Automotive Innovation (2021) 4:373–389
https://doi.org/10.1007/s42154-021-00161-1

# Simulator Coupled with Distributed Co-Simulation Protocol for Automated Driving Tests

Max-Arno Meyer[1] · Lina Sauter[1] · Christian Granrath[1] · Hassen Hadj-Amor[2] · Jakob Andert[1]

Received: 18 December 2020 / Accepted: 30 August 2021 / Published online: 16 October 2021
© The Author(s) 2021

**Abstract**
To meet the challenges in software testing for automated vehicles, such as increasing system complexity and an infinite number of operating scenarios, new simulation methods must be developed. Closed-loop simulations for automated driving (AD) require highly complex simulation models for multiple controlled vehicles with their perception systems as well as their surrounding context. For the realization of such models, different simulation domains must be coupled with co-simulation. However, widely supported model integration standards such as functional mock-up interface (FMI) lack native support for distributed platforms, which is a key feature for AD due to the computational intensity and platform exclusivity of certain models. The newer FMI companion standard distributed co-simulation protocol (DCP) introduces platform coupling but must still be used in conjunction with AD co-simulations. As part of an assessment framework for AD, this paper presents a DCP compliant implementation of an interoperable interface between a 3D environment and vehicle simulator and a co-simulation platform. A universal Python wrapper is implemented and connected to the simulator to allow its control as a DCP slave. A C-code-based interface enables the co-simulation platform to act as a DCP master and to realize cross-platform data exchange and time synchronization of the environment simulation with other integrated models. A model-in-the-loop use case is performed with the traffic simulator CARLA running on a Linux machine connected to the co-simulation master xMOD on a Windows computer via DCP. Several virtual vehicles are successfully controlled by cooperative adaptive cruise controllers executed outside of CARLA. The standard compliance of the implementation is verified by exemplary connection to prototypic DCP solutions from 3rd party vendors. This exemplary application demonstrates the benefits of DCP compliant tool coupling for AD simulation with increased tool interoperability, reuse potential, and performance.

**Keywords** Distributed co-simulation protocol · Co-simulation · Automated driving · Traffic simulation · Tool coupling

**Abbreviations**

| | |
|---|---|
| ADS | Automated driving system |
| API | Application programming interface |
| CACC | Cooperative adaptive cruise control |
| DCP | Distributed co-simulation protocol |
| FMI | Functional mock-up interface |
| FMU | Functional mock-up unit |
| HIL | Hardware-in-the-loop |
| HRT | Hard real-time |
| MIL | Model-in-the-loop |
| NRT | Non-real-time |
| PDU | Protocol data unit |
| SIL | Software-in-the-loop |
| SRT | Soft real-time |
| SUT | System under test |
| TCP | Transmission control protocol |
| UDP | User datagram protocol |

✉ Max-Arno Meyer
meyer_max@vka.rwth-aachen.de

[1] Teaching and Research Area Mechatronics in Mobile Propulsion, RWTH Aachen University, Aachen, Germany

[2] FEV Software and Testing Solutions SAS, Trappes, France

## 1 Introduction

Virtual testing by the means of simulation is essential for the verification and validation of automated vehicles as solely physical testing is time and resource consuming to cope with large test scopes [1–4]. The safety of the intended functionality (SOTIF) certification of automated driving systems (ADS) recommends a mixture of simulation-based tests, tests in controlled environments, and real-world driving to achieve a meaningful coverage

of possible driving scenarios [5]. Due to increasing testing efforts with the degree of automation, simulation methods are being developed which allow a frontloading of test activities to perform more tests per time [6, 7]. At the same, a key challenge in virtual prototyping of upcoming highly automated driving systems (HADSs) is capturing the diversity of a vehicle's environment with respect to the entirety of possible operating scenarios of the system under test (SUT) [8]. High complexity, perceptual range, and degree of environmental interaction of HADS require complex cyber-physical plant models for closed-loop testing in simulation. A virtual test frame is composed of the controlled vehicle with its sensors and motion control as well as the controlled vehicle's context including the perceived environment and all actors interacting with the SUT such as the driver and other traffic participants [3]. Sophisticated context models include behavior and uncertainty simulation, further increasing model complexity [9]. For collaborative embedded systems (CES), multiple controlled vehicles and data exchange via networks between vehicles and roadside units have to be considered additionally in the test frame to analyze the SUT behavior in collaborative system groups [3].

Models that form a suitable virtual test frame require different cyber-physical simulation disciplines such as kinematics simulation, powertrain simulation, and communication network simulation [9]. Often it is necessary to model a 3D representation of the world and render the scene to realistically simulate sensor data streams such as camera images and Radar or Lidar point clouds. The different plant model components, features, and simulation disciplines require specialized tools for model implementation and execution, which is why sophisticated closed-loop simulations for ADS often rely on co-simulation to combine and run heterogeneous component models in a single test frame. The large landscape of available specialized tools for individual simulation disciplines strengthens this trend [9].

Striving for fast and at the same time highly complex and numerically correct co-simulations leads to a field of tension as higher model complexity and fidelity result in more computationally intensive models with slower execution times. These problems have to be counteracted by new methods to speed up models and efficiently co-simulate model components and by new platforms to run the respective simulations. This is especially applicable for model-in-the-loop (MIL) and software-in-the-loop (SIL) simulations where the upper limit of simulation speed is not defined by real time.

For the realization of virtual test frameworks for ADS by the means of co-simulation, interfaces between heterogeneous component models are established to enable interoperability. The dynamic mutual exchange of several simulation tools to calculate the global behavior of a system consisting of several subsystems is referred to as a coupling of these tools. Each subsystem's behavior depends on the behavior and generated outputs of the other subsystems [10].

Apart from proprietary solutions being established on the basis of specific use cases, several co-simulation standards exist, which can be applied for ADS simulations. Standards such as high-level architecture (HLA) [11] or functional mock-up interface (FMI) [12], are tool independent and define non-functional interoperability via metadata formats for model exchange, data models, application layer protocols, or synchronization patterns to make simulation models interoperable [9, 13]. These standards often rely on a master–slave model architecture to achieve synchronization [14]. While already widely utilized standards such as FMI focus on interoperability only on the model level, the more recent distributed co-simulation protocol (DCP) [15] extends this concept to standardized interoperability on an application protocol level to run simulations on distributed hardware platforms [13].

The usage of DCP has previously focused on real-time simulations incorporating a plant model and real hardware operated on test benches combined in one test frame. For instance, Baumann et al. [16] coupled a small-scale vehicle testbed with a cross-domain simulation model. Krammer et al. [17] showed how DCP can be generally applied to couple different simulation tools and platforms in co-simulation for the realization of X-in-the-Loop use cases.

The DCP technology is of great potential value for coupling heterogeneous models and simulators for virtual ADS testing which highly depends on co-simulation, but it is yet to be applied in this field. Due to the highly specialized models, many of them are platform exclusive and incompatible with model exchange formats. This is especially true for 3D-rendered environment models that rely on a specific graphics engine to run as well as network simulators [9]. In order to combine these models with others and the SUT without the usage of standardization, tailor-made interfaces for a specific combination of tools must be developed that are limited to one toolset and use case. Such tool coupling solutions have recently been integrated into a lot of commercial products for simulation-based ADS testing. The lack of standardized tool and platform coupling options limits the flexibility with respect to model and tool choices and increases the effort for setting up virtual test frames which vary strongly with the SUT and test scenarios. A standardization of non-functional interoperability on the platform level would enable fully modular test frames incorporating not only FMI compatible models, but also models previously bound to specific tools and platforms. It would ease the realization of hardware-in-the-loop (HIL) and vehicle-in-the-loop (VIL) applications as virtual test frame components can be interfaced with hardware test benches or test vehicles. At the same time, it allows for an efficient distribution of models across platforms to speed up MIL and SIL simulations,

e.g., up to the point of coupling virtual machines running different simulators and models in cloud environments.

In this paper, the focus is on the development of a DCP compliant implementation to ease and enhance the coupling of simulators and platforms for simulation-based testing of ADS. The implementation shall cover an interface for a co-simulation master which enables it to communicate via DCP as well as a reusable DCP slave skeleton which is to be used to interface to a 3D environment and vehicle simulator including rendering.

The paper is structured as follows. First, it is explained how DCP addresses requirements of this tool coupling use case which are not covered by other co-simulation standards. Related works concerning the implementation and application of co-simulation standards are reviewed. Technical requirements for the DCP implementation are raised, followed by deriving the implementation concept. Subsequently, two software components developed for realizing simulator coupling for ADS testing with DCP are presented. The implementation is then evaluated with respect to standard compliance and by the means of a demonstrator which is developed to enable virtual testing of CES.

## 2 Related Works

With the high need for model integration methods, a lot of distributed co-simulation involving different tools or platforms has previously been realized in the field of ADS testing. Tool coupling has proven to be of great value especially for cross-simulation-domain use cases such as combined vehicle dynamics and network simulation or for the integration of rendering into simulations.

Buse et al. [18] connected a HIL test bench to a microscopic traffic simulator and a network simulator via a tailor-made interface creating a platform for testing control units relying on real-time interactive vehicle-to-X communication. The implemented interface coordinates all three simulators and is proved to be soft real-time capable in a demonstrator scenario. A similar co-simulation solution incorporating tool coupling of MATLAB with the traffic simulator VISSIM running on Windows operating system and the network simulator ns-3 running on Linux was developed by Choudhury et al. [19]. The coupling was established with virtual machines coupled via the sockets application programming interface (API), and the utilization of the framework was shown with the impact assessment of infrastructure-to-vehicle data. The authors have identified that the implemented non-standardized coupling is not flexible and propose establishing a generic runtime infrastructure instead.

A tool coupling solution for the 3D vehicle and environment simulator CARLA, which is also used in this paper, is presented by Stevic et al. [20]. ROS and Autoware as prototyping platforms were connected to CARLA using a ROS bridge component for the simulator. Yamaura et al. [21] highlighted the benefits of 3D engines in simulation frameworks for ADS and realized a co-simulation integrating Simulink, Dymola, and the Unity game engine using the OpenMETA tool suite which provides horizontal integration between design tools [22]. As a tool suite, OpenMETA is not designed as a standard prescribing the way interoperability is achieved. In order to interface with a tool, an interpreter must be created that is specific for each tool. Luttkus et al. [23] highlight the need for a unified interface to integrate different virtual test vehicles in a co-simulation to reduce the model integration efforts for multi ego simulations.

The presented research shows the high integration efforts for setting up prototypes for tool coupling solutions, and some papers also highlight the limited flexibility of proprietary interfaces. Especially Yamaura et al. [21] derived the requirement of a unified approach for coupling interfaces. Similar challenges have also been identified in the area of HIL testing, when it comes to integrating real-time systems such as testbeds with simulation models. This was one of the motivations for the development of the DCP standard which is currently the only standard covering the communication layer that is mandatory for tools and platforms to interoperate [16]. Establishing DCP support in a tool or platform is meant to guarantee the non-functional interoperability with other tools supporting the standard. As such, the standard is also promising for ADS co-simulations considering the large tool landscape.

DCP has been successfully implemented by Baumann et al. [16] and Krammer et al. [24] with the focus on proving the concept of DCP itself for the integration of real-time systems into simulations. The work aims at proving individual aspects of DCP, e.g., the reduction of configuration efforts. This paper focuses on evaluating the suitability of DCP to ease tool integration efforts for co-simulations in the ADS domain. Therefore, this new DCP implementation targets the prominent use case of integrating 3D simulators. It will also be investigated to what extent components of a DCP implementation can potentially be reused for future tool couplings in case DCP support is not already established.

## 3 Interoperability Concept

In Section 1, some benefits of DCP usage for simulation-based ADS testing have been highlighted. In this section the requirements leading to the applicability of the standard in this field are elaborated, leading to a concept for a new DCP implementation. Additionally, a representative and frequently occurring use case and a toolchain are selected, which reflect these requirements.

### 3.1 Requirements Definition

The following requirements have been raised for tool coupling interfaces and the involved tools in the field of ADS co-simulation based on the presented high-level goals and characteristics of ADS co-simulation. The requirements are written to take into account both the established approaches as well as the lack of more unified interfaces between ADS simulators identified in the analysis of relevant research.

(1) RQ1 (master–slave scheme)
The first requirement is that the co-simulation shall use a master–slave synchronization scheme incorporating one simulation master and multiple simulation slaves in order to synchronize multiple heterogeneous models. The simulated time is synchronized by the master across all simulation slaves. Thus, the following requirements apply to the master, the slave, or the interoperability interface to be implemented that connects both of them. This synchronization scheme is the most widespread one in the ADS and automotive domain including commercial simulators, standards, and state-of-the-art research.

(2) RQ2 (support for software and model formats)
A co-simulation master is required that supports the integration and performant execution of multiple models on its platform with native support for software and model formats widespread in the ADS domain. At least the master shall support the integration of C/C + + code, MATLAB/Simulink models, and functional mock-up units (FMUs).

(3) RQ3 (standardized communication layer for tool coupling)
In order to support the incorporation of models, software, or hardware that are incompatible with the co-simulation master tool or platform and therefore cannot be integrated using established methods, a standardized, non-functional interoperability interface shall be implemented that allows for the integration of autonomous tools and platforms as co-simulation slaves. In case an external tool or platform is required for simulating a slave model, a direct execution within the master is not possible as it requires the execution within a single multithreaded process [9]. Standardization of the communication shall enable the master to delegate any standard-compliant tool on which the particular model is executed no matter if it is running on the same or a stand-alone hardware platform.

(4) RQ4 (tool coupling pre-conditions)
In order to establish this standardized interface, both the master as well as potential slaves must enable interaction with external platforms, e.g., via an API. This includes reading and writing model variables as well as monitoring or controlling different states of the simulation.

(5) RQ5 (functional independence of interface)
The selected interface standard shall be non-functional and applicable independent of the model functionality inside the slave. The implementation inside the master shall be independent of the specific co-simulation slave to be integrated with respect to the tool and the functionality. The interface for different slaves shall use a unified structure to ease the configuration of the master and to enable the reuse of implemented components for interfacing new slaves. This satisfies the need for high flexibility of multi-platform co-simulation for ADS testing.

(6) RQ6 (execution modes)
Also, the co-simulation master and interface must support faster-than-real-time or non-real-time (NRT), soft real-time (SRT), and hard real-time (HRT) simulation to ensure maximum flexibility with respect to potential test environments and software or hardware under test.

(7) RQ7 (enabling time synchronization)
The master as well as the application and presentation layer protocol of the interface, which provides the cooperation pattern for the communication partners and data syntax, must enable the time synchronization of both platforms for NRT, SRT, and HRT use cases [25].

(8) RQ8 (data exchange and parametrization)
Moreover, the interface has to enable the exchange of data during runtime and provide an option for the master to initialize, parametrize and terminate the externally running slave model.

(9) RQ9 (transport layer protocols for tool coupling)
The interoperability interface shall support multiple applicable transport layer protocols, but at least the two IP network standards Transmission Control Protocol (TCP) and user datagram protocol (UDP). TCP can be used to natively ensure data integrity featuring sequence numbers, an acknowledgment system, and checksums while UDP can be effectively used if data transfer speed is prioritized over integrity.

### 3.2 Technical Concept

This work focuses on the added value of a standardized communication layer between co-simulation tools as it is only provided by the DCP standard for ADS testing use cases [13]. Based on the presented requirements, the DCP standard has been analyzed with respect to its applicability. To investigate the potential added value it is also exemplary to be compared with the FMI standard. A use case and a feasible toolchain have been identified to serve as the baseline for the implementation of the standardized tool coupling interface.

### 3.2.1 Applicability of the DCP Standard

As a companion standard, DCP is designed to be compatible with the commonly used FMI standard in the way that FMUs can be integrated within DCP slaves [15]. Table 1 shows a comparison of the two standards with respect to the applicable raised requirements. While FMI does allow for tool coupling by embedding a bridge to external software or hardware inside an FMU, in contrast to DCP, it does not define any constraints for the communication between the master and slave tools or platforms. As a result, while the model interface is standardized with FMI, the tool or platform coupling is proprietary and invisible to the master. DCP covers this aspect and standardizes the integration of real-time and non-real-time systems into simulation environments and is using a master–slave interoperability scheme [17]. Systems can represent either complete hardware platforms or simulation tools running on a single platform which can be executed in NRT, SRT, or HRT mode. In contrast to FMI, the supported execution modes for DCP slaves are shared with the master. As such, the DCP standard satisfies RQ1, RQ3, and RQ6. DCP utilizes messaging as an integration mechanism by exchanging packages of data called protocol data units (PDU). It provides an unambiguous structure for the slave and incorporates the slave model functionality into the system simulation. Therefore a state machine is standardized which has to be implemented to control the slave's internal behavior and to set up the master–slave coordination and data exchange [15]. State changes are either actively called by the master or handled internally by the slave. In any case, after each state change, the master needs to be notified about the new status. States are provided inter alia for the parametrization of the slave and handling of runtime synchronization and data exchange, which meets RQ7 and RQ8. Interoperability is achieved without the standardization of any functionality realized by the slave. Instead, functional aspects such as the inputs, outputs, and parameters are documented in the DCP slave description file which provides all static configuration data of the slave to the master [16]. Only the schema of such a DCP slave description file is explicitly defined as part of the DCP standard while it has to be filled by the implementer of the slave. As a result, DCP supports the exchange of master or slave in line with RQ5. It does also guarantee the support of various transport layer protocols covering TCP and UDP (RQ9).

### 3.2.2 Use Case Definition

The theoretical analysis of the DCP standard suggests that it is a good candidate to achieve standardized and flexible tool coupling solutions for ADS co-simulations. To investigate this in practice, a use case is designed incorporating a DCP compliant interface between a co-simulation master and an external co-simulation slave from the ADS domain. Fig. 1 shows an overview of the realized use case including the features of DCP and the selected tools linked to the raised requirements. A 3D vehicle and environment simulator with runtime rendering is chosen to be integrated as a DCP slave for this work as it represents one of the most prominent cases of tool- or platform-bound models required in ADS simulation, and there is a large diversity in available simulators.

Based on RQ1, RQ2, RQ4, RQ6, and RQ7 the tool xMOD by FEV Software and Testing Solutions GmbH was selected as a co-simulation master. The tool is specifically built for this role and designed to support a wide range of different model integration use cases in the automotive domain [9], [27], [28]. DCP is unspecific in the implementation of the master. This freedom on the master's side allows for the parallel usage of DCP and other model integration methods, e.g., model exchange according to the FMI standard, in a single co-simulation. In spite of not providing specific implementation rules for the master, DCP provides certain

**Table 1** Comparison of FMI and DCP standard with respect to raised requirements for tool coupling in the field of ADS simulation

| RQs | FMI | DCP |
|---|---|---|
| RQ1 (master–slave scheme) | Yes | Yes |
| RQ3 (standardized communication layer for tool coupling) | Tool coupling option via interface FMU, but no standardized master–slave communication | Yes |
| RQ5 (functional independence of interface) | Yes | Yes |
| RQ6 (execution modes) | Supported modes are undefined and not in the standard scope | Communication layer supports NRT, SRT and HRT, supported modes are defined for each slave |
| RQ7 (enabling time synchronization) | Definition of fixed communication points of a tool coupling FMU, master is responsible for synchronization mechanism | Synchronization states for slaves are defined, synchronization mechanism is not defined, master is responsible for synchronization |
| RQ8 (data exchange and parametrization) | Yes | Yes |
| RQ9 (transport layer protocols) | Not addressed by FMI | Native support of TCP, UDP, Bluetooth, USB and CAN |

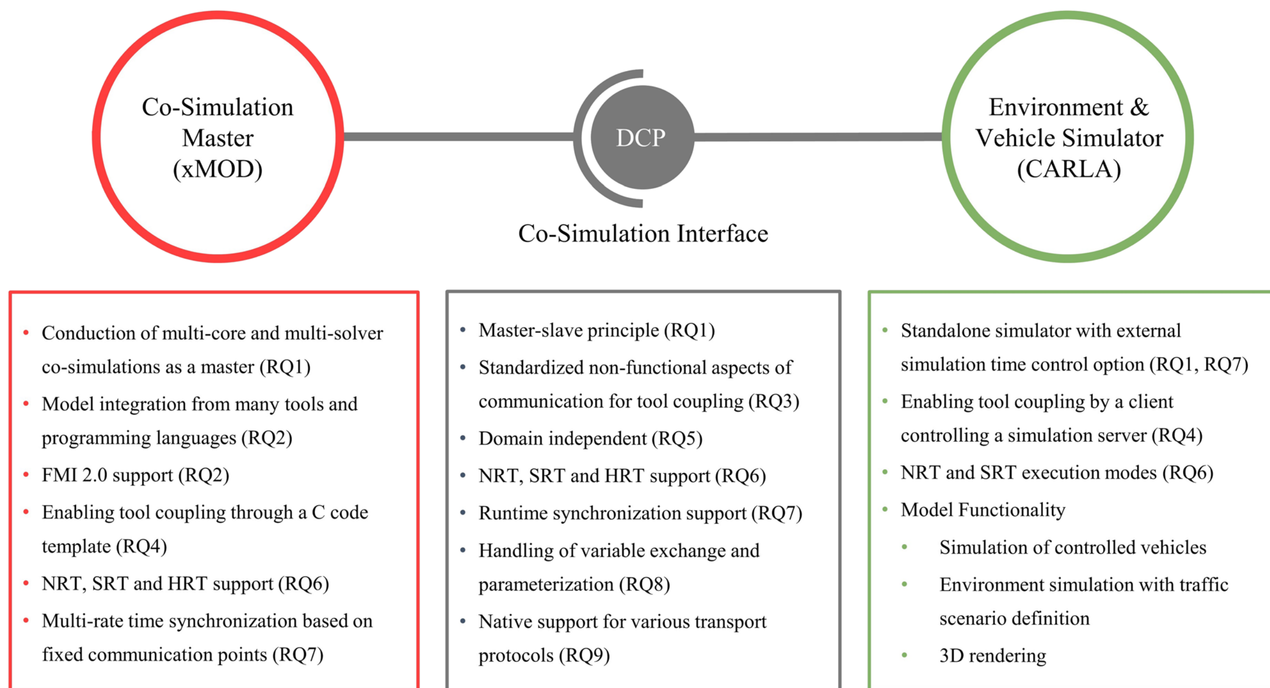


**Fig. 1** Technical concept of co-simulation interface (modified based on Ref. [26])

tasks for the master that has to be realized to turn the co-simulation master into a DCP master. One task is analyzing the DCP slave description file. Other tasks for the master contain to establish a connection to the slave, to provide it with configuration data, to trigger certain state changes, and to handle time synchronization depending on the execution mode which can be NRT, SRT, or HRT [17].

For the role of the vehicle and environment simulation as well as 3D rendering software that is going to be transformed into a DCP slave, the open-source tool CARLA [29] is chosen. CARLA is tailored to the evaluation of automated driving functions. It simulates the road, infrastructure as well as vehicles and their motion which can be controlled manually by a SUT or by traffic behavior models. The scenery and traffic scenario are also rendered based on Unreal Engine [30]. The virtual environment and objects can be captured by several simulated sensors [9]. Bound to the Unreal Engine, CARLA's simulation models are incompatible with standardized model export and rely on tool or platform coupling solutions to be integrated with other models or software, which makes it a good fit for tool coupling solutions (RQ1). The simulation runs on a local server and can be manipulated before and during runtime by a client via APIs which allows for the DCP slave implementation as the client that can be partly reused and serve as a baseline for future implementations. It is mainly designed for SRT execution with real-time visualization, but as the simulated time can be controlled from the outside, NRT execution is also supported. As such, the simulator also fulfills RQ4 and RQ7. With the presented feature set and as one of the most widely used simulators in ADS development (see Refs. [9], [31], [20]), CARLA qualifies for the DCP slave role.

## 4 Software Implementation

The software implementation of the tool coupling consists of two main components, a communication interface in xMOD functioning as a DCP master and a client to control CARLA's simulation server and transform it into a DCP slave (see Fig. 2). DCP compliance of the client has been achieved by creating a reusable DCP wrapper as described in Sect. 4.1. Subsequently, the original client which controls the server of the simulator is embedded into this wrapper. The communication interface that enables the master to act as a DCP master has been embedded into the structure of a proprietary C-code template for co-simulation slaves in xMOD. The main functionalities of the DCP master are analyzing an XML file to gather all necessary information on the DCP slave and providing communication with the DCP slave, as described in Sect. 3.2. In the last step, the DCP slave and master are coupled to achieve time synchronization and communication during runtime.

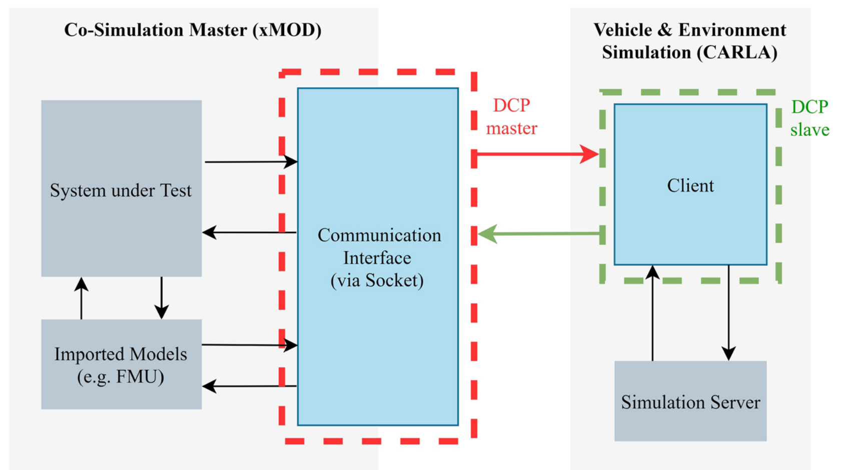


**Fig. 2** Overview of tool coupling implementation

## 4.1 DCP Wrapper

The server of the vehicle and environment simulator CARLA can be controlled by a client via an API (see Fig. 2). This client is realized as a DCP slave. To make the solution tool independent, a DCP wrapper is implemented. The DCP wrapper provides the state chart structure of the slave and the communication in the form of protocol data units (PDU) as defined in the DCP standard. Each state is implemented as a class in Python. A super-state is realized via inheritance of functions and parameters. Each class in the DCP wrapper has the same main structure. It consists of three main functions. The first function, “init”, is called after entering a state and sends a notification PDU to communicate the successful entering of the new state. Afterward the main function, “run”, defines the state-dependent behavior. This includes waiting for signals from the master, calculating outputs, or enabling communication via additional ports. The last main function, “state_change”, triggers state changes.

In DCP, two kinds of state change triggers exist, internal and external. External state changes are triggered by a state change PDU sent by the master. Internal state changes are triggered by the slave fulfilling specific tasks of a state. Thus the slave either checks constantly in the main function “run” for a PDU from the master, as shown in Fig. 3, or processes all internal calculations to fulfill the tasks defined in the “run” function. The received PDUs are checked by If-queries, according to the protocol. In any case, the slave informs the master about state changes.

Additionally, all sent messages by the slave also have to be packed into PDUs according to the protocol. The PDUs are created in PDU-type-specific functions inside the wrapper by packing the data into bytes following the protocol’s rules. The DCP specific rules include the exact order and size of the data for each type of PDU that has to be sent in one packet. For output and input PDUs which contain the functional variables exchanges between the master and the slave, only the header is predefined by the protocol. The packaging of the data itself is determined by the master and communicated to the wrapper via a configuration PDU. The wrapper also receives the number of sockets and their settings which are used to transfer outputs and inputs from the

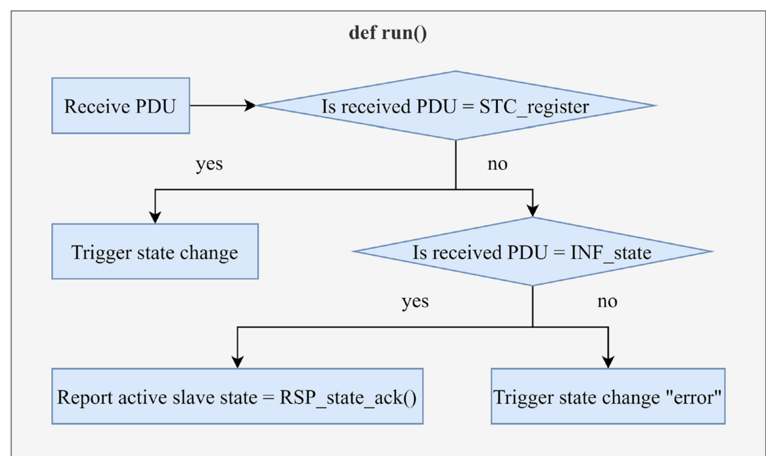


**Fig. 3** Continuously executed If-queries in function “run” for externally triggered state change

master via configuration PDUs. The master reads supported transport protocols from the XML slave description. In general, DCP allows UDP, TCP, Bluetooth, USB, and CAN for communication [15]. The implemented wrapper is prepared for TCP and UDP communication.

The general state chart defined in the DCP standard consists of 19 states, 15 of which are integrated into the developed Python wrapper according to the described method. The creation of the superstate "NonRealTime" and its three sub-states which can only be used in DCP's NRT mode for sample-based synchronization of master and slave is currently omitted. The reason for that is the implementation of a sample-based synchronization mechanism not only for NRT but also for SRT and HRT execution. It guarantees the synchronization of the simulated time in the DCP slave with the co-simulation master's simulated time while the master controls whether the simulated time progresses as fast as possible (NRT) or is synchronized to absolute time (SRT, HRT) outside of the DCP coupling. Although the developed mechanism theoretically is compatible with the NRT superstate (see Section 4.4), an integrated solution could not have been applied for DCP's SRT and HRT mode due to the missing states. Due to the use case focusing on SRT operation with real-time visualization in CARLA, a separate version including the "NonRealTime" superstate is not implemented.

Another DCP state that is not considered is "ERROR-RESOLVED". This state shall enable a reset of the DCP slave and can be internally entered if an error occurred but is healed. This mechanism can prevent the termination of the slave. As the implemented DCP slave does not feature such self-healing functionalities, this state is not relevant.

### 4.2 Client Integration into DCP Wrapper

To use the environment simulation, a client has to be implemented and connected to the simulator's server. The server runs the simulation itself and does all necessary calculations. The client controls the server and its settings. Also, it provides an interface to provide data for the calculations and receive data from the server.

In order to read and set values in the simulation, a Python API is provided by the tool vendor to connect to the server during runtime. To simulate the vehicle, environment, and SUT, the client has to set up the simulation via the Python API accordingly to the use case. To follow the DCP standard, the client is embedded into the DCP wrapper. Thus, the phases of the simulator (Initialization, Running, Stopping) are mapped to specific DCP states.

These implemented connections of the DCP states and the communication between client and server are shown in Fig. 4. During the initialization phase of CARLA general parameters, such as sample time and operation mode as well as scenario-specific parameters, such as the number of vehicles, weather, and map, must be set. These settings have to be known to the clients that communicate them to the server. Thus, the first step of the initialization phase is to connect the server and client of the vehicle and environment simulation. In that DCP defines that the output variables are sent to the master in state "SENDING_I" for the first time, the initialization must be completed at last when this state is entered. Additionally, exiting the state "CONFIGURING" is only possible if a predefined start condition is met. This start condition depends on the parameters received by the master in the state "CONFIGURATION". The configuration parameters can be divided into two groups. The first group consists of DCP specific parameters such as resolution or sample time, which is defined as the fixed time step length used for the iterative calculation inside the DCP slave, transport protocol parameters and communication type. The second group is used to parametrize the DCP slave's functionality from the master and e.g. includes the sample time, number of ego vehicles, traffic participants, and map choice in the CARLA simulator.

The execution time of the vehicle and environment simulation is controlled by the client via the Python API by triggering each calculation sample individually during runtime (CARLA synchronous mode). According to DCP, the execution time is only allowed to proceed during the superstate "Run". During runtime the slave exchanges inputs and outputs for the simulated scenario with the master. For the vehicle and environment simulation in CARLA, these include motion data such as the position, orientation, velocity, and acceleration of vehicles provided to the master and vehicle control data such as acceleration, brake, and steering received from the master. The DCP compliant client continuously forwards new input variables to the simulation server and reads the recent output variables from it. Static data as the position of infrastructure is provided only once during the state "SENDING_I".

During CARLA's stopping phase, the scenario which was built during the initialization phase has to be dismantled for the server to be closed safely. Thus, this phase is handled during the DCP state "STOPPING". The dismantling includes the deletion of vehicles and traffic participants as well as changing back the operation mode to unsynchronized mode.

### 4.3 Preparation of DCP Master

The co-simulation master software xMOD is adapted to function as a DCP Master. To achieve the functionalities of a DCP Master, which are listed in Section 3.2, the option of the software to include C-code into the co-simulation as a slave is used. As DCP does not provide a specific structure for the master, the C-code structure provided by xMOD is

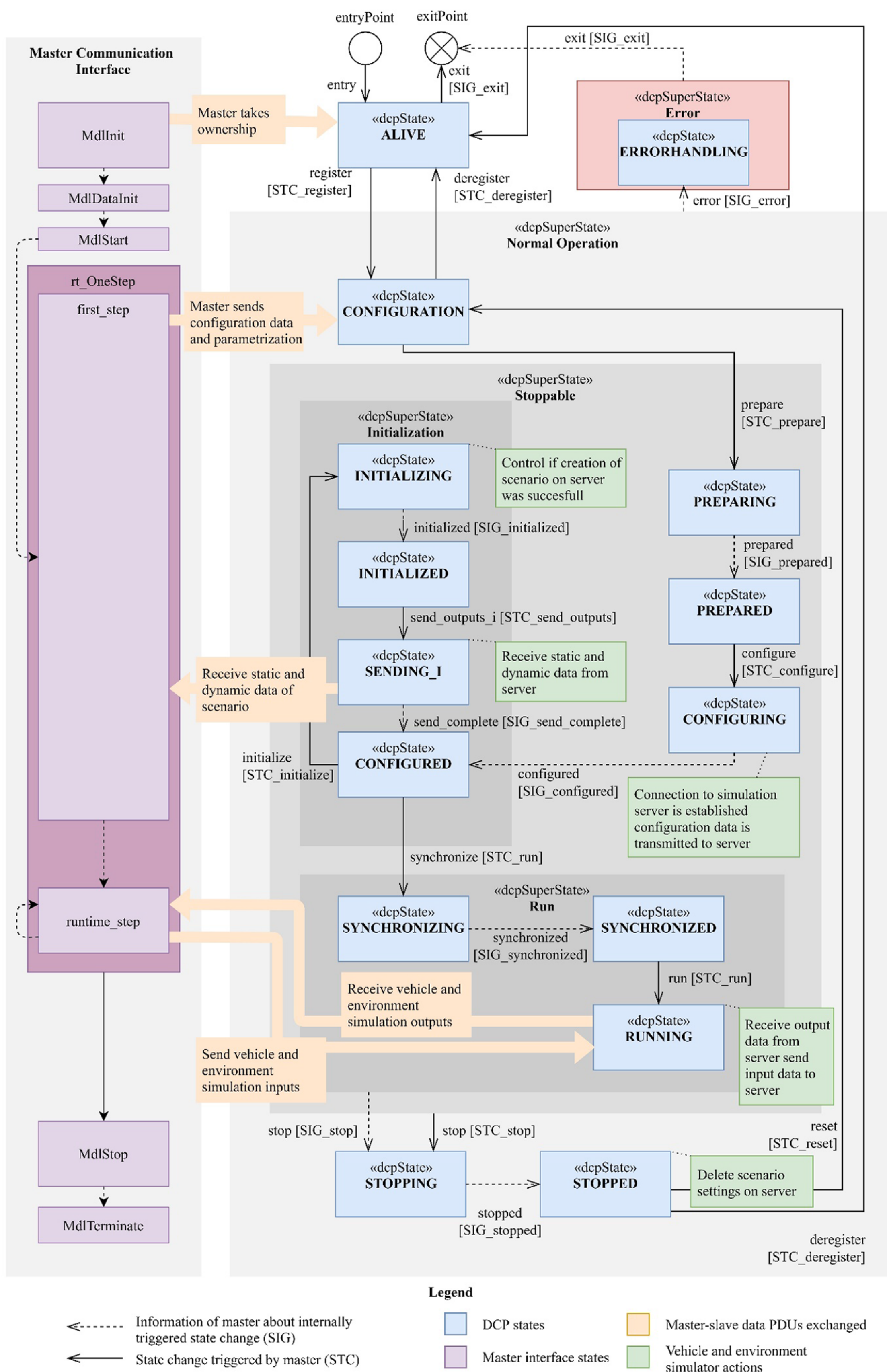


**Fig. 4** Implemented DCP slave functionality mapped to co-simulation master functions (realized DCP states as defined in Ref. [15])

used. The resulting architecture of the master interface is a series of states called in a specific sequence shown on the left-hand side of Fig. 4.

During the "MdlInit" function all available data of the slave is stored in objects. This is done by an implementation of a new class that has objects and parameters for all possible data types of a DCP slave. This concept provides the master with the data at any time after the initial analysis of the DCP slave description file. The master reads the XML file and extracts the information, where protocols and operating modes are supported by the slave and what kind of parameters, inputs, and outputs are available. This data is used to create sockets for communication with the slave. The master dynamically adapts to the number and type of inputs and outputs specified in the file and accordingly passes the data to other tools in the simulation platform.

The messages sent to and received from the slave have to be in PDU format. To transform the data into PDUs and to extract it from PDUs, a function for each kind of PDU is implemented that writes the data into a buffer in accordance with DCP and returns a pointer to the message. Thus, the master can access these functions at any time during the simulation to send any kind of PDU to the slave.

The master uses a fixed-step time discretization. The function "rt_OneStep" is called for each time step once the initialization phase is finished and the simulation is running. For the DCP implementation, it is divided into two parts because the parameters that have to be communicated to the slave in order to finish the configuration phase of DCP and switch from the state "CONFIGURED" to the superstate "Run" are only available from the first call of "rt_OneStep" in the master. However, the parameters only have to be transmitted once before the simulation starts. Therefore, the parameters are transmitted in the sub-function "first_step" which is called only once while the sub-function "runtime_step" handles repetitive tasks because it is called each time step. To define which sub-function is called an if-query is used.

## 4.4 Master–Slave Interaction

To achieve proper interoperability the states of the prepared DCP master and the vehicle and environment simulator prepared as a DCP slave must be mapped to each other. Additionally, a DCP compliant time synchronization mechanism has to be set up. Based on the DCP data model, the master is informed at any time about the current state of the slave. Also, the master has to trigger state changes regularly and exchange PDUs which do not trigger state changes but are used to exchange simulation model data (see Fig. 4).

While the master executes the function "first_step", the slave runs through all states until the superstate "Run" is entered. During simulation runtime, the master is in the function "runtime_step" and the slave is in super state "Running". In these states, the simulated time synchronization of the co-simulation master and the environment simulator is handled. This means that the slave's calculation steps that represent a fixed simulated time interval are aligned with the calculated simulated time inside the master. The simulated time is generally independent of absolute time. For NRT operation, the simulated time is normally accelerated as much as possible on the respective platform. For real-time simulation, a synchronization must be performed to achieve that one unit of elapsed simulated time corresponds to the same unit of absolute time [17]. To operate properly, DCP requires time synchronization in its superstate "Run", but according to the specification it explicitly excludes mechanisms to achieve synchronization [15]. However, as explained in Section 4.1, the additional super state "NonRealTime" defines a time-step-based computation of results by the slave for DCP's NRT execution mode. As triggering the execution of calculation steps by the master impacts the progression of simulated time in the slave, the states inside this super state are inseparable from a solution for synchronization. Due to the high relevance of SRT simulation for the use case, a synchronization mechanism was implemented instead, which is based on the exchange of data PDUs in the state "Running" and is applicable for NRT, SRT, and HRT operation, while xMOD as a master supports the controlling of the relation between simulated and absolute time. Formal support of the "NonRealTime" superstate requires a new implementation variant where data PDUs used for synchronization must be partly converted into control PDUs for state changes, and time step operations must be wrapped in additional states.

To realize synchronization of simulated times for fixed step time discretization, xMOD determines an execution order for all slaves based on their sample times and signal dependency. A slave must provide the possibility to trigger a calculation step and must provide the information that a calculation step has been conducted to enable xMOD to arrange the parallel and serial execution of models and keep the simulated time of each slave synchronized to its own simulated time which can optionally be synchronized with absolute time. The eventual wait times between time steps for each tool are not known before runtime, because the calculation speeds vary and the execution order is unknown to the slaves.

Slave and master need to interchange data to calculate the next simulation step. As shown in Fig. 5, the slave listens for inputs from the master, sent as PDUs. Afterward, the DCP communication interface in the master transitions into an idle state until it receives further notifications from the DCP slave. During this waiting period, xMOD performs other tasks such as executing other models included in the co-simulation. The slave sends the received data to the environment simulation server,

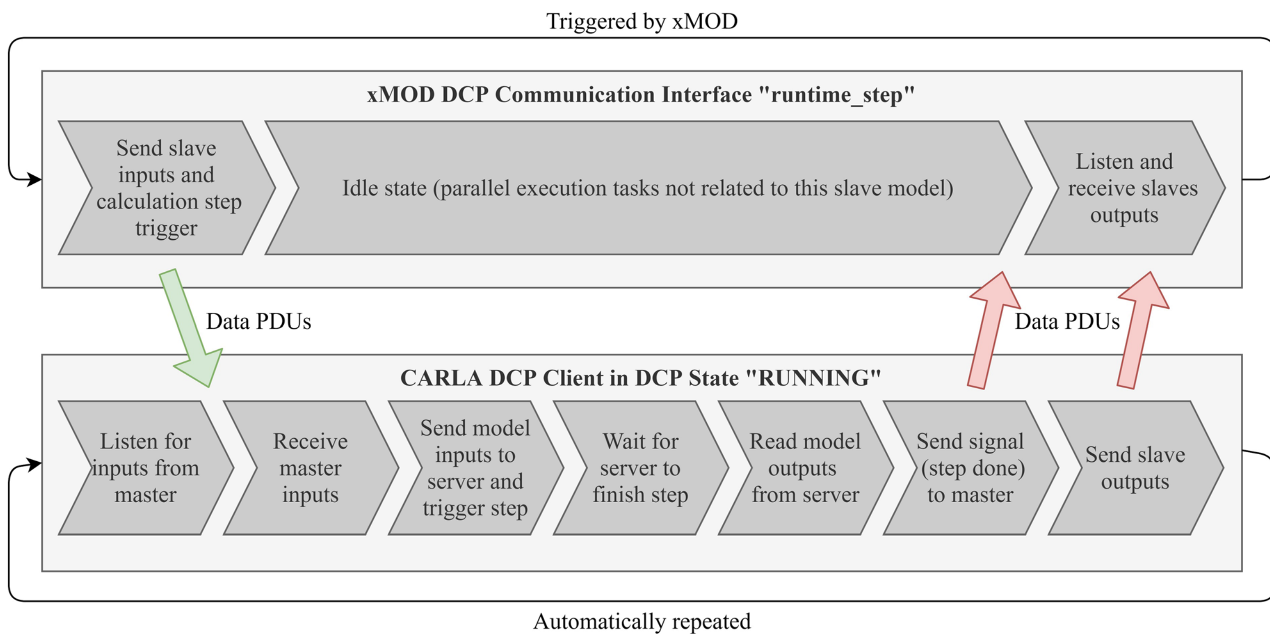


**Fig. 5** DCP compliant mechanism for synchronization of simulated times for xMOD (DCP Master) and CARLA (DCP Slave)

triggers the next time step, and then waits until the server finished processing the data and calculating the time step. Subsequently, the server provides the data to the DCP compliant client which sends a signal to the master to notify it that the output data is available. The master communication interface leaves its idle state and starts listening again on the specific ports where it receives the PDUs the slave is sending. By applying this method, the master is not loaded by constantly listening on all ports. Once all output data has been received the master's communication interface is ready for the next "runtime_step" which is triggered by xMOD when the DCP slave shall perform the next calculation step. The DCP slave client automatically starts listening for new inputs from the master again. As such, the process is repeated each time step.

By utilizing data exchanged as PDUs in the explained manner, time synchronization is achieved. This architecture is applicable for HRT, SRT, and NRT execution. For real-time operation, additional pre-conditions must be guaranteed. The co-simulation master has to be capable of synchronizing the global simulated time to real-time, and all slaves including the DCP slave have to finish a calculation step and provide the output data to the master in a shorter time frame than the chosen step size for the specific slave model.

## 5 Validation

For the validation of the implementation, two main steps were processed. First, the wrapper was tested on DCP compliance. Second, the interoperability interface between the co-simulation master (DCP Master) and the 3D vehicle and environment simulator (DCP Slave) was tested in a MIL simulation for the evaluation of the collaborative cruise control (CACC) function.

### 5.1 Verification of Standard Compliance of DCP Slave Wrapper

The DCP wrapper was tested during an interoperability verification event at the Jubilee Symposium in Lund/Sweden in 2019 of the Modelica Association responsible for the DCP standard. The goal of the event was to cross-test different DCP slave or master implementations and to perform a protocol-based verification of these as described by Krammer et al. [32] in a joint operation to ensure their correctness and also the applicability of the standard specification. Multiple scenarios for testing the

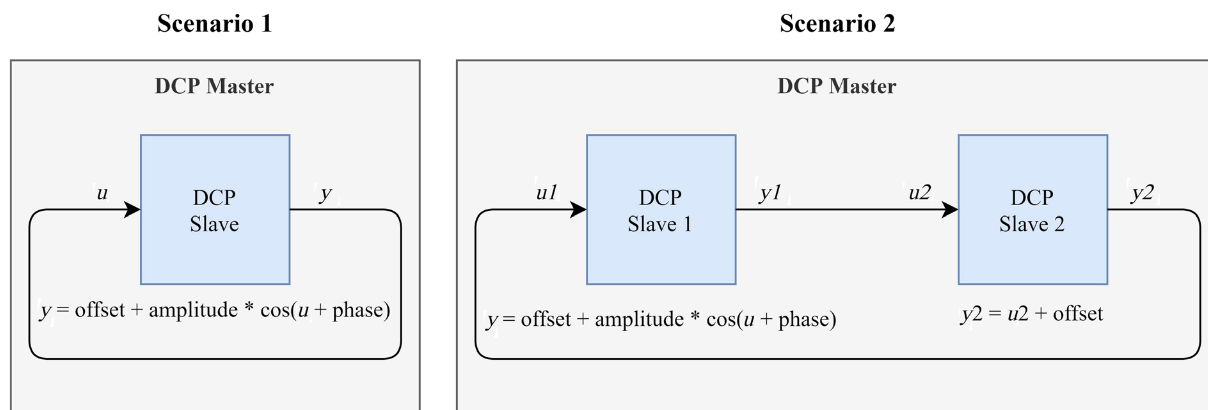


**Fig. 6** Two exemplary test scenarios defined for DCP interoperability verification [33]

participants' solutions were decided on, two of which are shown in Fig. 6.

In the first test scenario, the output of one DCP slave is fed back to the input of the same slave while this slave is controlled by a DCP master. The expected functionality during runtime was transforming a single input variable to a single output variable with a trigonometric function including various parameters. In the second test scenario, a DCP slave with this functionality had to be executed in series in a closed-loop with another DCP slave applying an offset to the input signal. The functionality inside the implemented DCP wrapper had to be changed to fit into the scenarios. Instead of operating as the CARLA client, the provided trigonometric function is calculated. The number of outputs and inputs had to be reduced to one, respectively, and the number of parameters to three to cover amplitude, offset and phase. Accordingly, the main function of the states "CONFIGURATION", "CONFIGURING", "INITIALIZING", "SENDING_I", and "RUNNING" was adjusted. The UDP transport protocol was used in all test scenarios for the exchange of input and output data. The transport protocol utilized by the implemented wrapper is selected by adjusting one parameter that can be set by the master when creating the communication sockets.

The communication protocol, the state machine with its transitions, and the configuration of the implemented slave were successively tested using two masters to stimulate it by sending PDUs. The set-up during the plugfest included one local network which connected different PCs running DCP masters and slaves via a network switch. During the test run the state transitions as well as the requested and responded PDUs including the inputs and outputs were monitored manually. By now, protocol-based DCP verifications can alternatively be carried out automatically using the DCP Tester [32].

During the DCP plugfest it was possible to verify the creation of communication sockets, correct sending and receiving of PDUs, and the correct series of state changes. Therefore, the listed functionality was cross-checked and the intended functionality was shown. These test results confirm that the DCP slave is able to handle the executed sequences of sent and received PDUs. However, it does not guarantee that it is fault-free and will never violate the standard specification [32].

## 5.2 Validation Through a Demonstrator

In order to validate the coupled implementations including both the DCP master in xMOD and the DCP slave client controlling CARLA, a demonstrator was set up.

### 5.2.1 Demonstrator Scenario

The co-simulation master and slave were executed on two different workstations to benefit from the performance increase with distributed platform support of DCP. The co-simulation master xMOD was executed on a workstation running Windows 10 as an operating system. The other workstation ran the DCP wrapper with the embedded CARLA client and the 3D vehicle and environment simulation on the CARLA server on a Linux operating system. These two workstations were connected via a local area network (LAN) to enable communication via TCP sockets.

The SUT for the demonstrator was a cooperative adaptive cruise control (CACC) function. CACC regulates the velocity of a vehicle to keep the safety distance to the preceding vehicle based on sensor and vehicle-to-vehicle communication data. The safety distance is defined as the time interval needed to close the gap to the preceding vehicle at the current velocity considering the reaction time [3].

This model was programmed in MATLAB/Simulink, and multiple instances were imported into the co-simulation set-up as FMUs. The CACC models are used to control vehicles in the scenario simulation in CARLA. The output

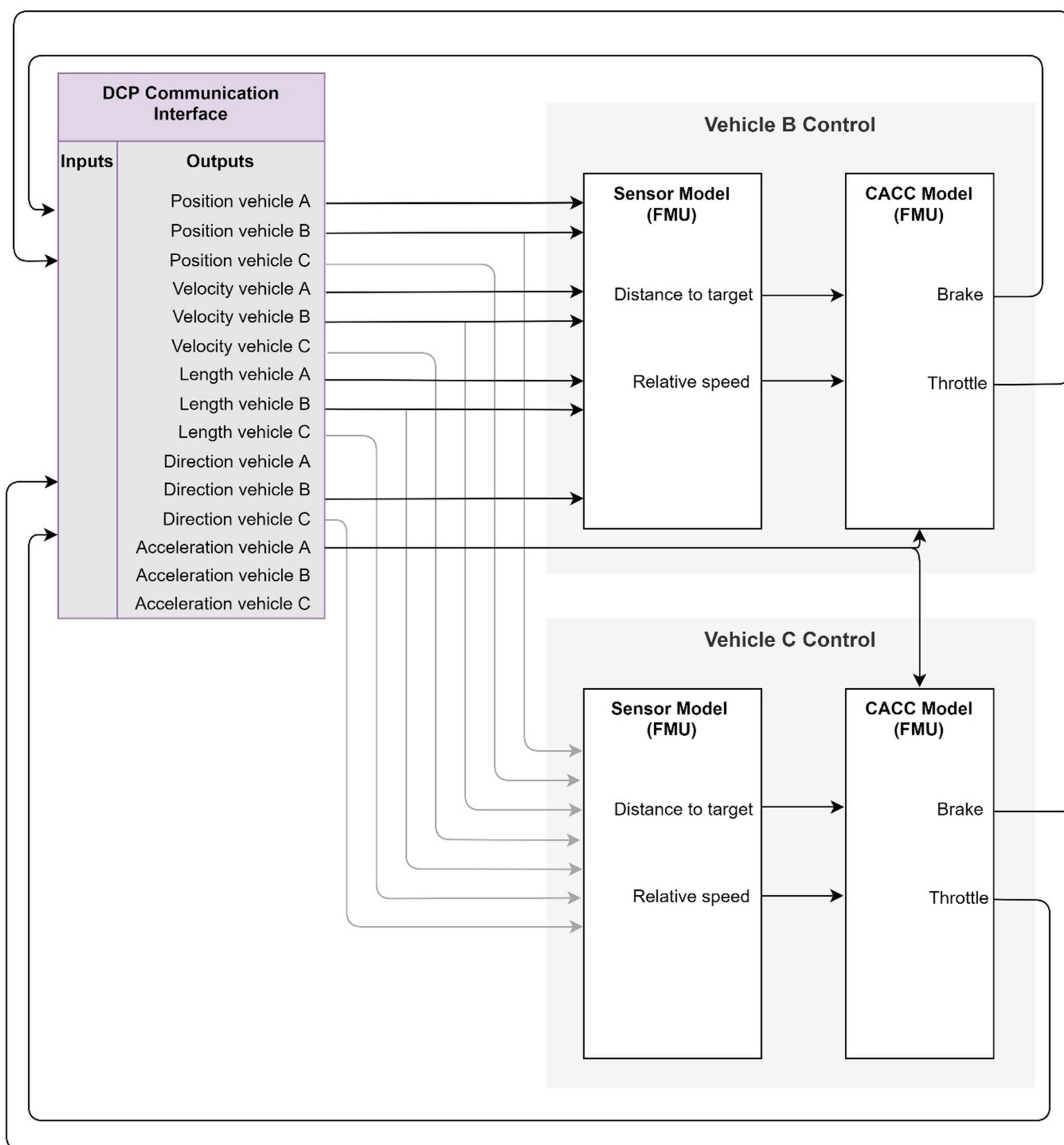


**Fig. 7** Co-simulation model setup for testing DCP implementation

variables from the vehicle and environment simulation processed by the DCP slave client are object-based, thus the world coordinates of each object are provided. The CACC model needs relative values such as a sensor would provide. Therefore an object-based sensor model is integrated into the co-simulation between the DCP communication interface and the CACC model in xMOD which results in the final model configuration (see Fig. 7). The sensor model converts absolute positions into the distance and relative speed with respect to the target as they would be provided by real sensors. The vehicle-to-vehicle communication is simulated in a simplified way by providing both CACC models with the actual acceleration of the platoon leader from CARLA that would be provided via an ad-hoc network in reality. The throttle and brake of the following vehicles are calculated by the two corresponding CACC models and provided as input to the DCP communication interface which forwards the information to CARLA.

A platooning scenario on a highway was chosen as a typical use case of CACC. The platoon consists of three vehicles that are initialized at a distance of 300 m from a roadblock as shown in Fig. 8 (left). The vehicle models are provided

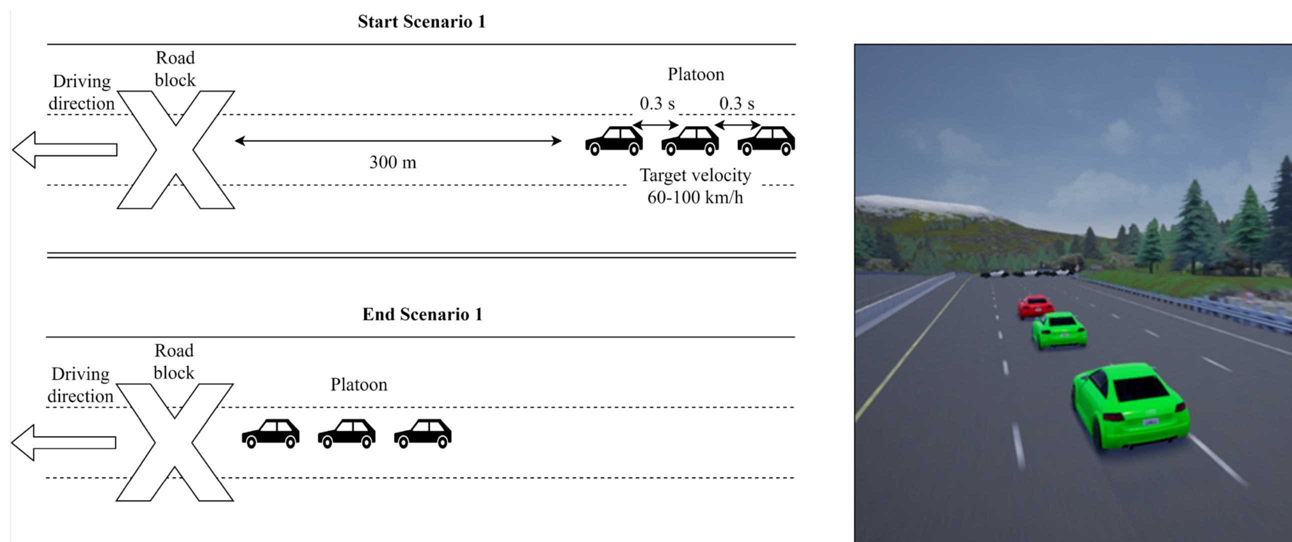


**Fig. 8** Defined (left) and simulated (right) scenario for testing xMOD – CARLA DCP implementation

by CARLA as well as the map and layout of the highway. The platoon leader is controlled by CARLA and configured to approach the roadblock starting at a constant speed before decelerating and coming to a still stand before the roadblock. The two following vehicles are controlled by the CACC models. The integrated CACC prototype FMU has been developed and tested in the CrESt research project and was previously described and analyzed in depth [3], [34]. Thus, this evaluation focuses on the established DCP tool coupling solution between xMOD and CARLA and its applicability for tool integration in ADS co-simulations, while simulation results are not analyzed for functional correctness. In particular, the DCP implementation is also analyzed with respect to execution modes, data exchange, and synchronization of simulated time. The presented solution has also been integrated into a tool platform for co-simulation-based analysis of collaborative embedded systems [9].

### 5.2.2 Demonstrator Results

The execution of a demonstrator scenario running the DCP slave with other, non-DCP slave models was successful (cv. Fig. 8 (right)). The demonstrator verifies multiple requirements of the implemented interoperability interface. Via the standardized DCP interface, the functionality of a vehicle and environment simulator including rendering was integrated into the co-simulation for CACC testing without the need for proprietary solutions for synchronization or data exchange. The distribution of the models on different hardware platforms such as PCs, enabled by DCP, is demonstrated. Distributing the models is an advantage in terms of calculation speed due to the extra performance an additional system provides. Also, the tools do not all need to be compliant with the same operating system.

The demonstrator was run in NRT as well as SRT mode successfully. The implemented data exchange and synchronization mechanisms for simulated times were investigated. It was found that the received and sent data between the DCP master and slave at the beginning and at the end of each time step, as described in Sect. 4.4, match and always correspond to the correct simulated times. This proves that the simulated times in xMOD and CARLA are synchronous, and data exchange is reliable. The TCP transport protocol was used for this analysis to exclude information loss on the physical layer of communication.

The execution of the simulation revealed that the simulated time could not be synchronized to absolute time as the absolute time to carry out a step occasionally exceeds the sample time. This means the simulation cannot be run in SRT or faster-than-real-time mode on the current platform. A detailed analysis to what extent the calculation speed is related to the interoperability interface has not yet been conducted as the speed is not only impacted by the interface, but also by the computation power of the machines, the complexity of the models, and the efficiency of the simulators. However, it was found that Python as a programming language used for the DCP slave and CARLA API slows down the overall simulation. This was proven by implementing an alternative client in C++ using CARLA's newer C++ API that showed a significant acceleration of the simulated time. The C++ implementation was not fully DCP compliant, but it provided proprietary communication for the presented test scenario using the same sample time and data exchanged at simulation runtime. In this specific scenario, the average absolute time to calculate a simulation step was reduced by

approximately 30%, enabling faster-than-real-time simulation. Additionally, it was found that it is possible to accelerate the simulation by increasing the number of PDUs sent via a single socket and by reducing the number of sockets as this saves resources spent for listening on ports and establishing connections. If the number of sockets is reduced too much, the likelihood of packet collisions increases. Thus, by reducing the number of sockets, not all sent packets will be received correctly. For TCP connections, this leads to the retransmission of PDUs, which costs performance and counteracts the positive effect.

## 6 Critical Reflection

This paper focuses on the application of the DCP standard in the development of automated driving functions with the example of a CACC controller. To prove the usability of the standard in this domain, a DCP interoperability interface was implemented and demonstrated in the use case of the CACC development. The scope is limited to the non-functional coupling of software tools for the realization of the use case. In particular, the DCP standard was implemented for the coupling of the co-simulation platform xMOD as a master with a 3D vehicle and environment simulation in CARLA as a slave. The non-functional compatibility and successful realization of a distributed co-simulation on two platforms were successfully demonstrated with the implemented toolchain. This illustrates how standardized, non-functional tool or platform coupling via DCP can be used effectively in co-simulation for virtual testing of ADS. The spreading of DCP implementations in this field eases the setup of co-simulations, especially considering the large tool landscape, as non-functional interoperability of DCP compatible simulators is guaranteed. Functional compatibility of the two simulation components is not focused or ensured by the described concept as it is not standardized by DCP and must therefore be ensured manually for specific applications. Opaque data types, which are relevant in specific SIL use cases for inter- or intra-communication of virtualized ECUs, are currently not considered by the DCP standard. Thus DCP is not applicable for co-simulation interfaces between these SIL components. To cover such use cases opaque data types could be considered in future revisions of DCP, e.g., by using dynamic length byte arrays in the PDU-oriented data model. Alternatively, a data exchange mechanism relying on a serialization library could be investigated. This would increase the flexibility to exchange more complex data structures between co-simulation partners and could reduce the DCP implementation effort, but would also potentially negatively impact the performance depending on the chosen library. The highlighted standardization gaps such as functional standardization for ADS simulation components as well as non-functional extensions could be filled by following the idea of layered standards introduced by FMI. Extensions of a standard's, e.g., DCP's base definitions could be defined and, in case of achieving a high degree of adoption and importance, could be integrated as optional or mandatory in future revisions.

The implemented toolchain uses the commercial tool xMOD as a co-simulation master. This tool has been extended by the functionality of the DCP capability. The use of other co-simulation platforms as masters is not considered in this publication. Even if other platforms exist and are basically suitable, adaptations for DCP compatibility must be implemented to enable suitability.

For the DCP slave realization, the client of the tool CARLA was combined with a universal DCP wrapper. The validity and reuse potential of the wrapper concept could be proven within the plugfest and by the examined use case. Nevertheless, further validation with other tools is useful to show the general validity. This is especially true since the plugfest uses firmly simplified and functionally primitive models and the focus was on the examination of formal compatibility. The investigation of functionally complex models of other tools should therefore be additionally validated in further steps. The implemented wrapper and synchronization mechanism do not support the superstate "NonRealTime" for DCP's NRT execution mode in their current state, which should be adjusted in a future revision. It was found that DCP's NRT mode is currently not compatible with a universal, step-based synchronization mechanism based on data PDUs that satisfies DCP's SRT and HRT modes, although it is also applicable for NRT simulations. In the light of ongoing investigation on the topic of synchronization with DCP (see [35]) and aiming for a unified solution, an issue was raised in the DCP standardization group.

Furthermore, the studies were limited to co-simulation incorporating software models as test objects. A coupling with real hardware components such as sensors was not part of the consideration and the theoretical benefits provided by a DCP compliant tool coupling for such use cases should be investigated in more detail in the future.

Especially the use of hardware components requires the HRT capability of the co-simulation. It should be further evaluated to what extent the discovered simulation speed behavior is related to the DCP implementation. Further investigations are necessary to investigate HRT which is not supported by CARLA in addition to SRT and NRT. Especially the use of the UDP instead of TCP to potentially speed up the simulation should be additionally investigated in this context. When using UDP, the previously explained possible packet collisions in case of high socket loads should be considered as these can lead to PDUs being lost. Therefore, the communication and thus the calculations will become less reliable which must be investigated for the specific

application with respect to countermeasures and acceptable drop rates.

## 7 Conclusions

The main goal of this work is to develop a DCP implementation to achieve a standardized integration of autonomous simulation tools into a co-simulation. The realization of a DCP master functionality inside the co-simulation master xMOD and transformation of the CARLA simulator into a DCP slave demonstrate the applicability of DCP for ADS use cases successfully. It is shown that different simulators can be coupled effectively for co-simulation using this approach. Widespread industrialization of the DCP technology in ADS simulation tools could potentially drastically minimize the efforts for creating tool coupling interfaces by ensuring interoperability on the communication layer. As a next step, a detailed study of the saved efforts, and also advantages and disadvantages in general when using DCP or alternative coupling standards instead of proprietary simulator couplings should be conducted. K. Albers et al. presented the first analysis of this in Ref. [9] and found a higher implementation effort for implementing standardized simulator couplings and effort savings when reusing it. Under this assumption, the overall efficiency is increased in the long term which is an argument for using standardized solutions such as DCP in new simulation domains as long as the technical boundaries of a standard do not contradict the requirements of the use case.

It is demonstrated that DCP fulfills all major requirements of interfaces between ADS simulation tools and can be used by co-simulation masters alongside other co-simulation and model exchange methods. The chosen approach for the DCP slave using a functionally independent wrapper shows how components of a DCP implementation can be reused for future realizations.

Finally, some technical aspects have been identified that are worth investigating further such as the carry-over of the implementation to other tools and platforms or time synchronization mechanisms with DCP. Specifically, the applicability of DCP in cloud environments, e.g., to couple virtual machines, should be investigated as there is a need in ADS development to provide highly time-efficient software-in-the-loop frameworks. Regarding the impact of DCP compliance on the overall calculation speed, an in-depth analysis should be performed. This includes tracking of absolute times for the individual operations within one time step such as communication delays, processing times, and calculations performed by the simulators. Furthermore, the options for HRT simulations in the field of ADS can be explored including the evaluation of UDP and other transport layer protocols.

**Funding** Open Access funding enabled and organized by Projekt DEAL. This work was supported in part by the German Ministry of Education and Research (BMBF) under grant 01IS16043.

## Declarations

**Conflict of interest** The authors declare that they have no conflict of interest.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

## References

1. ECE/TRANS/WP. 29/2019/34/Rev. 1: Framework document on automated/autonomous vehicles. Economic Commission for Europe, Inland Transport Committee, World Forum for Harmonization of Vehicle Regulations, 178th session, Geneva, June 2019
2. Wood, M., Wittmann, D., Liu, S., et al.: Safety first for automated driving. https://www.daimler.com/dokumente/innovation/sonstiges/safety-first-for-automated-driving.pdf (2019). Accessed 25 October 2020
3. Meyer, M., Granrath, C., Feyerl, G., Richenhagen, J., Kaths, J., Andert, J.: Closed-loop platoon simulation with cooperative intelligent transportation systems based on vehicle-to-x communication. Simul Model Pract Theory (2021). https://doi.org/10.1016/j.simpat.2020.102173
4. Cioroaica, E., Albers, K., Boehm, W., Pudlitz, F., Granrath, C., Rosen, R., Wehrstedt, J.C.: Development and evaluation of collaborative embedded systems using simulation. In: Böhm, W., Broy, M., Klein, C., Pohl, K., Rumpe, B., Schröck, S. (eds.) Model-Based Engineering of Collaborative Embedded Systems. Springer, Berlin (2021)
5. ISO PAS 21448:2019: Road vehicles – Safety of the intended functionality. International Organization for Standardization, Geneva, Switzerland (2019). https://www.iso.org/standard/70939.html
6. Schmidt, S., Elbs, M.: Automotive systems engineering enabled by virtual prototypes. In: Bargende, M., Reuss, H., Wiedemann, J. (eds.) Internationales stuttgarter symposium proceedings. Springer, Wiesbaden (2018)
7. Kapinski, J., Deshmukh, J.V., Xiaoqing, J., Hisahiro, I., Butts, K.: Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques. IEEE Control Syst. Mag. **36**(6), 45–64 (2016)
8. Hakuli, S., Krug, M.: Virtual integration in the development process of ADAS. In: Winner, H., Hakuli, S., Lotz, F., Singer, C. (eds.) Handbook of Driver Assistance Systems. Springer, Berlin (2015)
9. Albers, K., Bolte, B., Meyer, M., Terfloth, A., Wißdorf, A.: Tool support for co-simulation-based analysis. In: Böhm, W., Broy, M., Klein, C., Pohl, K., Rumpe, B., Schröck, S. (eds.) Model-based

engineering of collaborative embedded systems. Springer, Berlin (2021)
10. Modelica Association Project FMI: Functional mock-up interface specification. Version 64748c4. Modelica Association, Linköping, Sweden. https://fmi-standard.org/docs/3.0-dev (2021). Accessed 3 March 2021. Licensed under CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0/legalcode)
11. Morse, K.L., Petty, M.D.: High level architecture data distribution management migration from DoD 1.3 to IEEE 1516. Concurrency and Computation: Pract. & Exper. **16**, 1527–1543 (2004)
12. Blochwitz, T., Otter, M., Akesson, J., et al.: Functional Mockup Interface 2.0: The standard for tool independent exchange of simulation models. In: Otter, M., Zimmer, D. (Eds.) Proceedings of the 9th International MODELICA Conference, Munich, Germany, 3–5 September 2012. Linköping Electronic Conference Proceedings, vol. 76, pp. 173–184. Modelica Association and Linköping University Electronic Press, Linköping (2012)
13. Granrath, C., Meyer, M., Ewald, J., et al.: EleMA: A reference simulation model architecture and interface standard for modeling and testing of electric vehicles. Transportation (2020). https://doi.org/10.1016/j.etran.2020.100060
14. Nguyen, V.H., Besanger, Y., Tran, Q.T., et al.: Using power-hardware-in-the-loop experiments together with co-simulation for the holistic validation of cyber-physical energy systems. In: IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), Torino, Italy, 26–29 September 2017. Conference Proceedings, pp. 1–6 (2017)
15. Modelica Association Project DCP: Distributed Co-simulation Protocol (DCP). Specification Document 1.0.0. Modelica Association, Linköping, Sweden. https://dcp-standard.org/assets/specification/DCP_Specification_v1.0.pdf (2019). Accessed 5 October 2020. Licensed under CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0/legalcode)
16. Baumann, P., Krammer, M., Driussi, M., et al.: Using the distributed co-simulation protocol for a mixed real-virtual prototype. In: 2019 IEEE International Conference on Mechatronics (ICM), Ilmenau, Germany, 18–20 March 2019. Conference Proceedings, pp. 440–445 (2019)
17. Krammer, M., Benedikt, M., Blochwitz, T., et al.: The distributed co-simulation protocol for the integration of real-time systems and simulation environments. In: Proceedings of the 50th Computer Simulation Conference, Bordeaux, France, 9–12 July 2018
18. Buse, D.S., Dressler, F.: Towards real-time interactive V2X simulation. In: 2019 IEEE Vehicular Networking Conference (VNC), Los Angeles, CA, USA, December 2019. Conference Proceedings, pp. 114–121 (2019)
19. Choudhury, A., Maszczyk, T., Math, C.B., Li, H., Dauwels, J.: An integrated simulation environment for testing V2X protocols and applications. Procedia Comput. Sci. **80**, 2042–2052 (2016)
20. Stević, S., Krunić, M., Dragojević, M., Kaprocki, N.: Development and validation of ADAS perception application in ROS environment integrated with CARLA simulator. In: 2019 27th Telecommunications Forum (TELFOR), Belgrade, Serbia, 26–27 November 2019
21. Yamaura, M., Aréchiga, N., Shiraishi, S., et al.: ADAS virtual prototyping using Modelica and Unity co-simulation via OpenMETA. In: The First Japanese Modelica Conferences, Tokyo, Japan, 23-24 May 2014. Linköping Electronic Conference Proceedings, vol. 124, pp. 43-49. Modelica Association and Linköping University Electronic Press, Linköping (2014)
22. Sztipanovits, J., Bapty, T., Neema, S., Howard, L., Jackson, E.: OpenMETA: A model- and component-based design tool chain for cyber-physical systems. In: Bensalem, S., Lakhneck, Y., Legay, A. (eds.) From programs to systems. The systems perspective in computing lecture notes in computer science. Springer, Berlin (2014)
23. Luttkus, L., Mikelsons, L., Baumann, P., Kotte, O.: A simulation based interaction analysis of automated vehicles. In: SummerSim '19: Proceedings of the 2019 Summer Simulation Conference, Berlin, Germany, July 2019. Article 6, pp. 1–11. Society for Computer Simulation International (2019)
24. Krammer, M., Schuch, K., Kater, C., et al.: Standardized integration of real-time and non-real-time systems: The Distributed Co-simulation Protocol. In: 13th International Modelica Conference, Regensburg, Deutschland, 4–6 March 2019
25. Kumar, S., Dalal, S., Dixit, V.: The OSI model: Overview on the seven layers of computer networks. Int. J. Innov. Res. Sci. Technol. (IJIRST) **2**(3), 461–466 (2014)
26. Meyer, M., Granrath, C., Jäckel, N., Wachtmeister, L.: Methods for the development of collaborative embedded systems in automated vehicles. ATZ Electron. Worldw. **15**, 58–63 (2020). https://doi.org/10.1007/s38314-020-0294-z
27. Ewald, J., Orth, P., Granrath, C., Andert, J.: Model-based systems engineering for standardized simulation frameworks: Case study development of electrical vehicles. In: Proceedings of the 41st International Vienna Motor Symposium, 22–24 April 2020
28. Ben Gaid, M., Corde, G., Chasse, A., et al.: Heterogeneous model integration and virtual experimentation using xMOD: Application to hybrid powertrain design and validation. In: 7th EUROSIM Congress on Modeling and Simulation (EUROSIM'10), Prague, Czech Republic, September 2010
29. Dosovitskiy, A., Ros, G., Codevilla, F., López, A., Koltun, V.: CARLA: An open urban driving simulator. In: 1st Conference on Robot Learning (CoRL), Mountain View, California, USA, 13–15 November 2017
30. Epic Games, Inc.: Unreal Engine. https://www.unrealengine.com. Accessed 25 October 2020.
31. Codevilla, F., Müller, M., López, A., Koltun, V., Dosovitskiy, A.: End-to-end driving via conditional imitation learning. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, 21–25 May 2018. Conference Proceedings, pp. 4693–4700 (2018)
32. Krammer, M., Kater, C., Schiffer, C., Benedikt, M.: A potocol-based verification approach for standard-compliant distributed co-simulation. In: Proceedings of Asian Modelica Conference 2020, Tokyo, Japan, 8–9 October 2020
33. Modelica Association: DCP Plug Fest 2019 announcement. https://dcp-standard.org/news/2019/06/19/plug-fest-announcement.html. Accessed 25 October 2020
34. Kaths, J., Meyer, M., Granrath, et al.: Virtual test drives with multiple vehicles under test for the evaluation of collaborative assisted and automated driving functions. In: Proceedings of ATZlive - Automated Driving 2020, 13–14 October 2020
35. Krammer, M., Ferner, P., Watzenig, D.: Clock synchronization in context of the Distributed Co-Simulation Protocol. In: 2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE), Graz, Austria, 4–8 November 2019. Conference Proceedings, pp. 1–6 (2019)