to fill in the code for the capture and action methods. The methods provided by the DMF upon which the user depends are:

- getAllNearbyVehicles
- getVehiclesDownstream
- getVehiclesUpstream
- getCurrentSimTime

The first three methods return a vector containing nearby vehicles either all of them or only vehicles in front of vehicles behind the EGO vehicle. Each vehicle is represented by the NearbyVehicle object, which contains all the relevant information about the vehicle, such as relative distance and speed to the EGO vehicle, acceleration, relative position, and other vehicle states or parameters. These methods are meant to be used in the capture method, in which the user is observing the traffic and labeling relevant vehicles as target vehicles. When the user-defined criteria are fulfilled, the startAction method is to be implemented, with a specified action duration time and an optionally selected pause time after the action's end. During the time of the respective action, the takes control of the target vehicles. After the action time expires, the TFSS internal model takes back the control of the target vehicles. Through these methods, the DMF provides the user with the following main functionalities:

- Providing the user with the updated list of nearby vehicles;
- Offering the user the ability to select target vehicles and manipulate them;
- Keeping the list of target vehicles updated during the action;
- Scheduling the *capture* and *action* methods in time;
- Taking and releasing the control of the vehicles from and to the user;

Another essential feature of the DMF is that it has two separate modes of operation. In both of these modes, previously defined functionalities are present, but they require different approaches and implementations for different modes of operation. The DMF can operate both in Vissim only and in co-simulation between CarMaker and Vissim.

3. Use Case Application of the DMF

As introduced in Section 2, the use case for the DMF refers to the co-simulation between CarMaker and Vissim software, adjusting Vissim with the use of the DMF. The concept of the co-simulation framework is depicted in Figure 4 and the adjustment of Vissim using the DMF has already been introduced in Figure 1.

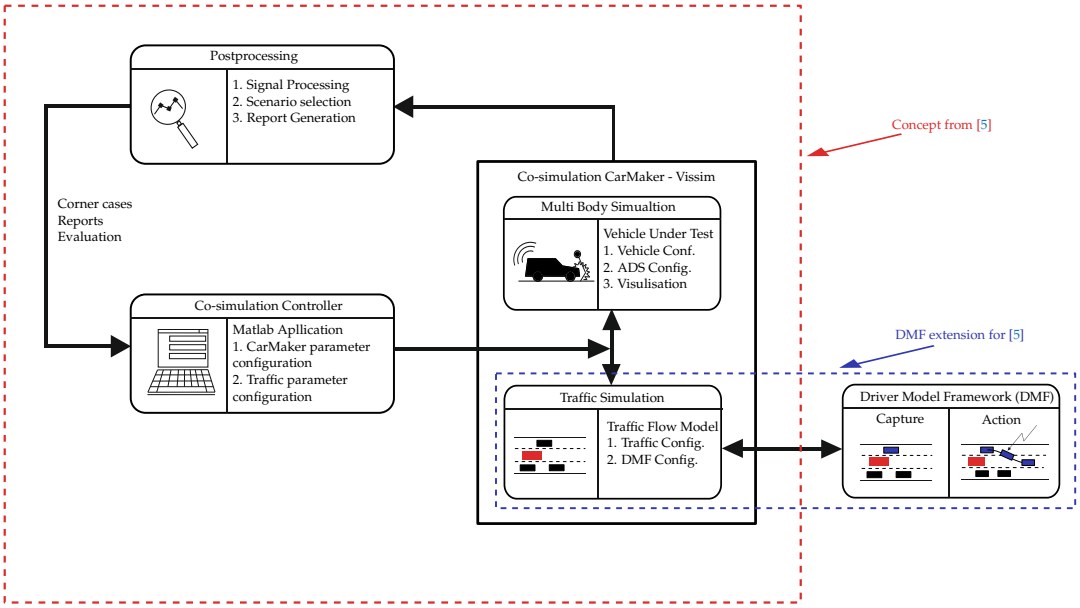

Figure 4. The concept of the co-simulation framework from [5] adjusted with the DMF from Figure 1.

This adjustment using the DMF was realized by implementing the STM which was introduced in [17]. Defined maneuvers were provoked in the vicinity of an EGO vehicle in order to force the tested vehicle into a challenging situation for the ADS. Using statistical data from accidents in Austria, the maneuvers which were provoked were classified into lateral and longitudinal maneuvers. These longitudinal and lateral maneuvers were implemented using the DMF, in which they could be parameterized and used for the generation of SCSs in the vicinity of the EGO vehicle. In [17] the comparison between a testing procedure with and without the STM is shown. In this testing procedure, an EGO vehicle equipped with adaptive cruise control and an automated lance change algorithm was tested on 10,000 simulation kilometers. The SCSs considered for evaluation purposes were collisions, and the criticality assessment criteria were those defined in [8]. It was observed that collisions could be generated, and very critical and critical scenarios from [8] increased by 1859 and 2320 over the course of 10,000 simulation kilometers, respectively. The second use case of the framework involves using the DMF only with Vissim. As a result of this, a vehicle with automated driving functions could be developed and implemented in the traffic environment, using the same DMDI from Vissim. In [19], a longitudinal control unit was developed using the Vissim DMDI in order to test the performance of automated vehicles on a single-lane road. A similar approach is presented in [20], where the impact of an emergency control function on mobility and safety was evaluated. The research work presented in [21,22] emphasizes the usage of the DLL interface for the analysis and evaluation of connected and autonomous vehicles in traffic. Using it for the purpose of testing an ADS, the advantage of this approach lies in the avoidance of couplings with other vehicle simulation software tools, such as CarMaker, VTD and others. In this case, the simulation times, implementation efforts and the need for additional tools could be decreased. A possible disadvantage of this approach is the simple point mass models of vehicles provided in Vissim. This issue can also be solved by developing and integrating single-track or more complex vehicle models using the same DMDI interface. Table 1 provides the software specifications of the the DMF in the default version. For the DMF with the implementation of the STM, the code information is provided in Table 2. The first use case requires IPG CarMaker, and for the second use case it is possible to implement the provided DMDI directly in Vissim on any test road. The simple braking example with the STM and DMF are provided for research and development purposes.

Table 1. DMF software information and code link for the default DMF.

Nr.	Code Metadata Description	Description
C1	Current code version	v1
C2	Permanent link to code/repository used for this code version	https://github.com/ftgTUGraz/DriverModel_Framework (accessed on 14 February 2021)
C3	Code Ocean compute capsule	none
C4	Legal Code License	GPL-3.0 License
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	C++, PTV Vissim 11.00-14
C7	Compilation requirements, operating environments & dependencies	Visual Studio 2019

Table 2. DMF software information and code link for the implementation of the STM.

Nr.	Code Metadata Description	Description
C1	Current code version	v1
C2	Permanent link to code/repository used for this code version	https://github.com/ftgTUGraz/DriverModel_STM (accessed on 14 February 2021)
C3	Code Ocean compute capsule	none
C4	Legal Code License	GPL-3.0 License
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	C++, IPG CarMaker 8.1.1 (optional), PTV Vissim 11.00-14
C7	Compilation requirements, operating environments, & dependencies	Visual Studio 2019

4. Conclusions

With the presented driver model framework, the user can implement and adjust driver models for traffic participants using the traffic flow simulation software Vissim. By that, traffic participants for testing purposes of a particular automated driving system of the vehicle under test can be tested on a virtual basis. Using the co-simulation between Vissim and CarMaker, a use case of the presented software framework was shown. An upgrade of the co-simulation with the presented framework for testing automated driving systems increases the benefit of the Vissim and CarMaker co-simulation environment. For future research, the software framework will be adjusted for implementing more realistic vehicle dynamics to the neighboring traffic vehicles, including vehicle based environment sensors as well for improved model validity and Vehicle-to-X communication for impact analysis of automated driving systems on traffic.

Author Contributions: Conceptualization, D.N.; methodology, D.N. and A.P.; software, D.N. and A.P.; validation, A.E., B.R., and D.N.; formal analysis, D.N. and B.R.; investigation, D.N.; resources, D.N., M.F., and B.R.; data curation, D.N.; writing—original draft preparation, D.N., M.F., and A.P.; writing—review and editing, A.E., M.F., and B.R.; visualization, D.N.; supervision, A.E. and B.R. All authors have read and agreed to the published version of the manuscript.

Funding: This work is founded by the Austrian Federal Ministry of Transport, Innovation and Technology as part of the FFG Program "EFREtop".

Data Availability Statement: Not applicable.

Acknowledgments: Open Access Funding by the Graz University of Technology

Conflicts of Interest: No conflict of interest exists: We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

References

1. Hallerbach, S.; Xia, Y.; Eberle, U.; Koester, F. Simulation-Based Identification of Critical Scenarios for Cooperative and Automated Vehicles. *SAE Int. J. Connect. Autom. Veh.* **2018**, *1*, 93–106. [CrossRef]
2. Semrau, M.; Erdmann, J. Simulation Framework for Testing ADAS in Chinese Traffic Situations. In Proceedings of the 2016 SUMO User Conference, Berlin, Germany, 23–25 May 2016; pp. 103–115.
3. Barthauer, M.; Hafner, A. Coupling traffic and driving simulation: Taking advantage of SUMO and SILAB together. *EPiC Ser. Eng.* **2018**, *2*, 56–66. [CrossRef]
4. Haberl, M.; Fellendorf, M.; Rudigier, M.; Kerschbaumer, A.; Eichberger, A.; Rogic, B.; Neuhold, R. Simulation assisted impact analyses of automated driving on motorways for different levels of automation and penetration. In Proceedings of the Mobil.TUM 2017, Munich, Germany, 4–5 July 2017.

5. Nalic, D.; Eichberger, A.; Hanzl, G.; Fellendorf, M.; Rogic, B. Development of a Co-Simulation Framework for Systematic Generation of Scenarios for Testing and Validation of Automated Driving Systems. In Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 27–30 October 2019; pp. 1895–1901. [CrossRef]
6. Ulbrich, S.; Menzel, T.; Reschka, A.; Schuldt, F.; Maurer, M. Defining and Substantiating the Terms Scene, Situation, and Scenario for Automated Driving. In Proceedings of the 2015 IEEE 18th International Conference on Intelligent Transportation Systems, Las Palmas, Gran Canaria, Spain, 15–18 September 2015; pp. 982–988. [CrossRef]
7. Mugur, T. Enhancing ADAS Test and Validation with Automated Search for Critical Situations. In Proceedings of the DSC 2015, Berlin, Germany, 16–18 September 2015.
8. Junietz, P.; Bonakdar, F.; Klamann, B.; Winner, H. Criticality Metric for the Safety Validation of Automated Driving using Model Predictive Trajectory Optimization. In Proceedings of the IEEE Conference on Intelligent Transportation Systems, ITSC, Maui, HI, USA, 4–7 November 2018; pp. 60–65.
9. Feng, S.; Feng, Y.; Sun, H.; Bao, S.; Zhang, Y.; Liu, H.X. Testing scenario library generation for connected and automated vehicles, part II: Case studies. *IEEE Trans. Intell. Transp. Syst.* **2020**, 1–13. [CrossRef]
10. Kalra, N.; Paddock, S.M. *Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?* RAND Corporation: Santa Monica, CA, USA, 2016. Available online: https://www.rand.org/pubs/research_reports/RR1478.html (accessed on 14 February 2021).
11. Wachenfeld, W.; Winner, H. The Release of Autonomous Vehicles. In *Autonomous Driving: Technical Legal and Social Aspects*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 425–449.
12. Amersbach, C.; Winner, H. Defining Required and Feasible Test Coverage for Scenario-Based Validation of Highly Automated Vehicles. In Proceedings of the 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 27–30 October 2019; pp. 425–430. [CrossRef]
13. Fellendorf, M.; Vortisch, P. Microscopic Traffc Flow Simulator VISSIM. In *Fundamentals of Traffic Simulation*, 1st ed.; International Series in Operations Research & Management Science; Springer Science + Business Media: Berlin, Germany, 2010; Volume 145, pp. 63–94.
14. Statistisches Bundesamt. Verkehr: Verkehrsunfälle. Technical Report Reihe 7, Statistisches Bundesamt. 2018. Available online: https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Verkehrsunfaelle/Publikationen/Downloads-Verkehrsunfaelle/verkehrsunfaelle-jahr-2080700177004.pdf (accessed on 16 February 2021).
15. Nalic, D.; Pandurevic, A.; Eichberger, A.; Rogic, B. Design and Implementation of a Co-Simulation Framework for Testing of Automated Driving Systems. *Sustainability* **2020**, *12*, 10476. [CrossRef]
16. Feng, S.; Yan, X.; Sun, H.; Feng, Y.; Liu, H.X. Intelligent driving intelligence test for autonomous vehicles with naturalistic and adversarial environment. *Nat. Commun.* **2021**, *12*, 748. [CrossRef] [PubMed]
17. Nalic, D.; Li, H.; Eichberger, A.; Wellershaus, C.; Pandurevic, A.; Rogic, B. Stress Testing Method for Scenario-Based Testing of Automated Driving Systems. *IEEE Access* **2020**. [CrossRef]
18. Barceló, J. (Ed.) *Fundamentals of Traffic Simulation*; Springer: NewYork, NY, USA, 2010; Volume 145.
19. Wang, Y.; Wang, L. Autonomous vehicles' performance on single lane road: A simulation under VISSIM environment. In Proceedings of the 2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), Shanghai, China, 14–16 October 2017; pp. 1–5. [CrossRef]
20. Thus, J.J.; Kang, J.; Park, S.; Park, I.; Lee, J. Automated emergency vehicle control strategy based on automated driving controls. *J. Adv. Transp.* **2020**, *2020*, 3867921.
21. Ard, T.; Dollar, R.A.; Vahidi, A.; Zhang, Y.; Karbowski, D. Microsimulation of energy and flow effects from optimal automated driving in mixed traffic. *Transp. Res. Part C Emerg. Technol.* **2020**, *120*, 102806. [CrossRef]
22. Papadoulis, A.; Quddus, M.; Imprialou, M. Evaluating the safety impact of connected and autonomous vehicles on motorways. *Accid. Anal. Prev.* **2019**, *124*, 12–22. [CrossRef]