IEEE Copyright Notice

DOI: 10.1109/IESES.2018.8349934

Publisher version: https://ieeexplore.ieee.org/document/8349934

Distributed Real-Time Co-Simulation as a Service

Markus Mirz, Steffen Vogel, Bettina Schäfer, Antonello Monti
Institute for Automation of Complex Power Systems
E.ON Energy Research Center
RWTH Aachen University
Aachen, Germany
Email: {mmirz, stvogel, bschaefer, amonti}eonerc.rwth-aachen.de

***Abstract*—The increase of faster dynamics in power systems has led to growing interest in new simulation solutions, especially in the field of hardware-in-the-loop and real-time simulation. Due to the size of power systems, detailed simulation of the faster dynamics is only feasible for a section of the system, whereas the rest is usually modeled as an infinite power bus. The aim of this work is to present a solution which would allow the representation of a significant portion of the dynamics that are usually not captured by the infinite power bus approach and enable the joint simulation among multiple simulation laboratories to share this dynamic model of the network beyond the boundaries of the detailed simulation. Furthermore, the presented architecture should allow the virtualization of each of these laboratories in cases where this coarse model of the neighboring grid sections is sufficient. First distributed simulation examples show the current status of our implementation of the architecture presented here.**

I. Introduction

The current evolution of the grid characterized by a higher penetration of renewable energy sources is significantly changing the main dynamics of the power system. This change is reflected in two main aspects:

- Higher volatility of the generation input
- Faster dynamics due to reduction of the mechanical inertia connected to the system

This process has attracted a growing interest in the development of new simulation solutions and in particular a significant development of the concepts of real-time simulation and hardware-in-the-loop (HIL). These technologies are widely used both in the analysis and in the development of advanced automation solutions intended to overcome the new challenges that power system dynamics are facing [1]. The lack of knowledge about the behavior of a power grid widely based on power electronics devices is asking for new simulation analysis able to give insight at different levels of details: from the slow dynamics of interaction of the large system scenarios to the complex non-linear interactions of switching devices at local level. On the other hand, it is becoming more and more clear that the computational complexity is such that the problem can not be solved by a single laboratory. Furthermore, availability of data on electrical infrastructure is also sometimes not easy to share among scientists. This scenario is calling for the development of a new architecture of real-time simulation that could support integration of knowledge among different players in the creation of large and meaningful scenarios [2]. The work proposed in this paper fits exactly in this direction of research, providing a platform which facilitates the joint research on large scale power systems.

II. The Vision

Main goal of the work described in this paper is the creation of a network of laboratories sharing knowledge and creating large scenarios of interaction to tackle the appropriate level of complexity. The idea is that to achieve such goals two components are necessary:

- A programmable system to support data interchange among laboratories set-up in real-time
- A simulation platform that is able to model only the dynamics that are compatible with the bandwidth of the interface.

Experimental results show that, with the current internet infrastructure, it is feasible to reach round-trip times within some milliseconds within distances in the order of hundred kilometers [3]. Such performance is in line with the main dynamics present at transmission level typically in the order of few hertz. Coherently, it would be interesting to have available a real-time simulation with a resolution in the range of milliseconds. It should be clarified that while operating at this time step does not allow the representation of every possible dynamic interaction among different sections of a network, on the other hand it provides a much more significant representation than what is possible within a single lab. In effect, currently, given the computational limits of whatever simulation platform, each laboratory needs to set a boundary condition to the analysis which is typically described in terms of infinite power bus. The architecture and the combination of tools proposed in the paper allows the substitution of such ideal hypothesis with an interaction model able to capture a significant portion of the dynamic interaction.

One step further is the virtualization of the laboratory in the cloud. Given that it is not reasonable that all the laboratories will be permanently connected, it is important to define a process that supports the substitution of the laboratory with its virtual representation. Each institution could run its own real-time simulation either in the cloud or locally in their facilities depending on their current activity while still contributing to the overall network. This approach has two benefits:

- Other participants always have some kind of power system representation available which they can connect to
- Each participant can move his simulation from the cloud to his local simulation environment to connect hardware at any time without coordination of other participants.

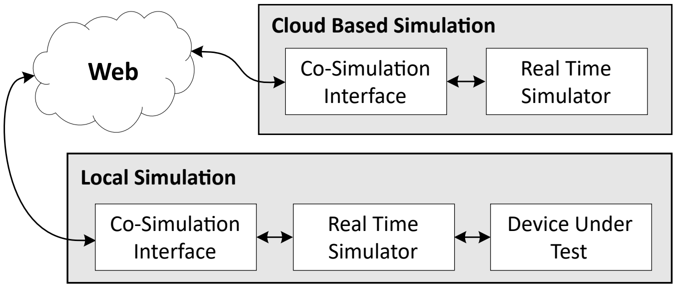


Fig. 1. Co-simulation as a service concept.

Today's availability of cloud computing resources and the free software, which was employed for this study, can form the basis for an always running real-time simulation collaboratively operated by the research community. Open protocols and the free software presented in this paper are key ingredients for this idea.

Other research disciplines already collaboratively operate such test beds. Examples are the Decentralized Network 42 (DN42), which builds a virtual overlay network on top of the real Internet [4]. Operated by individuals and researchers, this network is used for training purposes and experimentation. Annual *Battle the Mesh* events are hosted by a community built around the development of open source mesh network stacks [5].

III. Key Components of the Architecture

A. Geographically Distributed Real-Time Simulation

Geographically distributed real-time simulation as special case of distributed simulation has the advantage of being able to share computational load among several processing units. However, it is required to synchronize the simulators involved in the simulation. While distributed simulation does not specify the type of network connection between the simulators, geographically distributed simulation excludes high speed network interconnection, such as InfiniBand, since the connection can only be established through the Internet. Section III-B will further elaborate the challenges of this approach.

Apart from increasing the overall computational power of the simulator, there are more advantages of distributed real-time simulation [6], [7]. The following list summarizes the most important ones:

- Hardware and software in different simulation sites can be shared to facilitate remote software-in-the-loop (SIL) and (power-)hardware-in-the-loop (PHIL)
- Knowledge exchange is facilitated and encouraged since several research groups may work on the same case study without the need to move personnel and equipment
- Confidential data does not need to be shared as each laboratory can be responsible for simulating its own part of the model locally, solely exchanging interface variables with other interconnected systems, imitating the real world where regional or national power grids are interconnected through tie-lines
- Several algorithms to control, manage, or regulate systems can be tested in laboratories where no realistic models of the environment (e.g. power grid model) are available

B. Modelling of the Appropriate Resolution

As explained in Section III-A, the simulators are connected through the Internet. This results in a typical round-trip time (RTT) between two simulators in the range of milliseconds. If electromagnetic transient (EMT) values are exchanged among the simulators, a simulation time step of $10\,\mathrm{ms}$ ($50\,\mu\mathrm{Hz}$ AC) or smaller is required according to the sampling theorem. Besides, the typical simulation time step of EMT simulators is $50\,\mu\mathrm{s}$. Therefore, the relation between simulation time step and data exchange time step is very large and a lot of samples have to be sent at once.

The solution proposed [7] overcomes the problem of data size by compressing the information through an EMT-Phasor-Interface. This requires extraction of the phasor information and may alter simulation data. Such a data resolution calls for an appropriate simulation solver able to operate on the same time-scale and working directly in the phasor domain.

An alternative solution to data compression could be the simulation with static phasors. However, this approach would only cover simulations at fixed system frequency.

IV. VILLASnode Gateway

Communication between real-time solver instances is realized by a dedicated gateway called VILLASnode. VILLASnode is part of the VILLASframework, a toolkit for distributed real-time simulation which is released as open source software under the GLPv3 license [8]. The gateway is a software component which runs on a real-time optimized Linux machine which also executes the DPsim solver (see section V).

Responsibilities between the solver and the gateway are clearly separated. This modular approach allows to use the components in combination as well as with other software. The gateway supports a variety of pluggable transports like UDP/IP, ZeroMQ, nanomsg, IEC61850 Sampled Values & GOOSE as well as interfaces to commercial digital real-time simulators, like OPAL-RT and RTDS.

In this paper, the gateway provides two interfaces for exchanging simulation data between the solver and over the Internet as depicted in Figure 2

A. Shared-memory Interface

A shared memory interface is used to exchange simulation data between one or more processes on a single compute node. In this paper, a shared memory region is used to exchange

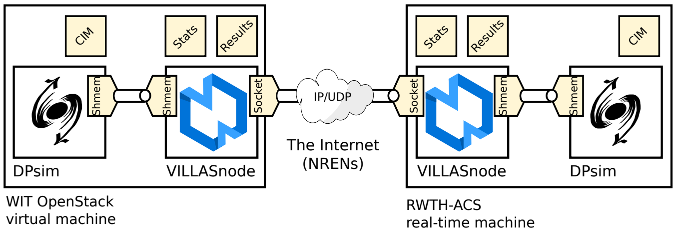


Fig. 2. Co-simulation architecture.

simulation data between the real-time phasor solver DPsim and the gateway.

The interface consists of two simple lock-free single-producer single-consumer (SPSC) queues for bidirectional communication between the processes. Shared-memory has been chosen over other inter-process communcation methods as it requires operating system (OS) support only for the initial setup. During a running simulation the OS remains in the background and does not disturb the execution of the solver. This is a key requirement for executing the solver in real time.

The shared memory interface has also been used for the first simulation case described in section VIII-B.

B. Network Interface

The communication between multiple gateways is realized by UDP/IP packets which are exchanged over the public Internet and the national research and education networks (NREN). Additional meta data such as time stamps and sequence numbers which is being exchanged is used to monitor the current quality of the connection and drop invalid or reordered packets.

By using Linux's NetEm queueing discipline, real network communication characteristics such as latency, packet loss and reordering can be emulated. This feature is utilized to anticipate the results of the distributed co-simulation. As part of the traffic control (TC) subsystem, NetEm is a special queuing discipline which is tightly integrated into Linux networking stack [9], [10] and allows good emulation of real communication characteristics in real-time.

V. DPsim - Dynamic Phasor Real-Time Simulator

As described in Section III-B, the exchange of time-domain or EMT values among simulators imposes strong requirements on the sampling rate. Therefore, previous work already introduced dynamic phasors as a means of exchanging data in the frequency domain rather than the time domain [3]. The difficulty lies in the extraction of the dynamic phasors from the time domain signal for every simulation step. Here, the DPsim simulator offers an alternative. Instead of simulating the entire system in EMT, the simulation operates on dynamic phasors. This method avoids the conversion from the time domain to the frequency domain and improves the possible sampling rate with regard to Shannon's sampling theorem. Simulation results which demonstrate this advantage of dynamic phasors compared to EMT are presented in [11].

The dynamic phasor approach is combined with modified nodal analysis and resistive companion method. Therefore, the majority of network components is represented by the network admittance matrix whereas more complicated models such as synchronous generators are solved separately and interfaced through, for example, a Norton equivalent model [12].

VI. Scheduling and Synchronization

A completely synchronized execution of both solvers over the Internet is not possible for two reasons:

- First, the communication latency of about 15 ms between the two sites exceeds the simulation time step of 1 ms.
- Secondly, the unreliable nature and the inherent jitter makes it impossible to provide any guarantees.

Therefore, all solvers are started synchronously in reference to a global point in time which was agreed upon before. During the execution of the co-simulation, every solver proceeds in real-time with its own time step without synchronization to its peers. The interface quantities described in section VIII are exchanged at periodical intervals which can, but do not have to, be equal to the simulation time step. This scheme allows multiple rates to be used for participating simulators as well as the interface itself.

To guarantee a synchronized start of the simulation as well as to avoid them drifting apart, all simulators must be synchronized to a global clock. This synchronization source is most conveniently provided by the global positioning system (GPS) and distributed by existing time synchronization protocols such as the network time protocol (NTP) or the precision time protocol (PTP/IEEE-1588).

VII. Network measurements

Figure 3 shows the geographical location of routing hops used in the pan-European laboratory infrastructure which has been used to conduct the following simulations. The black line

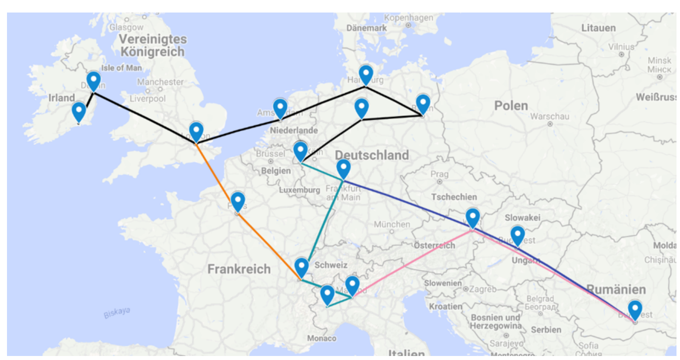


Fig. 3. Network connection of Pan-European Laboratory infrastructure.

in this map shows the communication link between Waterford Institute of Technology (WIT) in Ireland and RWTH Aachen University (RWTH) in Germany. In total, the connection is comprised of 15 routers/hops. Figure 4 shows the distribution of round-trip time as well as the geographical distance across those hops. The average RTT measured between RWTH and

WIT is $47.7\,\mathrm{ms}$ and was between $50\,\mathrm{ms}$ and $60\,\mathrm{ms}$ during the execution of the test simulations described in Section VIII.

These results show that the geographical distance between the sites is the main contributor of the RTT. The RTT is

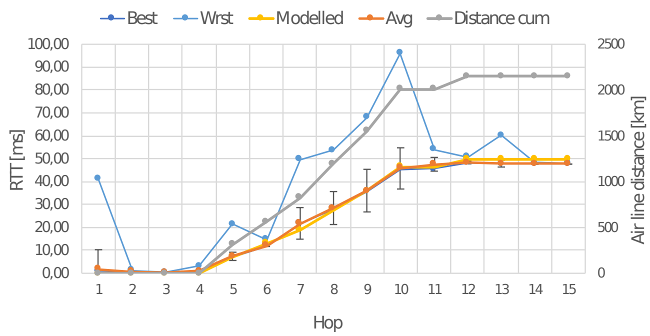


Fig. 4. RTT over communication hops.

only one, but important, metric for evaluating the quality of a connection for real-time simulation. Another metric of interest is the maximum packet rate, which we can use to exchange simulation samples while keeping the packet loss below a certain threshold. For this test, VILLASnode is used to generate sample streams at different packet rates and number of values.

Figure 5 shows the cumulative distribution of the RTT between RWTH and WIT in dependency to the sending rate. Not shown in this figure is the loss and reordering of packets, which gets significant with higher rates. At higher rates, the connection is susceptible to third party network traffic which causes loss and reordering.

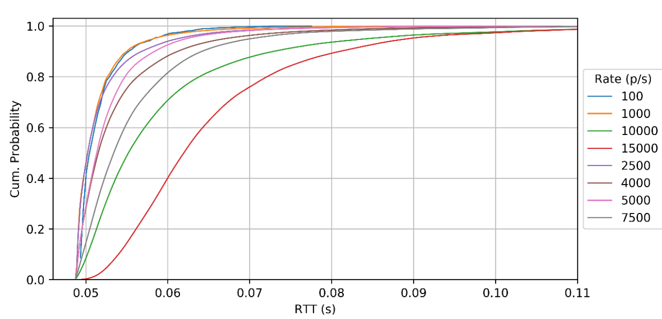


Fig. 5. Cumulative probability of RTT.

VIII. Co-simulation Example

The RTT delay measurements described before, allow an estimation of the dynamics that are able to propagate through the laboratory interface. However, real-time simulation test cases provide us with more details on the expected dynamic interaction between the distributed simulation laboratories since also the simulators themselves introduce latency.

A. Circuit Model

To validate the estimation of the delay and demonstrate the functionality of the interface between VILLASnode and DPsim, we present the simulation results for a simple circuit as depicted in Figure 6. The circuit consists of an AC voltage

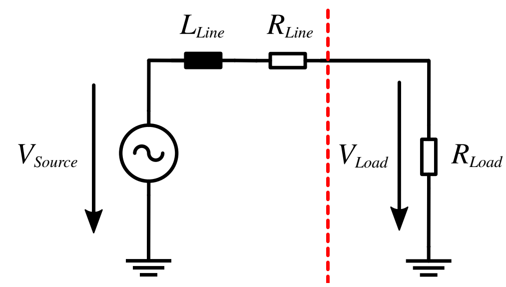


Fig. 6. Circuit model for distributed simulation.

source of $V_{Source} = 10\,\mathrm{kV}$ peak voltage with a resistance of $R_{Source} = 1\,\Omega$, an RX-series element of $R_{Line} = 1\,\Omega$ and $L_{Line} = 1\,\mathrm{H}$ and a load resistance of $R_{Source} = 10\,\Omega$. Internally, the voltage source is transformed to its Norton equivalent.

For a first test simulation, the circuit depicted in Figure 6 is split into two parts, indicated by the red line, and each part is simulated in a different location. The two locations that we consider are our institute, RWTH, and WIT. The source and line model are simulated in RWTH on a dedicated Linux workstation while the load resistance is simulated in the WIT OpenStack installation.

The depicted circuit is simulated for four different cases. First, both parts are simulated within one instance of DPsim. This case serves as reference for the three co-simulation cases. Secondly, the circuit is simulated using two separate instances of DPsim running on one computer that are interconnected via VILLASnode in the most optimal way. Then, the co-simulation is executed including a network latency emulation between two DPsim instances running on the same machine. Finally, one DPsim instance is running on a machine in RWTH while the other DPsim instance is executed in WIT.

The simulation time step is $1\,\mathrm{ms}$ in all cases. In comparison to EMT based simulation, this time step is relatively large and allows us to exchange interface quantities without down sampling. EMT simulations with $50\,\mathrm{\mu s}$ time step would require a decimation to reduce the amount of data. At $t = 1\,\mathrm{s}$, the load resistance is changed from $10\,\Omega$ to $8\,\Omega$. This change is an ideal step, which would not appear like this in a real setup. Still, it is an interesting test case because the step introduces very high frequencies and, therefore, presents the worst-case scenario for the distributed co-simulation and dynamic phasor approach since only the fundamental phasor is used.

The ideal transformer model (ITM) [13] which is shown in Figure 7 interconnects the two network solutions where t_D describes the latency introduced by the interface. The ITM is introduced as follows: The external voltage source is used in the left part of the circuit whereas the external current source is integrated into the right side. The voltage and current values which are exchanged between the two simulations are the complex voltage and current values split into real and imaginary part.

It should be noted that the bandwidth of the signal passing through the interface can be increased by considering also the phasors of higher harmonics. Therefore, fast electromagnetic phenomena in one section of the network could be seen in the remote section of the network. Still, the bandwidth of a control loop which is across the co-simulation interface is restricted by the delay t_D of the interface.

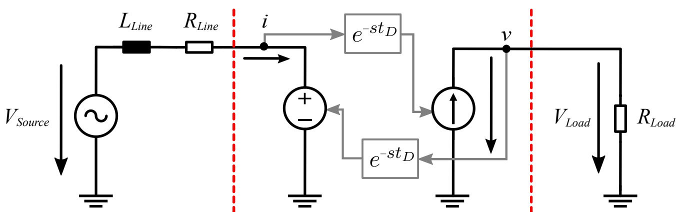


Fig. 7. Separated circuit with ITM and communication delay.

B. Local Co-Simulation Separated

For the co-simulation case where both DPsim instances are coupled in an optimal way, the communication delay is rather small but already visible. The delay between the monolithic reference simulation and the co-simulation is about $3\,\mathrm{ms}$ as can be seen in Figure 8. Since there is no communication network in between the two DPsim instances, this delay is only introduced by the software interface. DPsim is interfaced to VILLASnode through a shared memory segment and the two VILLASnodes instances, one for each DPsim instance, are exchanging data through a UDP connection.

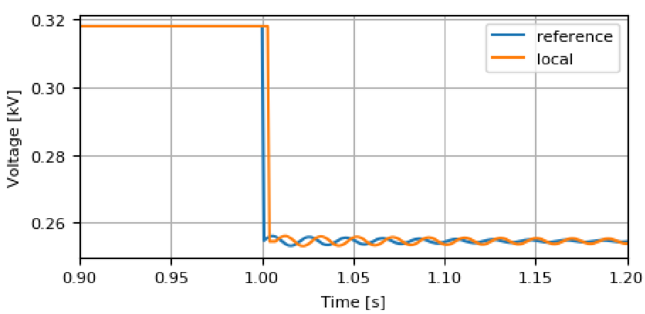


Fig. 8. Comparison of integrated reference simulation and local co-simulation.

C. Local Co-Simulation with Emulated Network Latency

In this case, the communication delay is emulated using the NetEm feature of VILLASnode. The properties of the artificial delay for sending data are set to $30\,\mathrm{ms}\pm1\,\mathrm{ms}$, normal distributed for each VILLASnode. This results in an emulated RTT of about $60\,\mathrm{ms}$ which is close to the measured RTT in Section IV-B. Therefore, the delay in simulation should be comparable to the distributed case. As can be observed in Figure 9, the resulting delay in the simulation is about $32\,\mathrm{ms}$.

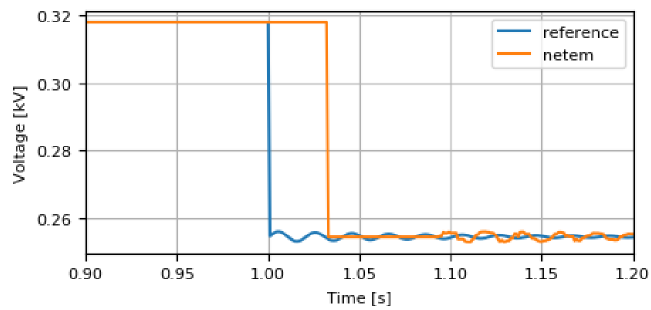


Fig. 9. Comparison of integrated reference simulation and local co-simulation with emulated network latency.

D. Distributed

As depicted in Figure 10, the distributed simulation between RWTH and WIT features a delay of about $33\,\mathrm{ms}$. Hence, the measurement of the delay as in Section IV-B and the simulation including emulated delay as presented in Section VIII-C allow a very good prediction of the behavior of the distributed simulation.

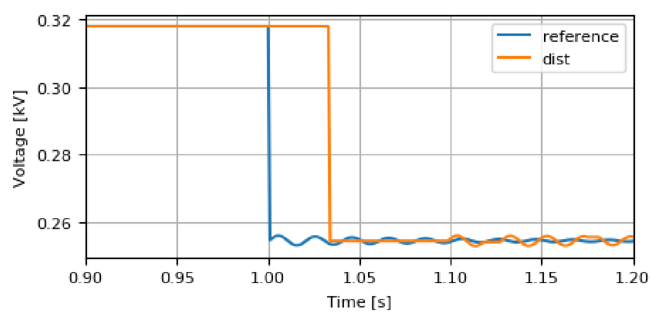


Fig. 10. Comparison of integrated reference simulation and distributed co-simulation among RWTH and WIT.

IX. Simulation of Slow Dynamics

The previous simulations are intended to show how the communication latency affects the distributed simulation in the extreme case. The main purpose of the proposed simulation solution is the investigation of slower dynamics. Slow dynamics can be represented by dynamic phasors very accurately as shown in Figure 11. Here, the voltage source is changing its frequency by $-1\,\mathrm{Hz}$ over $200\,\mathrm{ms}$. Figure 11 shows the EMT reference simulation which is simulated with a simulation time step of $50\,\mu\mathrm{s}$ and the absolute value as well as the shifted version of the dynamic phasor values are simulated with a time step of $1\,\mathrm{ms}$. The shifted version is the result of the complex phasor signal shifted by $50\,\mathrm{Hz}$ in the frequency domain. It can be seen that the dynamic phasor values are following the transient very well even though the simulation time step is much larger. In this case, the simulation is integrated and not interfaced through VILLASnode. This is an example of the dynamics that we want to be able to propagate between distributed simulators in the future.

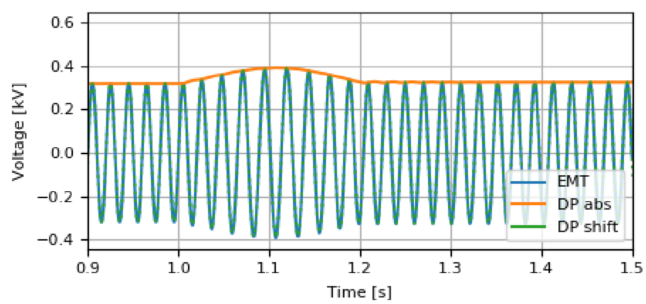


Fig. 11. Frequency change simulated for dynamic phasors and EMT.

X. Conclusion

Initial test cases have shown that DPsim and the VILLASnode gateway can be used to conduct simple geographically distributed simulations in real time over the Internet. Besides, it can be seen that the latency caused by VILLASnode is very small compared to the latency expected from the network connection with the co-simulation partner WIT.

It is presented that the communication network measurements and the simulation test cases are coherent. The comparison with the network connectivity measurement in Section IV-B shows that predictions of the delay in the simulation can be very accurate based on the network measurements.

Furthermore, the simulation highlights an important condition for the validity of co-simulation results. The dynamics of variables which are propagated between the simulators should not be faster than the communication delay or otherwise the results will differ from the integrated simulation results. This is exactly why dynamic phasors can be an important tool to support distributed real-time simulation. They allow the mapping of high frequencies to lower frequencies which decreases the impact of the communication delay. Hence, the dynamic behavior of the grid model beyond the detailed simulation in one laboratory can be improved as stated in Section I.

XI. Outlook

Upcoming tests must validate these results using larger networks including multiple sources and detailed models, such as synchronous generators.

The transition of frequency changes as shown in [11] should be investigated. If fast events are of interest, the simulation could be extended to include dynamic phasors of higher harmonics as well. However, for system frequency control investigations, as conducted in the Horizon 2020 project RESERVE the fundamental phasor is the most important one [14]. In the frame of RESERVE, the presented components are developed as part of a pan-European real-time simulation infrastructure for the validation of innovative approaches to system level automation based on innovative ancillary service provision.

In the simulation case described here, no data processing is applied to the exchanged values. In the future, current simulation values at the interface could be extrapolated from previous values. This could further improve the correct propagation of fast dynamics through the co-simulation interface.

The achievable data exchange rates discussed in section IV-B are dependent on the current conditions of the communication medium. Future work could investigate adaptive methods to determine the optimal sending rate and its effects on the simulation fidelity. The real-time protocol (RTP) commonly used for streaming of audio / video content is a good candidate and can be used as the basis to develop a new variant of that protocol for exchanging simulation data.

XII. Acknowledgment

This project has received funding from the European Unions Horizon 2020 Framework Programme for Research and Innovation under grant agreement no 727481.

References

[1] M. D. O. Faruque, T. Strasser, G. Lauss, V. Jalili-Marandi, P. Forsyth, C. Dufour, V. Dinavahi, A. Monti, P. Kotsampopoulos, J. A. Martinez, K. Strunz, M. Saeedifard, X. Wang, D. Shearer, and M. Paolone, "Real-time simulation technologies for power systems design, testing, and analysis," *IEEE Power and Energy Technology Systems Journal*, vol. 2, no. 2, pp. 63–73, June 2015.

[2] C. F. Covrig, G. D. Santi, G. Fulli, M. Masera, M. Olariaga, E. F. Bompard, G. Chicco, A. Estebsari, T. Huang, E. Pons *et al.*, "A european platform for distributed real time modelling & simulation of emerging electricity systems," 2016.

[3] M. Stevic, S. Vogel, A. Monti, and S. D'Arco, "Feasibility of geographically distributed real-time simulation of hvdc system interconnected with ac networks," in *2015 IEEE Eindhoven PowerTech*, June 2015, pp. 1–5.

[4] (Accessed 2017-09-27) Decentralized network 42. [Online]. Available: http://www.dn42.net

[5] (Accessed 2017-09-27) Wireless Battle Mesh. [Online]. Available: http://battlemesh.org

[6] E. Bompard, A. Monti, A. Tenconi, A. Estebsari, T. Huang, E. Pons, M. Stevic, S. Vaschetto, and S. Vogel, "A multi-site real-time co-simulation platform for the testing of control strategies of distributed storage and v2g in distribution networks," in *2016 18th European Conference on Power Electronics and Applications (EPE'16 ECCE Europe)*, Sept 2016, pp. 1–9.

[7] M. Stevic, S. Vogel, M. Grigull, A. Monti, A. Estebsari, E. Pons, T. Huang, and E. Bompard, "Virtual integration of laboratories over long distance for real-time co-simulation of power systems," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Oct 2016, pp. 6717–6721.

[8] FEIN Aachen e. V. (Accessed 2017-09-27) VILLAS Framework. [Online]. Available: http://www.fein-aachen.org/projects/villas-framework

[9] S. Hemminger and others, "Network emulation with NetEm," in *Linux conf au*, 2005, pp. 18–23. [Online]. Available: https://www.rationali.st/blog/files/20151126-jittertrap/netem-shemminger.pdf

[10] (Accessed 2017-09-27) netem. [Online]. Available: https://wiki.linuxfoundation.org/networking/netem

[11] M. Mirz, A. Estebsari, F. Arrigo, E. Bompard, and A. Monti, "Dynamic phasors to enable distributed real-time simulation," in *2017 6th International Conference on Clean Electrical Power (ICCEP)*, June 2017, pp. 139–144.

[12] L. Wang, J. Jatskevich, V. Dinavahi, H. W. Dommel, J. A. Martinez, K. Strunz, M. Rioual, G. W. Chang, and R. Iravani, "Methods of interfacing rotating machine models in transient simulation programs," *IEEE Transactions on Power Delivery*, vol. 25, no. 2, pp. 891–903, April 2010.

[13] W. Ren, M. Steurer, and T. L. Baldwin, "Improve the stability and the accuracy of power hardware-in-the-loop simulation by selecting appropriate interface algorithms," *IEEE Transactions on Industry Applications*, vol. 44, no. 4, pp. 1286–1294, July 2008.

[14] (Accessed 2017-09-27) RESERVE. [Online]. Available: http://www.re-serve.eu/

全数字实时仿真器——HYPERSIM

周保荣[1]，房大中[1]，Laurence A. Snider[2]，陈家荣[2]
(1. 天津大学电力自动化及能源工程学院，天津市 300072；2. 香港理工大学电机工程学系，香港)

摘要：介绍了加拿大HYDRO-QUEBEC公司开发的一种功能强大、界面友好、接口方便的全数字实时仿真软件HYPERSIM。这种全数字实时仿真软件是为在线实时应用设计的，也可进行离线仿真，可用于电力系统、电力电子及电机拖动分析。A/D和D/A接口、用户代码模块(UCB)接口以及与商业软件MATLAB和EMTP等接口极大地扩充了程序应用的灵活性。文中通过3机9节点系统的仿真实例说明了程序的性能。

关键词：电力系统仿真；实时仿真器；静止无功补偿器；并行计算

中图分类号：TM743

0 引言

电力系统实时仿真器经历了从第1代模拟仿真器即暂态网络分析器，到第2代模拟一数字混合仿真器，再到第3代全数字仿真器的发展过程。HYPERSIM是加拿大HYDRO-QUEBEC公司开发的一种基于并行计算技术、采用模块化设计、面向对象编程的电力系统全数字实时仿真软件，目前具有Unix，Linux，Windows等3种版本。这种实时仿真软件既可在SUN Unix工作站或Linux/Windows PC机上进行离线仿真，也可在SGI Unix超级计算机或Linux PC-Cluster上与实际电力系统器件联接而进行实时在线仿真。HYPERSIM提供了电磁仿真的准确性、并行处理器强大的计算能力以及离线仿真的灵活性，比传统的模拟仿真器应用更加灵活、简单、廉价，可用于电力系统、电力电子及电机拖动分析，用于HVDC及FACTS设备动态性能测试，以及控制系统性能测试、继电保护和重合闸装置闭环测试。

1 HYPERSIM软件组成及功能

1.1 软件组成

HYPERSIM软件包由图形用户界面模块HYPER、代码生成模块、波形显示及分析模块SPECTRUM和命令控制模块CNTR组成。

a. 图形用户界面模块HYPER

该模块由工作空间、元件模板、画图模板等3部分组成，可构建电力系统结构图、控制系统模块图和设置元件参数等。图形用户界面有两种操作模式：编辑模式和仿真控制模式。在编辑模式下，用户可利用元件模板中的电力系统器件，如发电机、变压器、线路和开关构建电力系统网络拓扑；可利用元件模板中控制器件构建各种控制系统，如发电机励磁调节器、调速器、FACTS和HVDC控制器；也可利用画图模板中的工具添加文字说明或其他装饰性、解释性图画。仿真开始前，用户必须在仿真控制模式下选择离线或在线仿真模式，选择仿真平台和仿真步长；仿真开始后，HYPERSIM自动进行任务分配、代码编辑及生成。用户可在仿真控制模式下通过元件的控制面板修改该元件参数。在仿真控制模式下，通过元件控制面板或文件选择信号探测器后，HYPERSIM就可以与波形显示及分析模块SPECTRUM通信。

b. 代码生成模块

代码生成器用来分析网络拓扑，分解线路、母线、控制元件及其子系统为不同的任务，自动将任务合理分配给各并行处理器，使各任务之间通信负担最小。代码生成器为每一任务生成C代码，编辑后可直接在SUN或SGI等工作站离线运行，或下载到SGI等实时仿真器在线运行。

c. 波形显示及分析模块SPECTRUM

该模块可以处理信号探测器获取的信号，观察波形，对波形进行代数或频谱分析。SPECTRUM可读取商业软件MATLAB和EMTP的仿真结果，显示波形。

d. 命令控制模块CNTR

该模块是命令文本，执行与图形用户界面中的操作指令相同的操作。命令可通过键盘或文件输入，因此，一系列操作可在命令控制模块中编程后顺

收稿日期：2002-08-20；修回日期：2002-12-19。

序执行,无须用户干预。

1.2 建模能力

HYPERSIM 具有强大的建模能力,包含了丰富的电力系统及控制系统元器件。无源器件有电阻器、电容器、电感器、变压器、线路,以及非线性元件——避雷器和变压器饱和支路等;开关器件有理想开关、断路器、晶闸管、二极管、GTO 等;有源器件包括HVDC 变换器、SVC、交直流电动机、非线性负荷、发电机等。简单的控制系统可用HYPERSIM 的控制器件建模。此外,HYPERSIM 提供了两种对复杂控制系统建模的方法:一种是HYPERSIM 的用户代码模块(UCB——user coded blocks)功能;另一种是在MATLAB 中用SIMULINK 工具箱设计控制系统。

2 HYPERSIM 求解方法

2.1 并行算法

输电线路互联的电站组成电力系统网络,各电站信号以光速在输电线路中传播。然而,传输信号具有传输延迟,这种延迟随线路长度而变化。因此,当仿真步长小于传输延迟时,对电站和线路并行仿真是可行的。电站包括无源器件、电源、电动机、控制系统。大多数控制系统的时间常数远大于仿真步长,因此,这些控制系统也可独立仿真,与电站节点方程并行处理,整体准确性不受影响。HYPERSIM 将传输线路、电站、控制系统分解为并行子任务,这些子任务分配到可用的处理器。每一仿真时步开始,分别计算每一子任务;仿真时步结束,各子任务相互交换信息。

2.2 节点求解法

不同于MATLAB 变步长仿真技术,HYPERSIM采用固定步长隐式梯形法。同EMTP 一样,HYPERSIM 将储能元件——电感器和电容器等值为电阻并联电流源,建立节点导纳矩阵,采用节电导纳方程求解网络变量。每一仿真时步用节点导纳矩阵和注入电流求解节点电压。当网络拓扑因开关动作改变时,修改节点导纳矩阵,采用LU 分解法重新对节点导纳矩阵求逆。

3 HYPERSIM 接口

HYPERSIM 的主要优点在于它是一个开放结构的仿真软件,有许多方便灵活的接口。在仿真文件开发过程中,通过UCB 接口,提供用户一个开发环境和集成方法,增加HYPERSIM 建模能力和应用灵活性;通过MATLAB 接口,可充分发挥商业软件的优势,提高仿真文件的开发效率;通过EMTP 接口,可将EMTP 和HYPERSIM 仿真文件相互转换,因而对常规电磁仿真程序完全兼容;通过A/D、D/A 接口,可将硬件设备联接到虚拟的电力系统环境中进行实时仿真。此外,通过命令控制模块CNTR,用户可在HYPERSIM 的基础上进行二次开发,充分利用HYPERSIM 强大的计算功能,进行更为复杂和特殊的分析。下面详细介绍各个接口。

3.1 UCB 接口

UCB 接口提供了一个开发环境和集成方法。在用户代码模块中,可以利用类似C 语言的编程方法,开发网络模块和控制模块。用户代码经过编辑后可以集成到HYPERSIM 中,成为标准模块。在图形用户界面元件模板中,一种特殊的图标表示UCB。用户可以像使用其他标准模块一样使用UCB,将UCB 与其他标准模块互联,构建仿真文件。

3.2 MATLAB 接口

MATLAB 是一种功能强大的科学计算软件。SIMULINK 是MATLAB 中设计控制系统的工具箱。SIMULINK 构建的控制系统经过Real time Workshop 工具转化为C 语言,可以集成到HYPERSIM软件中。波形显示及分析模块SPECTRUM 可以直接读取MATLAB 的仿真结果,显示和处理波形。

3.3 EMTP 接口

HYPERSIM 在图形用户界面模块的编辑模式下可以将EMTP 数据文件导入,在HYPERSIM 的图形用户界面工作空间中自动形成电力系统网络接线图,形成一个HYPERSIM 仿真文件。在图形用户界面的仿真控制模式中,可将HYPERSIM 文件导出,形成一个EMTP 仿真文件。波形显示及分析模块SPECTRUM 可以读入EMTP 的仿真结果,显示和处理波形。

3.4 A/D 和D/A 接口

A/D 和D/A 接口使HYPERSIM 与外部硬件设备相连,形成闭环回路,可用于FACTS、继电保护、自动重合闸等设备的实时闭环测试。

4 应用实例

选择文献[4]中3 机9 节点电力系统(WSCC)作为应用例子,用HYPERSIM 建模、仿真和演示HYPERSIM 准确的计算性能,强大的建模能力,以及方便灵活的接口技术。为了充分演示HYPERSIM的性能,对3 机9 节点系统做了修改,在母线C 处安装±100 Mvar 的并联静止无功补偿器(SVC)。

图1 为在HYPERSIM 工作空间中的3 机9 节点系统接线图,与母线G1 相联的POW 单元提供一个同步信号,作为开关动作及SPECTRUM 数据采集的时间基准。

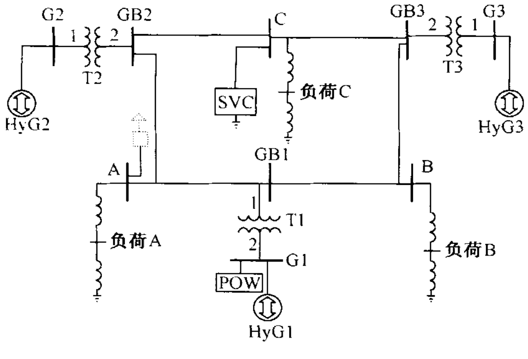


图1 WSCC 系统接线图
Fig.1 Configuration of WSCC power system

4.1 模型介绍

修改后的3机9节点系统包括3个发电机、1个FC-TCR 类型的SVC 以及谐波过滤器。HYPERSIM提供了3种发电机模型:水力涡轮发电机、火力涡轮发电机和双轴式涡轮发电机。本文算例采用具有恒定励磁电压和机械功率的水力涡轮发电机模型。为了避免数值振荡,本文采用分布参数传输线模型和恒定阻抗负荷模型。HYPERSIM 提供了具有内部控制系统、也有外部控制系统接口的SVC 模型。为了演示HYPERSIM 的建模能力和方便的接口技术,本文采用晶闸管、电感器、电容器构建FC-TCR 类型的SVC 主电路,用MATLAB 的SIMULINK 工具箱构建SVC 的控制系统。

图2所示SVC 电路包括星形联接的固定电容器、晶闸管控制电抗器(TCR)、谐波过滤器。固定电容器发出100 Mvar 无功,TCR 最大吸收200 Mvar 无功。开关用于闭合或断开SVC 与母线C 的连接。电压测量1和2输入三相瞬时电压到SIMULINK SVC 控制器,SIMULINK SVC 控制器产生等间距的6个脉冲。脉冲经过浮点数/整数转换,通过HYPERSIM 的MUX 控制元件将6个输入信号拟合为1个到晶闸管的输出信号。

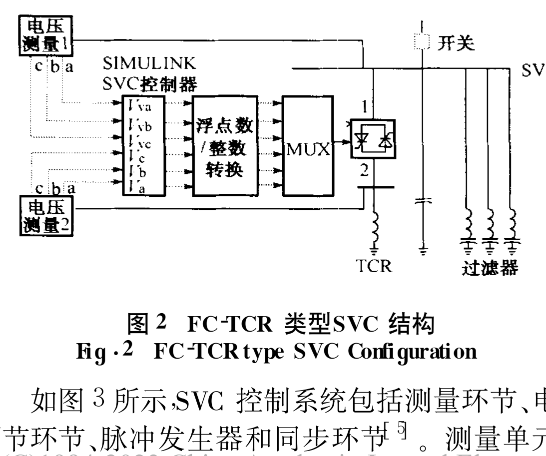


图2 FC-TCR 类型SVC 结构
Fig.2 FC-TCR type SVC Configuration

如图3所示,SVC 控制系统包括测量环节、电压调节环节、脉冲发生器和同步环节[5]。测量单元产生一个与基频电压均方根成正比的直流量,这个直流量作为电压调节器的输入。电压调节器进行闭环电压调节,测量电压和参考电压的偏差经比例积分控制器,产生所需电纳。同步单元产生与电压同步的三角波,三角波与触发角 α 比较,产生触发脉冲,控制晶闸管的闭合。

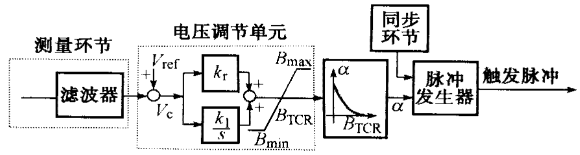


图3 SVC 控制系统示意图
Fig.3 Functional control scheme for FC-TCR static var compensator

应用MATLAB 的SIMULINK 工具箱构建SVC 控制系统,采用MATLAB 中Real time Workshop 工具生成C 代码。HYPERSIM 中SimulCr 模块集成SIMULINK 生成的SVC 控制系统到HYPERSIM 中。图2中SIMULINK SVC 控制器就是由SIMULINK 构建生成的SVC 控制器模块。

4.2 仿真结果

采用50 μs 的仿真步长,仿真时间为1s,仿真期间改变SVC 的参考电压,观察SVC 的响应速度。图4中实线和虚线分别表示SVC 测量电压 V_{mea} 和参考电压 V_{ref} 的有效值。参考电压0.2s 时从0.99跃变为1.03;0.4s 时跃变为0.83;0.65s 时跃变为0.99。从仿真结果可看出,SVC 在几个周期之内实现电压的调整。

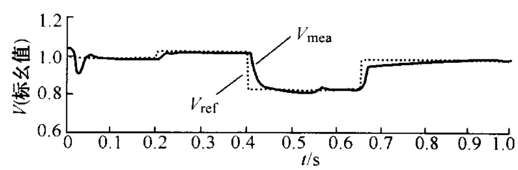


图4 SVC 参考电压和测量电压曲线
Fig.4 Curves of SVC reference voltage and measured voltage

图5是图4曲线的局部放大,表示SVC 参考电压 V_{ref} 在0.2s 从0.99跃变为1.03时参考电压和测量电压的变化曲线。

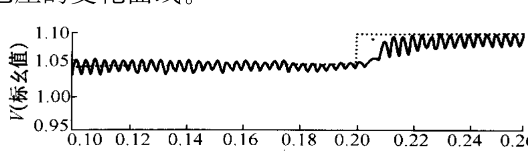


图5 SVC 参考电压和测量电压局部放大曲线
Fig.5 Zoomed in curves of reference voltage and measured voltage

图6表示与图5中电压变化相对应的A相TCR触发脉冲、TCR电流及TCR电压波形的变化曲线。

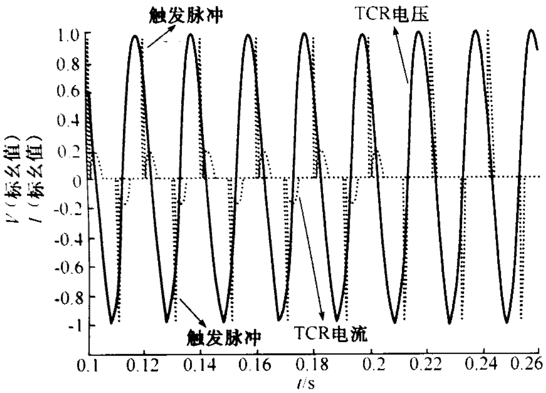


图6 A相TCR电流、电压、触发脉冲
Fig.6 Current through TCR, voltage across TCR and firing pulse in phase A

5 结语

与传统的电磁仿真程序EMTP,EMTDC等相比,HYPERSIM不仅提供了电磁仿真的准确性,而且提供了实时仿真的快速性。此外,通过模拟和数字输入输出接口,HYPERSIM可与外部硬件设备相连,形成闭环回路,可进行继电保护闭环测试、FACTS或HVDC控制系统研究、交直流混合系统研究等。

与传统的实时仿真器TNA相比,HYPERSIM应用更加灵活、方便。与实时数字仿真器RTDS相比,HYPERSIM具有高性能价格比、易安装维护等特点。HYPERSIM用户不仅可在SGI超级计算机上做电力系统暂态仿真,而且可在SGI超级计算机上运行其他商业软件,充分利用超级计算机的性能。RTDS实质上是为实时仿真电力系统暂态过程专门设计的并行计算机系统,只能运行RTDS的软件。

与EMTP和RTDS相比,HYPERSIM面临的挑战在于电力工程师缺乏应用HYPERSIM进行仿真的经验,缺乏充足的技术资源。但可以预言,随着HYPERSIM软件的不断完善,电力工程师仿真经验的积累,HYPERSIM将会在工业和科研领域得到广泛应用。

参考文献

1 Sybille G, Giroux P. Simulation of FACTS Controllers Using the MATLAB Power System Blockset and Hypersim Real time Simulator. In: Proceedings of 2002 Power Engineering Society Winter Meeting. New York(USA): 2002. 488～491

2 Barry A O, Guay F, Guerette S, et al. Digital Real time Simulation for Distribution Systems. In: Proceedings of 2000 IEEE 9th International Conference on Transmission and Distribution Construction, Operation and Live line Maintenance. Montreal(Canada): 2000. 252～258

3 Kaddouri A, Khodabakhchian B, Dessaint L A, et al. New Generation of Simulation Tools for Electric Drives and Power Electronics. In: Proceedings of the IEEE 1999 International Conference on Power Electronics and Drive Systems, Vol 1. Hong Kong (China): 1999. 348～354

4 Anderson P M, Fouad A A. Power System Control and Stability. New York: IEEE Press, 1994

5 IEEE Special Stability Control Working Group. Static Var Compensator Models for Power Flow and Dynamic Performance Simulation. IEEE Trans on Power Systems, 1994, 9(1)

周保荣(1974—),男,博士研究生,研究方向为电力系统控制。E-mail: eebrzhou@tju.edu.cn

房大中(1946—),男,博士,教授,博士生导师,研究领域为电力系统分析与控制。

Laurence A. Snider,男,博士,高级讲师,研究方向为电力系统实时仿真和电能质量。

THE FULLY DIGITAL REAL-TIME SIMULATOR——HYPERSIM

Zhou Baorong[1], *Fang Dazhong*[1], *Laurence A. Snider*[2], *K. W. Chan*[2]
(1. Tianjin University, Tianjin 300072, China)
(2. Hong Kong Polytechnic University, Hong Kong, China)

Abstract: HYPERSIM, a new powerful fully digital real time simulator with convenient interface developed by HYDRO-QUEBEC Co., is introduced in this paper. This simulator with on line and off line modes can be utilized as a simulation tool for the analysis of power systems, power electronics and drive systems. The flexibility of its application is enhanced by the interfaces for A/D and D/A, user coded block (UCB), and some commercial software such as MATLAB and EMTP. A static var compensator (SVC) simulation on the three-generator nine-bus test system demonstrates the performance of the simulator.

Key words: HYPERSIM; power system simulation; real time simulator; SVC; parallel computation

SoftwareX 10 (2019) 100253

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

DPsim—A dynamic phasor real-time simulator for power systems

Markus Mirz *, Steffen Vogel, Georg Reinke, Antonello Monti

Institute for Automation of Complex Power Systems, RWTH Aachen University, Aachen, Germany

ARTICLE INFO

Article history:
Received 17 December 2018
Received in revised form 12 April 2019
Accepted 21 May 2019

Keywords:
Real-time simulation
Dynamic phasors
Co-simulation

ABSTRACT

DPsim is a real-time capable solver for power systems that operates in the dynamic phasor and electromagnetic transient (EMT) domain. This solver primarily targets co-simulation applications and large-scale scenarios since dynamic phasors do not require sampling rates as high as EMT simulations do. Due to the frequency shift introduced by the dynamic phasor approach, the sampling rate and rate of data exchange between simulators can be reduced. DPsim supports the Common Information Model (CIM) format for the description of electrical network topology, component parameters and power flow data and it is closely integrated with the VILLASframework to support a wide range of interfaces for co-simulation. Simulation examples demonstrate the accuracy of DPsim and its real-time performance for increasing system sizes.

Code metadata

Current code version	v1.0.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2018_244
Legal Code License	GPLv3
Code versioning system used	git
Software code languages, tools, and services used	C++, python
Compilation requirements, operating environments & dependencies	DPsim can be compiled for Linux, OSX and Windows. The main dependencies are: gcc, Eigen, Python, CIM++, VILLASnode, Sundials. A Dockerfile with all dependencies is included in the repository.
If available Link to developer documentation/manual	https://fein-aachen.org/projects/dpsim/
Support email for questions	mmirz@eonerc.rwth-aachen.de

Software metadata

Current software version	v1.0.0
Permanent link to executables of this version	https://hub.docker.com/r/rwthacs/dpsim
Legal Software License	GPLv3
Computing platforms/Operating Systems	Linux docker container
Installation requirements & dependencies	Only a docker installation is required.
If available, link to user manual	https://fein-aachen.org/projects/dpsim/
Support email for questions	mmirz@eonerc.rwth-aachen.de

1. Motivation and significance

DPsim introduces the dynamic phasor (DP) approach to real-time power system simulation. The motivation is to remove the requirement of proportionality between the simulation time step

* Corresponding author.
E-mail address: mmirz@eonerc.rwth-aachen.de (M. Mirz).

https://doi.org/10.1016/j.softx.2019.100253

and the highest frequency of simulated signals. Especially for power electronics and geographically distributed real-time simulation, this is an interesting feature. Currently, the focus of DPsim is more on the application in distributed real-time co-simulation than power electronics. The idea is that the larger the simulation step, the smaller the impact of the communication delay between simulators, which are geographically distributed.

Distributed real-time co-simulation is motivated by large system simulations requiring more simulation capacity than is locally available and by the possibility of Hardware-In-the-Loop (HIL) testing with hardware under test and simulators at different locations. Using dynamic phasors for this application is a fairly recent development although the dynamic phasor, or the envelope function concept, is well known and was introduced in power electronics analysis in [1] as a generalized state space averaging method.

The authors of [2] have developed a distributed real-time simulation laboratory by applying a communication platform as a simulator-to-simulator interface in order to enable remote and online monitoring of interconnected transmission and distribution systems. Each simulator carries out simulations in time domain, while the variables exchanged at the interconnected nodes, i.e. decoupling point, are in the form of time-varying Fourier coefficients. The electromagnetic transient (EMT) values cannot be exchanged at every simulation step due to the communication delay. This delay may be directly incorporated into the physical power system model using traveling wave transmission line models [3]. However, electromagnetic waves travel about 15 km in 50 μs, a time which equals the typical step time in real-time EMT power system simulation. This means that a communication delay in the range of milliseconds would have to be compensated with a line length of several hundred kilometers which may require large and unrealistic changes to the original system model. Therefore, this method is suited for applications on local simulation clusters where the delay is only few simulation steps, but not for internet-distributed simulation, where the expected delay is often tens of milliseconds. In the latter case, the insertion of a transmission line model with the required parameters into the model would have a severe impact on the behavior of the system. Without compensation, the communication delay might cause large errors and even instability of the simulation as shown in [4] for a delay of more than 30 ms.

Starting from the concept presented in [2], we propose interfaces and system-wide simulation state variables based on dynamic phasors. This approach has two benefits explained in [5], which presents a comparison of DP and EMT simulations conducted in DPsim. First, it takes advantage of the computational efficiency of dynamic phasors in scenarios that involve bandpass signals with large center frequencies, as in the case of switching power electronics. Secondly, it increases the simulation time step thus reducing the difference between the local simulation time step size and the round trip time between simulators.

The approach employed in [2] requires the extraction of phasor information from the EMT signals. Therefore, the transparency of the interface is not guaranteed since the interface algorithm may alter the exchanged signals. This extraction step is eliminated by using dynamic phasors as state variables. Because of these features, DPsim is now part of the distributed real-time co-simulation platform described in [6], which is also the base for the experiment described in [2]. A first example of distributed real-time co-simulation using DPsim is presented in [7]. This example shows the impact of the latency in geographical distributed simulation on the results and how the latency can be modeled a priori to running actual simulations.

Besides, there are other relevant open-source initiatives in the field of power system simulation to be mentioned. In particular the Modelica community is very active in adapting Modelica environments for large scale power system cases [8,9] and providing comprehensive libraries of models [10,11]. These initiatives aim to enable large scale, fast simulation of power systems using open-source components but the focus is slightly different compared to the work presented here. The primary objective of DPsim is to assure a deterministic time step in terms of simulated and computation time to provide real-time capability required for Hardware-in-the-Loop (HIL) experiments. This is why in DPsim higher order solver methods are avoided and the system is split into subsystems, which are solved separately. The aforementioned related work does not seem to rely on such compromises since a deterministic time step is not required. While Modelica allows a convenient definition of physical models, DPsim does not intend to provide a solution in this regard. The idea is rather to extend the set of available models in DPsim by generating C-code from Modelica models to represent components connected to the network. These can be linked to DPsim and solved by the ODE solver that was integrated into DPsim for this purpose.

Another related work which is not open-source but also DP based is described in [12]. Similarly to DPsim, the authors have developed a simulator based on the DP approach. However, the focus does not seem to be fast simulation or real-time simulation since the simulator was developed in Python and there are no measures described in order to speed up the simulation. Furthermore, the validation is focusing on the correctness of the simulation rather than simulation speed.

2. Software description

The main theoretical building blocks of DPsim are the dynamic phasor concept and the modified nodal analysis (MNA). Dynamic phasors [13] have various names in scientific literature. Depending on the field and application, they are known as generalized averaging method [1], shifted frequency analysis [14], equivalent envelope [15,16] or baseband signal [17]. Here, the term dynamic phasors is selected because it is widely known in the power system community. The basic idea of dynamic phasors is to approximate a time domain signal x with a Fourier series representation as shown in (1)

$$x(\tau) = \sum_{k} X_k(t) e^{jkw_s(\tau)} \quad (1)$$

where $\tau \in (t-T, t]$. The kth coefficient is determined by

$$X_k(t) = \langle x \rangle_k(t) = \frac{1}{T}\int_{t-T}^{t} x(\tau) e^{-jkw_s(\tau)} d\tau \quad (2)$$

where ω_s is the fundamental system frequency and $k\omega_s$ are its harmonics.

MNA is used for the representation of the network as a linear equation system, whereas complex components, such as electrical machines, connected to the network are treated by a separate ODE solver. Depending on the simulation scenario, the admittance matrices of the required network topologies can be preprocessed, i.e. factorized, before simulation start to guarantee a deterministic simulation time step.

The simulation kernel of DPsim is extended with interfaces to support different use cases such as circuit or system simulation, batch simulation, co-simulation and HIL testing. DPsim supports the Common Information Model (CIM) [18] as native input for the description of electrical network topologies, component parameters and load flow data, which is used for initialization. Users can interact with the simulation kernel via Python bindings, which can be used to script the execution, schedule events, change parameters and retrieve results. Python scripts have been proven an easy and flexible way to codify the complete workflow of a

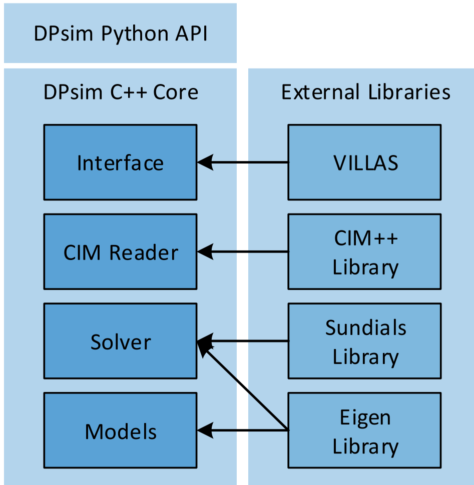


Fig. 1. Overview of DPsim's main components and dependencies on external libraries.

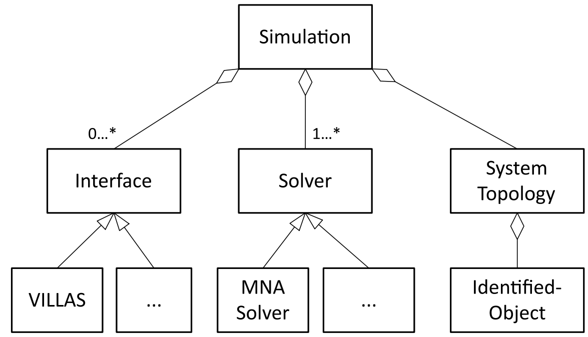


Fig. 2. Diagram of the main classes of DPsim.

simulation from modeling to analysis and plotting, for example in Jupyter notebooks using Numpy and Matplotlib. Furthermore, DPsim supports co-simulation and interfaces to a variety of communication protocols of commercial hardware via the integration of DPsim with the *VILLASframework* [19], that enables large-scale co-simulations and HIL experiments.

2.1. Software architecture

2.1.1. Module structure

The DPsim simulation kernel and its component models are implemented in the C++ programming language. The availability of good compilers and highly optimized software libraries, such as *Eigen* [20], *Sundials* [21] and *VILLASnode*, which is part of the VILLASframework, for C++ were key factors for this decision.

The core of DPsim consists of models and solvers as depicted in Fig. 1. Interfaces to other software, hardware or data are supported through external libraries. Grid data in CIM standard format is imported using the *CIM++* library [22]. Communication with other software, e.g. real-time simulators, control and monitoring software, as well as hardware is provided by the VILLASframework. For linear algebra operations, DPsim uses the Eigen library. The MNA solver of DPsim uses the Eigen LU factorization and *Eigen::Dense/Sparse::Matrix* are the standard data types for all matrix variables. The Sundials solver package is included in DPsim to provide more complex ODE solvers for components connected to the network.

Compilation from source code requires a Git, a C++11 compiler and CMake 3.6. Tested compilers are Clang, GCC, MSVC and Intel's ICC. As a base operating system Ubuntu 18.04 LTS or Fedora 29 are recommended. For real-time execution a Linux 4.9 kernel with the *PREEMPT_RT* patch-set is recommended.

2.1.2. Class hierarchy

Simulation is the main interface class for users to control the simulation state. The attributes of a *Simulation* instance hold all the information that specifies one simulation scenario in DPsim. The *Simulation* holds references to the solvers, interfaces and the power system model information. This hierarchy is presented in Fig. 2. The complete class hierarchy diagram can be found in the developer's documentation of DPsim [23]. All solvers inherit from *Solver*, e.g. *MNASolver* and *ODESolver*, and all interfaces from *Interface*, e.g. for VILLASnode. All component models, power system, signal etc., derive from the class *IdentifiedObject*. The main attribute of this class is a unique identifier, which is equivalent to the *mRID* in CIM.

2.1.3. Parallelization

In order to utilize the multiple processor cores of modern computer systems and speed up the simulation, the computation of one time step is split up into multiple tasks. These tasks are defined by the various parts of the simulation (like instances of *Solver* and *Interface*). An internal scheduler creates a task graph by analyzing which variables are modified and used by the tasks. This task graph is a directed acyclic graph with nodes representing tasks and edges representing data dependencies. The scheduler uses this graph to distribute the tasks onto multiple threads such that these dependencies are upheld. DPsim implements different scheduling algorithms for this purpose, the simplest of which being level scheduling as used in [24]. This algorithm divides the tasks into ordered levels such that each task only depends on tasks in a previous level. To simulate a time step, these levels are then executed in order by distributing all tasks of one level evenly among the available threads.

To further optimize the simulation performance for large networks when using multiple processors, a decoupling transmission line model can be introduced. In this model a transmission line is represented using equivalent current sources and resistances at the line ends that are not topologically connected as described in [25, ch. 6.2]. Instead, the ends are indirectly connected by updating the values of the sources based on the values on the other end only after a delay τ, which depends on the line's parameters. As subsystems that are connected using this line model are not connected in a strict topological sense, they can be solved independently and in parallel. This significantly reduces the computational effort to simulate large systems.

2.2. Software functionalities

DPsim supports both dynamic phasor and EMT simulation. Furthermore, DPsim is optimized for real-time simulation but it is also possible to run the same simulation scenario offline meaning that the simulation is executed as fast as possible. These different simulation modes are compatible with the user and simulation interface which are covered in the following.

2.2.1. User interface

Network models can be directly defined in the C++ code. This technique is complemented by a CIM importer, which allows the user to directly load network models from CIM-XML files. This proves to be an adequate form to describe network topology and component parameters, which are required by the solver. This CIM importer relieves the user from defining the model in plain C++ code. However, CIM-XML is not suitable for the definition of more complex simulation scenarios with time varying parameters or topology changes caused by contingencies in the system (e.g. breaker events or faults). Therefore, DPsim features Python bindings to most parts of the C++ programming interface. A scripting language like Python is used to define scenarios by leveraging the flexibility of a general purpose imperative language. This allows the user to write a single Python script to:

- Describe the network topology and parameters
- Load a network from CIM data and optionally extend it
- Define a simulation scenario with events or parameter changes
- Execute the simulation and analyze or plot the simulation results

2.2.2. Simulation interface

Interfacing the simulation kernel is desirable for multiple reasons:

- Real-time exchange of simulation signals for co-simulation or HIL testing
- User interface for online monitoring and control of the simulation
- Logging of simulation results for offline analysis
- Import of time series data, for example load and production profiles

For commercial simulation tools these interfaces are a major selling point as new protocols and standards and DAQ cards are introduced continuously. Their implementation and maintenance is time consuming and seemingly never ending. By design, DPsim tries to avoid this pitfall by leaving the implementation of interfaces, data formats and protocols to a separate project. VILLASnode, a component of the VILLASframework project, handles input/output as well as translation between different protocols. DPsim focuses on solving the system model and provides only a single type of interface, shared memory, to the VILLASnode gateway. Interfaces to external systems, databases, files or the web interface are then handled by the wide range of supported interfaces of the VILLASnode gateway, which in this case acts as a proxy between the shared-memory interface to DPsim and the outside world. Responsibilities are clearly separated. This allows the development of new interfaces without having to modify the simulation kernel itself.

In addition, DPsim can use the VILLASnode interfaces for co-simulation with other simulators or remote DPsim instances. In such a scenario, DPsim is usually coupled using an Ideal Transformer Model (ITM). Fig. 3 shows a decoupled model, which exchanges voltages in one and current signals in the opposite direction to control respective sources. For a phasor simulation, the exchanged signals are complex-valued attributes which are passed via the shared-memory interface to VILLASnode which further sends them to a remote simulator using one of VILLAS' supported protocols (e.g. MQTT, UDP, ZeroMQ, …). For geographically distributed simulations, VILLASnode can implement interface algorithms to compensate for the inherent communication latencies when executed in real-time over a high latency connection such as the internet. Alternatively, the implementation of a Discrete Fourier Transform (DFT) in VILLASnode allows for a coupling of the phasor-based DPsim with other EMT-based simulation tools like OPAL-RT or RTDS.

DPsim exchanges simulation data with the VILLASnode gateway via a shared-memory region. The execution of DPsim and VILLASnode as independent processes is crucial in real-time simulation scenarios as the main simulation kernel must not be interrupted by background activities such as data logging to a possibly blocking database.

Furthermore, DPsim has its own simple logging module to write results to CSV files for archival and post processing. This method is easy to setup and convenient for small simulations where post processing and analysis of simulation results is done in MATLAB or Python. In the long term, the internal CSV logging functionality is planned to be incorporated into VILLASnode and enhanced by the support of additional data formats like HDF5 as used by MATLAB.

3. Implementation and empirical results

To complement the previous overview of DPsim's architecture, the next subsection explains implementation specifics affecting the real-time performance of DPsim. The real-time performance of DPsim is demonstrated in the following subsection. The remaining two subsections demonstrate the correctness of the solution computed by DPsim against Matlab Simulink. The first simulation validates only the network solution, which is computed by the MNA solver. The second simulation features a combination of the MNA solver for the network solution and an ODE solver for the numerical integration of the synchronous generator equations.

3.1. Implementation details

Only a compiled language like C++ with minimal runtime overhead is suitable for the implementation since DPsim is targeting simulation time steps in the range of milliseconds to microseconds on off the shelf computing hardware. Great care was taken to avoid memory allocation during the actual simulation. Whenever possible, DPsim utilizes low order integration methods and avoids iterative solver strategies to minimize computation time. That is why the network part is handled by the MNA solver specifically developed for DPsim.

DPsim is compatible with Windows, macOS and Linux operating systems. Eventually, the operating system configuration can have a large impact on the real-time performance. To guarantee real-time execution, DPsim leverages several Linux real-time features such as the real-time capable *SCHED_FIFO* scheduler, real-time signals, the *timerfd* interface, or control groups (*cgroups*). Many of these features are nowadays incorporated in the standard Linux kernel but have their origin in the *PREEMPT_RT* patch-set. The patch-set is slowly integrated into the mainline kernel but still exists and can be applied to further improve the real-time performance by enabling preemption of critical sections such as interrupt handlers. The capability to preempt critical sections in the operating system kernel reduces the overall system latency and therefore helps to ensure strict deadlines at each time-step interval.

Real-time execution on Windows or macOS is not supported. Best real-time performance was achieved on a recent Intel x86_64 multi-core machine with optimized BIOS settings to avoid interruptions of the system by the System Management Mode (SMM). To do so, DPsim execution threads are isolated from remaining system processes using Linux's *cgroup* feature. This reduces the impact of background jobs in the system on the real-time performance.

As a good start for real-time optimizations, we recommend Redhat's Real-time Guide [26] with its *tuned* tool and the *realtime* profile. An updated list of optimization options can be found in the DPsim documentation [23].

To control the time step, DPsim is using *timerfd* interface in Linux environments. The *timerfd* interface allows for the configuration repetitive interval and one-off timers. It uses blocking file descriptors to suspend the execution of the simulation loop until the beginning of the next interval. This approach is more

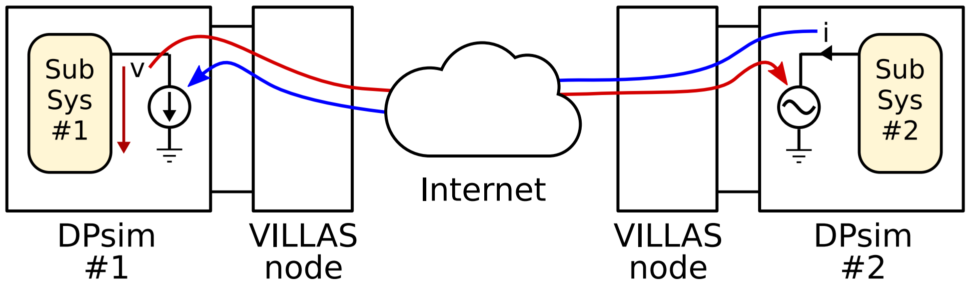


Fig. 3. VILLASnode as a gateway for distributed co-simulation with DPsim.

efficient than the use of the more common *timer* interface, which relies on signals. At the same time, *timerfd* is more accurate as calls to *sleep* or *nanosleep* as they suffer of a lingering drift to non-equidistant execution intervals. The *timerfd* is also used to schedule the synchronized start of distributed simulations as described in the introduction.

Shared-memory is a common method for inter-process communication (IPC) used on symmetric multi-processing (SMP) machines. It enables user processes to exchange data without the involvement of the OS kernel. Shared-memory IPC allows both processes, the solver and the gateway, to be executed in parallel while streaming their data with minimal latency over a queue in the shared memory region. The queue is implemented as a lock free multiple producer/multiple consumer (MPMC) queue and relies on atomic operations of the processor to facilitate synchronization of the DPsim and VILLASnode processes. The lockfree queue is a thread safe data structure. It is used to pass samples of simulation data between the involved processes. As such *message passing* is used as the main paradigm to avoid data races. During the initialization phase, semaphores are used to avoid race conditions in the setup of the shared-memory regions. The absence of the operating system in the communication is crucial to avoid costly context switches, which have to be avoided in a real-time context. Using the shared-memory interface, the DPsim simulation loop can run uninterrupted in a high priority process. At the same time, VILLASnode gets assigned a lower priority for handling of possibly blocking disk or database accesses. In a hardware-in-the-loop simulation it might be necessary to have hard real-time capable interfaces to the real world. For this purpose, DPsim supports an arbitrary number of shared-memory interfaces at the same time. This allows the user to configure one VILLASnode process with a high priority for the control of PCIe FPGA or DAQ cards, and at the same time another VILLASnode process for low priority logging of simulation data in the background.

3.2. Real-time performance evaluation

DPsim is specifically designed for real-time simulation. To assess the effect of system size on the real-time performance of DPsim, a simple test network was copied multiple times and connected with additional transmission lines. Fig. 4a shows the WSCC 9-bus system, which was used for this purpose. The copies were connected in a ring-like topology using additional transmission lines at the nodes 5, 6, and 8. The average wall clock time needed to simulate one time step was measured for a simulation time period of 0.1 s with a time step of 100 µs. Each measurement was further averaged over ten simulations of the same system. The measurements were performed on a system running Ubuntu 16.04 on an Intel Xeon Silver 4114 processor featuring ten cores clocked at 2.2 GHz. The results are shown in Fig. 4b for two different configurations: For the normal simulation, the additional transmission lines were modeled using the Pi model and only one thread was used. For the parallel simulation, the decoupling transmission line model was used for the additional lines and ten threads were employed. As it can be seen, the use of multiple threads and the special transmission line model greatly reduces the wall clock time necessary for simulating a single time step. Even for 40 copies of the original system (resulting in a system size of 360 nodes), the wall clock time per step stays under the simulation time step of 100 µs, thus allowing for real-time simulation. For a small number of system copies, DPsim supports simulation time steps of about 10 µs, which is a typical time step for commercial EMT simulators. However, it should be noted that such small time steps are not the aim of DPsim since the simulation is conducted in DP and not EMT.

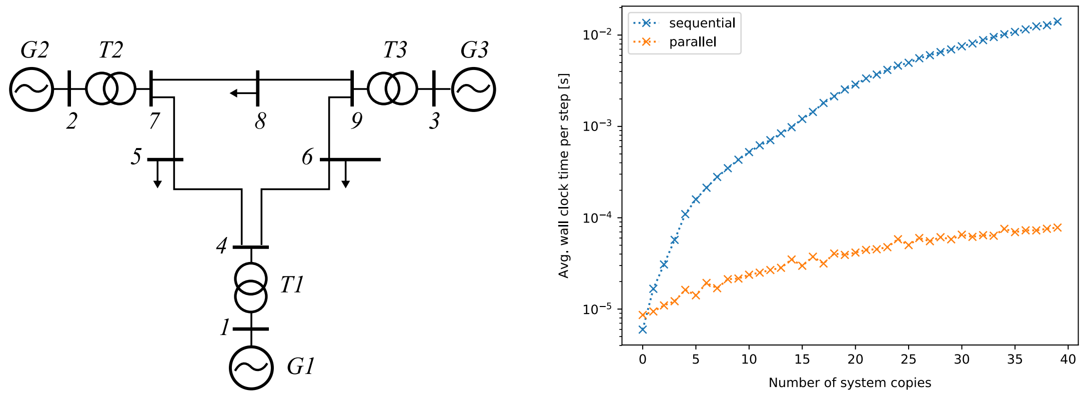
3.3. Validation of the MNA network solver

The next simulation case demonstrates the accuracy of the MNA network solver compared to Simulink results. Both simulators DPsim and Simulink are run with a simulation time step of 100 µs. It can be seen that for such small time steps DP simulations yield the same results as EMT. The larger the time step, the more will the results degrade. It is shown in [5] that the degradation of the results with larger time steps is smaller when using the DP approach compared to EMT.

Fig. 5a shows the simulated circuit, which is composed of one current source of 10 A, two resistors of 1 Ω, an inductor of 1 mH and a capacitor of 1 mF. The voltage source is set to its nonzero peak value at the beginning because it is following a cosine with zero phase shift. Therefore, the system is not starting from steady-state and a transient can be observed. The DPsim dynamic phasor results are transformed to time domain values and compared against Simulink EMT results. As can be seen from Fig. 5b, the results match.

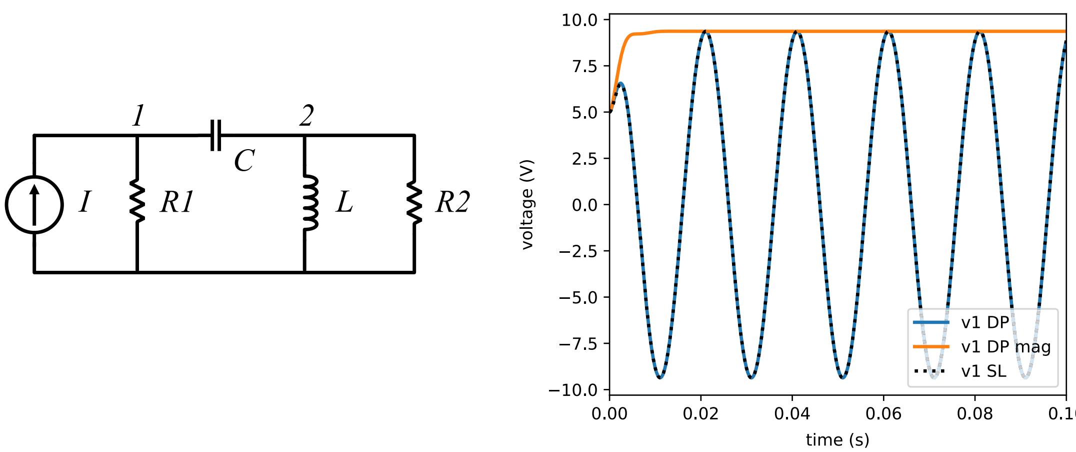
3.4. Validation of the ODE solver for components

The following example compares the results of Simulink and DPsim for a three phase synchronous generator fault. Here, the simulation time step is 50 µs for DPsim and Simulink. As in the previous simulation case, the DP approach would allow for larger time steps than EMT. A comprehensive study investigating this property and featuring synchronous generator models is presented in [27]. Initially, the load is 300 MW and the fault is applied at 0.1 s. The generator parameters are taken from example 3.1 in [28]. As in the previous example, the dynamic phasor results are transformed to time domain values before the comparison. Again, it is visible that the DPsim results match the Simulink results except when the fault is cleared (see Fig. 6). In contrast to DPsim, the fault in Simulink is not immediately cleared but at the next zero-crossing.



(a) (b)

Fig. 4. WSCC 9-bus system (a) and average wall clock time per step (b).



(a) (b)

Fig. 5. Example circuit (a) and DPsim dynamic phasor and Simulink EMT simulation results for node 1 (b).

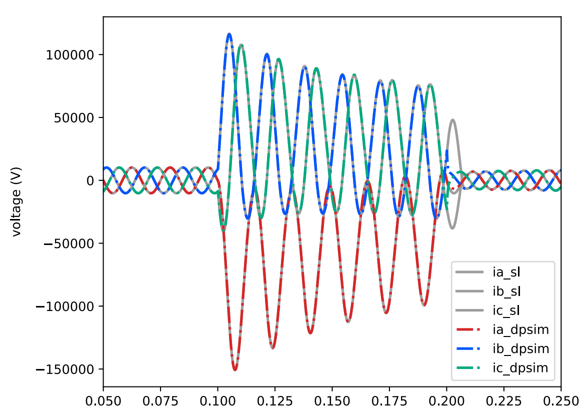


Fig. 6. DPsim dynamic phasor and Simulink EMT results for the synchronous generator three-phase fault example.

4. Illustrative examples

As mentioned in Section 2.2, there are two ways to define a circuit topology for DPsim: coding the topology using Python or C++ or importing it from CIM. The two options are presented by means of two examples: a circuit and a small power system. The first example presented in this section demonstrates the definition utilizing Python while the second example takes advantage of the CIM import function.

4.1. Defining a circuit simulation in python

The circuit of the previous section, Fig. 5, is taken as an example to demonstrate how circuits can be defined using DPsim's Python interface. The topology can be created in Python as depicted by Listing 1.

Listing 1: Python code to define a circuit.

```
# Nodes
gnd = dpsim.dp.Node.GND()
n1 = dpsim.dp.Node('n1')
n2 = dpsim.dp.Node('n2')

# Components
cs = dpsim.dp.ph1.CurrentSource('cs')
cs.I_ref = complex(10,0)
r1 = dpsim.dp.ph1.Resistor('r_1')
r1.R = 1
c1 = dpsim.dp.ph1.Capacitor('c_1')
c1.C = 0.001
l1 = dpsim.dp.ph1.Inductor('l_1')
l1.L = 0.001
r2 = dpsim.dp.ph1.Resistor('r_2')
r2.R = 1
```

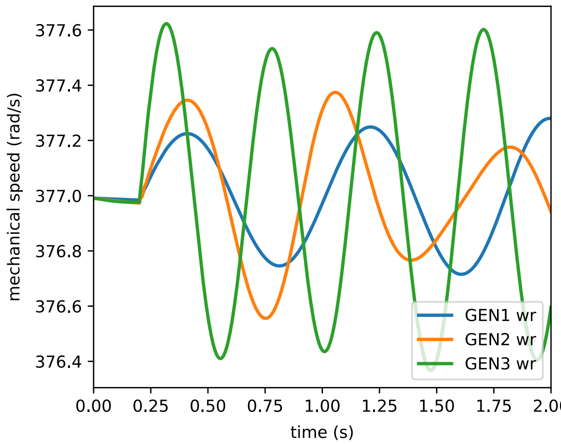


Fig. 7. WSCC 9-bus system mechanical speed simulation results after fault.

```
# Connections
cs.connect([gnd, n1])
r1.connect([n1, gnd])
c1.connect([n1, n2]);
l1.connect([n2, gnd]);
r2.connect([n2, gnd]);

system = dpsim.SystemTopology(50, [gnd, n1, n2], [cs, r1, c1, l1, r2]);
sim = dpsim.Simulation('circuit', system, timestep=0.0001, duration=0.1)
sim.start()
```

First, the nodes and components are declared and parameterized. Then, the connections between components and nodes are set and in the following step all network objects are added to the system topology. Finally, the system topology and parameters such as time step and final time can be used to create a simulation instance, which can be started, stopped and stepped through. Optionally, initial voltages could be assigned to the nodes. Since this is not the case here, the initial voltages are set to zero.

4.2. Simulating a power system defined in CIM

The next example presents the CIM loading functionality, which is used to read the data of the WSCC 9-bus system as displayed in Fig. 4a. The objects defined in the CIM file, e.g. *Terminal*, *TopologicalNode*, *SynchronousMachine*, are mapped to DPsim objects according to the *CIM::Reader* class of DPsim. The system frequency, 60 Hz, is not defined in the CIM file. Therefore, it needs to be specified before loading the CIM file. Furthermore, a fault is applied to node 9 of the imported system. To implement the fault, the system is extended by a switch connected to node 9, which connects the node to ground with a small resistance after 0.2 s. The switching action is created as an object of type *Event* and added to the *Simulation* instance.

Listing 2: Python code to import a topology from CIM.

```
# Read from CIM
files = glob.glob(' ../../ dpsim/Examples/CIM/WSCC09_RX_Dyn/*.xml')
system = dpsim.load_cim('WSCC9bus', files, frequency=60)

# Get existing nodes
gnd = dpsim.dp.Node.GND()
bus9 = system.nodes['BUS6']

# Add switch
sw = dpsim.dp.ph1.Switch('Switch')
sw.R_open = 1e9
sw.R_closed = 0.1
sw.is_closed = False
sw.connect([ bus9, gnd ])
system.add_component(sw)

sim = dpsim.Simulation('WSCC9bus', system,
timestep=0.0001, duration=2, init_steady_state=True)

sim.add_event(0.2, sw, 'is_closed', True)
sim.start()
```

In Fig. 7, it can be seen how the rotational speed of the generators starts to oscillate after the fault. The oscillation frequency depends on the mechanical inertia and as expected the generator with the largest inertia has the lowest oscillation frequency.

5. Impact

Since DPsim is open source, it can serve as a reference implementation for real-time power system simulators and a common basis for users working with different real-time simulation solutions. Having a common basis facilitates discussions on differences in simulation results of different tools. Besides, DPsim allows for real-time simulation on standard computing hardware, making real-time applications available to a wider range of researchers.

Thanks to its design, DPsim facilitates distributed real-time co-simulation, which promotes collaboration and allows the use of the simulation capacity of geographically distributed laboratories to support large scale scenarios [5]. Distributed real-time co-simulation allows also for better data privacy, enabling cooperation because, in a co-simulation, each entity can keep its data confidential and only exchange boundary variables. This could be interesting for confidential grid data but also black box device models where manufacturers cannot share implementation details.

The dynamic phasor approach is used here in a system level simulation in contrast to component level power electronics applications as in [1]. With an increasing share of power electronics in power systems, this approach supports the investigation of future grids.

DPsim is already used in the EU H2020 research project RESERVE [29] and it was developed as a solution to decrease the difference between communication delay and simulation time step in previous co-simulation projects, e.g. RT-Superlab [2]. DPsim is promoted by the FEIN association that also hosts its code and documentation [23].

6. Conclusions

The presented software project, DPsim, exploits the dynamic phasor approach to overcome the requirement for EMT simulation that the simulation time step needs to be proportional to the highest signal frequency. In doing so, DPsim facilitates distributed real-time simulation and lets users exploit simulation resources in different geographical locations. For this purpose, DPsim is programmed in the C++ language and has its own MNA based network solver. Despite having a C++ core, the DPsim allows for scripting simulations via the python interface and reading grid topologies in the standard CIM format via the CIM++ library. These features are demonstrated in two simulation examples, a circuit and a grid simulation. Likewise, the computational correctness of DPsim and its real-time performance are demonstrated by means of simulation examples. Furthermore, DPsim is tightly integrated with the VILLASframework that offers many interfaces to commercial real-time simulators and hardware.

Declaration of competing interest

We confirm that there are no known conflicts of interest associated with this publication.

Acknowledgments

This work was partly funded by the European Unions Horizon 2020 Framework Programme for Research and Innovation under grant agreement no 727481.

References

[1] Sanders SR, Noworolski JM, Liu XZ, Verghese GC. Generalized averaging method for power conversion circuits. IEEE Trans Power Electron 1991;6(2):251–9.
[2] Monti A, Stevic M, Vogel S, De Doncker RW, Bompard E, Estebsari A, Profumo F, Hovsapian R, Mohanpurkar M, Flicker JD, et al. A global real-time superlab: enabling high penetration of power electronics in the electric grid. IEEE Power Electr Mag. 2018;5(3):35–44.
[3] Schutt-Ainé JE. Latency insertion method (LIM) for the fast transient simulation of large networks. IEEE Trans Circuits Syst I 2001;48(1):81–9.
[4] Stevic M, Monti A, Benigni A. Development of a simulator-to-simulator interface for geographically distributed simulation of power systems in real time. In: Industrial electronics society, IECON 2015-41st annual conference of the IEEE. IEEE; 2015, p. 005020–5.
[5] Mirz M, Estebsari A, Arrigo F, Bompard E, Monti A. Dynamic phasors to enable distributed real-time simulation. In: Clean electrical power (ICCEP), 2017 6th international conference on. IEEE; 2017, p. 139–44.
[6] Vogel S, Mirz M, Razik L, Monti A. An open solution for next-generation real-time power system simulation. In: Energy internet and energy system integration (EI2), 2017 IEEE conference on. IEEE; 2017, p. 1–6.
[7] Mirz M, Vogel S, Monti A. First interconnection test of the nodes in pan-european simulation platform. RESERVE Library; 2017.
[8] Braun W, Casella F, Bachmann B, et al. Solving large-scale modelica models: new approaches and experimental results using openmodelica. In: 12 international modelica conference. Linkoping University Electronic Press; 2017, p. 557–63.
[9] Guironnet A, Saugier M, Petitrenaud S, Xavier F, Panciatici P. Towards an open-source solution using modelica for time-domain simulation of power systems. In: 2018 IEEE PES innovative smart grid technologies conference Europe. IEEE; 2018, p. 1–6.
[10] Casella F, Leva A, Bartolini A. Simulation of large grids in openmodelica: reflections and perspectives. In: Proceedings of the 12th international modelica conference, vol. 132. Linköping University Electronic Press; 2017, p. 227–33.
[11] Baudette M, Castro M, Rabuzin T, Lavenius J, Bogodorova T, Vanfretti L. OpenIPSL: Open-Instance power system library—Update 1.5 to "iTesla Power Systems Library (iPSL): A modelica library for phasor time-domain simulations". SoftwareX 2018;7:34–6.
[12] Martí AT, Jatskevich J. Transient stability analysis using shifted frequency analysis (SFA). In: 2018 power systems computation conference. IEEE; 2018, p. 1–7.
[13] Demiray T, Andersson G, Busarello L. Evaluation study for the simulation of power system transients using dynamic phasor models. In: Transmission and distribution conference and exposition: Latin America, 2008 IEEE/PES. IEEE; 2008, p. 1–6.
[14] Martí JR, Dommel HW, Bonatto BD, Barrete AF. Shifted frequency analysis (SFA) concepts for EMTP modelling and simulation of power system dynamics. In: Power systems computation conference. IEEE; 2014, p. 1–8.
[15] Strunz K, Shintaku R, Gao F. Frequency-adaptive network modeling for integrative simulation of natural and envelope waveforms in power systems and circuits. IEEE Trans Circuits Syst I Regul Pap 2006;53(12):2788–803.
[16] Suárez A. Analysis and design of autonomous microwave circuits, vol. 190. John Wiley & Sons; 2009.
[17] Proakis JG. Digital communications. New York: McGraw-Hill; 1995.
[18] Energy management system application program interface (EMS-API) - Part 301: Common information model (CIM) base. International Electrotechnical Commission; 2016.
[19] FEIN Aachen e. V., VILLAS framework. http://www.fein-aachen.org/projects/villas-framework. [Accessed 23 November 2018].
[20] Guennebaud G, Jacob B, et al. Eigen v3. 2010, http://eigen.tuxfamily.org.
[21] Hindmarsh AC, Brown PN, Grant KE, Lee SL, Serban R, Shumaker DE, Woodward CS. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. ACM Trans Math Softw 2005;31(3):363–96.
[22] Razik L, Mirz M, Knibbe D, Lankes S, Monti A. Automated deserializer generation from CIM ontologies: CIM++ - an easy-to-use and automated adaptable open-source library for object deserialization in C++ from documents based on user-specified UML models following the Common Information Model (CIM) standards for the energy sector. Comput Sci Res Dev 2018;33(1–2):93–103.
[23] FEIN Aachen e. V., DPsim. https://www.fein-aachen.org/projects/dpsim/. [Accessed 23 November 2018].
[24] Walther M, Waurich V, Schubert C, Gubsch I. Equation based parallelization of modelica models. In: Proceedings of the 10th international modelica conference. p. 1213–20.
[25] Watson N, Arrillaga J. Power systems electromagnetic transients simulation. IET; 2003.
[26] Red hat enterprise Linux for Real Time 7: Tuning Guide; 2018. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_for_real_time/7/pdf/tuning_guide/Red_Hat_Enterprise_Linux_for_Real_Time-7-Tuning_Guide-en-US.pdf.
[27] Zhang P, Marti JR, Dommel HW. Synchronous machine modeling based on shifted frequency analysis. IEEE Trans Power Syst 2007;22(3):1139–47.
[28] Kundur P, Balu NJ, Lauby MG. Power system stability and control, vol. 7. New York: McGraw-hill; 1994.
[29] RESERVE. http://www.re-serve.eu/. [Accessed 23 November 2018].