A CO-SIMULATION APPROACH USING POWERFACTORY AND MATLAB/SIMULINK TO ENABLE VALIDATION OF DISTRIBUTED CONTROL CONCEPTS WITHIN FUTURE POWER SYSTEMS

K. Johnstone, S. M. Blair, M. H. Syed, A. Emhemed & G. M. Burt
University of Strathclyde – UK
kevin.johnstone@strath.ac.uk

T. I. Strasser
AIT Austrian Institute of Technology – Austria
thomas.strasser@ait.ac.at

ABSTRACT

In power network analysis it is increasingly desirable to implement controller and power systems models within different software environments. This stems from, among other things, an increasing influence of new and distributed control functions within smart grids and a growing influence of market operations. The computation time resulting from use of multiple simulation environments can cause significant delays and constrain the number of scenarios considered. This paper introduces and compares several techniques for integrating external control system models into power systems models for time domain simulations. In particular, a new technique is reported in this paper for PowerFactory-MATLAB/Simulink co-simulation interfaces, which offers a significant advantage over alternative methods in terms of the reduction in simulation runtimes and flexibility for the end user.

INTRODUCTION

Realistic, large-scale validation of future power system operation is expected to require increased collaboration between specialized researchers and advanced laboratory facilities. The test systems may be complex and control algorithms increasingly distributed, requiring the contribution from different organizations and the use of multiple software packages and programming languages during development and testing. In particular, where complex controller models are required to be developed and tested, it may be advantageous to implement the power system simulation and controller model in more capable and flexible software packages. However, it is not always possible for one simulation tool to provide all the requirements for the validation of complex power systems with intelligent controls [1], [2]. This increases the need for co-simulation techniques that provide a more powerful test environment. It is essential that the integration and co-simulation of the control and power systems developed in different packages is efficient, both in terms of reducing the complexity of interfacing the packages by the user, and does not adversely affect the simulation accuracy or cause prohibitively long simulation run times [3].

The ELECTRA IRP project [4] has developed novel approaches to control the frequency and voltage within a future conceptual distributed power system architecture, called the "Web of Cells" (WoC) [5]. While new power system control methods [6] are often collaboratively developed and prototyped within MATLAB/Simulink, their performance is being appraised with test power systems modelled within DIgSILENT PowerFactory. Co-simulation promises effective means of cooperation between smart grid controller developers and power systems laboratories conducting validation studies, however conventional approaches [7] for co-simulation have been found to be ineffective due to the complexity and abundance of controller models which must be integrated. As demonstrated in this paper, this would result in unacceptably long simulation run times.

Whereas the design and development of advanced control schemes for future power systems is covered by ELECTRA IRP, a more formalized and holistic validation approach covering power systems, control, and communications issues are necessary before bringing such new approaches to the field, the ERIGrid project [8] addresses this need by providing a formalized description of test scenarios as well as corresponding enhanced validation methods for smart grid systems (including co-simulation, hardware-in-the-loop and laboratory experiments).

This paper analyses several methods for PowerFactory-MATLAB/Simulink integration in terms of the complexity to the user and the impact on simulation run time, and presents a new coupling method. This method allows the engineer to run a compiled version of a controller developed in Simulink from within PowerFactory directly, significantly reducing the computational overhead and therefore the simulation run time – without adding significant complexity to the user. This provides a mechanism for comprehensive distributed control solutions to be tested and validated for a range of frequency and voltage disturbances.

TEST CASE AND MOTIVATION

A suitable base case for the development of co-simulation techniques was first identified as the example test system given within the DIgSILENT PowerFactory technical tutorial. This example integrates a simple voltage control loop into the plant model of a synchronous generator. While relatively simple, the chosen test model and network include representative characteristics of more complex models, such as multiple inputs and outputs, variable parameters, continuous states and multiple control model instances. The test case consists of two synchronous generators connected through a tie line and a single load connected at the same bus as one of the generators, as indicated in Figure 1.

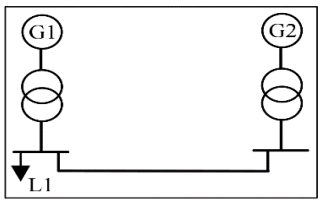

Figure 1: Simple test system for co-simulation method testing.

To illustrate the operation of the controller, an event is simulated which causes a step increase of 20% reactive power demand from the load in Figure 1, causing the generators to increase their reactive power outputs to compensate and support the network voltage. The voltage controller which is implemented is a standard "VCO type 16" excitation system, which exists as a black box model within the PowerFactory software (specifically, a ".ElmVco__16" object). Simulation times noted in this paper are averaged over five runs. Using a Python script to perform the simulation and calculated the run time, a 20 second simulation of the test system takes 0.2214 seconds to run with the "original" voltage controller installed. A Simulink excitation system model (based on the controller shown in Figure 2) is also available to use as a reference implementation, to allow the simulation results to be compared with the "original" excitation system.

The output signal of the voltage controller (i.e. generator 1 excitation voltage in per-unit) is used as the monitored variable in the testing and, in Figure 3, the simulation results are shown to be identical to the "original" model, however the simulation time is drastically increased – to 111.56 seconds, a factor of around 504.

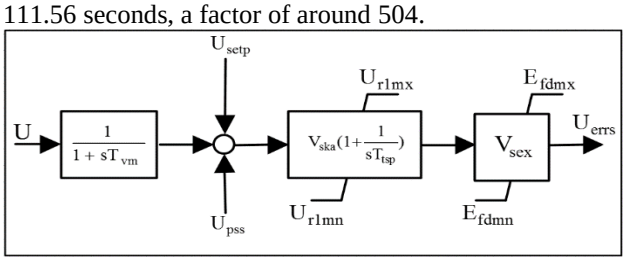

Figure 2: Block diagram of the voltage controller [7].

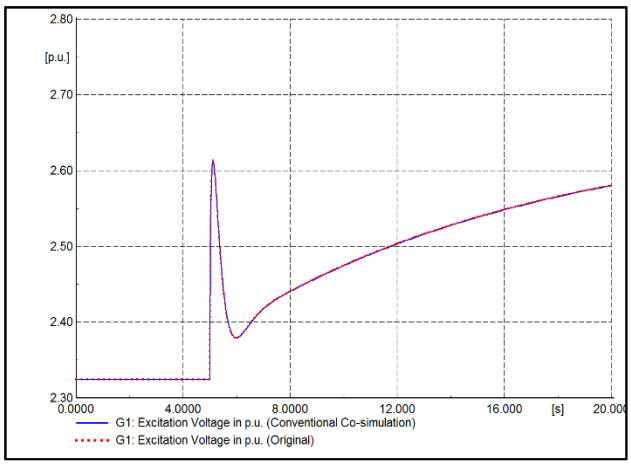

Figure 3: "original" vs. "conventional co-simulation" cases.

This is due to the fact that the conventional PowerFactory-MATLAB/Simulink method requires that the MATLAB program first be opened by PowerFactory, and then a new instance of the Simulink model is opened and simulated for each timestep of the simulation. This causes a severe slowing of the process of co-simulating the test network, rendering it impractical for more realistic full scale studies. A method which eliminates this bottleneck from the process, whilst not compromising the accuracy of the final results, is required. This example deals only with a very simple controller, with few inputs and a single output. Therefore, it can be expected that with more complex controller models and multiple instances, or the parallel integration of several different controllers, the simulation time overhead will be rendered infeasible, especially when the user is required to carry out many simulations and iterations of study cases, for example during sensitivity studies.

ALTERNATIVE METHODS AND RESULTS

A number of alternative methods for co-simulation have been considered and each evaluated with the test system.

Method 1: Native controller "re-implementation"

It is possible to re-implement the Simulink model in PowerFactory using the native control system design suite, known as DIgSILENT Simulation Language (DSL). This method is dependent on the user being fluent in the techniques required to build and implement control models using DSL. An advantage of this solution is that it is easier for the user to quickly make incremental changes to the controller model and rapidly test these changes in the simulation environment, when compared to the "conventional co-simulation" approach. This solution also does not require re-configuration of the MATLAB interface script whenever new variables or signals are created within the model – an extra step which adds to the implementation time and creates opportunities for errors to appear. The other key advantage of re-implementation of the model in DSL is that the user has full access to the time series data for all of the internal variables and signals of the controller during the simulation. The MATLAB approach does not allow this visibility of internal signals, instead treating the controller model as a black box, which makes the debugging of problems in the controller significantly more challenging. However, in projects where a large number of control functions have been implemented in Simulink, perhaps by an external partner, it may prove costly and infeasible to carry out the re-implementation process. For comparison, the simulation time for the case where the controller model is re-implemented in PowerFactory using DSL is 0.2394 seconds, which is slightly longer than the "original" case but still a factor of 466 faster than the "conventional co-simulation" case.

Method 2: Dynamic-Link Library (DLL) run externally

The Simulink Coder toolbox has functionality to automatically generate an efficient C code representation of a Simulink model [9]. This code can be used to compile a Dynamic-Link Library (DLL) which implements the original Simulink model. The "conventional co-simulation" approach (as described in the standard co-simulation approach from the PowerFactory User Manual) can be modified to interface with a DLL file, rather than with the Simulink model, i.e., PowerFactory's link with MATLAB, via a script file, calls a DLL version of the controller model instead of a Simulink version on each timestep. This method has the advantage of still giving the user access to the MATLAB scripting environment for programming flexibility and any data processing that they would like to perform, external to PowerFactory, during and after the simulation runtime. However, the method also requires the user to be familiar enough with MATLAB scripting to be able to perform some significant changes to the .m file and adapt the code so that MATLAB runs the DLL instead of the Simulink model in the simulation. However, this process could potentially be automated using a script, which mitigates this drawback.

The simulation time for this method was 9.97 seconds, which is a factor of 45 times slower than the "original" model implementation, but still 11.2 times faster than the "conventional co-simulation" approach. This is due to the fact that PowerFactory must still open the MATLAB program, which then performs each timestep by running the DLL file on each time step. As can be seen from Figure 4, further development of this technique is required to refine the accuracy of the results; however it is clear that the technique does provide marked benefits in terms of simulation run time.

Method 3: DLL run internally

The DLL file can also be referenced and run directly from within PowerFactory, as an ".ElmDsl" object (making reference to a block definition, where the user specifies the particular DLL file derived from an original model). Two different approaches have been compared: firstly with the DLL file derived from the re-implemented PowerFactory "DSL" version of the controller, and secondly with the DLL file derived from the Simulink version of the controller.

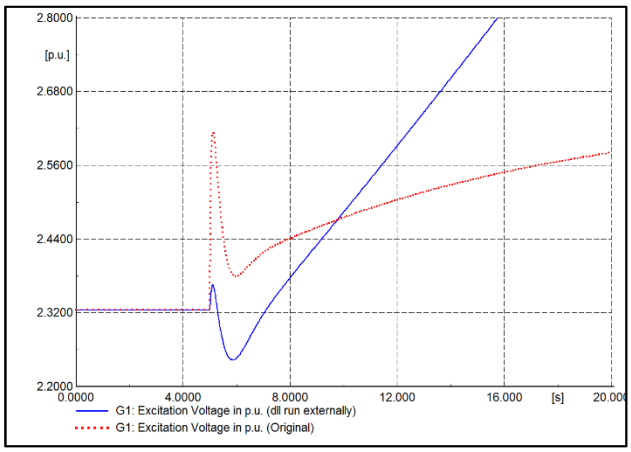

Figure 4: Results from "DLL run externally" test case.

Method 3a: DLL derived from DSL model

PowerFactory allows the user to automatically create a compiled version of any DSL model which has been built within the program itself. The model can be compiled in line with the instructions given in the User Manual, resulting in a DLL file which can be used in the subsequent time domain simulations. As shown in Figure 5, further work is required to refine the calculation of the initial conditions of the controller (because PowerFactory handles the calculation of initial conditions differently between internal and external models); however it can be seen that the controller model almost recovers to the nominal value during the event (Figure 6) and the results then are in agreement with the "original" controller in the steady state (Figure 7). Despite the DLL file being derived from the DSL model described in the "re-implement in DSL" case, these results show a marked improvement in the accuracy of the results over the previous case. The simulation time for this method

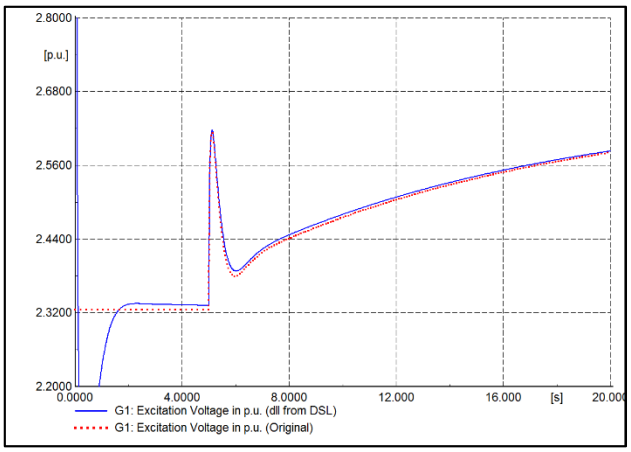

Figure 5: Results of "DLL from DSL" test case.

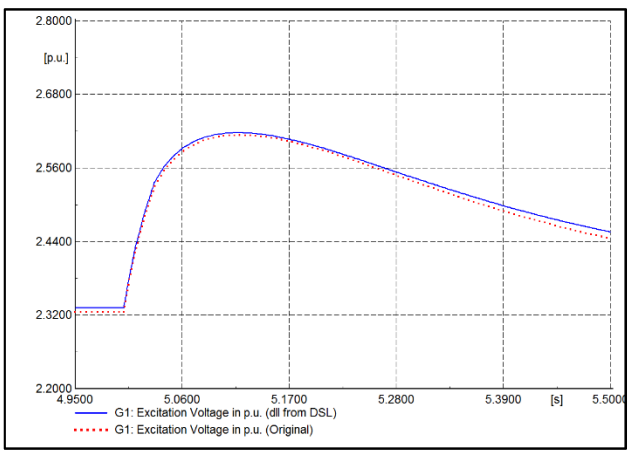

Figure 6: Deviation during event.

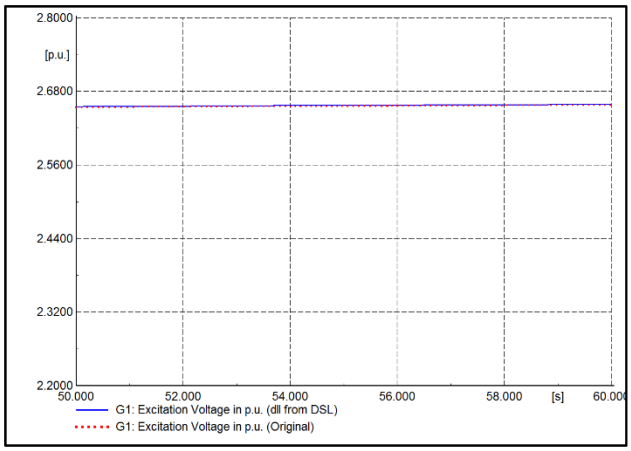

Figure 7: Results converge in steady state.

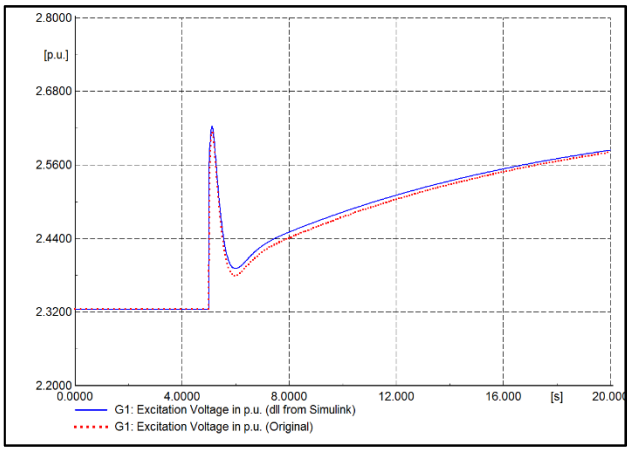

Figure 8: Results of ".dll from Simulink" test case.

was0.2484 seconds, which takes 1.12 times as long as the "original" model implementation, but is 449 times faster than the "conventional co-simulation" approach.

Method 3b: DLL derived from Simulink model

It is likely that a power system analyst using PowerFactory will not be immediately able or willing to re-implement a controller model that may have been developed by another user or users in the Simulink environment. It may not be necessary to monitor the internal signals of the controller model that has been provided and it is therefore appropriate to simply implement the finished controller – but avoiding the lengthy simulation times that are inherent in the "conventional co-simulation" approach from the User Manual. The method which achieves these goals includes the following general steps:

1. Use Simulink Coder to generate a C code implementation of the Simulink model, using the Embedded Real-time Target (ERT). The Simulink model must use the same timestep as the PowerFactory model (as for Method 2).
2. Create an empty PowerFactory DSL, with the same number of inputs and outputs as the Simulink model. Use PowerFactory to automatically generate C code representation of the DSL and a Visual Studio project (as for Method 3a).
3. Add the Simulink Coder C code to the generated Visual Studio project. Replace the contents of the *Initialise()* and *EvaluateEquations()* functions with calls to the appropriate Simulink Coder functions, and map the Simulink inputs and outputs to the PowerFactory variables.
4. Compile the project as a DLL, and access the DLL file from within PowerFactory.

Figure 8 to Figure 10 show the results for this study case in different levels of detail. It can be seen that the results match up well, however there is a slight deviation between the study case and the base case around the event, while the initial condition and the final value are entirely aligned with the base case. This error is likely to be eradicated with further development of the technique.

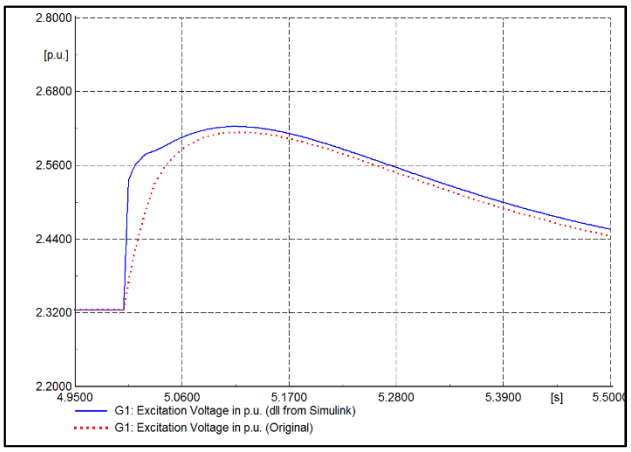

Figure 9: Deviation which occurs during event.

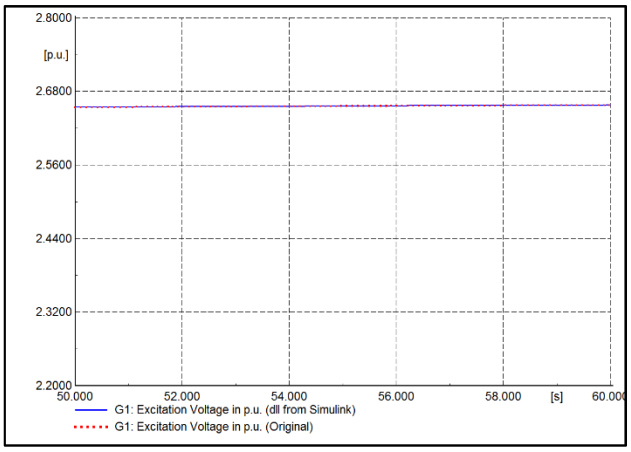

Figure 10: Results converge in steady state.

The simulation time for this study case was 0.3898 seconds, a factor of 1.76 times that of "original" case, but importantly still approximately 286 times faster than the "conventional co-simulation" case.

DISCUSSION

As the needs of power system analysts tend towards the use of a more diverse set of specialist controller system

modelling and network analysis tools, and reliance on the successful integration and co-simulation between these different tools, it is vitally important that usability and computational efficiency are ensured. This paper describes several different approaches for co-simulation. The packages PowerFactory and MATLAB/Simulink have been used for illustration in this paper and three alternative co-simulation methods have been compared to two benchmark scenarios. By producing a DLL version of the controller, the user can then choose to interface with MATLAB and have it run the DLL file, giving the flexibility provided by still being able to use the MATLAB scripting language for further data analysis. Alternatively, the MATLAB interface can be discarded entirely by using PowerFactory run the DLL file directly. This recommended method, which has been introduced in this paper, has proved successful by allowing an alternative technique for a power system control element designed in MATLAB/Simulink to be integrated into a DIgSILENT PowerFactory test power system, without the need to interface the two packages during the simulation. The method improves on the conventional co-simulation approach described in the PowerFactory User Manual by significantly reducing the time domain simulation runtimes, while maintaining accuracy of simulation results. These gains in simulation runtime have been achieved by having the entire simulation run within PowerFactory – using DLL calls where necessary – and therefore cuts out the lengthy process of running a new instance of the controller in Simulink in each timestep.

CONCLUSION

With relatively little work required by the user, a controller built originally in Simulink can be integrated in PowerFactory with virtually no penalty in terms of simulation times when compared with the case when the controller is re-implemented natively in PowerFactory. It is clear that eradicating the computationally intensive step of running a complete Simulink controller model on each timestep brings significant benefits, a lesson which might extend readily to other power system analysis software. While the test system is relatively simple in this case, it is expected that the benefits of the technique, in terms of simulation times and convenience, will be compounded in applications with large test systems and many controller models – a common feature of research on future distributed control systems and networks with large numbers of decentralised devices – yielding significant reductions in time and cost to users.

Acknowledgments

This work is partly supported by the European Community's Seventh Framework Program (FP7/2007-2013) under project “ELECTRA IRP” (Grant Agreement No. 609687) as well as by the European Community’s Horizon 2020 Program (H2020/2014-2020) under project “ERIGrid” (Grant Agreement No. 654113). Further information is provided at www.electrairp.eu and www.erigrid.eu.

REFERENCES

[1] A. Latif, et al. “An Alternative PowerFactory Matlab Coupling Approach”, International Symposium on Smart Electric Distribution Systems and Technologies (EDST), Vienna, Austria, Sept. 8-11, 2015.

[2] F. M. Gonzalez-Longatt, 2014, PowerFactory Applications for Power System Analysis, Chapter 15, Springer, 343-366.

[3] T. Strasser et al. “Towards Holistic Power Distribution System Validation and Testing – An Overview and Discussion of Different Possibilities”, in Cigré Session 46 – Set of Papers, pp. 1-10, Paris, France, Aug. 21-26, 2016.

[4] European Liaison on Electricity Committed Towards long-term Research Activity Integrated Research Programme, http://www.electrairp.eu/

[5] L. Martini, et al., “ELECTRA IRP approach to voltage and frequency control for future power systems with high DER penetrations”, in 23rd International Conference on Electricity Distribution CIRED, pp.1-4, Lyon, France, June 15-18, 2015.

[6] E. Guillo Sansano, et al. “Transitioning from centralized to distributed control: using SGAM to support a collaborative development of web of cells architecture for real time control”, CIRED Workshop, Helsinki, Finland, June 14-15, 2016.

[7] DIgSILENT PowerFactory 15, "User Manual", DIgSILENT GmbH, 2015.

[8] European Research Infrastructure supporting Smart Grid Systems Technology Development, Validation and Roll Out, http://www.erigrid.eu/

[9] A. J. Roscoe, S. M. Blair, G. M. Burt, “Benchmarking and optimisation of Simulink code using Real-Time Workshop and Embedded Coder for inverter and microgrid control applications”, 44th International Universities Power Engineering Conference (UPEC), Glasgow, UK, Sept 1-4, 2009.