Hybrid Power Systems Simulation and Optimization Utilizing SSP and FMI

Dai Araki[1] Magnus Sandell[2]

[1]Toshiba Digital Solutions Corporation, Japan, dai.araki@toshiba.co.jp

[2]Toshiba Europe Ltd., UK, Magnus.Sandell@toshiba-bril.com

Abstract

Collaborative model-based development of the hybrid power system often requires large-scale co-simulation and system parameter optimization. In this study, we investigate an architecture for parallel processing simulation of SSP (System Structure and Parametrization) and FMI (Functional Mock-up Interface), which enables high-speed computation by multi-core distribution. We combine Bayesian optimization and co-simulation, then we build a collaborative development platform for hybrid power systems design. We report performance experiments using hybrid electric vehicle simulation model published by JAMBE (Japan Automotive Model-Based Engineering center).

Keywords: Model exchange, FMI, SSP, Distributed co-simulation

1 Introduction

Hybrid power systems are a combination of power generation and energy storage systems (batteries and fuel cells). As shown in Figure 1, a hybrid power system consists of systems for both power supply and power demand stakeholders, and each stakeholder has its own methods and tools for system design and analysis. Hybrid electric vehicles (HEVs) also consist of various components such as engines, electric motors, DC-DC converters, DC-AC inverters, and batteries, and OEMs and suppliers work together for system design and analysis.

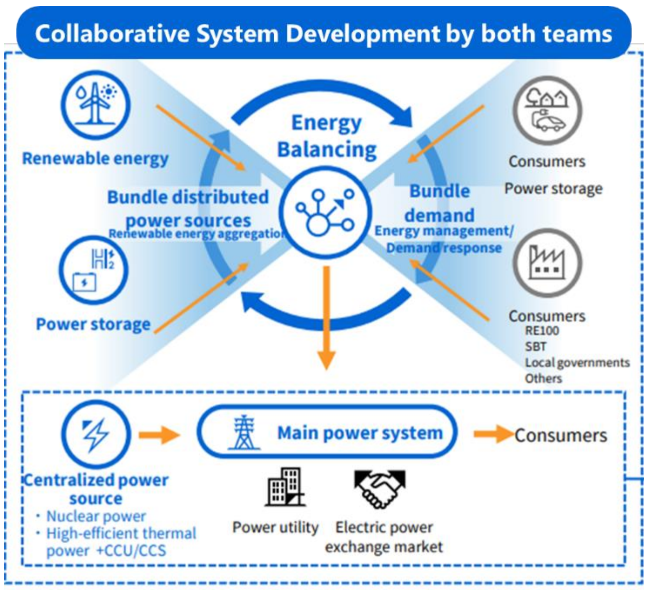

Collaborative System Development by both teams
Renewable energy
Energy Balancing
Consumers
Power storage
Bundle distributed power sources
Renewable energy aggregation
Bundle demand
Energy management/ Demand response
Power storage
Consumers
RE100
SBT
Local governments
Others
Centralized power source
・Nuclear power
・High-efficient thermal power +CCU/CCS
Main power system
Consumers
Power utility
Electric power exchange market

Figure 1. DX in Hybrid Power Supply Systems Design.

This means that system designing should consider different perspectives, which requires the use of several different tools and methodologies. However, connecting tools and exchanging information between different teams often leads to inefficiencies in system development. Since connecting tools from different vendors is not guaranteed to work and not supported by each vendor, it is necessary for the tool users to build and maintain their own environment for connecting tools.

For these reasons, interoperability standards for model exchanging between tools are important. FMI (Functional Mock-up Interface) and SSP (System Structure and Parametrization) are standardized by Modelica Association to establish interoperability between tools for model exchanging at various levels of abstraction.

SmartSE project of prostep ivip Association has published "SmartSE Recommendation" (SmartSE 2023) for simulation-based system design and decision making in collaborative development between multiple design teams across companies based on the utilization of FMI and SSP standard.

1.1 FMI (Functional Mock-up Interface)

FMI (Functional Mock-up Interface) is common interface and file format that allows simulation models to be passed between different tools. FMI standard is available on the official website (ref. FMI website). There are many tools over hundreds that support the FMI standard, and models can be exchanged between these tools.

FMU (Functional Mock-up Unit), which is zipped compressed file format, contains modelDescription.xml file in XML document format and a library file in binary format (DLL on Windows systems, SO on Linux systems) that implements the model's definition expressions or solver functions.

The format of modelDescription.xml is defined by the standard, and it contains information such as the names, types, and other attributes of the input and output signals of the model stored in the FMU, and a list of parameters that can be set and changed from outside the FMU.

There are two types of FMI standards "Model Exchange" interface and "Co-simulation" interface. Model Exchange FMU contains only the model equations.

Co-simulation FMU includes both model equations and solver.

Model Exchange interface assumes that the model equations in multiple FMUs are aggregated and computed by a single solver. This makes it difficult to speed up the calculation by distributed parallelization, and it is difficult to guarantee consistency between the results of a single solver and those of distributed parallel computation. The reason is that each individual Co-simulation FMU contains a model and a solver, so each FMU can be computed independently.

1.2 SSP (System Structure and Parametrization)

SSP (System Structure and Parametrization) is standard format for describing model and signal connection structures and other parameters necessary to conduct multi-domain co-simulation combining multiple FMUs. Specification of SSP can be obtained from the official SSP website (ref. SSP website).

Like FMU, SSP is a compressed file in zip format, and its interior consists of several sub-formats and its interior consists of several sub-formats. SSD (System Structure Description) is an XML file that describes the hierarchical structure, connection relationships, and functional structure of the entire FMU network.

1.3 Distributed co-simulation standards

In collaborative development through model exchange, there are many opportunities for large-scale simulation that combine model components created by multiple teams. Most commercial simulation tools calculate models sequentially, so when model parts are exchanged between companies or design teams using FMI and SSP, the simulation speed decreases as the number of FMUs for model parts increases.

Distributed co-simulation is expected to speed up large-scale simulations by running models in parallel. Distributed co-simulation may be run in a multi-core distribution within a single machine, on a set of machines interconnected via a local area network, or on globally distributed computers communicating via the Internet.

Typical distributed co-simulation standards include IEEE 1516 HLA and DSP.

IEEE 1516 HLA (High Level Architecture) (IEEE 1516) standardizes distributed co-simulation and standard interface for connecting multiple heterogeneous simulators. HLA defines controller called RTI (Run-time Infrastructure) which provides services such as data distribution and time synchronization among multiple connected simulators (called federates in HLA). Through this RTI, simulators are combined in a star-like network configuration for distributed simulation.

DCP (Distributed Co-simulation Protocol) (ref. DCP website) is a new protocol developed for the purpose of connecting real-time systems (HILS or prototype machines) and simulators. DCP protocol has two modes: real-time mode that connects a real-time system (actual device) and a simulator, and non-real-time mode that is used to connect virtual simulators. In both modes, the role of controller (equivalent to RTI in HLA standard) is limited compared to HLA. For data communication, each simulator node communicates directly with the other simulator node on a point-to-point basis. DCP protocol configures mesh-type network for distributed simulation. In the real-time mode of DCP protocol, each node runs using its own timer, so there is no explicit synchronization of time between nodes. In the non-real-time mode, the controller sends clock signal to each node, and the simulator on each node runs explicit synchronized to the clock signal.

1.4 Contributions of this paper

In this paper, we design a multi-core distributed simulator which performs high speed co-simulations that connect many model parts (FMUs). Next, we combine Bayesian optimization and distributed co-simulation to create a toolset that can automatically perform parameter optimization of hybrid power systems design.

2 Computation of distributed co-simulation

This section considers distributed co-simulation for SSP and FMI that can perform parallel computation with multi-core distribution.

The upper part of Figure 2 shows the sequence of execution of the calculations of each simulator connected to the distributed simulation, the time synchronization for distributed simulation, and the data distribution. The lower part of Figure 2 shows the case of simulation by sequential computation without distributed computation as a comparison.

In the upper part of Figure 2, each of the three simulators computes independently and in parallel until the logical time set as the synchronization timing (coupling point), at which point the simulators exchange output values with the other simulators. After that, the three simulators again perform calculations in parallel and exchange output values with the other simulators at the next synchronization timing. This process is repeated. This figure shows a simple case in which the communication step size of the three simulators is the same and does not vary, but there may be other cases in which the synchronization period varies, or in which each simulator has a different synchronization period or synchronization timing.

In contrast, in the case of sequential calculations as in the lower part of Figure 2, which do not involve distributed calculations, the three calculations are repeated in sequence using a set of simulators. Most commercial simulation tools are considered to be sequential.

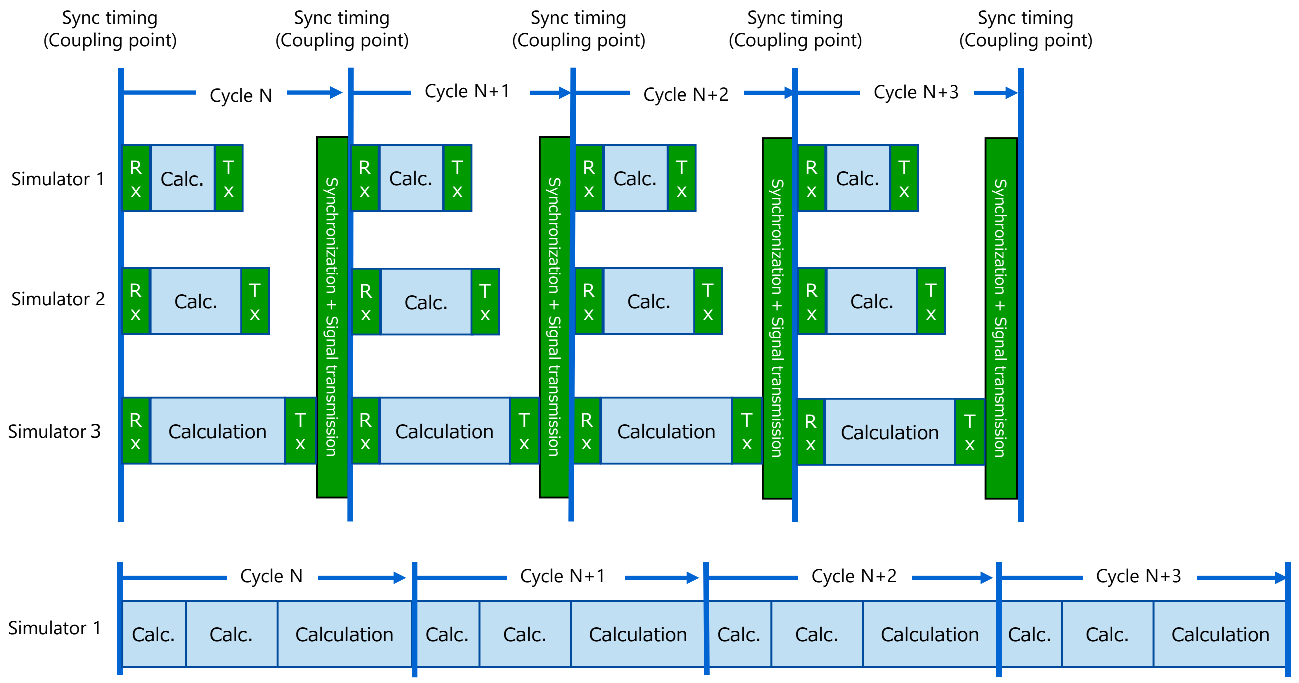


Figure 2. Comparison of computation between distributed co-simulation and sequential co-simulation.

Comparing distributed and sequential computation cases, the distributed simulator is expected to increase simulation speed compared to the sequential case. However, the simulation speed does not increase linearly with the degree of parallelism. In the case of distributed computation, there is the overhead of synchronization and signal transmission, so it is important how lightly this process can be made and how much the computation time can be shortened. It should also be assumed that the computational complexity of each model component is not uniform, so even if distributed computation is used, there is a tendency for the speed to decrease when there is a model with a large computational complexity.

In Figure 2, all simulators synchronize at the same period, but in general, each simulator may synchronize at a different period. DSP described in section 1.3 is suitable for distributed co-simulations that synchronize at a single cycle, but not for distributed simulations that synchronize at different cycles, because it synchronizes by distributing clock signals from the controller node to the slaves. On the other hand, IEEE 1516 HLA is suitable for both single-period and multi-period synchronization because it schedules slave nodes by supervising the global time in the controller node. Therefore, this paper adopts the IEEE 1516 HLA mechanism.

3 Distributed SSP-FMI co-simulation

3.1 Architecture of distributed simulator

This section shows the architecture of SSP-FMI simulator with distributed computation shown in Figure 3. SSP-FMI simulator was developed using distributed co-simulation platform VenetDCP from Toshiba Digital Solutions (ref. VeneDCP website).

"FMI Executable" loads and executes Co-simulation 2.0 interface FMU file. FMI standard defines the function APIs used to initialize the FMU and execute model computation. These functions are stored in DLL binary library file in the FMU. "FMI Executable" unzips FMU file, obtains information of input/output signals and parameters from modelDescription.xml file, and calls the function APIs in DLL binary library file to drive and simulate the imported FMU. In co-simulation where multiple FMUs are running, multiple FMI executables are launched for individual FMUs to achieve parallel distributed computation.

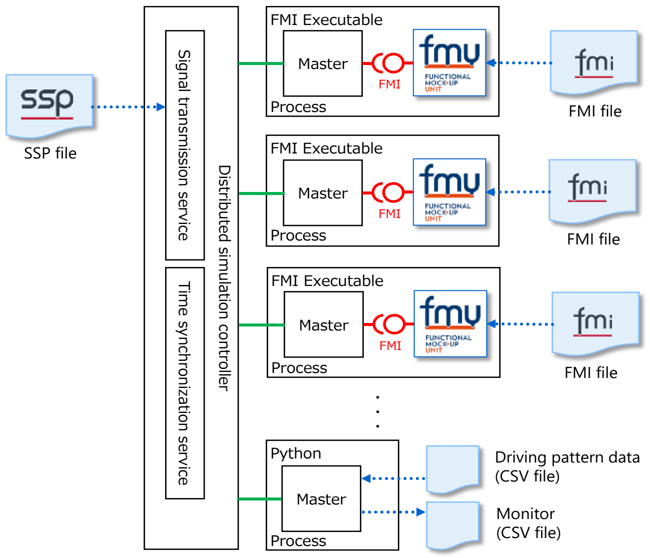
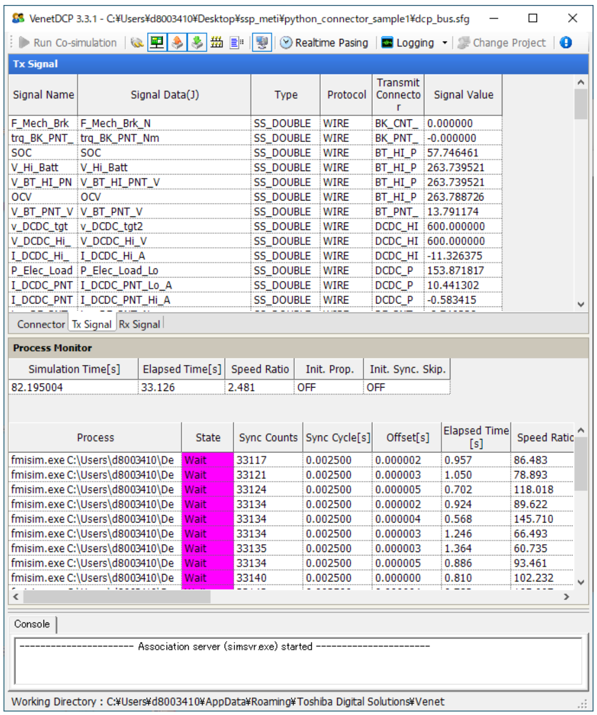

SSP file
Signal transmission service
Time synchronization service
Distributed simulation controller
FMI Executable
Master
FMI
Process
FMI file
Python
Driving pattern data (CSV file)
Monitor (CSV file)

Figure 3. Architecture of distributed SSP & FMI simulator.

Recording and monitoring of the test data time series signal input and output signal time series are performed using Python.


VenetDCP 3.3.1 - C:¥Users¥d8003410¥Desktop¥ssp_meti¥python_connector_sample1¥dcp_bus.sfg

Tx Signal

Signal Name	Signal Data(J)	Type	Protocol	Transmit Connector	Signal Value
F_Mech_Brk	F_Mech_Brk_N	SS_DOUBLE	WIRE	BK_CNT_	0.000000
trq_BK_PNT_	trq_BK_PNT_Nm	SS_DOUBLE	WIRE	BK_PNT_	-0.000000
SOC	SOC	SS_DOUBLE	WIRE	BT_HI_P	57.746461
V_Hi_Batt	V_Hi_Batt	SS_DOUBLE	WIRE	BT_HI_P	263.739521
V_BT_HI_PN	V_BT_HI_PNT_V	SS_DOUBLE	WIRE	BT_HI_P	263.739521
OCV	OCV	SS_DOUBLE	WIRE	BT_HI_P	263.788726
V_BT_PNT_V	V_BT_PNT_V	SS_DOUBLE	WIRE	BT_PNT_	13.791174
v_DCDC_tgt	v_DCDC_tgt2	SS_DOUBLE	WIRE	DCDC_HI	600.000000
V_DCDC_Hi_	V_DCDC_Hi_V	SS_DOUBLE	WIRE	DCDC_HI	600.000000
I_DCDC_Hi_	I_DCDC_Hi_A	SS_DOUBLE	WIRE	DCDC_HI	-11.326375
P_Elec_Load	P_Elec_Load_Lo	SS_DOUBLE	WIRE	DCDC_P	153.871817
I_DCDC_PNT	I_DCDC_PNT_Lo_A	SS_DOUBLE	WIRE	DCDC_P	10.441302
I_DCDC_PNT	I_DCDC_PNT_Hi_A	SS_DOUBLE	WIRE	DCDC_P	-0.583415

Connector | Tx Signal | Rx Signal

Process Monitor

Simulation Time[s]	Elapsed Time[s]	Speed Ratio	Init. Prop.	Init. Sync. Skip.
82.195004	33.126	2.481	OFF	OFF

Process	State	Sync Counts	Sync Cycle[s]	Offset[s]	Elapsed Time [s]	Speed Ratio
fmisim.exe C:\Users\d8003410\De	Wait	33117	0.002500	0.000002	0.957	86.483
fmisim.exe C:\Users\d8003410\De	Wait	33121	0.002500	0.000003	1.050	78.893
fmisim.exe C:\Users\d8003410\De	Wait	33124	0.002500	0.000005	0.702	118.018
fmisim.exe C:\Users\d8003410\De	Wait	33134	0.002500	0.000002	0.924	89.622
fmisim.exe C:\Users\d8003410\De	Wait	33134	0.002500	0.000004	0.568	145.710
fmisim.exe C:\Users\d8003410\De	Wait	33134	0.002500	0.000003	1.246	66.493
fmisim.exe C:\Users\d8003410\De	Wait	33135	0.002500	0.000003	1.364	60.735
fmisim.exe C:\Users\d8003410\De	Wait	33134	0.002500	0.000005	0.886	93.461
fmisim.exe C:\Users\d8003410\De	Wait	33140	0.002500	0.000000	0.810	102.232

Console

```
--------------------- Association server (simsvr.exe) started ---------------------
```

Working Directory : C:¥Users¥d8003410¥AppData¥Roaming¥Toshiba Digital Solutions¥Venet

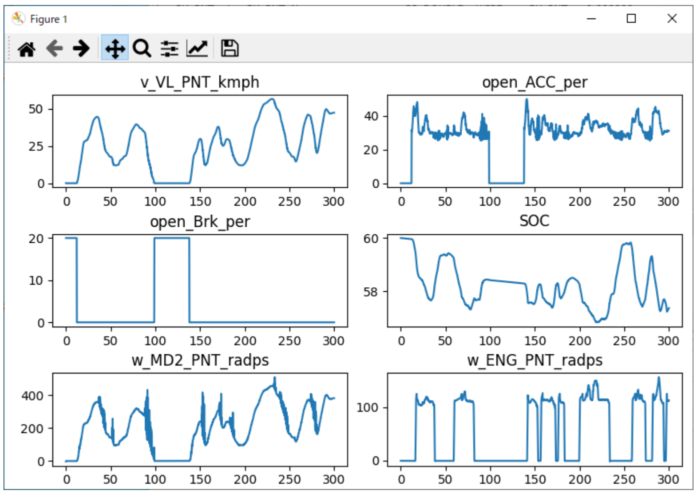


Figure 4. Example of SSP-FMI Simulation (series parallel hybrid vehicle model).

"Distributed simulation controller" provides the signal transmission service and the time synchronization service between FMI Executables and Python.

This mechanism corresponds to the RTI (Run-time Infrastructure) of the IEEE1516 HLA. The signal exchange service also reads SSP file and sets up the signal connection relationship between FMUs and parameters.

Since "FMI Executable," "Python," and "Distributed simulation controller" are independent processes, each process will be distributed across multiple CPU cores when run on a multi-core CPU machine. The number of processors and CPU core allocation can be changed using the processor affinity option in Microsoft Windows. Processor Affinity, also called CPU pinning, allows the user to assign a process to use only a few cores.

Inter-process communication between "FMI Executable," "Python," and "Distributed simulation controller" uses shared memory between processes on the same machine and TCP communication between processes on different machines.

Figure 4 shows the execution screen of the SSP-FMI simulator, with the "Distributed simulation controller" screen on the left and the simulation output signal time series on the right, plotted as a graph using Python's Matplotlib library (ref. Matplotlib website).

3.2 Performance evaluation using hybrid vehicle simulation

This section reports the performance evaluation of distributed SSP-FMI simulation by using series parallel hybrid electric vehicle (HEV) simulation model shown in

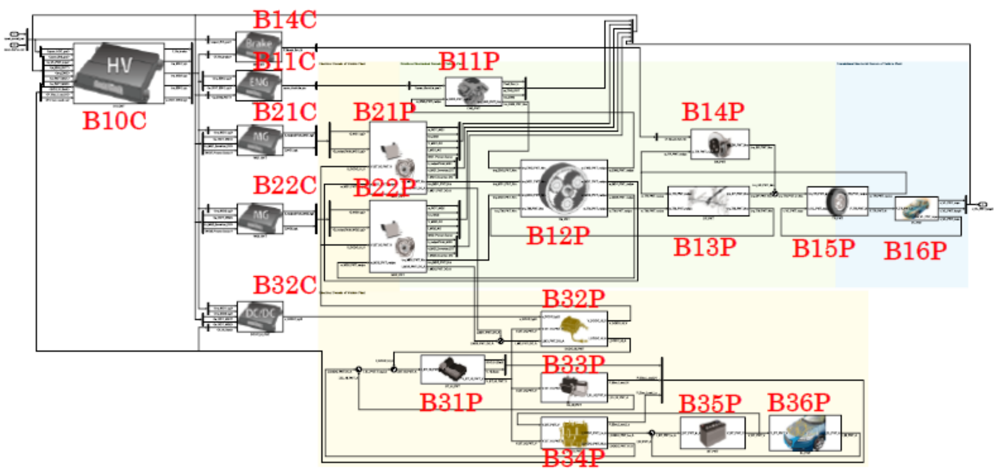


Figure 5. Series parallel hybrid electric vehicle model (ref. JAMBE HEV model).

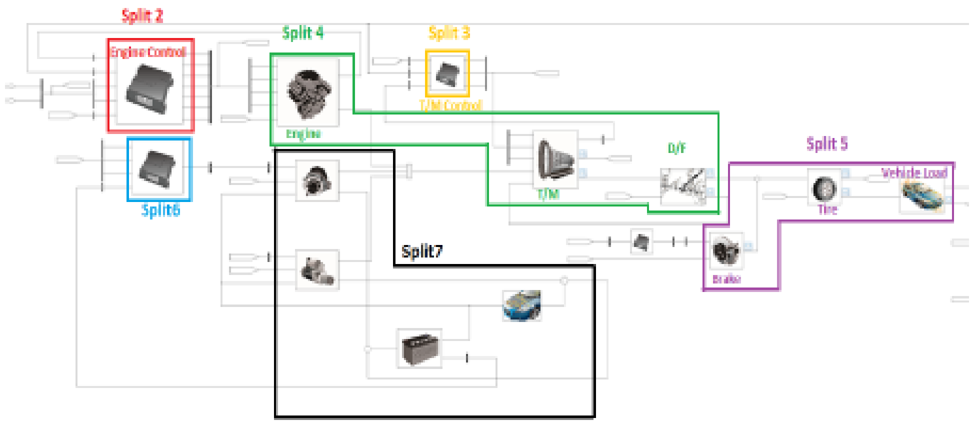


Figure 6. CVT vehicle simulation model (ref. JAMBE CVT model)

Figure 5 and continuously variable transmission (CVT) engine vehicle simulation model shown in Figure 6. Both simulation model is published by JAMBE (Japan Automotive Model-Based Engineering center).

The original model was built in MathWorks Simulink. We divided Simulink model and exported in FMI format. The number of FMI files for the hybrid vehicle model is 21 files divided by the block units of BXXX in Figure 5, and the CVT vehicle model is 7 files divided by the block units of color marks in Figure 6.

Each simulation was performed with the input of 1800 second standard driving pattern of WLTC (Worldwide harmonized Light vehicles Test Cycles) class 3b developed by UNECE (United Nations Economic Commission for Europe). Both simulations were run with a sampling time of 2.5 millisecond which is the same as the original JAMBE model.

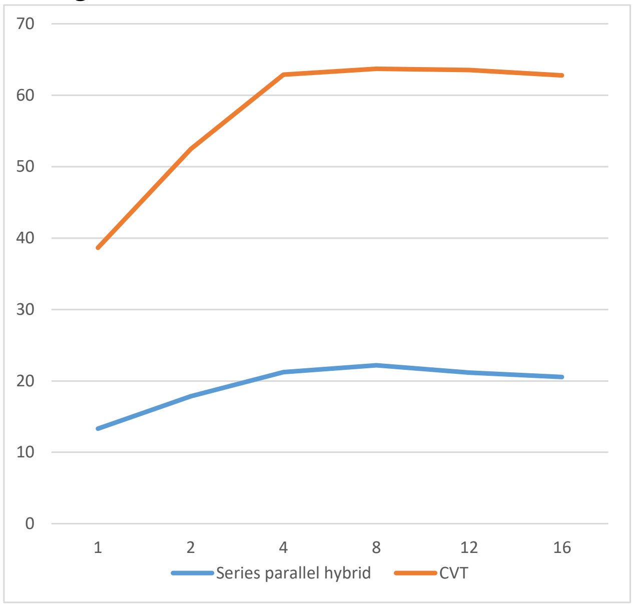


Figure 7. Number of CPU cores (threads) and RTF (Core i7-10870H)

We measured the RTF (real-time factor) of simulation speed using two different machines. The first machine equips Intel Core i7-10870H processor with 8 cores and 16 threads. The second machine equips Intel Core i7-8700 processor with 6 cores and 12 threads.

We used Windows 10 as the OS. The SSP-FMI simulator we developed allows the user to select the number of CPU cores (threads) used in the calculation using the processor affinity option of Windows, and we compared the simulation speed when using a single CPU and when using multiple CPU cores (threads).

Figure 7 shows the measurement results on the machine with Intel Core i7-10870H processor. The vertical axis represents RTF. The horizontal axis is the number of CPU cores (threads) used. The upper measurement is for a CVT vehicle simulation with 7 FMUs, and the lower measurement is for a hybrid vehicle simulation with 21 FMUs. Figure 8 shows the measurement results using the machine with Intel Core i7-8700 processor, and the notation is the same as in Figure 7.

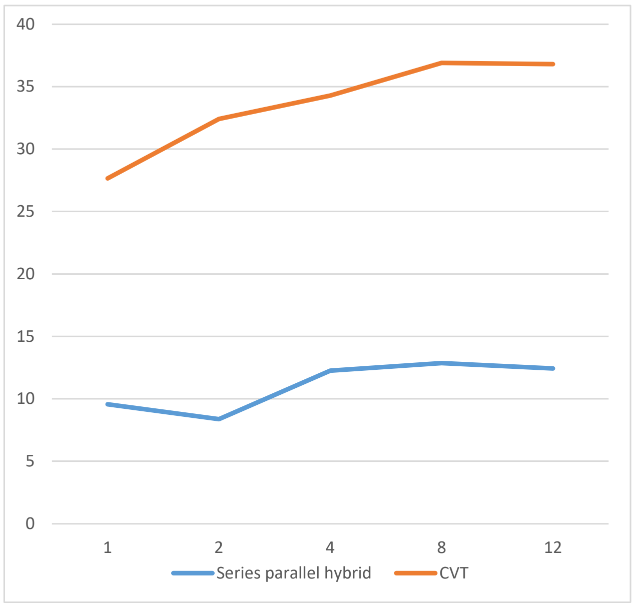


Figure 8. Number of CPU cores (threads) and RTF (Core i7-8700)

The measurement results show that increasing the number of CPUs used in a distributed calculation can increase the speed by up to a factor of two compared to a calculation using a single CPU. However, it was also found that speed increases only up to 8 CPU cores (threads) and that speed tends to decrease slightly when more than 8 CPU cores (threads) are used. This is thought to be because excessive use of processor CPU cores (threads) affects the execution of non-simulation processes and has the opposite effect on the performance of distributed computation.

Table 1. Comparison of RTF for each FMI model. (Series parallel hybrid vehicle)

Parts	Speed ratio
DCDC_HI_CNT	84.755
BK_CNT	79.029
BT_HI_PNT	74.171
BT_PNT	70.812
BK_PNT	66.966
DCDC_HI_PNT	63.065
DCDC_PNT	60.866
DF_PNT	58.56
Driver	53.997
EL_HI_PNT	54.604
EL_PNT	52.353
ENG_CNT	49.95
ENG_PNT	48.118
MD1_PNT	45.65
HV_CNT	26.651
MD2_PNT	42.494
MG1_CNT	45.802
MG2_CNT	42.333
TM_PNT	35.417
TR_PNT	38.759
VL_PNT	35.661
Total	21.785

Table 1 compares RTF of each individual component FMU in a hybrid vehicle simulation using 21 FMUs with 8 threads on a Core i7-10870H, measuring the execution time that "FMI Executable" was running. It can be seen that the simulation speed varies by a factor of several depending on the complexity of the model included in the FMU and the amount of calculation. The overall simulation RTF is 21.785 while the RTF of the single FMU of HV_CNT (hybrid control controller) is close to this at 26.651, indicating that the calculation of HV_CNT is the overall speed-determining factor. This shows that the overall simulation speed tends to be dragged down by the computationally intensive FMU, and that no further speed-up can be expected in the hybrid vehicle simulation even if the number of threads is increased to 8 or more.

By measuring the overall RTF and RTFs of each individual FMU in this manner, it is believed that it is possible to determine the optimal number of CPU cores (threads) that will provide the maximum simulation speed in the parallel distributed SSP-FMI simulator.

4 System parameter optimization using SSP-FMI co-simulation

4.1 Framework of system parameter optimizetion

This chapter describes an application of SSP-FMI co-simulation to system parameter optimization.

Collaborative and rapid development of hybrid power supply systems often requires various configuration and many control parameters to be optimized. It also requires to facilitate model exchange between partners while keeping confidentiality of model. We think distributed co-simulation utilizing model interoperability standard FMI and SSP and Bayesian optimization will be solve the problems.

Figure 9 illustrates the framework of collaborative development platform for hybrid power suppy systems. In this framework, model parts are collected from partners in FMI format and optimum parameter set can be searched by Bayesian optimizer and distributed co-simulation.

4.2 Optimization set-up

A flowchart for optimization functionality is shown in Figure 10, where an initial value is first generated (either randomly or by user input). The SSP file for the co-simulation is then modified, where the values of the parameters to be optimized over are changed. This allows the co-simulation to be run for the chosen parameter values.

Once the co-simulation has finished, the cost function to be optimized can be extracted from the output. Based on the parameter and output values, an optimization module can determine the next point to evaluate.

4.3 Optimization algorithms

Compared to most optimization problems, the cost function in co-simulation is often time- and resource-consuming. The system may comprise a large number of subsystems and even with parallel and distributed simulation, it can take a long time to evaluate its performance. Furthermore, the inner workings of the subsystems are often not known to the co-simulation master as they may originate from different vendors or developers. Hence, we can treat the cost function as a black box which is expensive to evaluate.

For this kind of optimization problem, a suitable algorithm is Bayesian optimization (Brochu09). This works by placing a Gaussian prior on the function and updating the posterior distribution based on the observed input and output values. This can be used to compute the best next value to evaluate, where criteria such as “probability of improvement” and “expected improvement” can be used. The advantage of Bayesian optimization is that it requires few evaluations of the cost function, as opposed to, e.g., evolutionary methods

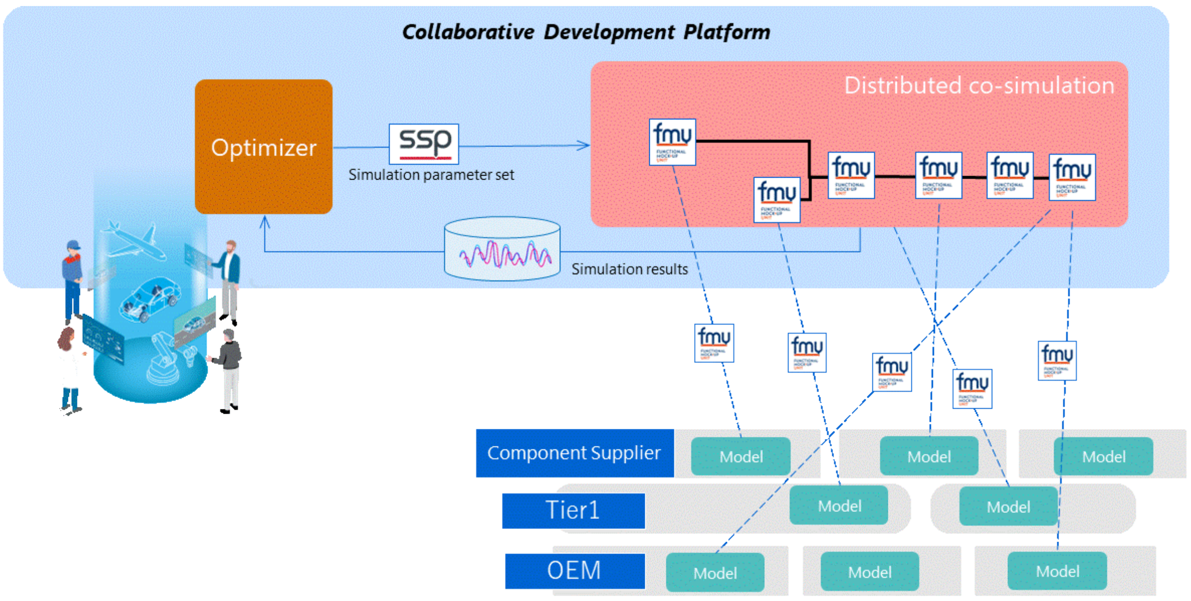


Figure 9. Framework of collaborative development platform.

(NSGA-II, genetic algorithms, particle swarms, etc.) which need many evaluations (Emmerich18).

It should be pointed out that it is possible to consider more than one cost function. Multi objective optimization is common in large and complex systems, where there are many metrics by which a system performance can be measured in. These are often conflicting, and an optimal trade-off is sought. Bayesian optimization can be extended to multi objective optimization, e.g., with efficient algorithms such as TSEMO (Bradford18).

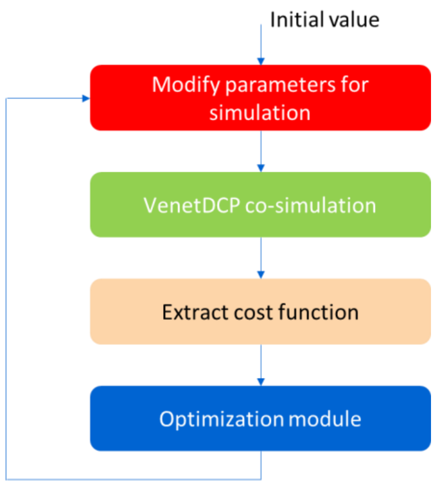


Figure 10. Optimization principle with co-simulation.

4.4 Optimization example

As an example of system parameter optimization, we considered the JAMBE HEV model. In particular we considered the role the clutch thresholds play in the propulsion. As shown below, the clutch helps activate electric only (Figure 11a) or hybrid-electric assist (Figure 11b). This is determined by, among other things, a threshold to open and a threshold to close it (measured in vehicle speed, km/h). These two thresholds were chosen as the system parameters to optimize over. The cost function was set as the fuel consumption during the WLTC class3b test drive cycle shown in Figure 12.

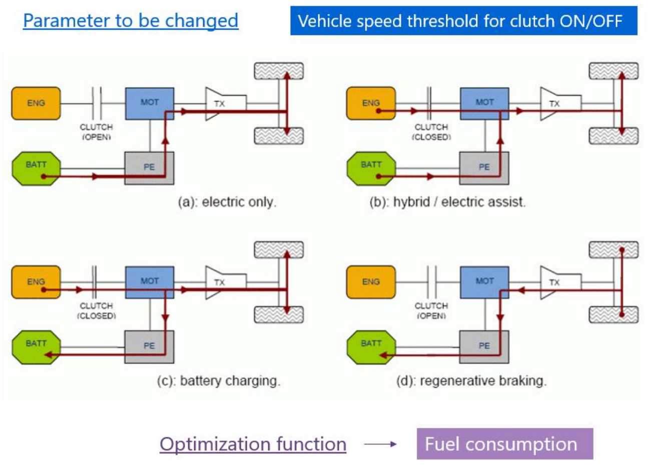


Figure 11. Scenario for the considered JAMBE model.

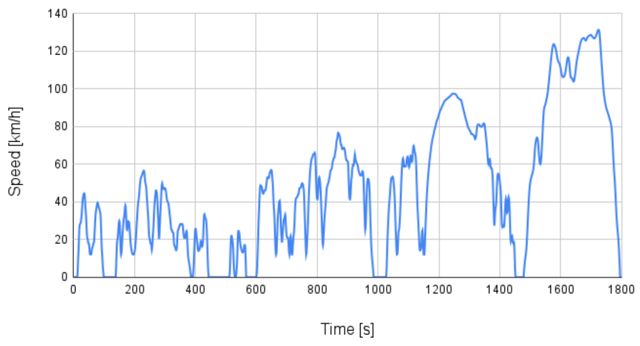


Figure 12. WLTC Class 3b test driving cycle consisting of Low, Medium, High and Extra High phases.

4.5 Optimization results

Using the MATLAB toolbox, Bayesian optimization of the clutch thresholds was implemented. Note that this allows the *Optimization module* in Figure 10 to output a suggested next value to evaluate in the next co-simulation, which was based on the "expected improvement" criterion. The cost function was extracted from the co-simulations as the fuel consumption after the 1800 seconds drive cycle. This is measured in km/l, so we are looking for its maximum.

The results are shown in Figure 13, where the fuel consumption (in km/l) is plotted against the two clutch thresholds parameter which defined in the JAMBE HEV model. HV_CNT_Clutch_ON_threshold_vel_kmph means that closing clutch is possible above this speed. HV_CNT_Clutch_OFF_threshold_vel_kmph means that opening clutch is prohibited below this speed.

To appreciate the optimization result, we also performed an exhaustive evaluation for all feasible parameter values. This involved around 400 co-simulations, whereas the optimization approach only used 10. Compared with the default parameter values which defined in the JAMBE HEV model (red circle), the optimized values (blue circle) showed a 2% improvement in fuel consumption. Although this is quite small, more gains can be had by considering other blocks, possibly in combination with each other.

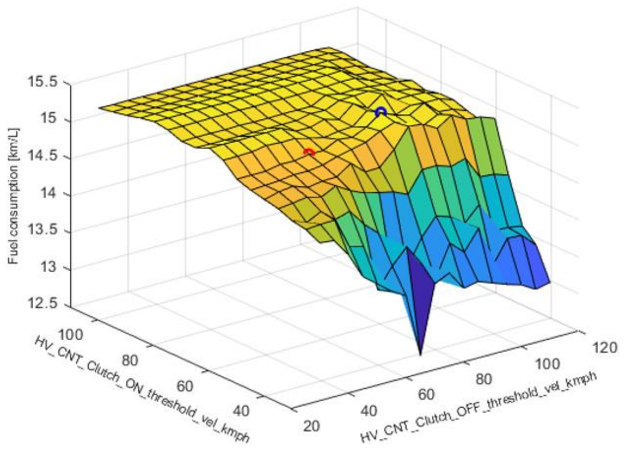


Figure 13. Optimization results for the co-simulated JAMBE model. The red and blue circles represent the default and optimized operational values, respectively.

5 Conclusion

FMI and SSP standards establish model exchange at various levels of abstraction and interoperability between tools. In this paper, we investigated a configuration of SSP-FMI simulator that enables parallel computation by multi-core distribution. We also examined application of SSP-FMI simulator to the system parameter optimization.

We are planning to apply collaborative development platform to the development of electric vehicles (integration of batteries, BMS, power trains, vehicle dynamics, etc), hybrid electric aircrafts (hydrogen fuel cells with batteries and high-performance electric motors) and offshore wind turbines (optimize efficiency and cost by wind & wave prediction).

We also plan to support co-simulation interface of FMI3.0 standard, whose official specification will be issued in 2022, and the newly introduced Scheduled Execution (SE) interface with the distributed parallel simulation in this paper.

References

SmartSE (2023). "SmartSE Recommendation V3 (Smart Systems Engineering Collaborative Simulation-Based Engineering Version 3.0), prostep ivip Association, January 2023. https://www.prostep.org/fileadmin/downloads/PSI_11_V3_SmartSE_Rec_and_Part_A-I.zip

FMI - Functional Mock-up Interface) https://fmi-standard.org

SSP - System Structure and Parameterization https://ssp-standard.org

IEEE1516. "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)", DOI： 10.1109/IEEESTD.2010.5553440 https://standards.ieee.org/ieee/1516/3744/

DCP - Distributed Co-Simulation Protocol https://dcp-standard.org/

VenetDCP - Distributed Co-Simulation Platform https://www.global.toshiba/ww/products-solutions/manufacturing-ict/venetdcp.html

Matplotlib - Visualization with Python https://matplotlib.org/

JAMBE HEV model. "Fuel efficiency models and manuals for series parallel hybrid 2 vehicles" https://www.jambe.jp/system/link.aspx?cid=200091

JAMBE CVT model. "Fuel efficiency model and manual for CVT" https://www.jambe.jp/system/link.aspx?cid=200101

Brochu, E., Cora, M., and de Freitas, N. (2009). "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning". Technical Report TR-2009-023, Department of Computer Science, University of British Columbia. arXiv:1012.2599.

Emmerich, M.T.M., Deutz, A.H. A tutorial on multi objective optimization: fundamentals and evolutionary methods. Nat Comput 17, 585–609 (2018). https://doi.org/10.1007/s11047-018-9685-y

Bradford, E., Schweidtmann, A.M. & Lapkin, A. "Efficient multi objective optimization employing Gaussian processes, spectral sampling and a genetic algorithm". J Glob Optim 71, 407–438 (2018). https://doi.org/10.1007/s10898-018-0609-2