Synchronization of Hybrid Models in the Automated Driving Simulation

Wojciech Baron*, Christoph Sippl†, Kai-Steffen Hielscher* Reinhard German*
* Computer Networks and Communication Systems
Friedrich-Alexander-Universität Erlangen-Nürnberg
{wojciech.baron, kai-steffen.hielscher, reinhard.german}@fau.de
† Pre-development of Automated Driving, Audi AG
christoph.sippl@audi.de

2022 IEEE 95th Vehicular Technology Conference (VTC2022-Spring) | 978-1-6654-8243-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/VTC2022-Spring54318.2022.9860798

***Abstract*—Developing and testing automated driving functions on public roads or proving grounds is time-consuming and expensive. Driving in virtual environments is increasingly seen as an eligible approach to counteract these drawbacks. The virtual environment is provided by one or more simulation tools and is coupled with software modules of the automated driving function. This results in a feedback loop that controls a virtual vehicle. Since simulation runs are carried out on off-the-shelf computers, determinism issues may occur due to race conditions. However, determinism is a desirable property in order to be able to reasonably benchmark the automated driving function. This work identifies execution patterns of software modules and shows how they can be synchronized by means of mapping to discrete event semantics. The resulting simulation setup produces repeatable results independent of the available computing and network power. The results are demonstrated using an exemplary driving scenario and compared to paced real-time simulation.**

***Index Terms*—Hybrid Discrete Event, Software-in-the-loop, Automated Driving Simulation**

I. Introduction

The future of transportation lies in automation. The launch of highly automated vehicles is facing challenges in implementation combined with an unprecedented, massive testing requirement. The testing effort has its origin in the altered role of the human and the software as the driver of the vehicle. In the case of advanced driver assistance systems (ADAS), the human is the main instance responsible for controlling the vehicle, and he or she is merely supported in performing the driving task. As the degree of automation increases, the boundary shifts and the human only forms the fallback level in the occurrence of a fault, or the driving responsibility is eliminated entirely [1]. This is also reflected in the automotive safety integrity level (ASIL) rating. If ASIL D is assumed (highest safety level), only one safety-critical fault is permitted every 10^8 operating hours [2]. For a potential highway pilot and an average speed of $100km/h$, this corresponds to a distance of 10 billion test kilometers just to determine whether or not an error occurred in this time frame. From this point of view, mere safeguarding on public roads or proving grounds does not seem feasible. It is promising to manage the immense test effort with data replay or with simulation techniques. Replay mechanisms have the downside that traffic scenarios are reproduced statically and only what-if analyses can be conducted. Despite high relevance, there is no interaction between the dynamic traffic and the automated driving function (ADF). This can only be achieved by simulation, in which driver models actually react dynamically to the behavior of the ADF and vice versa.

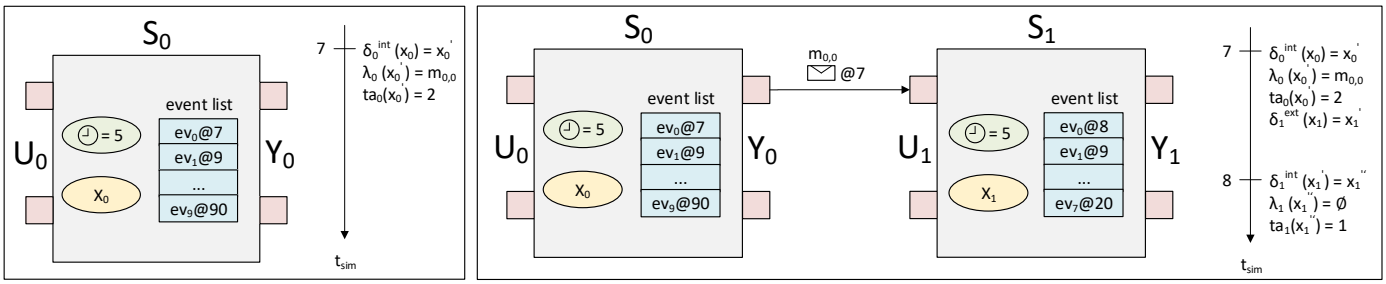
One issue with simulation is certainly its credibility. How should the driven test kilometers or test scenarios be regarded or valued in relation to a real drive? A potential variability in the simulation results is related to the credibility issue and does not exactly build trust. The variation can be caused e.g. by the interconnection of distributed simulation and software and arising synchronization issues. In distributed simulation systems, synchronization is based on partially or totally ordering events by simulation time. For example, conservative or optimistic synchronization methods for discrete event (DE) models are used [3]. In contrast, software modules are always executed in relation to physical time. If the software is distributed, the physical time can be synchronized by means of a time synchronization protocol like precision time protocol (PTP) [4]. The software modules are time- or data-triggered. Scheduling and synchronization on a physical time basis does not prohibit determinism issues. It is known that in this case a distributed real-time simulation is not deterministic due to causality violations caused by message timing, order or loss [5]. Conversely, the use of conservative synchronization algorithms in software modules is not supported because they are defined solely for DE models. In this work, this problem is addressed with a Hybrid DE approach by mapping execution patterns derived from software modules to DE models. This permits the use of established and upcoming conservative synchronization algorithms in this kind of setting and thereby enables deterministic coupling of simulation models and software modules.

II. Fundamentals

For a better understanding of the contribution, the basics of ADF simulation and DE simulation techniques are introduced.

A. Software-in-the-Loop

The interconnection of simulation and software is referred to as software-in-the-loop (SIL) simulation. The goal is to stimulate the software through the simulation so that execution



(a) single (b) coupled

Fig. 1: Sample execution of DE simulation units (SUs).

outside its usual physical deployment is enabled. Time and cost expenses can be reduced e.g. due to the lack of dedicated hardware requirements that result from the execution in a target environment. In the case of automated driving (AD) the number of prototype vehicles and test drives on public roads or proving grounds can be decreased.

In the ADF SIL the ADF controls a virtual vehicle (called ego) that drives through a virtual world and has to master designated challenges in so-called scenarios. The ego vehicle is equipped with sensor models that mimic those in the prototype vehicle and through which it perceives a restricted and noisy snapshot of the virtual world. The inertia of the ego vehicle is replicated by a vehicle dynamics model and the parameterization corresponds to that of a real vehicle. The virtual world including road network, traffic signs and other traffic participants is provided by a submicroscopic traffic simulator. Depending on the specific use case, additional simulation models or tools may be coupled, such as network simulation in the case of cooperative sensing or cooperative driving. This composite of multiple simulation tools and models is referred to as co-simulation [6]. Synchronization is required within the co-simulation to ensure that all simulation events are processed in a causally consistent order. The need for synchronization stems, among other things, from the varying run times of the models in relation to their simulation time progress. The simulation time progress within the co-simulation must be coordinated.

The environmental data provided by the co-simulation is consumed by the ADF that controls the virtual ego vehicle. The exact implementation of an ADF is up to the intellectual property of the respective developers. However, all ADFs share common characteristics. The ADF is a distributed real-time system. The ADF must withstand real-time guarantees within its true physical execution environment. It is distributed because (1) it consists of multiple software modules or tasks and (2) it performs complex computations that must be carried out on parallel, distributed or even heterogeneous hardware. The general operation of the ADF can be explained with robot architecture models such as the sense-model-plan-act (SMPA) [7] model. First, the environment is perceived by installed sensors. The incoming data flows are merged and a unified model of the environment is constructed. Based on parameters such as the destination address and the desired speed and restrictions resulting from the internal environmental model, the next maneuvers are planned and a target trajectory is generated. Finally, control variables such as the steering angle and acceleration are derived from the target trajectory and the actuators are steered consequently.

B. Discrete Event Simulation

DE simulation units are formally described in the discrete event system specification (DEVS) [8]. Minor adjustments are made in [9] to explicitly express that no output needs to be performed by a DE SU. A DE SU S_i is defined as follows:

$$\begin{aligned} S_i &= \langle X_i, U_i, Y_i, \delta_i^{ext}, \delta_i^{int}, \lambda_i, ta_i, q_i(0) \rangle \\ \delta_i^{ext} &: Q_i \times U_i \to X_i \\ \delta_i^{int} &: X_i \to X_i \\ \lambda_i &: X_i \to Y_i \cup \{\phi\} \\ ta_i &: X_i \to \mathbb{R}_0^+ \cup \infty \\ q_i(0) &\in Q_i \\ Q_i &= \{(x, e) \mid x \in X_i \ \wedge 0 \leq e \leq ta_i(x)\}, \end{aligned} \tag{1}$$

where X_i denotes the internal state of S_i, U_i denotes the input to S_i, Y_i denotes the output of S_i, and $q_i(0)$ denotes the initial state of S_i. The function δ_i^{ext} is referred to as the external transition function that computes a new state of S_i based on the current total state q_i and an input event u_i. The function δ_i^{int} is referred to as the internal transition function and computes a new state of S_i based on the current state. The value e denotes the elapsed simulation time since the last internal or external transition. The total state q_i is the tuple of the internal state x_i and the elapsed time e. The function λ_i is referred to as the output event function and computes the output of S_i based on the current state. The function ta_i is referred to as the time advance function that returns the simulation time delta until the next internal event of S_i.

Fig. 1 provides an example of how the theoretical construct works. Internal events are denoted as $ev_i@t$, where i is the event index and t is the simulation timestamp of the event. External events can be regarded as messages and are denoted as $m_{i,j}@t$, where i is the receiver's input port index, j is the message index, and t is the simulation timestamp. In

Fig. 1a, the operation of a single DE unit is first outlined. The DE SU S_0 has two input ports, two output ports, an internal state, an event list and a simulation clock. S_0 is located in simulation time 5. The event list contains the next internal events to be executed, sorted in ascending order by their simulation timestamp. The next internal event ev_0 takes place at simulation time 7. An external state transition by δ_0^{ext} is not possible because S_0 is unconnected and cannot receive external events. At simulation time 7, an internal state transition occurs using δ_0^{int} due to ev_0. The internal state changes from x_0 to $x_0^{'}$. The output function λ_0 produces the output event m_0, which is however unconnected. The event ev_0 is removed from the event list and new internal events can be generated and enqueued in the event list, but this is not the case in this example. Finally, the time advance ta_0, which describes the relative time to the next internal event, is set to $9 - 7 = 2$.

The second example in Fig. 1b is an extension of the first example. S_0 is unchanged, except that an output port of S_0 is connected to an input port of S_1. The sequence starts identically to the one from the first example, because the next internal event of S_0 is earlier than of S_1. This time, the event m_0 is consumed by S_1 at time 7 and leads to an external state transition by δ_1^{ext} that modifies the internal state from x_1 to $x_1^{'}$. At simulation time 8, S_1 performs an internal state transition. S_1 emits no output events and the output function λ_1 returns an empty set.

C. Synchronization

The above example shows a sequential solution to the co-simulation problem. Parallel Discrete Event Simulation (PDES) research [10] is concerned with the parallel yet causally correct execution of the coupling problem. The execution is considered valid if the parallel execution yields the same results as the sequential execution or if all internal and external events of each SU are executed in ascending simulation time order. There are two classes of synchronization algorithms: conservative and optimistic [3]. In optimistic algorithms, regular state saving is performed. Detected causality violations lead to a rollback and reset the simulation to a state from the past. With conservative algorithms, blocking occurs when correct execution cannot be ensured. The simulation units know when to block and wait for possibly incoming events based on a threshold value called lower bound timestamp (LBTS). It is calculated from the commitment of each simulation unit to send only outgoing events with a minimum simulation time. In addition, scrambling of individual messages may occur. However, since these must be presented as input values in ascending simulation time order, a timestamp order (TSO) queue is frequently used.

The operation of the TSO queue is illustrated in Fig. 2. Two messages reside in the upper receive buffer. The message $m_{1,0}$ in the bottom receive buffer is in transit. The message in transit has a lower simulation timestamp than the messages already received. So consuming these messages would be erroneous. This is prevented by the LBTS and the TSO queue. The

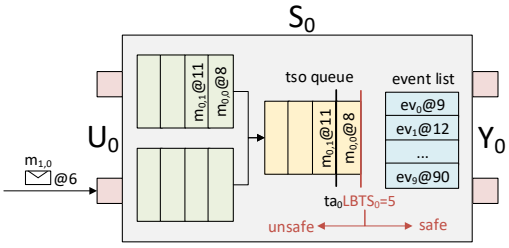


Fig. 2: DE SU with a TSO queue in a blocked state.

TSO queue contains all received messages sorted in ascending simulation time order. In addition, the ta_0 and $LBTS_0$ values are marked. The LBTS value in this case is 5 and is therefore smaller than all messages in the TSO queue. The messages are thus considered unsafe to consume, since messages with a simulation timestamp greater than 5 can still be received. The time advance ta_0 is also greater than $LBTS_0$ and thus unsafe, so S_0 must block until new messages arrive and the LBTS is increased.

III. Related Work

There are numerous related works presenting frameworks or evaluating case studies for AD in simulation. For the most part, the type of synchronization applied is only discussed marginally. In [11] high-severity real-world collision scenarios are reproduced in simulation and human-driven drivers are replaced with an ADF. The simulation runs are not deterministic and from several simulation runs the worst-case is selected. Riedmaier et al. propose a validation approach for simulation-based testing of ADFs [12]. Multiple simulation runs are performed for each scenario and it is shown that the simulation results do not differ significantly. In [13] experiences in using the popular Robot Operating System (ROS) framework in the context of AD are shared. The authors remark on problems concerning determinism, which they attribute to execution reordering, non-atomic message delivery and dropped data. These examples indicate that reproducible simulation runs cannot be assumed to be granted. As a solution, the means of choice is typically to perform multiple simulation runs. Finally, in [14] a relative time synchronization approach for SIL simulations is proposed. The approach is based on a global logical clock but only supports periodic execution units.

The combination of miscellaneous models of compuation (MoCs) within one simulation system is a prevalent research topic. Extensions to the function mock-up interface (FMI) standard are proposed that enable the support of continuous time (CT) and DE simulation [15]. This is achieved for instance with flags that indicate, whether an event is active or absent. In the survey of Gomes et al. [9] many other works are listed, which couple CT and DE simulation techniques. Two main approaches have been identified: Hybrid DE and Hybrid CT. A DE or CT orchestration is used respectively and all non-compliant simulation units are wrapped to behave in accordance with the orchestrator. A popular example of the Hybrid DE methodology is [16]. The work connects the

most widespeard CT and DE standsrds FMI and High Level Architecture (HLA) and the HLA run-time infrastructure (RTI) orchestrates the simulation system in a timestepped manner. In [17] statecharts, i.e. DE elements, are integrated into an FMI simulation. This requires state saving and restoration as multiple simulation step sizes are trialed. The research framework Ptolemy II [18] takes a completely different approach. It allows the combination of hybrid domains with continuous and discrete properties through the utilization of standardized execution models such as kahn process network (KPN) or synchronous dataflow (SDF). The approaches usually consider only the combination of DE and CT simulation and no software. Ptolemy's approach is promising, but restrictive because all elements must be designed according to a standard execution model, which is often not the case.

Last but not least, this work is inspired by the work of McLean et al. [5]. The authors discuss why real-time distributed simulations cannot be assumed to be deterministic. They identify non-determinisms sources lying in issues with message loss, message ordering and message timing. To solve this problem, code blocks are assigned a virtual latency. Subsequently, synchronization within the system is performed with reference to the simulation time. However, this work does not address different MoCs.

The contribution of this paper is a Hybrid DE approach that allows deterministic coupling of simulation components with software components. This is accomplished by the adoption of conservative synchronization and an algorithm for determining the time advance of data-triggered software components.

IV. Methodology

ADF software modules are stimulated by simulation signals and feed their output values back into the simulation. A simulation system is not a real-time system and through the distribution, causality violations such as (1) message timing, (2) message order, and (3) message loss faults may occur, making the overall system non-deterministic and thus non-reproducible. Conversely, the simulation must also wait for control values from the software. Synchronization is required. The proposed approach is to use conservative synchronization approaches from DE simulation. These allow parallel execution of individual modules and still guarantee determinism.

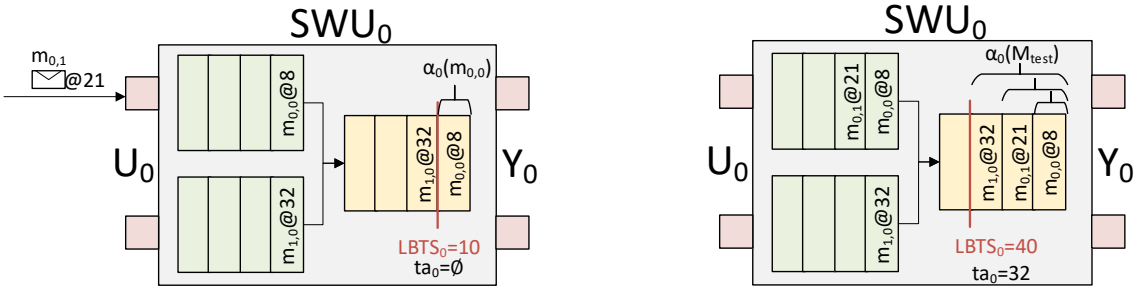
Software modules have a lot in common with DE simulation modules. Likewise, they possess input ports, output ports, and an internal state. What distinguishes them, however, is when they perform calculations and that they do not provide a time advance, which is essential for conservative synchronization. Software modules can be time- or data-triggered, i.e. they are activated after a time interval has elapsed or upon arrival of certain input data. Because time-triggered modules can be treated like timestepped simulation modules, the coupling of data-triggered models is a greater challenge. For time-triggered modules, computation takes place periodically and the time advance ta can therefore be set to the period. For data-triggered modules, no restrictions are made with regard to the activation. For example, activation could occur on each new input data sample, on multiple samples of input data, when the input data meets certain criteria, or on any combination of these criteria. Thus, the activation function $\alpha(m)$ is a function of the received input data samples $m_{i,j} \in \mathbb{M}$ and it returns a boolean value $\{false, true\} \in \mathbb{B}$, which indicates whether the computation has been activated:

$$\alpha : \mathbb{M} \rightarrow \mathbb{B}. \tag{2}$$

SUs and software units (SWUs) have in common that input and output ports can be defined for both of them. They also both have an internal state. What differentiates them, especially with regard to the use of conservative synchronization algorithms, is that SWUs are unable to provide a known time advance. This makes conservative time synchronization infeasible. In this paper, it is shown how to compute the time advance ta using a declared activation function α. This allows the use of conservative time synchronization for SWUs and thus the deterministic operation of these in a SIL simulation.

There are several challenges in combining the co-simulation and the distributed real-time software. The co-simulation defines its causal correctness in a simulation time reference and the software in a physical time reference. Co-simulation may not be able to stimulate the ADF software fast enough or with adequate timing because it cannot reliably keep up with real-time progress. Besides, the software itself is not executed on its actual target hardware, but on simulation machines, which affects its timing behavior. Thus, a coupling to a simulation time reference instead of a physical time reference seems more appropriate and flexible. Here, however, the problem opens up that the coupling of data-triggered software modules with existing simulation synchronization algorithms is not well-defined. The proposed approach exactly addresses this problem.

The evaluation of the activation function α is best done by examining the data samples in a given TSO queue. This is because the TSO queue is divided into safe and unsafe data samples by the LBTS. Consideration of the unsafe data samples can be excluded, as this would lead to non-determinisms. Thus, only the safe data samples from the TSO queue are considered. If all the safe data samples do not fulfill the activation function α, which corresponds to a false return value, the SWU must block and hold back the time advance. This in turn also blocks all subsequent SWUs. When checking the safe data samples, the smallest possible data sample set that satisfies the activation function must be found. Thus, the activation function is first applied to the oldest safe data sample and the considered set is increased by one data sample at a time until the SWU is activated, or until the TSO queue no longer contains any safe data samples. If the SWU is activated by the activation function, the computation of the SWU is triggered and a time advance is announced. The time advance corresponds to the timestamp of the most recent data sample (maximum simulation timestamp) from the minimum set of data samples that have activated the SWU.



(a) blocked (b) unblocked

Fig. 3: Sample operating of the time advance determination algorithm.

Algorithm 1 $DetermineTimeAdvance(lbts, \mathbb{M}_{tso})$

1: $ta \leftarrow \emptyset$
2: $\mathbb{M}_{test} \leftarrow \emptyset$
3: $\mathbb{M}_{safe} \leftarrow \{(m_i, t_i) \in \mathbb{M}_{tso} | t_i < lbts\}$
4: $n \leftarrow |\mathbb{M}_{safe}|$
5: **for** $(i \leftarrow 0; i < n; i \leftarrow i + 1)$ **do**
6: $\quad \mathbb{M}_{test} \leftarrow (\mathbb{M}_{test}, dequeue(\mathbb{M}_{safe}))$
7: $\quad$ **if** $(\alpha(\mathbb{M}_{test}))$ **then**
8: $\quad\quad ta \leftarrow \max_{(m_i,t_i)\in\mathbb{M}_{test}} t_i$
9: $\quad\quad break$
10: $\quad$ **end if**
11: **end for**
12: **return** ta

The concept for determining the time advance is presented in more detail in Algorithm 1. The return value of the time advance ta is predefined with an empty set. If the conditions of the activation function are not met, no time advance can be returned and the SWU blocks. The data samples of interest $\mathbb{M}_{safe}$ are solely the safe samples from the TSO queue. Considering the unsafe data samples as well would trigger the activation more quickly, but it would not be deterministic or reproducible. Testing for activation occurs at most as many times n as there are safe samples in the TSO queue. The test samples $\mathbb{M}_{test}$ for the activation function α start with the oldest data sample from the safe sample set $\mathbb{M}_{safe}$. If no activation takes place, one safe sample after the other is gradually added to the test sample set $\mathbb{M}_{test}$ and activation is checked each time. If the activation function α is triggered, the time advance ta is set to the simulation time of the most recent data sample from the test sample set $\mathbb{M}_{test}$ and the time advance is returned. This leads on the one hand to the activation of the computation of the SWU and on the other hand to the unblocking of subsequently interconnected SUs or SWUs and thus to a simulation time progress in the overall SIL simulation system.

An example of the operation of the algorithm is given in Fig. 3. The activation function for the SWU_0 component is $\alpha_0 = u_0 \wedge u_1$. This means that the SWU_0 is triggered when one data sample from each of the two input ports is present. In Fig. 3a, SWU_0 is in a state where it has received a data sample on each of its two input ports. There is in addition a transient message for the first input port that has not yet been received. The $LBTS$ is 10 and is thus located between the two received messages. Thus, only the message $m_{0,0}$ belongs to the safe samples and is therefore checked with the activation function. The activation function does not trigger and SWU_0 blocks.

In Fig. 3b there are two changes compared to the previous state: (1) the transient message has been received and (2) the $LBTS$ has been increased to 40. Due to the increased $LBTS$, all 3 data samples now classify as safe samples. The activation function is called three times with $\{m_{0,0}\}$, $\{m_{0,0}, m_{0,1}\}$ and $\{m_{0,0}, m_{0,1}, m_{1,0}\}$. The activation is successful on the third call, since there is now one sample from each of the two input ports in the test set. The time advance ta is set to the value 32, since this corresponds to the value of the most recent data sample $m_{1,0}$ of the successful test sample set. The data from the successful test sample set is delivered and the computation of SWU_0 is scheduled.

V. Implementation

We have implemented the concepts described above on top of the framework Functional Engineering Platform (FEP). FEP is a scalable execution environment for distributed real-time and simulation systems and aims to cover all X-in-the-Loop (XIL) use cases [19]. The concepts were implemented on top of the FEP version 2.3.0. Since version 3.0.0b[1] FEP is open source and available to the public. FEP utilizes Data Distribution Service (DDS) at its communication layer. DDS [20] is a data-centric publish-subscribe (DCPS) communication middleware known from many applications with soft real-time requirements. Efforts are being made to use DDS for applications with hard real-time requirements. The architecture of DDS consists of domain participants, which can exchange data via topics. For this purpose, domain participants have publishers to offer data and subscribers to manage data subscriptions. A data writer is created in the publisher for each offered data set and a data reader is created in the subscriber for each subscribed data set. We represent SUs and SWUs as domain participants with one publisher and one subscriber.

[1]FEP3 SDK: https://github.com/cariad-tech/fep3_sdk, accessed 01-04-2022

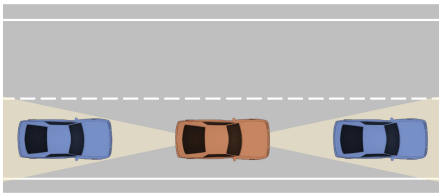


Fig. 4: The examined car-following scenario.

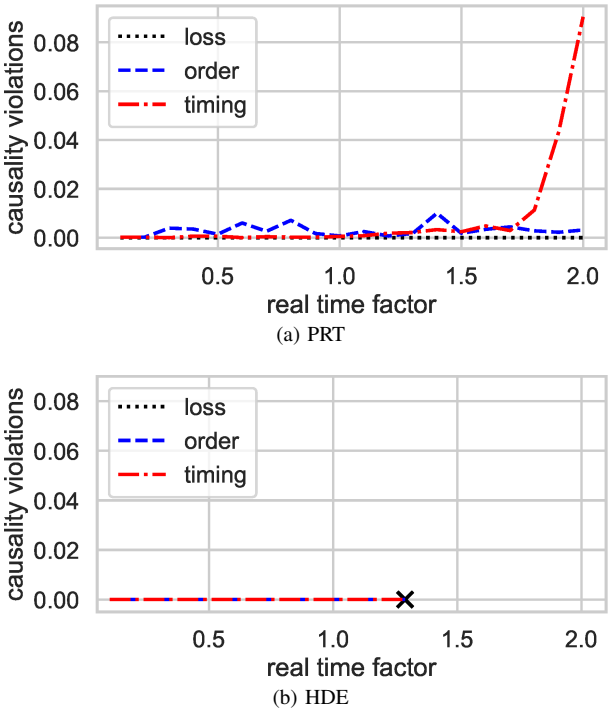
The subscriptions for the input ports are managed by the subscriber. The reception of data samples at specific input ports is performed by a data reader each. The same applies to the output ports. The publisher manages the publications and the sending at specific output ports is performed by a data writer each. In [21], we have presented a conservative synchronization algorithm, that allowed distributed simulation with time- and event-triggered simulation models on top of this framework. In this paper, the extension to support data-triggered software modules is presented. The data-triggered SWUs are a derivative of event-driven SUs that contain an activation function and set their time advance based on the presented algorithm. The following case study is conducted on this foundation.

VI. Case Study

The presented methodology is evaluated by means of a case study. A car-following scenario is analyzed, the simulation execution of which is performed with two different synchronization mechanisms: (1) the proposed Hybrid DE approach HDE and (2) a paced real-time approach PRT. In the PRT approach the simulation time progress correlates with the real-time progress. A real-time factor can speed up or slow down the simulation speed linearly. In the proposed HDE approach there is no relationship between simulation time and real time. The simulation runs as fast as causality violations can be excluded.

The car-following scenario is depicted in Fig. 4. The automated ego vehicle is equipped with two sensors to detect surrounding vehicles and environment. The front and rear vehicles alternately accelerate and decelerate, shifting the target gap for the ego vehicle. This scenario is particularly well suited for testing synchronization approaches because it is transparent and still meets all the characteristics of a SIL simulation: (1) divergent data flows from the traffic simulator to the sensor models, (2) merging data flows from the sensor models to the ADF and (3) feedback loop from the ADF back to the traffic simulator. The simulation step of the traffic simulator is set to $10ms$ and the sensor models have a period of $40ms$. The ADF is regarded as one software unit and operates data-triggered on the arrival of each new sensor value.

For both PRT and the proposed HDE approach, the number of occured causality violations within a simulation run is measured. For the PRT approach, the real-time factor is varied between 0.1 and 2.0 in steps of 0.1 and the measurement is taken in each case. This corresponds to 20 simulation runs. The HDE is conducted in a as-fast-as-possible manner.



(a) PRT

(b) HDE

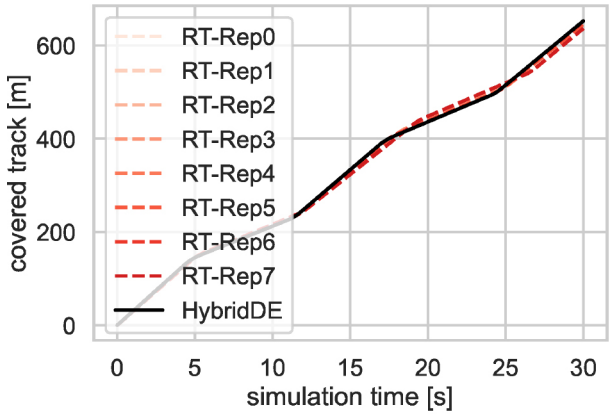
Fig. 5: The occurrence of causality violations within the SIL.

The scenario is stopped after a simulation time of $100s$. The results are presented in Fig. 5. No causality violations due to message loss occur in either case because the underlying DDS protocol is configured to be reliable.

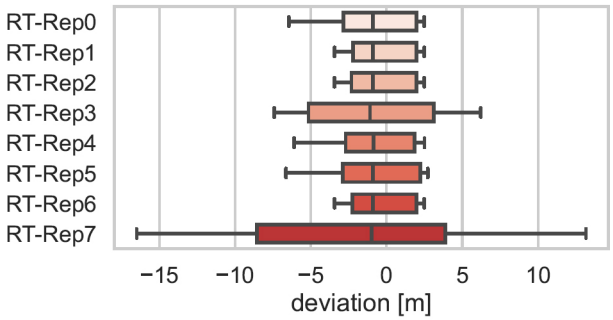
For PRT, no causality violations occur only in the case of a real-time factor of 0.1. For this purpose, time synchronization and scheduling with scaled real-time is compared with the proposed approach. An exponential relationship can be observed between increasing the real-time factor and thus the simulation speed and causality violations due to incorrect timing of messages. This makes sense in so far as the computational share of SUs and SWUs decreases proportionally with an increased simulation speed compared to the idle share. In the idle share, however, the messages are transmitted. Since this interval is smaller, it is more likely that messages arrive late. With a real-time factor of 2.0, the proportion of messages with incorrect timing is already close to 10%. No correlation can be observed between the simulation speed and changes in the order of messages. The message order can change regardless of the message frequency and proportion of the idle share.

For HDE, a simulation speed with a real-time factor of 1.29 is established. All types of causality violations are successfully prevented. Any real-time factor smaller than the maximum value of 1.29 can be achieved by combining HDE with PRT and additional waiting for the physical clock. A simulation speed greater than a real-time factor of 1.29 is impossible to obtain with HDE.

Now we qualitatively examine 8 runs of the PRT approach with one run of the HDE approach. With the HDE approach, only one simulation run is necessary because it always pro-



(a) absolute



(b) deviation

Fig. 6: Covered track of the ego vehicle.

duces the same results. In Fig. 6 the driven distance of the ego vehicle is examined. The repetitions of the PRT runs differ from the HDE run. Moreover they also differ among one another. In Fig. 6b the HDE run is taken as the baseline and the deviation of the PRT repetitions to the baseline is formed. The deviations between the repetitions have different magnitudes in each case. With increasing simulation time, we could not detect any divergence behavior. However, this is also related to the investigated scenario. In a car-following scenario the leading vehicle sets the pace for the ego vehicle. For general scenarios, divergence cannot be precluded.

VII. Conclusion

In a SIL with distributed simulation and software modules, causality violations cannot be excluded for a paced real-time execution. Known synchronization algorithms from simulation can prevent violations, but are not defined for software modules. The presented HDE approach closes this gap by deriving a time advance with the help of a TSO buffer. In a case study it is shown that this eliminates all causality violations and thus determinism can be guaranteed. Future work will investigate what proportion of causality violations also translate into actual errors. It will also be investigated whether a timestepped synchronization approach can also lead to success and how this relates to simulation performance.

Acknowledgment

This research is supported by AUDI AG.

References

[1] P. Koopman and M. Wagner, "Challenges in Autonomous Vehicle Testing and Validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, p. 15–24, Apr 2016.

[2] A. Höfer and M. Herrmann, *Scenario-based approach for developing ADAS and automated driving functions*. Springer Fachmedien Wiesbaden, 2017, p. 215–225.

[3] S. Jafer, Q. Liu, and G. Wainer, "Synchronization methods in parallel and distributed discrete-event simulation," *Simulation Modelling Practice and Theory*, vol. 30, p. 54–73, Jan 2013.

[4] "IEC/IEEE International Standard - Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEC 61588:2009(E)*, pp. 1–292, 2009.

[5] T. McLean and R. Fujimoto, "Repeatability in Real-Time Distributed Simulation Executions," in *Proceedings Fourteenth Workshop on Parallel and Distributed Simulation*. IEEE Comput. Soc, 2000, p. 23–32.

[6] G. Schweiger, G. Engel, J. Schoeggl, I. Hafner, C. Gomes, and T. Nouidui, "Co-Simulation – an Empirical Survey: Applications, Recent Developments and Future Challenges," in *MATHMOD 2018 Extended Abstract Volume*. ARGESIM Publisher Vienna, 2018, p. 125–126.

[7] J. Hertzberg, K. Lingemann, and A. Nüchter, "Roboterkontrollarchitekturen," in *Mobile Roboter*. Springer, 2012, pp. 317–333.

[8] Y. Van Tendeloo and H. Vangheluwe, "An Introduction to Classic DEVS," *ArXiv*, May 2018.

[9] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, "Co-Simulation: A Survey," *ACM Comput. Surv.*, May 2018.

[10] R. M. Fujimoto, R. Bagrodia, R. E. Bryant, K. M. Chandy, D. Jefferson, J. Misra, D. Nicol, and B. Unger, "Parallel discrete event simulation: The making of a field," in *2017 Winter Simulation Conference (WSC)*. IEEE, Dec 2017, p. 262–291.

[11] J. M. Scanlon, K. D. Kusano, T. Daniel, C. Alderson, A. Ogle, and T. Victor, "Waymo Simulated Driving Behavior in Reconstructed Fatal Crashes within an Autonomous Vehicle Operating Domain," 2021.

[12] S. Riedmaier, D. Schneider, D. Watzenig, F. Diermeyer, and B. Schick, "Model Validation and Scenario Selection for Virtual-Based Homologation of Automated Vehicles," *Applied Sciences*, vol. 11, no. 1, p. 35, Dec 2020.

[13] N. Valigi, *Lessons Learned Building a Self Driving Car on ROS*, ser. Studies in Computational Intelligence. Springer International Publishing, 2021, vol. 895, p. 127–155.

[14] S. Lee, B. I. Hwang, K.-B. Seo, and W. J. Lee, "Relative Time Synchronization of Distributed Applications for Software-in-the-Loop Simulation," in *2016 IEEE Intl Conference on Computational Science and Engineering (CSE)*. IEEE, Aug 2016, p. 753–756.

[15] F. Cremona, M. Lohstroh, D. Broman, E. A. Lee, M. Masin, and S. Tripakis, "Hybrid co-simulation: it's about time," *Software & Systems Modeling*, vol. 18, no. 3, p. 1655–1679, Jun 2019.

[16] M. U. Awais, P. Palensky, A. Elsheikh, E. Widl, and S. Matthias, "The High Level Architecture RTI as a master to the Functional Mock-up Interface components," in *2013 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, Jan 2013, p. 315–320.

[17] J. Denil, B. Meyers, P. De Meulenaere, and H. Vangheluwe, "Explicit Semantic Adaptation of Hybrid Formalisms for FMI Co-Simulation," in *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, ser. DEVS '15. San Diego, CA, USA: Society for Computer Simulation International, 2015, p. 99–106.

[18] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng, "Heterogeneous Concurrent Modeling and Design in Java," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2008-28, Apr 2008.

[19] C. Stadler and T. Gruber, *Functional Engineering Platform — A Continuous Approach Towards Function Development*. Springer International Publishing, 2016, p. 69–84.

[20] G. Pardo-Castellote, "OMG Data-Distribution Service: Architectural Overview," in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.* IEEE, 2003, p. 200–206.

[21] W. Baron, C. Sippl, K. Hielscher, and R. German, "Repeatable Simulation for Highly Automated Driving Development and Testing," in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, 2020.