Article

Design and Implementation of a Co-Simulation Framework for Testing of Automated Driving Systems

Demin Nalic[1,†]*, Aleksa Pandurevic [2,†], Arno Eichberger[3,†] and Branko Rogic[4,‡]

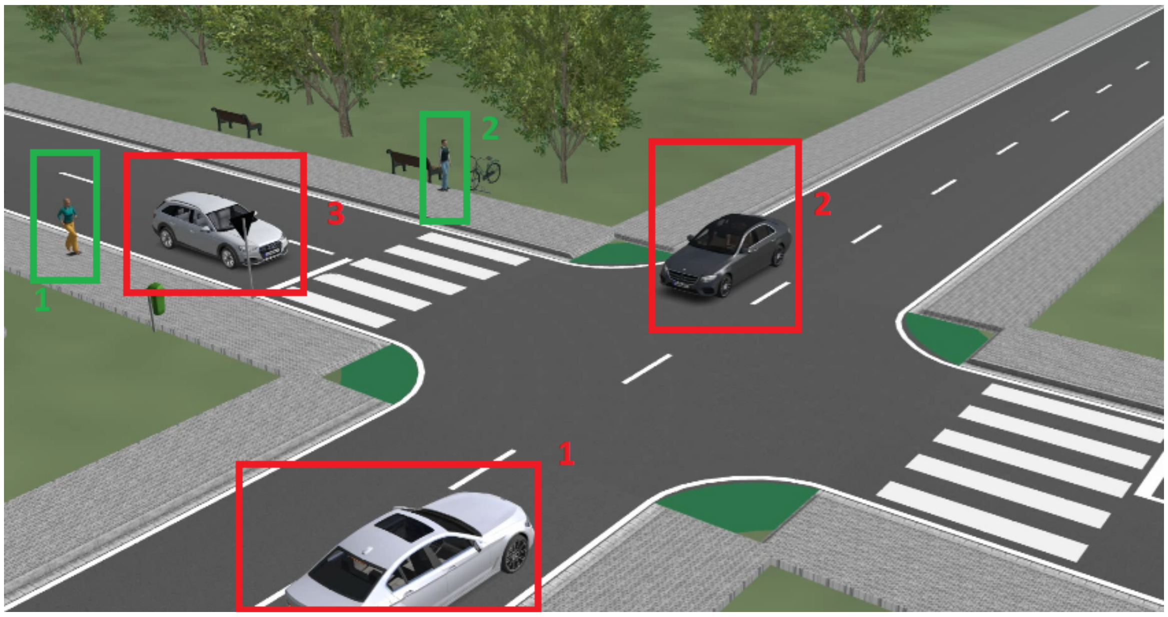
1 Institute of Automotive Engineering, TU Graz; demin.nalic@tugraz.at
2 Institute of Automotive Engineering, TU Graz; pandurevic@tugraz.at
3 Institute of Automotive Engineering, TU Graz; arno.eichberger@tugraz.at
4 MAGNA Steyr Fahrzeugtechnik AG Co & KG; branko.rogic@magna.com
† Current address: Inffeldgasse 11/2, 8010 Graz, Austria
‡ Current address: Liebenauer Hauptstraße 317, 8041 Graz, Austria

Abstract: The increasingly used approach of combining different simulation software in testing of automated driving systems (ADS) increases the need for potential and convenient software designs. Recently developed co-simulation platforms (CSP) provide the possibility to cover the high demand on testing kilometers for ADS by combining vehicle simulation software (VSS) with traffic flow simulation software (TFSS) environments. The emphasis on the demand of testing kilometers is not enough to choose a suitable CSP. The complexity level of the used vehicle, object, sensors and environment models is essential for valid and representative simulation results. Choosing a suitable CSP raises the question of how the test procedures should be defined and constructed and what the relevant test scenarios are. Parameters of the ADS, the environments, objects, sensors in VSS as well as traffic parameters in TFSS can be used to define and generate test scenarios. In order to generate a large number of scenarios in a systematic and automated way, suitable and appropriate software designs are required. In this paper we present a software design for CSP based on the Model-View-Controller (MVC) design pattern and implementation of a complex CSP for virtual testing of ADS. Based on this design, an implementation of a CSP is presented using the VSS from IPG Automotive called CarMaker and the TFSS from PTV Group called Vissim. The results have shown that the presented CSP design and the implementation of the co-simulation can be used to generate relevant scenarios for testing of ADS.

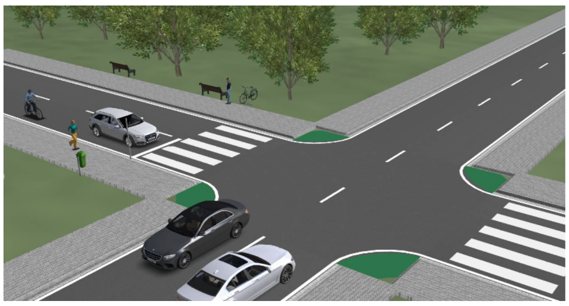
Keywords: ADAS simulation; scenario generation; automated driving; Testing; innovation in mobility; self-driving cars; transportation

1. Introduction

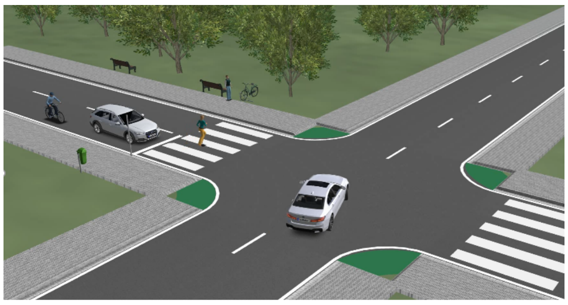
Testing and validation procedures are essential for safety approval and acceptance of ADS. Recent studies have shown that the main concerns regarding the acceptance of ADS in Austria are reliability and safety, see [1]. For ADS with Level 3+ automation levels (defined as in [2]), the test and validation procedures are very complex. The high demand on testing kilometers as well as real-word testing (RWT) is time and resource-consuming, see [3], [4] and [5]. Besides the costs and time demand for RWT, future technologies presented in [6] and impact analysis of ADS on traffic [7],[8] are very difficult to implement and test in real world. To illustrate the complexity of RWT for a special scenario including several participants, we present an example for an intersection in Fig. 1. In it, we have 3 different cars and 2 pedestrians placed on an intersection.



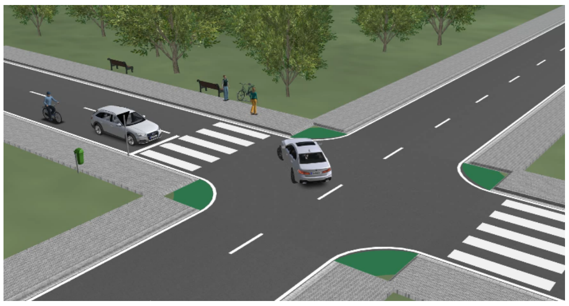
(a) Intersection scenario with three vehicles in red and two pedestrians in green rectangles.



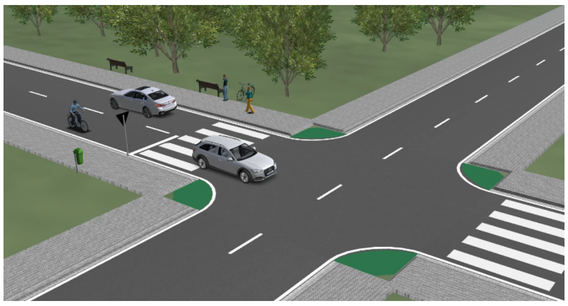
(b) Time Frame 1 - Vehicle 2 is crossing the street while the other vehicles are waiting. The pedestrian 1 is moving to the zebra crossing and pedestrial 2 is standing still.



(c) Time Frame 2 - Vehicle 1 starts moving to left toward the zebra crossing while vehicle 3 is standing still. The pedestrian 1 is in the middle of the the zebra crossing and pedestrian 2 is standing still.



(d) Time frame 3 - Vehicle 1 is moving very close to the zebra crossing. The pedestrian 1 crossed the zebra crossing and pedestrian two is standing still.



(e) Time frame 4 - Vehicle 1 continues driving on the new lane and vehicle 3 started driving in the initial lane.

Figure 1. Intersection scenario with five objects, three vehicles and two pedestrian. The cyclist which occurs in time frame 1 is not considered for actual scenario calculation.

If we define parameters with defined ranges for all intersection participants, it is possible to generate different scenarios on this intersection by combining these parameters. In this case, an exemplary amount of parameters and parameter ranges with five elements are defined. The parameters which are defined are the vehicle initial velocities $v^i_{veg} \in$ [10 km/h, 15 km/h, 20 km/h, 30km/h, 40 km/h], the vehicle routes $r^i_{veh} \in$ [1,2,3,4,5], the lateral displacement from the vehicle route $r^i_{lat} \in$ [0.25 m, 0.5 m, 1.0 m, 1.5 m, 2.0 m], the pedestrian routes $r^j_{ped} \in$ [1,2,3,4,5] and the pedestrian initial velocities $v^j_{ped} \in$ [0 km/h, 2.5 km/h, 5 km/h, 7.5 km/h, 10 km/h] with the index i representing the number of vehicles and index j the number of pedestrians in the intersection from Fig. 1. By calculating all possible combinations of these parameters we could create 3125 scenarios. If one scenario takes 10 minutes in real world, we would need 31250 minutes of RWT, which is around 22 days of RWT. Considering higher parameter ranges and more participants, it would jump to an infinite number of scenarios for these specific intersections. Comparing this with a continuous simulation where one such scenario in CarMaker could be calculated in 5 seconds, all scenarios could be calculated and tested in 52 minutes assuming these parameter ranges. It should be mentioned here that even if it is possible to reduce the calculation and generation time of scenarios, not all combinations are relevant in full factorial testing for the system under test. To reduce the number of scenarios to only relevant scenarios, combinatorial testing methods presented in the research works of [9], [10], [11] can be used. By using combinatorial testing, it is possible to define less parameters and parameter interactions where not all parameter combinations are taken into account. Using suitable virtual platforms and simulation tools for generation of test scenarios in testing of ADS is a way to reduce and overcome RWT. To ensure representative and valid data by virtual platforms, those platforms should provide the possibility to implement and develop realistic and complex vehicle, traffic, object, sensor, environment models and other real-world elements which are relevant for the vehicle under test. These virtual platforms can be found in the research works presented in [12] [13], [14]. The platforms have a common quality – they combine vehicle simulation software (VSS) with a TFSS in a co-simulation. Such CSP provides the opportunity to test the ADS in an approximately realistic environment, thus reducing the efforts in RWT. In this paper we present an approach where, with the combination of a VSS with TFSS, a vehicle under test is tested in a stochastic traffic environment in order to create relevant scenarios. The stochastic traffic environment provides the possibility of detection of scenarios which cannot be found easily by exhaustive combinatorial testing. The stochastic traffic environment corresponds to approximately real test-driving conditions, where a vehicle under test is tested as in RWT procedures. To create such an environment, the approach is using two components, one for modelling of the ego vehicle with its automated driving functions, and another for modelling the traffic flow. For the vehicle simulation, the widely used software in automotive engineering community is CarMaker and for the traffic flow simulation Vissim presented in [15]. The VSS CarMaker provides a virtual multi-body simulation which is a computer-modelled representation of a vehicle, see [16]. Additionally, it provides the possibility to implement ADS, sensor models, adjustments and models of objects in the virtual environment of CarMaker. The traffic flow model in Vissim is calibrated by using measured data from an official test road in Austria [17] and is presented in [12]. The concept and the modelling part of CSP, vehicle under test and the traffic environment are also presented in [12]. The main goal of this paper is to show and present a design of a CSP with an implementation using the co-simulation between CarMaker and Vissim for generation of scenarios in testing ADS (e.g. [18] and [19]). The results and efficiency of the presented co-simulation framework are summarized in the Sec. 5.

2. Co-Simulation Framework

The high-level overview of the co-simulation framework (CSF) can be seen as a group of three components interacting with each other, as represented in Fig. 2. In a broad sense, this existing components, remaining tasks are the development of the co-simulation controller (CSC) and communication between CSC, CarMaker and Vissim. Communication between Vissim and CarMaker is handled through the Vissim Interface for CarMaker, see [20]. Vissim provides a useful COM

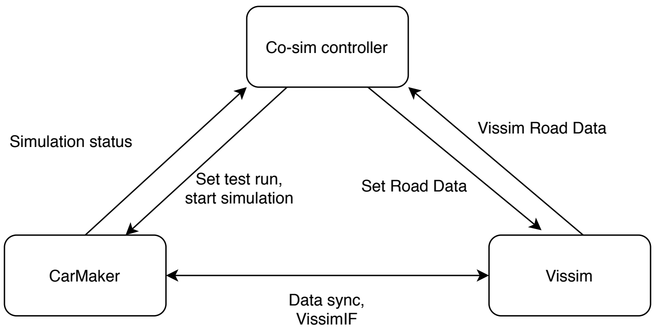


Figure 2. A high-level overview of the software architecture presenting the three components of the co-simulation framework.

interface through which the communication between CSC and Vissim is handled, see [21] and [22]. Communication between CSC and CarMaker is done in two ways, by sending *Tcl* commands through the *cmguicmd* interface and setting the Simulink model parameters pragmatically. Considering the whole framework, the most development is done regarding CSC. Using object-oriented concepts, data and the behavior related to that data are coupled together. Another goal of introducing object-oriented concepts was a separation of logic from the user interface related code.

3. Software Design

Software Design is the essential part of this research. CSC is broken into multiple components where each is described with at least one class without violating any of the functional requirements.

3.1. Defining Functional Requirements

The functional requirements for the software design are shown in Tab. 1

Requirement 1	Simulation
Requirement 2	Data Storage
Requirement 3	Road File
Requirement 4	Simulation Control
Requirement 5	Input Validation

Table 1. Functional requirements on the co-simulation framework.

The main functional requirement from Tab. 1 is the first one. It is divided in additional parts.
First Part: Simulation of one specific scenario over many kilometers.
The co-simulation framework provides the possibility to choose the test road, the Vissim version, the testing kilometers and the specific scenario test run for scenario-generation purposes. In CarMaker, a scenario test run is a specified simulation run which comprises the vehicle under test with all functionalities, a defined test road, environment objects and test definition. All given parameters shall be checked by the system for validity before starting the simulation. A scenario test run is test run defined with a configured vehicle model and traffic simulation.
Second Part: Simulation of multiple scenarios over many kilometers.
The co-simulation framework provides the possibility to choose the root folder of containing scenarios and a selection of those which should be simulated. Each scenario test run is simulated over a given number of kilometers. All given parameters are checked by the system for validity before starting the simulation.

3.2. Design

For the design of the co-simulation framework a GUI is implemented using a model-view-control (MVC) pattern [24]. Applying MVC pattern, elements of view component are decoupled from the framework logic and the rest of the code is broken into smaller yet meaningful components which interact with each other. For implementation of MVC Design Pattern guidelines from [23] were used. The interaction between components can be seen in Figure 3.

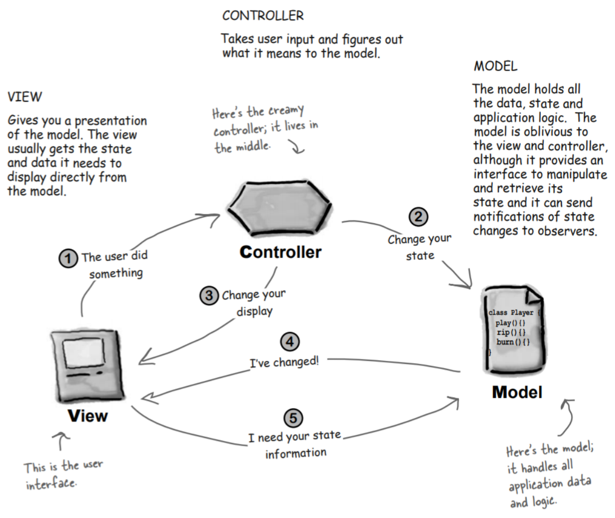

CONTROLLER
Takes user input and figures out what it means to the model.
MODEL
The model holds all the data, state and application logic. The model is oblivious to the view and controller, although it provides an interface to manipulate and retrieve its state and it can send notifications of state changes to observers.
VIEW
Gives you a presentation of the model. The view usually gets the state and data it needs to display directly from the model.
Here's the creamy controller; it lives in the middle.
Controller
1 The user did something
2 Change your state
3 Change your display
4 I've changed!
5 I need your state information
View
Model
This is the user interface.
Here's the model; it handles all application data and logic.

Figure 3. Representation between components of MVC (image taken from [23]).

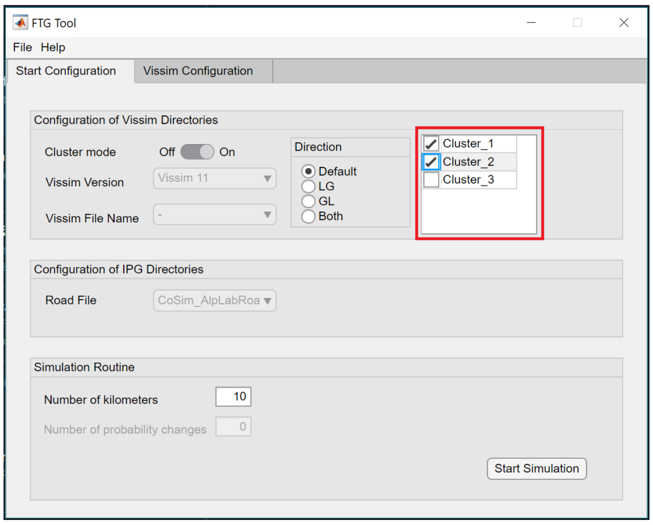
In the co-simulation framework, these three components can be distinguished as *View*, *Controller* and *Model*. The *View* component is *singleton View* class which describes the appearance of the user interface. All user interface elements are defined there. The *Controller* component is a *singleton Controller* class which acts as an interface between *View* and *Model* components. It updates the state of the *Model* component, handles return values of the *Model* functions and based on them, generates adequate graphic response using *View* elements. Compared to the previous two components, the *Model* component is not a single class. There is a *singleton Model* class but it is just the main part of the model component. The *Model* component is composed of the additional classes which are responsible for the framework logic and for holding all necessary data for the simulation process. The control logic of framework is located in *StartSimulation* function. It differentiates between two different modes of operation, the manual and the cluster mode. The model class is also responsible for the inter-process communication with Vissim and CarMaker. When the *StartSimulation* function is called, the *Model* component configures both Vissim and CarMaker and executes simulations.

4. Application

Two separate modes of simulations can be distinguished from functional requirements, manual and automatic or cluster mode, see [12]. In manual mode, the user is allowed to generate a very specific scenario where they can wage different parameters such as distributions, composition of vehicles and traffic in the tool. Automatic or cluster mode is meant for exhaustive fully automated testing of a model over multiple kilometers.

4.1. Cluster Mode

Automatic or cluster mode of the co-simulation is a simulation of predefined scenarios created using real traffic measurements of a specific road, see [12]. With this data provided, the user wants to observe the behavior of the modeled vehicle over varying driving distances. In this mode, the user can choose the path to the root folder where clusters containing predefined scenarios are located.



Cluster mode Off [✓ On]
Vissim Version: Vissim 11
Vissim File Name: -
Direction: [✓ Default] LG GL Both
✓ Cluster_1
✓ Cluster_2
Cluster_3
Road File: CoSim_AlpLabRoa
Number of kilometers: 10
Number of probability changes: 0

Figure 4. Selecting clusters loaded from the cluster's root path in GUI.

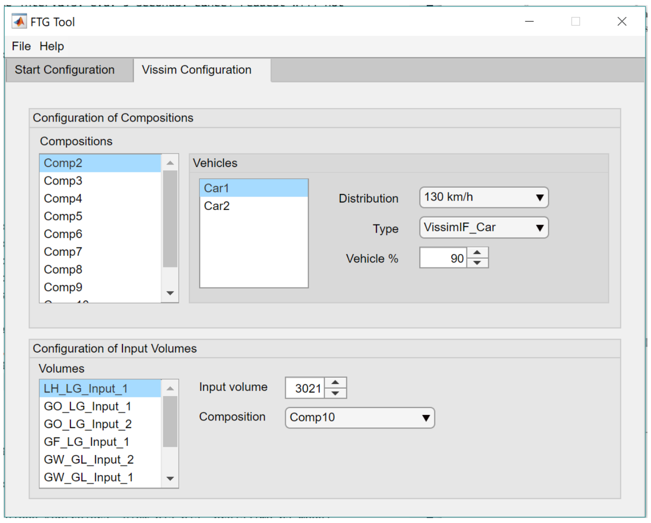
All clusters found in the selected root folder are displayed to the user in GUI and made available for selection as presented in Fig. 4. Execution of cluster mode can be separated into two phases, initialization phase and simulation phase. Initialization phase comprises:

1. **Loading CarMaker testrun file**
   File with specified direction and Vissim road file is loaded into the CarMaker.
2. **Loading Vissim road file**
   Road from the selected cluster is loaded into the Vissim and it is modified by changing the random seed value. The random seed value changes the traffic procedure without changing the parameterization. This value represents the stochastic nature of the road traffic.
3. **Setting the number of kilometers to be covered**
   Number of kilometers specified by the user is set in the Simulink model using *set_param* function.
4. **Starting simulation**
   Simulation is started by setting *SimulationCommand* parameter of the Simulink model to *start*. That is also done using *set_param* function.

At this point, *simulation phase* begins and therefore the status of the running simulation has to be checked. For those purposes, timers are used as described in section 4.3. If simulation is stopped, the covered kilometers are checked. If the simulation status reached a given number of kilometers, the CSC can proceed in three following ways:

1. Restart the simulation on the same road in opposite direction.
2. Switch to the following cluster.
3. Finish if no more clusters are left.

During the simulation phase, the progress bar is presented to the user with three purposes: the first one is that the user is aware of the running simulation status. The second purpose is the possibility of cancelling the simulation via the stop button, which is part of the progress bar. Displayed progress bar deactivates the whole user interface so that no changes of the simulation settings can be made until the stop button is pressed, and that is its third purpose. The obvious advantage of this mode is the automation of a process which, if done manually, would consume more time. The data generated by Vissim and CarMaker is stored in such way that it can be accessed by date, mode or creation time stamp. Post-processing is done at the end where data for all clusters is acquired. With all generated data, the configuration file is also stored in which the configuration for the latest simulation run can be found, including cluster root folder path, selected clusters, number of kilometers per cluster and random seeds used for every simulation step.



Comp2
Comp3
Comp4
Comp5
Comp6
Comp7
Comp8
Comp9

Car1
Car2

Distribution 130 km/h
Type VissimIF_Car
Vehicle % 90

LH_LG_Input_1
GO_LG_Input_1
GO_LG_Input_2
GF_LG_Input_1
GW_GL_Input_2
GW_GL_Input_1

Input volume 3021
Composition Comp10

Figure 5. Vissim configuration tab.

4.2. Manual mode

The main difference between Manual and Cluster mode is that in manual mode, user does not use predefined clusters for the simulation, but rather individual Vissim road files which can be easily modified over the GUI. All the data related to the chosen Vissim road file is loaded into the internal data structure using COM interface and displayed in the separate tab in the GUI as presented in Fig. 5. Vissim related data is held by the *VissimData* class which, when initialized, reads the data from Vissim and has the responsibility of filling that data structure. The process of loading the data is done in the *VissimData* constructor which is called upon the selection of the Vissim file. All operations on the data are handled by *VissimData* class. Having this class provides convenient extraction of Vissim data. It also enables having multiple Vissim files loaded into the memory. Switching between them is a lot faster than loading all the data all over again. When all the data from Vissim is retrieved, the data is placed into the structures. Lists of compositions and volumes are created, where both lists are contained within *VissimData* class. Each volume has a reference to the corresponding composition, so it can keep the track of the current composition which is assigned to that particular volume. From this point, the CSF is ready for the simulation run, and again, code execution after pressing Start Simulation button is separated into two phases, initialization and simulation phase. At the beginning of the initialization phase, all changes of compositions and volumes are sent back to Vissim which is also the main difference between cluster and manual mode in this phase. Changes which user is able to perform are as follows:

1. Change vehicle input volume value
2. Change vehicle input composition
3. Modify composition by modifying vehicles of that composition
4. Change the speed distribution of selected vehicle
5. Change the type of selected vehicle
6. Change the vehicle percentage within selected composition

Rest of the initialization phase of manual mode is similar to the one in the cluster mode. In the simulation phase, the main difference to the cluster mode happens when all given kilometers are covered. Simulation does not continue with the different road file since there are no clusters to follow. *The only case when the simulation continues after a selected number of kilometers is when the direction option is set to both.* All performed changes are immediately reflected into the internal data structure which is loaded into the Vissim road file upon simulation start via COM interface. All the data generated by Vissim and CarMaker are stored in the same way as described in Cluster Mode.

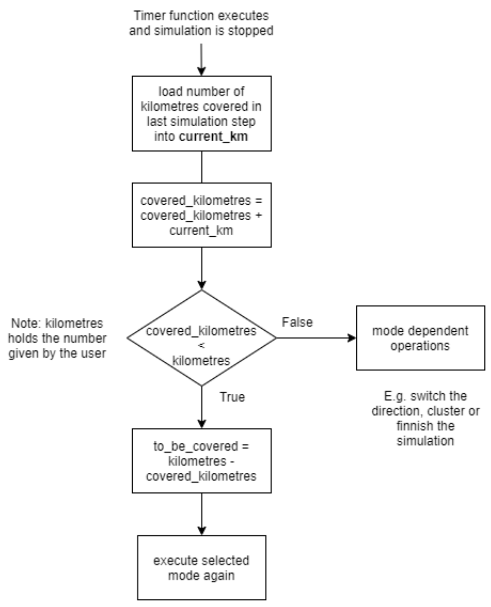

Timer function executes and simulation is stopped
load number of kilometres covered in last simulation step into **current_km**
covered_kilometres = covered_kilometres + current_km
Note: kilometres holds the number given by the user
covered_kilometres < kilometres
False
mode dependent operations
E.g. switch the direction, cluster or finnish the simulation
True
to_be_covered = kilometres - covered_kilometres
execute selected mode again

Figure 6. Control flow diagram showing the calculation of remaining kilometers.

4.3. Simulation status

Since MATLAB lacks conventional multi-threading capabilities, timers are used as a workaround to make the GUI responsive while the simulation is running. After the simulation is initialized, the timer is started. It schedules the execution of a MATLAB code, in this case, the code which checks the current simulation status. It checks whether simulation is stopped or still running. In the case that the simulation stops, the number of covered kilometers is checked, and it is decided if more kilometers have to be simulated in this configuration or not. The rest of the simulation flow is mode specific. Since scheduling of the code which checks simulation status is not time-critical, the *fixedSpacing* mode for timer is chosen. This mode avoids any possibility of interrupting the execution of a function during its execution. No new execution is scheduled until the function is executed, and that plays out as an advantage since the goal is to avoid frequent checks of the status, which is time-consuming and interrupts the simulation. It is also not desirable to fill a queue with scheduled function calls since the simulation can be stopped at any time. As a result of using timers, the CSF has responsive GUI allowing the user to stop the simulation, and even more important, it enables the checking of simulation status efficiently, since there is no reliable option for triggering events upon simulation end, and repeatedly checking for status with delays in between keeping the system busy when not needed.

4.4. Covered Kilometers

Checking the covered kilometers is a common mechanism for both modes. It enables the simulation to run for any given length even though the road file is limited to some extent. As it can be seen in Sec. 4.3, the covered kilometers are checked periodically in the function executed by the timer. Out of the number of currently covered kilometers, new number of remaining kilometers is calculated. The workflow of this procedure can be best seen in Fig. 6. There is no way to capture a route length which will be driven, therefore a number of kilometers is given as the initial distance to be covered. If the remaining distance is shorter than the current route length, vehicle will drive the remaining number of kilometers and then stop. In case that the remaining distance is longer than the route, the simulation will stop when the end of the route is reached and covered distance will be saved in the workspace. That addition in Simulink is necessary in order to obtain precise number of covered kilometers by the model vehicle.

Method	Context
cmguicmd	Setting distance to be covered
set_param	Starting and stopping simulation

Table 2. Different approaches of interacting with CarMaker.

Task 1	Setting the newly generated random seed value.
Task 2	Loading compositions, vehicles, vehicle distributions, vehicle types, vehicle percentages, vehicle input volumes.
Task 3	Setting modified compositions, volumes and vehicles.

Table 3. Task list for the interaction with Vissim.

4.5. Communication with CarMaker

Communication with CarMaker is done in two ways - through the *cmguicmd* interface, provided by CarMaker in order to send *Tcl* commands [25] and through setting the Simulink model properties using the *set_param* function, see Tab. 2.

Besides these two options, certain parameters are set using additional scripts, such as driving direction in the simulation, path in which generated data will be stored and which Vissim road path should be applied. Since there is no clear interface for modifying any of these parameters, scripts which modify necessary configuration files are introduced. They are simple C++ scripts which match given attribute name and assign the new value to it.

4.6. Communication with Vissim

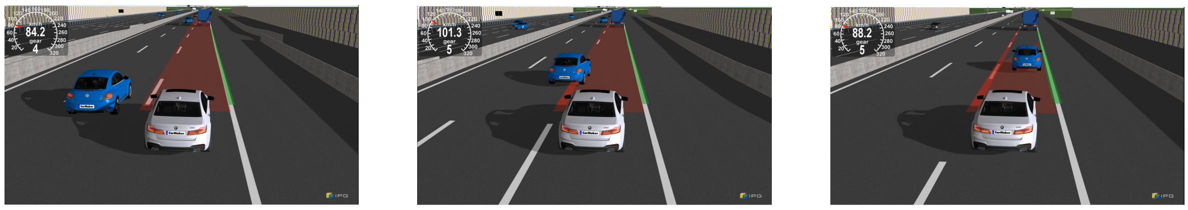
Interaction with PTV Vissim is completely executed by the COM interface [26]. It is used for the tasks described in Tab. 3. Before any of these operations is done, the creation of a new *actxserver* is needed which as an argument requires programmatic identifier (ProgID). It creates a COM server through which it is possible to communicate with Vissim. Depending on the Vissim version, different programmatic identifiers are passed. After starting *actxserver* and therefore starting Vissim, the desired road network has to be loaded in order to make any changes.

4.7. Direction Selection

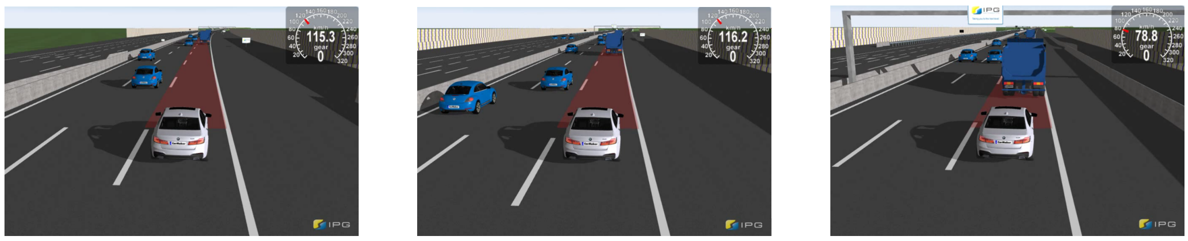
The direction in which the modeled vehicle will move is specified as an attribute in the testrun file. Since this is specific to the selected road, it is meant to be used only with predefined roads and clusters. Differences between the same test runs in different directions are not huge, therefore the scripts mentioned in Sec. 4.5 are used to set certain attributes to the specified value. Besides two directions in which simulation can take place, there is also possibility of choosing both directions. In this case, the simulation is running in both directions and it will cover the same specified number of kilometers in both of them.

4.8. Data Storage

The result of the successful simulation is generated data which is used for post-processing and data extraction. For every simulation step, one **.erg* the file is created by CarMaker which contains data from various ego vehicle sensors. Besides *erg* files, one configuration file is created containing all parameters used for simulation. Upon simulation start of either of two modes, the folder named by current date is created, e.g. 12-Feb-2020. Inside of it, another folder named by current mode is created, either Automatic or Manual. All the data created by simulations on that day will be stored in the same root folder named by that date with the exception of the user changing the data



(a) Near-collision caused by a lane change.



(b) Near-collision caused by speed reduction of the vehicle in front.

Figure 7. Two detected near-collisions caused in by different causes. The vehicle under test in the white vehicle and the blue vehicle are traffic participants generated by Vissim.

output path. Assuming that user is not changing data output path, all automatic mode simulations would be stored in *../12-Feb-2020/automatic/* and all manual mode simulations would be stored in *../12-Feb-2020/manual/*. To avoid mixing all the data, every simulation run gets its own folder named by the time of simulation start, e.g. 14-30-24 in which the *erg* files and the configuration file is placed. All *erg* files and corresponding info files are simply named by the number which defines the order of its creation.

5. Simulation Results

To demonstrate the effectiveness of the co-simulation framework we use a Level3+ Highway Chauffeur presented in [12] as the vehicle under test. For that purpose, one simulation run with 5000 kilometers is carried out. Focusing on safety critical scenarios, collisions and near-collisions are taken for the evaluation of the framework efficiency. Within these 5000 km of simulation, 13 collisions were detected and 658 near-collisions. Each of these detected scenarios belongs to the same defined scenario type but has a unique trajectory, vehicle states and surrounding area with traffic participants. As an example, two scenarios of the scenario type near-collision are presented in Fig. 7. In the first Fig. 7a we can see a near-collision which is caused by a vehicle in front while changing the lane in front of the vehicle under test. The Fig. 7b shows a near-collision caused by a truck in front of the vehicle under test while reducing the speed.

6. Conclusion

In this paper we presented the design and implementation of a co-simulation framework for testing ADS using MATLAB/Simulink, CarMaker and Vissim. As described in Sec. 3, all functional requirements have been defined and implemented. The results show that the implemented co-simulation framework is capable of acquiring data for scenario generation in ADS. Since the framework highly relies on CarMaker, Vissim and the interfaces which are provided by them, the implementation cannot eventually be passed onto another CSP. The chosen MVC pattern for the design of the co-simulation framework has proven as an efficient and convenient way to structure a CSP and can be used in general for all CSPs which combine VSS and TFSS.

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. and B.R.; data curation, D.N.; writing–original draft preparation, D.N. and A.P.; writing–review and editing, A.E. 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"

Conflicts of Interest: The authors declare no conflict of interest.

1. Wintersberger, S.; Azmat, M.; Kummer, S. Are We Ready to Ride Autonomous Vehicles? A Pilot Study on Austrian Consumers' Perspective. *Logistics* **2019**, *3*. doi:10.3390/logistics3040020.
2. SAE. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, 2019.
3. Kalra, N.; Paddock, S. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice* **2016**, *94*, 182–193. doi:10.1016/j.tra.2016.09.010.
4. Broggi, A.; Buzzoni, M.; Debattisti, S.; Grisleri, P.; Laghi, M.C.; Medici, P.; Versari, P. Extensive Tests of Autonomous Driving Technologies. *IEEE Transactions on Intelligent Transportation Systems* **2013**, *14*, 1403–1415. doi:10.1109/TITS.2013.2262331.
5. Stellet, J.E.; Zofka, M.R.; Schumacher, J.; Schamm, T.; Niewels, F.; Zöllner, J.M. Testing of Advanced Driver Assistance Towards Automated Driving: A Survey and Taxonomy on Existing Approaches and Open Questions. 2015 IEEE 18th International Conference on Intelligent Transportation Systems, 2015, pp. 1455–1462. doi:10.1109/ITSC.2015.236.
6. Azmat, M.; Kummer, S.; Moura, L.; Di Gennaro, F.; Moser, R. Future Outlook of Highway Operations with Implementation of Innovative Technologies Like AV, CV, IoT and Big Data. *Logistics* **2019**, *3*, 15. doi:10.3390/logistics3020015.
7. 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. 2017.
8. Makridis, M.; Mattas, K.; Ciuffo, B.; Alonso, M.; Thiel, C., Assessing the Impact of Connected and Automated Vehicles. A Freeway Scenario; 2018; pp. 213–225. doi:10.1007/978-3-319-66972-4_18.
9. Rogic, B.; Bernsteiner, S.; Samiee, S.; Eichberger, A.; Payerl, C. Konzeptionelle virtuelle Absicherung von automatisierten Fahrfunktionen anhand eines SAE Level 3 Fahrstreifenwechselassistenten. 2016. VDI/VW-Gemeinschaftstagung : Fahrerassistenz und automatisiertes Fahren ; Conference date: 08-11-2016 Through 09-11-2016.
10. Kuhn, R.; Kacker, R.; Lei, Y.; Hunter, J. Combinatorial Software Testing. *Computer* **2009**, *42*, 94–96. doi:10.1109/MC.2009.253.
11. Kuhn, R.; Lei, Y.; Kacker, R. Practical Combinatorial Testing: Beyond Pairwise. *IT Professional* **2008**, *10*, 19–23. doi:10.1109/MITP.2008.54.
12. 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*. 2019 IEEE Intelligent Transportation Systems Conference (ITSC), 2019, pp. 1895–1901. doi:10.1109/ITSC.2019.8916839.
13. Hallerbach, S.; Xia, Y.; Eberle, U.; Köster, F. Simulation-Based Identification of Critical Scenarios for Cooperative and Automated Vehicles. *SAE Technical Papers* **2018**, *2018-01-1066*. doi:10.4271/2018-01-1066.
14. Aparow, V.R.; Choudary, A.; Kulandaivelu, G.; Webster, T.; Dauwels, J.; d. Boer, N. A Comprehensive Simulation Platform for Testing Autonomous Vehicles in 3D Virtual Environment. 2019 IEEE 5th International Conference on Mechatronics System and Robots (ICMSR), 2019, pp. 115–119. doi:10.1109/ICMSR.2019.8835477.
15. Fellendorf, M.; Vortisch, P., Microscopic Traffc Flow Simulator VISSIM. In *Fundamentals of Traffic Simulation*, 1 ed.; Springer Science+Business Media, 2010; Vol. 145, *International Series in Operations Research & amp; Management Science*, pp. 63–94.
16. Varga, B.O.; Moldovanu, D.; Mariaşiu, F.; Iclodean, C.D. Simulation in the Loop of Electric Vehicles. In *Modeling and Simulation for Electric Vehicle Applications*; Fakhfakh, M.A., Ed.; IntechOpen: Rijeka, 2016; chapter 1.

17. Seebacher, S.; Datler, B.; Erhart, J.; Greiner, G.; Harrer, M.; Hrassnig, P.; Präsent, A.; Schwarzl, C.; Ullrich, M. Infrastructure data fusion for validation and future enhancements of autonomous vehicles' perception on Austrian motorways. 2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE), 2019, pp. 1–7. doi:10.1109/ICCVE45908.2019.8965142.
18. Weber, N.; Frerichs, D.; Eberle, U. A simulation-based, statistical approach for the derivation of concrete scenarios for the release of highly automated driving functions. AmE 2020 - Automotive meets Electronics; 11th GMM-Symposium, 2020, pp. 1–6.
19. Menzel, T.; Bagschik, G.; Maurer, M. Scenarios for Development, Test and Validation of Automated Vehicles. 2018 IEEE Intelligent Vehicles Symposium (IV), 2018, pp. 1821–1827. doi:10.1109/IVS.2018.8500406.
20. Siebel, T. IPG | PTV | New Interface. *MTZ Worldwide* **2017**, pp. 36–39.
21. Tettamanti, T.; Horváth, M. A practical manual for Vissim COM programming in Matlab - 3rd edition for Vissim 9 and 10. 2018.
22. Joelianto, E.; Sutarto, H.; antariksa, s. Simulation of Traffic Control Using VissimCOM Interface. *Internetworking Indonesia Journal* **2019**, *11*, 55.
23. Freeman, E.; Freeman, E.; Bates, B.; Sierra, K. Head First Design Patterns **2004**.
24. Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.M. *Design Patterns: Elements of Reusable Object-Oriented Software*; Addison-Wesley Professional, 1994.
25. IPG. CarMaker Programmer's Guide, 2019.
26. PTV AG. PTV Vissim User Manual, 2018.