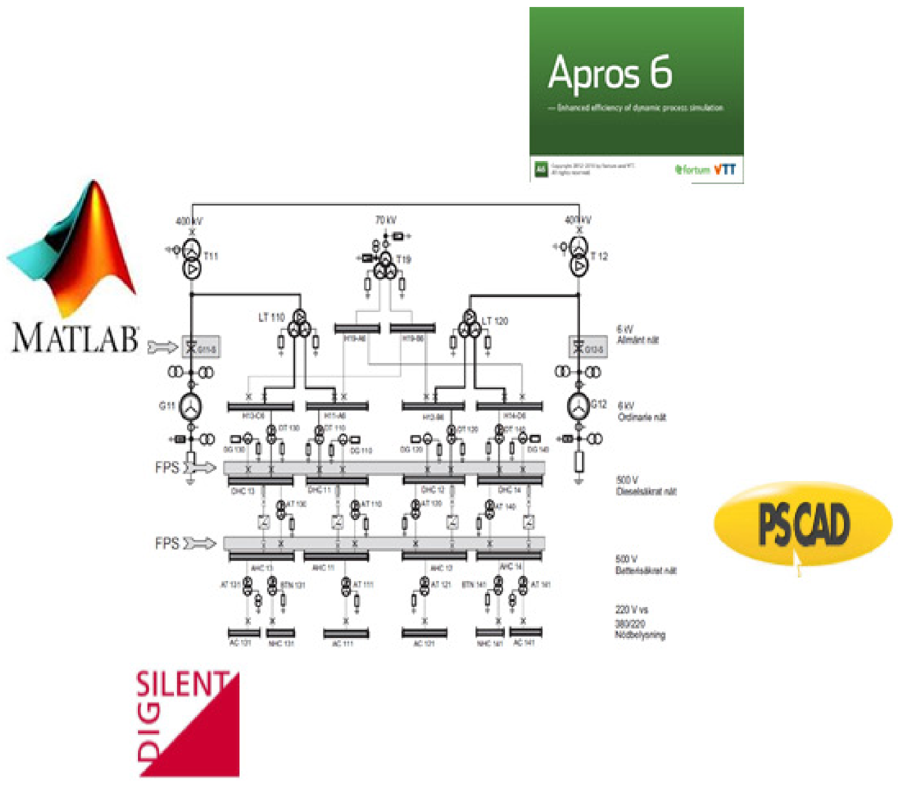


beyond the obvious

Description of the co-simulation platform for NPP

Poria Divshali | Seppo Hänninen | Pasi Laakso | Timo Korvola | Robert John Millar

VTT TECHNOLOGY 382

VTT TECHNOLOGY 382

Description of the co-simulation platform for NPP

Poria Divshali, Seppo Hänninen, Pasi Laakso & Timo Korvola

VTT Technical Research Centre of Finland, Ltd

Robert John Millar.

Aalto University, Finland

ISBN 978-951-38-8742-1

VTT Technology 382

ISSN-L 2242-1211
ISSN 2242-122X (Online)
DOI: 10.32040/2242-122X.2020.T382

JULKAISIJA – PUBLISHER

VTT
PL 1000
02044 VTT
Puh. 020 722 111
https://www.vtt.fi

VTT
P.O. Box 1000
FI-02044 VTT, Finland
Tel. +358 20 722 111
https://www.vttresearch.com

Preface

This work was carried out in the project “COSI - Co-simulation model for safety and security of electric systems in flexible environment of NPP”. The research partners of COSI project are VTT and Aalto University. The work supported mainly by SAFIR2022, The Finnish Research Programme on Nuclear Power Plant Safety 2019-2022, the GINO steering group, VTT and Aalto University. The work is motivated by the practical and theoretical problems studied in the project “ESSI- Electric systems and safety in Finnish NPP” of the previous SAFIR2018 programme.

The operating model of SAFIR2022 programme consists of a Management Board and four research area steering groups (SG) working under its supervision, as well as reference groups (RG) that are responsible for scientific and technical guidance of the projects. The administration of the programme is conducted by the administrative unit and Programme Director Jari Hämäläinen. COSI project belongs to “SG1 - Plant Safety and system approach safety” and “RG2 - Plant level analysis”.

A project-specific steering group has also been set up for the COSI project, which will, among other things, direct research and resolves confidential issues related to the project, as the project uses power plant self-generated electrical system, simulation models. The project-specific steering group consisting of the following members: Seppo Härmälä (Chairman, TVO), Jyrki Kykkänen (TVO), Ari Kanerva (Vice-chairman, Fortum), Juha Eriksson (Fortum), Juha Kemppainen (Fennovoima), Lauri Taivainen (Fennovoima), Monika Adsten (Energiforsk), Per Lamell (Forsmark/Vattenfall), Kim Wahlström (STUK), Samuli Hankivuo (STUK), Liisa Haarla (Fingrid), Minna Laasonen (Fingrid).

Thanks to Seppo Härmälä and Ari Kanerva for reviewing this report as official reviewer appointed by SAFIR 2022 RG2; and Per Lamell and GINO steering group to review the report and provide valuable comments and technical points. Thanks also the other member of the project-specific steering group for their suggestions to steer the project.

Espoo 22.12.2020
Authors

Contents

1. Introduction

A holistic model of a nuclear power plant (NPP), or generally any power plant, includes several domains with fundamentally different natures, such as thermomechanical loops, reactor physical models, automation and control models, an on-site electrical system, and an off-site transmission power system. From a conceptual point of view, this means that an NPP should be considered as a system-of-systems.

Each of these domains has a very different nature (continuous, discrete, stochastic, etc.), with different modelling assumptions. In this regard, significant efforts have been devoted to develop multi-domain simulation environments. Among them, Apros, a comprehensive software product for modelling and dynamic simulation of power plants, developed by VTT and Fortum [1], is one of the successful efforts, which has been used in more than 30 countries for numerous projects during the last 30 years. Although Apros is a powerful multi-domain simulation environment, especially with regard to the thermal process, nuclear reactor, and automation; its ability to model different dynamic events in power systems, e.g. asymmetrical faults, is somewhat limited.

In general, developing a multi-domain simulation environment, which supports all necessary domains, is not trivial to achieve due to the significant effort and expertise required. Models from different domains often need to be dealt with using a different time scale, a model of computation (MoC) and specialized solvers. Building a simulator capable of providing the appropriate environments, correct MoC, solvers and properly coordinating them internally, is expensive and maybe not worth the effort [2, 3]. Furthermore, domain-specific tools are usually equipped with validated component libraries for that domain and the correctness of models are verified by domain experts. Following these challenges, currently, there is no systematic way to study precisely the interactions of different domains in an NPP and to analyse the effect of these interactions on the safety of NPP.

However, analysis of operating experience and lessons learnt, which shall not be limited to domain-specific events, is an important input in the safety enhancements of an NPP. Especially, disturbances in electrical systems, both on-site and off-site, may influence the performance of the thermomechanical system, without proper reaction of the protection system, and lead to critical events endangering the safety. In this regard, loss of offsite power or loss of off-site power combined with emergency diesel generator common cause failure (station blackout) are not sufficient scenarios for the design basis. The plurality of relevant disturbances in electric systems is much larger. The recent changes in the electrical grid, such as the increased role of renewable energy sources, possible increased frequency of extreme weather conditions, and implementation of digital control systems for electric systems, mean that the previously thought design philosophy for electric systems of NPP may no longer be fully valid.

In these circumstances, the COSI project exploits the existing domain-specific simulation tools and develops a co-simulation platform based on the existing domain-specific tools. This co-simulation platform uses Apros to model thermal, reactor physical, and automation models. Then, in order to simulate the on-grid and off-grid electrical system and analyse their interaction, the co-simulation platform creates an interface for co-simulation of Apros with other power system tools, such as MATLAB/Simulink, PSCAD, and PowerFactory. In the first version, Simulink will be the power system simulator that is connected to the co-simulation platform. COSI Steering Group decided to study on-site electrical models from one of the Fortum NPPs, which was developed using MATLAB/Simulink. After analysing the advantages of the co-simulation platform, in this case, the project may develop further co-simulation platform to support other power system simulator, e.g. PSCAD.

A co-simulation model for the safety and security of electric systems in the flexible environment of NPP (COSI) project is part of SAFIR2022 programme for Nuclear safety. The project focuses on the electrical system of nuclear power plants and related grid effects. The focus of this project in WP1 has been to develop the abovementioned co-simulation platform. In the first year, the architecture of the platform was designed and reported in [4]. Here, the development of the first version of the co-simulation will be reported.

2. Co-simulation Implementation

In order to assess different systems using co-simulation, it is necessary to integrate the model of computation (MoC) behind a model or a simulator. The MoC represents the interactions between modules, components or phenomena and it is independent of the implementation technology (i.e. sequential or parallel) and language (i.e. Matlab, Python) [5].

The energy domain simulators often employ Dataflow MoC due to the fact that they derive mostly from sets of ordinary differential equations defining the state variables and the environmental factors of a system (e.g. steady-state simulations, electromagnetic transients or circuit simulations). However, ICT, market simulators and eventually control simulators often use the Discrete Event or Finite State Machine MoC. The discrete models react to events that occur at a given time instant and produce other events either at the same time instant or at some future time instant in chronological execution order. Combining discrete event and continuous simulation requires mixing different MoC, such as Discrete Events and Dataflow hierarchically. In these circumstances, the co-simulation architecture design must address the following issues:

- Data Exchange layout among simulators
- Data Exchange Intervals
- Time step handling
- Data exchange Protocol among simulators
- Initialising

A detailed description of the architecture design for the co-simulation platform has been reported in [4]. Figure 1 shows the final schematic of this architecture.

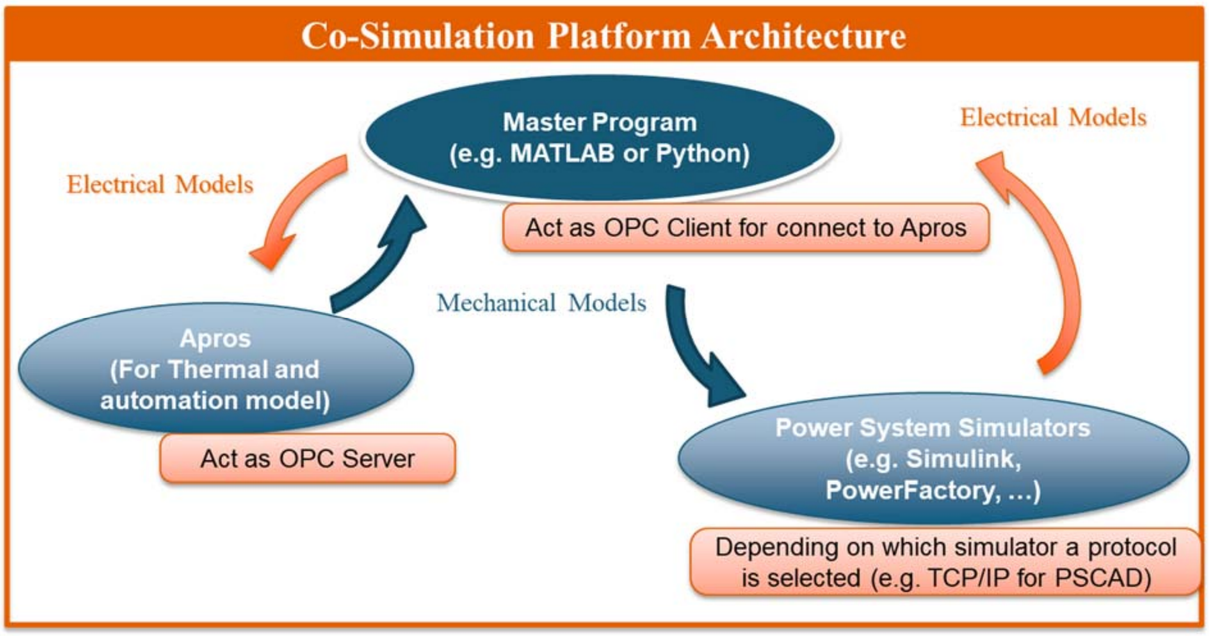

Co-Simulation Platform Architecture
Master Program (e.g. MATLAB or Python)
Act as OPC Client for connect to Apros
Electrical Models
Electrical Models
Mechanical Models
Apros (For Thermal and automation model)
Act as OPC Server
Power System Simulators (e.g. Simulink, PowerFactory, ...)
Depending on which simulator a protocol is selected (e.g. TCP/IP for PSCAD)

Figure 1. The proposed co-simulation architecture for the COSI project

According to this architecture, the co-simulation platform includes a Master Program, which can coordinate the co-simulation of different domains using domain-specific simulation tools. This Master Program connects to the Apros using Open Platform Communications (OPC) data connection. The connecting protocol could be different for other simulation tools that are connected to this co-simulation platform. For example, for connecting PSCAD, the FMI-Compliant Interface developed based on TCP/IP protocol developed in [5] can be used. In this section, the implementation of the chosen architecture will be described.

The main part of the co-simulation platform is the Master Program, which indeed is the implementation of the co-simulation architecture developed in [4]. The first version of the co-simulation platform develops the Master Program in MATLAB/m-file environment. However, it can be developed in any programing language that supports OPC data connection, e.g. Python. The detailed code of this Master Program in MATLAB is available in Appendix A and can be found as an open-source tool in [6]. The Master Program has the following main sections/functions.

- Set Co-simulation parameters
- Define Input Layout
- Create Data Structure and OPC HOST
- Initialising
- Co simulation loop
- Data exchange
- Progress report and plot results

The detailed code of these functions is given in Appendix A. The Master Code was developed in such a way that the user of the co-simulation platform does not need to change the code. However, the user needs to open Apros with administrator right, introduce the data layout and set the co-simulation parameters.

The co-simulation parameters include the co-simulation time, the time step of each simulator, the name of Simulink files that represent the electrical model (on-grid and off-grid), and the name of file and variable, which used in the Master Program to save the Steady-State data after initialising. The saving of initial data is accelerating the initialisation of the co-simulation system to be ready for simulation. Table I lists the name of the parameters that need to be set to configure the co-simulation.

Table I: The name of the parameters that need to be set to configure the co-simulation.

Variable Name	Description
CoSim.Ta	Simulation Time Step in Apros
CoSim.Ts	Simulation Time Step in Simulink
CoSim.SteadyTime	The time considers that electrical system reaches steady-state. Co-simulation start after this time.
CoSim.Tend	Stop time of co-simulation
CoSim.SimName	The name of the Simulink file includes electrical system model.
CoSim.IC_File	File Name that keeps the Initial Condition of Simulink
CoSim.IC	Variable Name that keeps the Initial Condition of Simulink

In addition to these parameters, the user needs to define the co-simulation layout, as an input file. The first version of the platform modelled Turbine Generator set and three different Pump-motor sets (BasicPump, MotorPump, and CommonPump). The Master Program just needs to have the name of the components that participate in the co-simulation and their initial input/output variables. Table II lists the variables that are assigned these inputs.

The sample code for Layout data input is in Appendix A2.

3. User Instructions

In order to co-simulate NPP using the first version of the Co-Simulation Platform, the user needs to perform the following steps:

Table II: The data layout inputs need to be modified in the co-simulation.

Parameter Name	Description
Layout.GenSet	A Cell array to keep the data of Turbine-Generator sets. Each raw keeps the data of one set. The array has 4 columns as follows: • Column 1, name [type: String]: Keep the Generator Name in Apros • Column 2, name [type: String]: Keep the Shaft Name in Apros • Column 3, Simulink Input (Pmech) [type: Float]: Initial mechanical power of Turbine (W) • Column 4 Simulink Output ([PE],[W]) [type: cell of Floats]: {Initial electrical active power (W), Initial rotation speed (RPM)}
Layout.BasicMotor	A Cell array to keep the data of Basic Motor-Pump sets. Each raw keeps the data of one set. The array has 3 columns as follows: • Column 1, name [type: String]: Keep the Pump Name in Apros • Column 2, Simulink Input (Tmech) [type: Float]: Initial mechanical Torque of Motor (KW) • Column 3 Simulink Output (W) [type: Float]: Initial rotation speed of motor (rad/sec) ω
Layout.MotorPump	A Cell array to keep the data of Motor-Pump (Motor Pump type) sets. Each raw keeps the data of one set. The array has 3 columns as follows: • Column 1, name [type: String]: Keep the Pump Name in Apros • Column 2, Simulink Input (Tmech) [type: Float]: Initial mechanical torque of Motor (N.m) • Column 3 Simulink Output (W) [type: Float]: Initial rotation speed of motor (rps)
Layout.ComPump	A Cell array to keep the data of Motor-Pump (Motor Common Pump) sets. Each raw keeps the data of one set. The array has 3 columns as follows: • Column 1, name [type: String]: Keep the Pump Name in Apros • Column 2, Simulink Input (Tmech) [type: Float]: Initial mechanical torque of Motor (N.m) Column 3 Simulink Output (W) [type: Float]: Initial rotation speed of motor (rps)

Apros Model

The Master Program is designed so that the user does not need to make many changes in the Apros model. After opening Apros with administrator right, the user needs to load the initial value from Apros and keep the name of the components that are participating in the co-simulation and their initial output. For example, the initial active power of the Shaft, or the output torque of the pumps (see Table II). In addition, in this implementation user needs to change the operation mode of Motor-pump (MC_CALCULATION_MODE)manually and put it in 3. The later version will change this mode automatically.

Besides, since the co-simulation platform runs the Apros model and Simulink simultaneously, the user can see all the results in Apros as well. Therefore, they can follow the simulation results (any variable) in the Apros Chart.

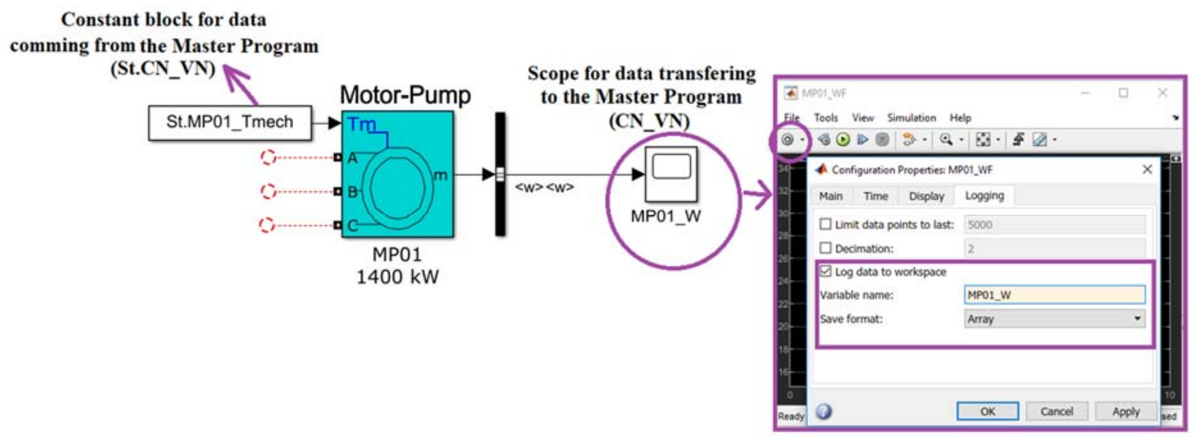
Simulink Model

The master program calls the Simulink model. The name is defined by the user as explained in Table I. In order to have input and output from Simulink, the user needs to modify the Simulink model so that the component uses the same name as in Apros. For example, the generator in Simulink that needs to be coupled to a turbine set in Apros needs to have the same name as the generator in Apros.

In addition, for all data transferring from Apros to Simulink (inputs), the user needs to put a constant block with the name of “St.CN_VN”. Also, for all outputs from Simulink that is intended to transfer through the Master Program to Apros, the user needs to put a scope, enable log data to workspace with an array format, and set the variable as “CN_VN”. Here, CN is the component name, according to Table II, and VN is the input/output variable name, according to Table II. Table III shows the exact structure of the input/output variable name for different co-simulation components, which need to be set by the user. Figure 2 shows the schematic of a motor-pump component, which illustrates how it looks like in Simulink.

Table III: Structure of an input/output variable name for the different co-simulation components in Simulink. The values in () are examples

Component type	Component name	Input variable name	Output variable name
Generator	CN (SG01)	St.CN_Pmeach (St.SG01-Pmech)	CN_PE (SG01_PE) CN_W (SG01_W)
Motor (co-simulating of basic pump)	CN (BP01)	St.CN_Tmeach (St.BP01-Pmech)	CN_WP (BP01_W)
Motor (co-simulating of MotorPump**)**	CN (MP01)	St.CN_Tmeach (St.MP01-Tmech)	CN_WF (MP01_W)
Motor (co-simulating of ComPump**)**	CN (CP01)	St.CN_Tmeach (St.CP01-Tmech)	CN_WF (CP01_W)


Constant block for data
comming from the Master Program
(St.CN_VN)
St.MP01_Tmech
Motor-Pump
Tm
A
B
C
m
MP01
1400 kW
<w> <w>
Scope for data transfering
to the Master Program
(CN_VN)
MP01_W
MP01_WF
File Tools View Simulation Help
Configuration Properties: MP01_WF
Main Time Display Logging
OK Cancel Apply

☐ Limit data points to last: 5000
☐ Decimation: 2
☑ Log data to workspace
Variable name: MP01_W
Save format: Array

Fig 2. The schematic of a motor (name: MP01) co-simulation component in Simulink with the required setting.

Master Program

As mentioned in Section II, the co-simulation platform runs both simulators, Apros and Simulink, and therefore the user does not need to run them separately. However, the user needs to determine which component should be involved in the co-simulation, and how long the co-simulation should last and what is the time step of each simulator. This information can be defined in variables according to the details explained in Tables I and II.

4. Proof of concept

In order to verify the co-simulation platform, small thermomechanical and electrical models have been used. The electrical model includes a Generator that through a unit transformer is connected to the high voltage grids and through an auxiliary transformer provides energy for three large motors. The generator coupled to a turbine shaft and motors coupled to the basic pump, motor pump, and common pump in Apros.

Fig. 3 shows the Simulink model of the test case, where the mechanical torque of the generator and motors comes from the Apros simulation (through the master program). It is important to mention that modelling the power system in MATLAB/Simulink gives the ability to model different types of dynamic events, such as single-phase faults, which are not possible when using Apros alone. Using the co-simulation to study the detailed dynamic behaviour of interaction of NPP and electrical system is the final goal of this project. However, the subject of this report is to demonstrate the principle of the co-simulation in a simple case-study. Fig. 4 depicts the Apros model, where the electrical power measurement and rotational speeds come from the Simulink model (through the master program).

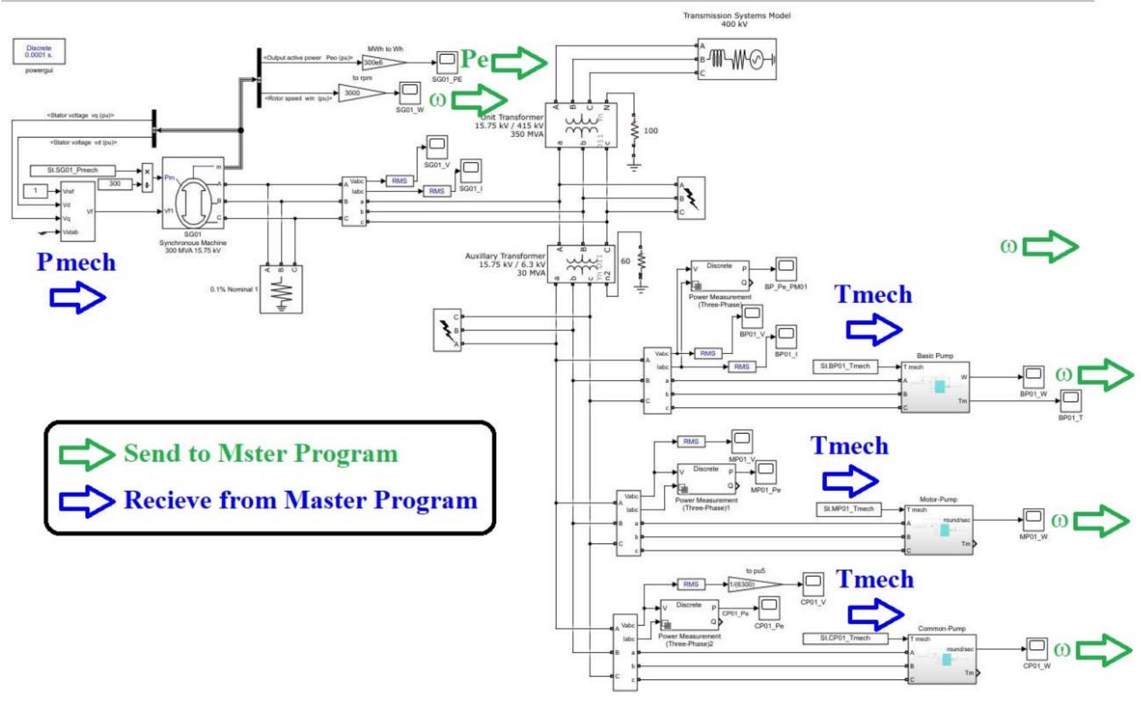

Pe
ω
Pmech
Tmech
ω
Tmech
ω
Tmech
ω
ω
Send to Mster Program
Recieve from Master Program
Transmission Systems Model 400 kV
Unit Transformer 15.75 kV / 415 kV 350 MVA
Auxiliary Transformer 15.75 kV / 6.3 kV 30 MVA
SG01 Synchronous Machine 300 MVA 15.75 kV
Basic Pump
Motor-Pump
Common-Pump

Figure 3. The Simulink model of the test case

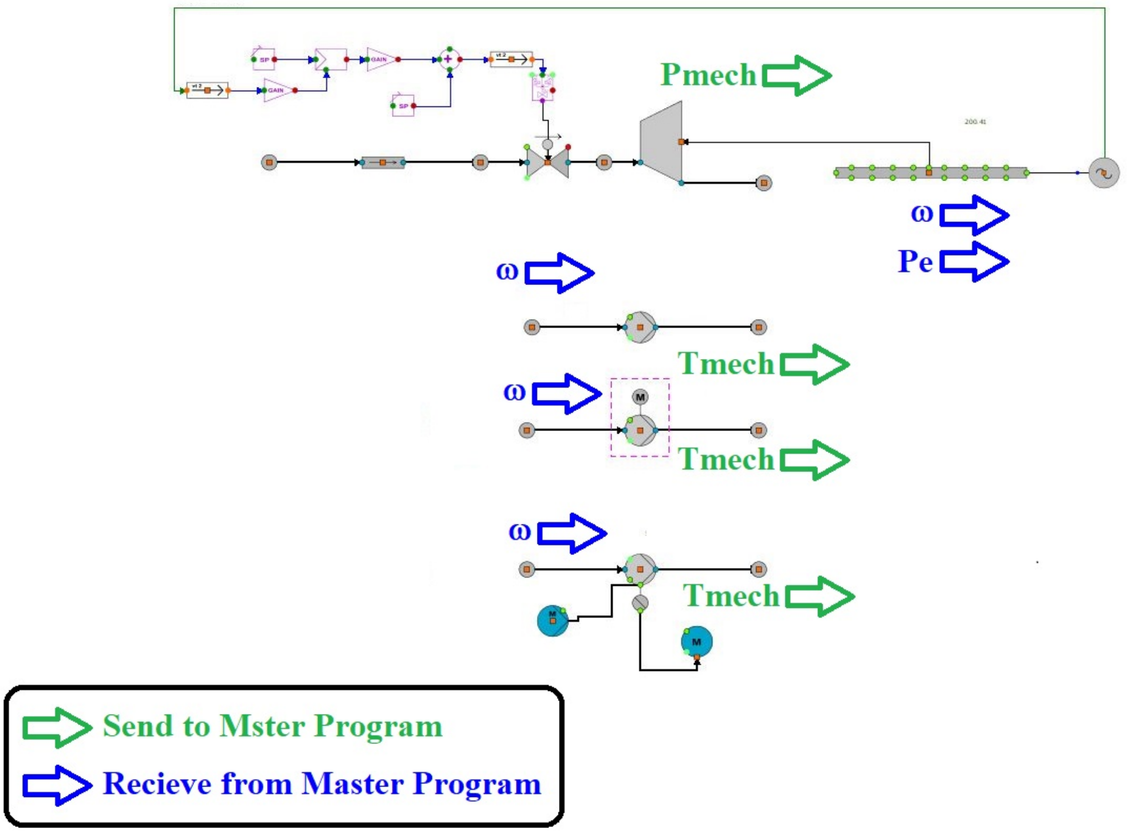


Figure 4. The Apros model of the test case

In order to show that the co-simulation platform is working, a two-phase fault is modelled at the motors' terminals. This fault between phases A and B, having 0.45 Ohm resistance and not grounded, happened in time 1.1 S, with the system reaching the steady-state condition before time zero. This fault leads to a change in the voltage of the motors' terminals, as can be seen in Figure 5. In an unsymmetrical fault, the voltage change might not be large enough to be detected easily by a conventional protection system. Regardless of the performance of the protection system, here the fault is not cleared to show the full effect of interaction between thermomechanical and electrical systems.

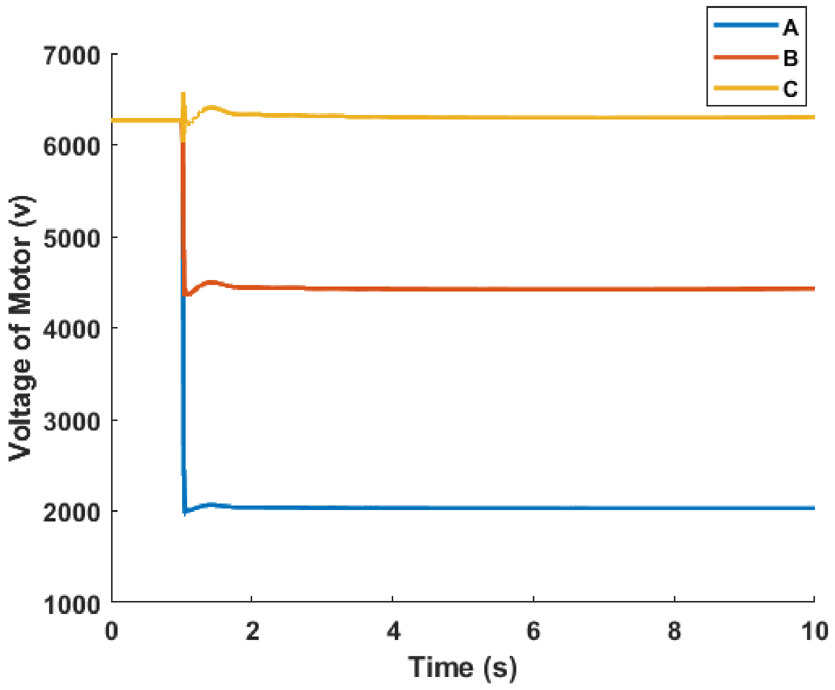


Figure 5. The voltage of the motors' terminal

Figure 6 shows the current of the first motor, coupled with the basic pump from Apros. This voltage change in the motors' terminals causes the electrical torque of the motors and their rotational speed to deviate. Figure 7 shows the rotational speed of these three motors. These motors are almost similar in their electrical model and parameters, but they are coupled to three different pump models in Apros.

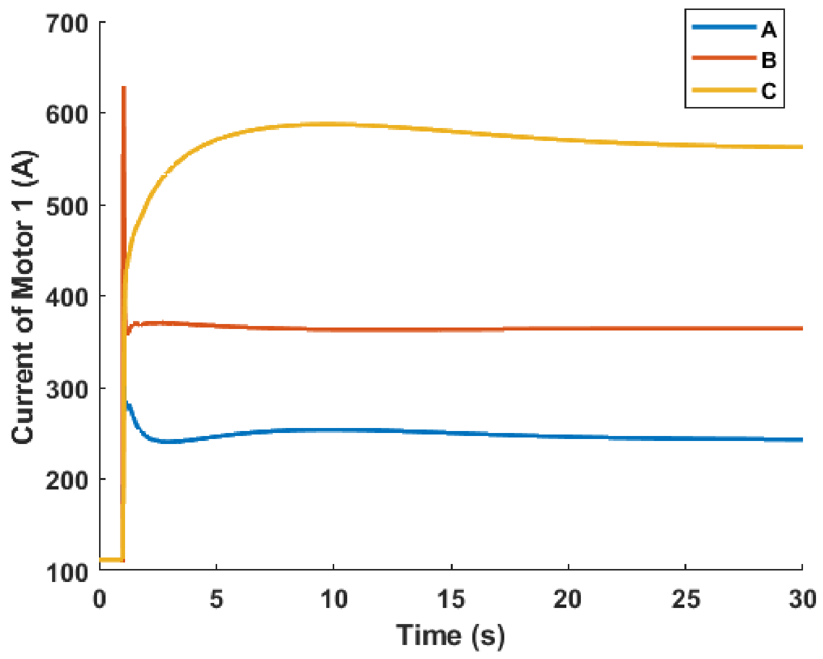


Figure 6. The current of the first motor, coupled to the basic pump from Apros

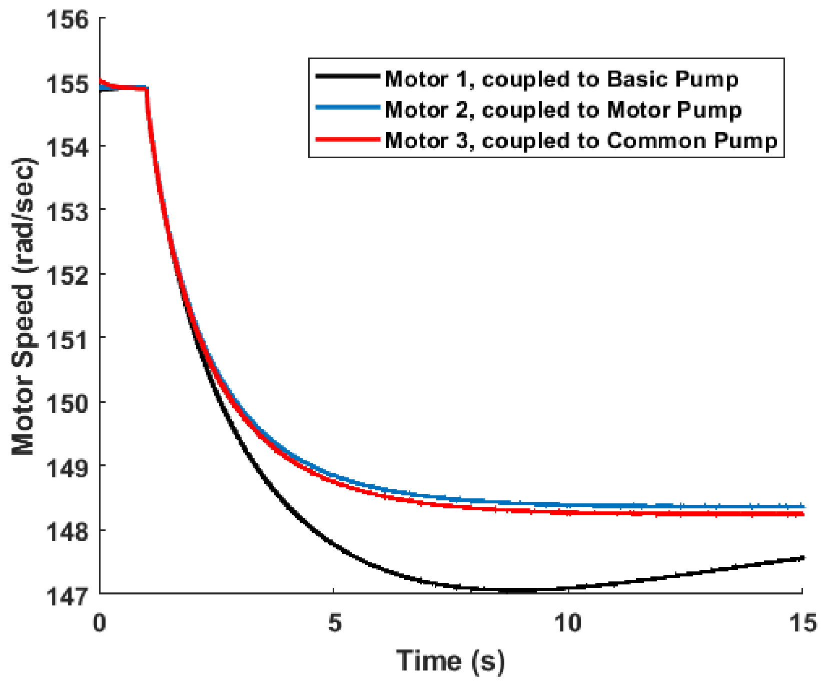


Figure 7. The rotational speed of motors, coupled to different pump models from Apros

The change in the rotational speed, change the torque of the mechanical pump, which can be seen in the Apros Models. Figure 8 shows the pump output from Apros, which acts as an input for Simulink. As Fig. 8 shown, in the basic pump model, the torque will be increased initially, which may lead to instability.

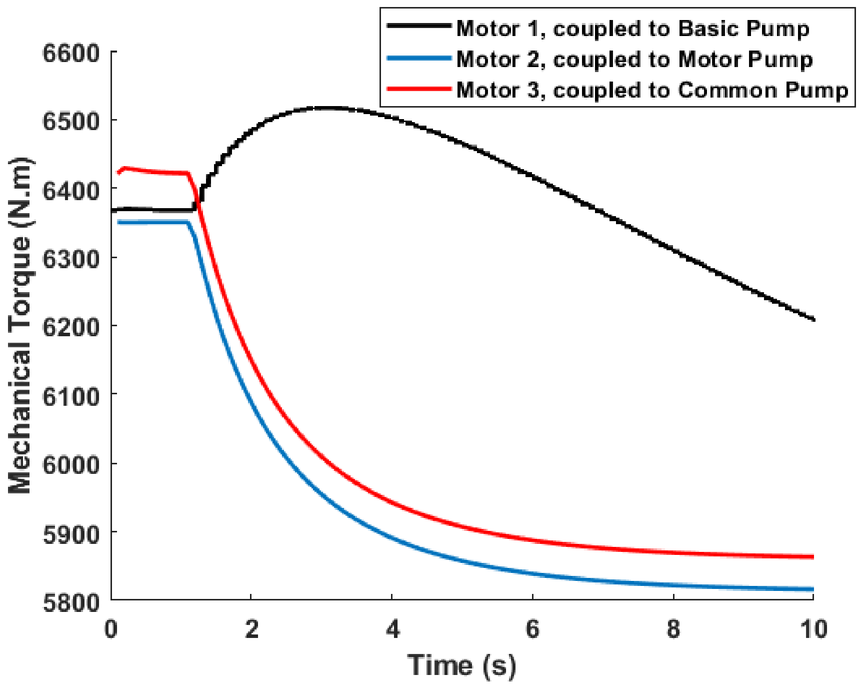


Figure 8. The mechanical torque output of different pumps, coupled to almost similar motors in Simulink

The simulated fault in the electrical system, a two-phase fault, creates disturbances in the motor speed and pump's torque as shown in Figs. 7 and 8. These disturbances lead to some other changes in the mechanical loops. For example, Fig 9 shows the change in the mass flow of the different pumps in the Apros model.

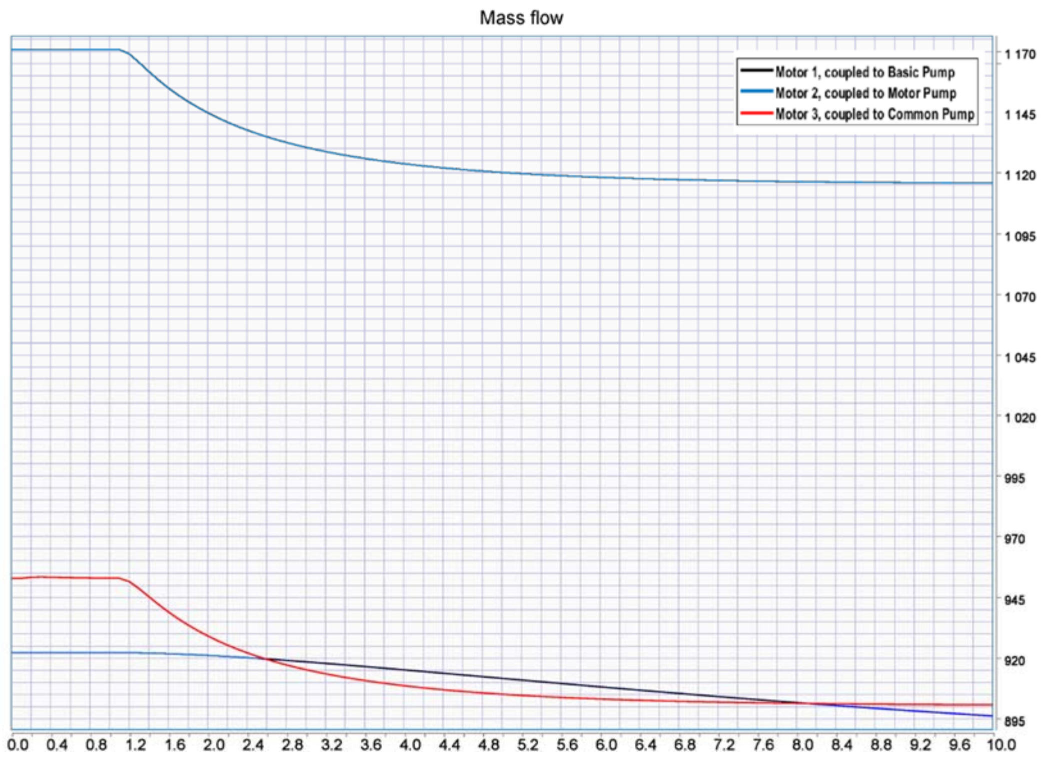


Figure 9. The mass flow of different pumps, coupled to almost similar motors in Simulink

This fault creates a disturbance in the Turbine-Generator as well. Fig. 10 shows the voltage of the generator terminal, that is not changing much. However, this disturbance can lead to a disturbance in the generator speed and also to the mechanical power of the turbine-shaft in the Apros model, as shown in Figs. 11 and 12.

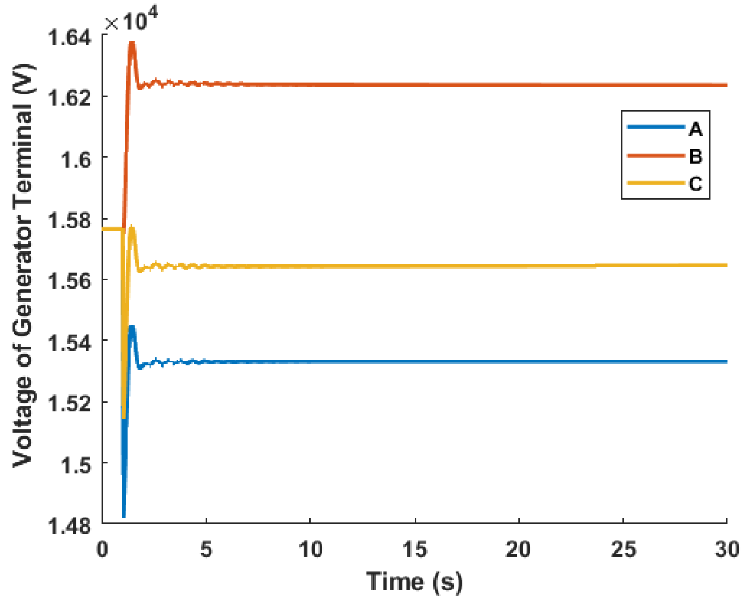


Figure 10. The voltage of the generator terminal

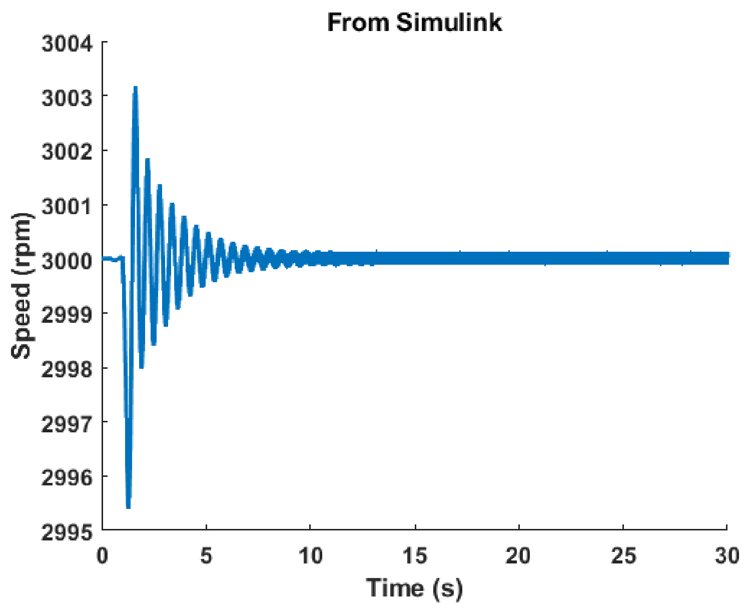


Figure 11. The generator speed

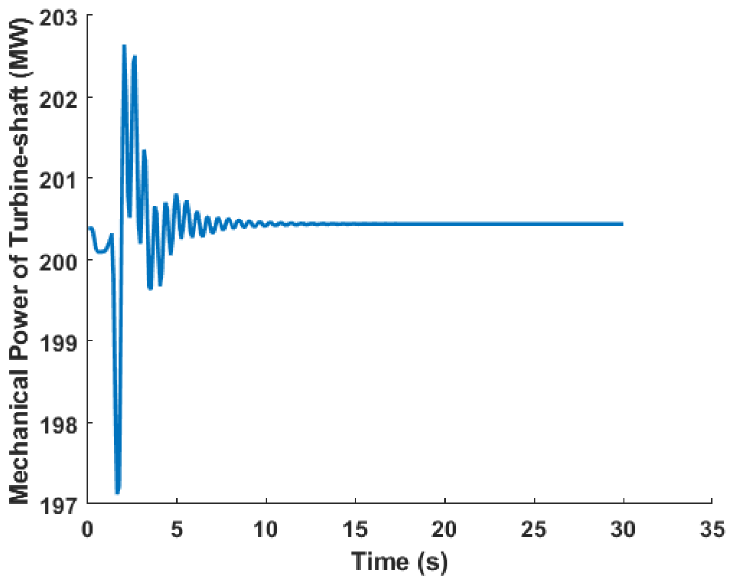


Figure 12. The mechanical power of the turbine-shaft in the Apros model

This simple co-simulation test shows that the disturbance of an event in one simulator can be transferred to another simulator using the co-simulation platform. This co-simulation allows the user to study the detailed effect of the interaction of different domains in the NPP.

To show the effect of the co-simulation of different simulators, the co-simulation is compared to the pure electrical system, when the mechanical torque is modelled as $k*\omega^2$. In this type of modelling, *k* is calculated for each pump-motor to have similar operating point pre-fault. Figure 13 shows the results mechanical torque of motors, which differ to those in Fig. 8 in dynamic behaviour, showing the impact of different co-simulation methods, especially in the case of basic pump models. This comparison shows that removing the exact model of a pump and replacing it with a simplified mechanical torque leads to a slight loss in the accuracy.

5. Conclusions

This report details the first version of the co-simulation platform and explains how a user can co-simulate an NPP using this platform. The first version of the co-simulation platform was developed to study the interaction of thermomechanical system and electrical system (on-site and off-site) using Apros and Matlab/Simulink. However, the platform is designed so that the different simulators, especially power system simulators, can be added to the platform later.

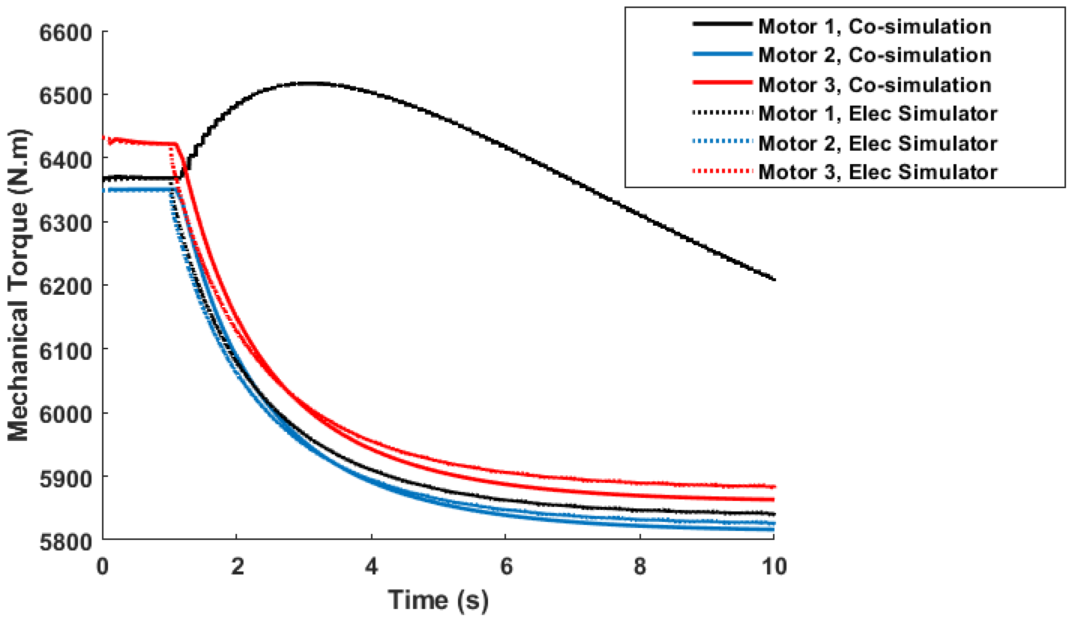


Figure 13. The mechanical torque of motors, solid lines: co-simulation result from Fig.8; dash lines: pumps are modelled as $k*\omega^2$ (No co-simulation)

In order to show the performance of the platform, a small model of an electrical system, including generators and three motors was developed in Matlab/Simulink, while the generator and motors coupled to Turbine and pumps were modelled in Apros. Among different pump models Apros, three of them, which are used more frequently in the modelling of NPP, have been implemented in the first version of this co-simulation platform.

The simulation results show that co-simulation works. All involved tools have their normal usage environment available that can be used to see simulation results and allow the user to edit and change the settings in all simulators simultaneously. This way, the model and tools that are developed in each domain-specific simulator, can be used directly by their domain expert, while the co-simulation platform can analyse the interaction of these different domains without simplifying the models of any of the domains.

Besides, the report compares the simulation results of the electrical system using co-simulation with those obtained using just the power system simulator. The common method that is widely used to study the electrical system, is to replace the pump model by a variable torque ($k*\omega^2$), and neglect the effects of the thermomechanical system. The comparison showed that this variable torque can be an acceptable equivalent in the case of a basic pump model. However, such a simplification cannot achieve the same results when more advanced pump models are used.

Reference

1 “Apros - Dynamic Process Simulation Software for Nuclear and Thermal Power Plant Applications,” http://www.apros.fi/en, accessed October 2020

2 Johnstone, K., Blair, S.M., Syed, M.H., Emhemed, A., Burt, G.M., Strasser, T.I.: “Co-simulation approach using PowerFactory and MATLAB/Simulink to enable validation of distributed control concepts within future power systems,” in “CIRED - Open Access Proceedings Journal” (2017), pp. 2192–2196

3 Latif, A., Shahzad, M., Palensky, P., Gawlik, W.: “An alternate PowerFactory Matlab coupling approach,” in “Proceedings - 2015 International Symposium on Smart Electric Distribution Systems and Technologies, EDST 2015” (2015)

4 Divshali, P., Hänninen, S., Laakso, P., Korvola, T.: “Architecture Design for NPP Co-Simulation Platform” (VTT Technical Research Centre of Finland, 2020)

5 Hasanpor Divshali, P., Laukkanen, M., Kulmala, A., *et al.*: “Smart Grid Co-Simulation by Developing an FMI-Compliant Interface for PSCAD,” in “CIRED 2019 Conference” (AIM, 2019)

6 Poria Divshali: “Co-Simulation platform V01 · master · Poria Divshali / SAFIR_COSI · GitLab,” https://gitlab.vtt.fi/poria.divshali/safir_cosi/-/tree/master/Vosimulation platform V01, accessed December 2020

Appendix A

The detailed code of the Master Program in MATLAB is available in this Appendix, and also can be found as an open-source tool in [6]. The Master Program includes the main body (PD_main.m) and 7 functions as follows:

A.1.main body of the Master Program (PD_main.m)

```
clear
clc
close all
warning off

%% Set File names && Initial Parameter

CoSim.Ta = 0.1;       % Apros Time Step
CoSim.Ts = 50e-6;     % Simulink Time Step

CoSim.SteadyTime = 20; % if there is no initial data for electrical
                    % systems, Simulink is run for CoSim.SteadyTime s
                    % to reach the steady-state (just one time)
CoSim.Tend = 15;      % Stop Time, Notice start time in co-simulation is 0

CoSim.SimName = 'PD_OnSiteElectricalGrid_Sample';     % Simulink file name
CoSim.IC_File = 'StedyState';          % Initial Condition File Name
CoSim.IC = 'OperP';                    % Initial Condition variable Name

%% Define Input Layout
% Co-simulation layout (File) need to be updated for co-simulation of a
% new NPP

Layout = PD_Layout();

%% Create Data Structure and OPC Host
% This Function create data Structure for both Simulink and Apros

DataStruc = PD_DataStructure(Layout);
OPC_Host = PD_OPC_setup(DataStruc);
connect(OPC_Host);

%% Initializing All Simulators

[St,Results] = PD_Initializing(DataStruc,OPC_Host,CoSim);
% Initializing Simulink
open(CoSim.SimName)
set_param(CoSim.SimName,'FastRestart','off','SaveFinalState','on',...
   'SaveFormat','Array','FinalStateName','FinalState',...
```

```
    'SaveCompleteFinalSimState','on','LoadInitialState','off');
try
    load([CoSim.IC_File num2str(CoSim.SteadyTime) '.mat']);
    disp('Start from Steady-state')
    eval(['IntState = ' CoSim.IC ';'])
catch
    disp('Initializing Simulink, Please Wait ....')

    SimulinlkOut = sim(CoSim.SimName,'StartTime','0',...
        'StopTime',num2str(CoSim.SteadyTime));
    IntState = SimulinlkOut.get('FinalState');

    eval([CoSim.IC ' = IntState;']);
    save([CoSim.IC_File num2str(CoSim.SteadyTime) '.mat'],CoSim.IC);
end

set_param(CoSim.SimName,'LoadInitialState','on',...
    'InitialState','IntState','FastRestart','on')

ItmN = size(DataStruc.AprInp,1) + size(DataStruc.AprOut,1) + ...
    size(DataStruc.AprExt,1) + 2;   % Item Number in OPC
OPCTemp = read(OPC_Host.Group);
AprIntTime =OPCTemp(ItmN).Value; % The internal time when Apros start
                                 % Co-simulation

%% Co Simulation

disp('Start Co-simulation ...')

Time = CoSim.SteadyTime;
IttrCnt = 1;
while Time < CoSim.SteadyTime + CoSim.Tend

    % Run Apros
    Do = ['do ' num2str(CoSim.Ta)];
    write(OPC_Host.Group.Item(ItmN-1),Do);

    % Run Simulink
    t_sim = tic;
    SimulinlkOut = sim(CoSim.SimName,'StopTime',num2str(CoSim.SteadyTime...
        +IttrCnt*CoSim.Ta));
    IntState = SimulinlkOut.get('FinalState');

    % wait until Apros finnish the task
    OPCTemp = read(OPC_Host.Group,'device');             %
    AprTime = OPCTemp(ItmN).Value - AprIntTime;
    while abs(AprTime - Time + CoSim.SteadyTime - CoSim.Ta) > 1e-6
        pause(CoSim.Ta/2)
        disp('puase 0.1')
        OPCTemp = read(OPC_Host.Group,'device');             %
        AprTime = OPCTemp(ItmN).Value - AprIntTime;
    end

    % Data Exchange and Keeping Simulation Results
    [St,Results] = PD_DataExchange(CoSim,DataStruc,SimulinlkOut,OPCTemp...
        ,OPC_Host,St,Results);

    % Report the progress
    PD_ProgRep(Time,CoSim.SteadyTime,CoSim.SteadyTime+CoSim.Tend,...
        CoSim.Ta,10,1);

    % Next itteration
    Time = Time + CoSim.Ta;
    IttrCnt = IttrCnt + 1;
end


%% Plot

PD_plot(DataStruc,Results)

%%
% reset Matlab OPC:
disconnect(OPC_Host);
opcreset;

set_param(CoSim.SimName,'FastRestart','off')
```

A.2.Input data (PD_Layout.m)

```
function Layout = PD_Layout()
% This Function defins Co-Simulation Layout (indicate all components that
% participating in co-simulations, e.g. pump-motor sets and
% turbine-generator sets.
% Also, any variables that needs to monitor/log in master program.

Layout.GenSet = {};
Layout.BasicMotor = {};
Layout.MotorPump = {};
Layout.ComPump = {};
Layout.log_Simulink = {};
Layout.log_Appros = {};

%% Generators Model
% Layout.GenSet = { 1)Generator Name, 2) Shaft Name, 3) Inputs, 4)Output;
%         ....
%              };
% 1)Generator Name : Exact Name in Appros and Simulink
% 2) Shaft Name : Exact Name in Appros and Simulink
% 3) Inputs: G_Pmech_ ... : Initial  Mechanical power of Turbine (MW)
% 4) Output: [ 4-1) G_PE_... 4-2)G_W_... ]
   % 4-2)G_PE_...: Initial electrical active power (W)
   % 4-1)G_W_...: Initial rotation speed (RPM)

Layout.GenSet = {'SG01' 'SG01_SH' 200.4 [202e6 3000]};

%% BasicPump Model
% Layout.BasicMotor = { 1) Name , 2) Inputs, 3)Output;
%          ....
%               };
% 1) Name : Exact Name in Appros and Simulink
% 2) Inputs: St.BP_Pmech_...: Initial Mechanical Power (kW) of Motor
% 3) Output: BP_WP_...: Initial Speed of Motor (%)

Layout.BasicMotor = {'BP01' , 1035.66, 98.534};


%% MotorPump Model
% Layout.BasicMotor = { 1) Name , 2) Inputs, 3)Output;
%          ....
%               };
% 1) Name : Exact Name in Appros and Simulink
% 2) Inputs: St.CP_Tmech0...: Initial Mechanical Torque of Motor (N.m)
% 3) Output: CP_Wf...: Initial rotational Speed of Motor-frequency (rps)

Layout.MotorPump = {'MP01' , 6468.91, 24.6459};


%% CommonPump Model
% Layout.ComPump = { 1) Name , 2) Inputs, 3)Output;
%          ....
%               };
% 1) Name : Exact Name in Appros and Simulink
% 2) Inputs: St.CP_Tmech0...: Initial Mechanical Torque of Motor (N.m)
% 3) Output: CP_Wf...: Initial rotational Speed of Motor-frequency (rps)
Layout.ComPump = {'CP01' , 6338.3, 48.4549};

%% Extra Log
% Simulink output name (Variable name in logging page of Scope (Type Array));
% Layout.log_Simulink={'Name 1';
%                   'Name 2';
%                    ...    ;
%                   };

Layout.log_Simulink={'BP01_V';
               };

% Apros output name (Not Complete Yet but must be defined by OPC);
% Layout.log_Simulink={'Component Name 1' 'Property name 1';
%                   'Component Name 2' 'Property name 2';
%                    ...    ;
%                   };
Layout.log_Appros = {'MP01' 'PU12_ACTIVE_POWER'
                 };
```

A.3.Create Data Structure (PD_DataStructure.m)

```
function Structure = PD_DataStructure(Layout)
% This Function create data Structure for both Simulink and Apros (OPC)

SimPar = {}; % Parameters act as Sim Input without Apros Output
SimInp = {};
SimOut = {};
SimExt = {}; % Extra log
AprPar = {};
AprInp = {};
AprOut = {};
AprExt = {}; % Extra log

%% Generator Parameters
for Cnt = 1: size(Layout.GenSet,1)
   Name = Layout.GenSet{Cnt,1};
   Sh_Name = Layout.GenSet{Cnt,2};

   CntSimInp = size(SimInp,1) + 1;
   CntSimOut = size(SimOut,1) + 1;

   % Inputs
   SimInp{CntSimInp,1} = ['St.' Name '_Pmech'];
   SimInp{CntSimInp,2} = Layout.GenSet{Cnt,3}(1,1);
   AprOut{CntSimInp,1} = [Sh_Name '!SH_POWER'];
   AprOut{CntSimInp,2} = Layout.GenSet{Cnt,3}(1,1);

   % Outputs
   SimOut{CntSimOut,1} = [Name '_PE' ];
   SimOut{CntSimOut,2} = Layout.GenSet{Cnt,4}(1,1);
   AprInp{CntSimOut,1} = [Name '!ES_GE_ACTIVE_POWER'];   %[Name '_PE!SP_VALUE']
   AprInp{CntSimOut,2} = Layout.GenSet{Cnt,4}(1,1);

   SimOut{CntSimOut+1,1} = [Name '_W'];
   SimOut{CntSimOut+1,2} = Layout.GenSet{Cnt,4}(1,2);
   AprInp{CntSimOut+1,1} = [Name '!ES_GE_SPEED_OF_ROTATION'];    % [Name
'_W!SP_VALUE']
   AprInp{CntSimOut+1,2} = Layout.GenSet{Cnt,4}(1,2);

   SimOut{CntSimOut+2,1} = [Name '_W'];
   SimOut{CntSimOut+2,2} = Layout.GenSet{Cnt,4}(1,2);
   AprInp{CntSimOut+2,1} = [Sh_Name '!SH_SHAFT_SPEED'];    % [Name '_W!SP_VALUE']
   AprInp{CntSimOut+2,2} = Layout.GenSet{Cnt,4}(1,2);
end

%% BasicPump Parameters
for Cnt = 1: size(Layout.BasicMotor,1)
   Name = Layout.BasicMotor{Cnt,1};

   CntSimInp = size(SimInp,1) + 1;
   CntSimOut = size(SimOut,1) + 1;

   % Inputs
   SimInp{CntSimInp,1} = ['St.' Name '_Pmech'];
   SimInp{CntSimInp,2} = Layout.BasicMotor{Cnt,2}(1,1);
   AprOut{CntSimInp,1} = [Name '!PU11_ACTIVE_POWER'];
   AprOut{CntSimInp,2} = Layout.BasicMotor{Cnt,2}(1,1);

   % Outputs
   SimOut{CntSimOut,1} = [Name '_WP' ];
   SimOut{CntSimOut,2} = Layout.BasicMotor{Cnt,3}(1,1);
   AprInp{CntSimOut,1} = [Name '_PU1!P_SPEED_OLD'];    %[Name
'_PU1!P_SPEED_OLD']; [Name '!PU11_SPEED_SET_POINT'];
   AprInp{CntSimOut,2} = Layout.BasicMotor{Cnt,3}(1,1);
end

%% Motor-Pump Parameters
for Cnt = 1: size(Layout.MotorPump,1)
   Name = Layout.MotorPump{Cnt,1};

   CntSimInp = size(SimInp,1) + 1;
   CntSimOut = size(SimOut,1) + 1;

   % Inputs
   SimInp{CntSimInp,1} = ['St.' Name '_Tmech' ];
   SimInp{CntSimInp,2} = Layout.MotorPump{Cnt,2}(1,1);
   AprOut{CntSimInp,1} = [Name '!PU12_PUMP_TORQUE'];
   AprOut{CntSimInp,2} = Layout.MotorPump{Cnt,2}(1,1);

   % Outputs
   SimOut{CntSimOut,1} = [Name '_WF'];
   SimOut{CntSimOut,2} = Layout.MotorPump{Cnt,3}(1,1);
   AprInp{CntSimOut,1} = [Name '_MC1!MC_MOTOR_SPEED'];     % [Name
'_MO1!MO1_SPEED'];  [Name '!PU12_MOTOR_SPEED'] [Name
'_MC1!MC_MOTOR_SPEED'];
   AprInp{CntSimOut,2} = Layout.MotorPump{Cnt,3}(1,1);
end

%% Common-Pump Parameters
for Cnt = 1: size(Layout.ComPump,1)
   Name = Layout.ComPump{Cnt,1};
   CntSimInp = size(SimInp,1) + 1;
   CntSimOut = size(SimOut,1) + 1;

   % Inputs
   SimInp{CntSimInp,1} = ['St.' Name '_Tmech'];
   SimInp{CntSimInp,2} = Layout.ComPump{Cnt,2}(1,1);
   AprOut{CntSimInp,1} = [Name '_PU1!PU2_TORQUE'];  %[Name
'_MO1!MO1_TORQUE'];
   AprOut{CntSimInp,2} = Layout.ComPump{Cnt,2}(1,1);

   % Outputs
   SimOut{CntSimOut,1} = [Name '_WF'];
   SimOut{CntSimOut,2} = Layout.ComPump{Cnt,3}(1,1);
   AprInp{CntSimOut,1} = [Name '_PU1!PU2_SPEED'];   % [Name '_WF!SP_VALUE'];
   AprInp{CntSimOut,2} = Layout.ComPump{Cnt,3}(1,1);
end

%% Extra Log
SimExt = Layout.log_Simulink;
for Cnt = 1: size(Layout.log_Appros,1)
   AprExt{Cnt,1} = [Layout.log_Appros{Cnt,1} '!' Layout.log_Appros{Cnt,2}];
end

%% Output
Structure.SimPar = SimPar;
Structure.SimInp = SimInp;
Structure.SimOut = SimOut;
Structure.SimExt = SimExt;
Structure.AprPar = AprPar;
Structure.AprInp = AprInp;
Structure.AprOut = AprOut;
Structure.AprExt = AprExt;
```

A.4.OPC Setup (PD_OPC_setup.m)

```
function out = PD_OPC_setup(Str)
% % Create OPC Setup based on the Data Structure
% 1) Input variables then 2) output variables and 3) later Extra output
% 4)Finally last two items are General Apros CoomandData


% Create the OPCDA object - daobj1
daobj1 = opcda('localhost', 'VTT.AprosOPCCOMDAserver.1');

% Create the Group object - grp1
ItmN = size(Str.AprInp,1) + size(Str.AprOut,1) + size(Str.AprExt,1) + 2;
grp1 = addgroup(daobj1, num2str(ItmN));
set(grp1, 'LogFileName', 'opcdatalog.olf');
```

```
%% Input Variable
ItmCnt = 0;
for Cnt = 1: size(Str.AprInp,1)
   eval(['itm' num2str(ItmCnt+Cnt) '=additem(grp1,"' Str.AprInp{Cnt,1} '");'])
   eval(['set(itm' num2str(ItmCnt+Cnt) ', "DataType", "double");'])
end

%% Output Variable
ItmCnt = Cnt;
for Cnt = 1: size(Str.AprOut,1)
   eval(['itm' num2str(ItmCnt+Cnt) '=additem(grp1,"' Str.AprOut{Cnt,1} '");'])
   eval(['set(itm' num2str(ItmCnt+Cnt) ', "DataType", "double");'])
end

%% Extra Result from Appros
ItmCnt = ItmCnt + Cnt;
for Cnt = 1: size(Str.AprExt,1)
   eval(['itm' num2str(ItmCnt+Cnt) '=additem(grp1,"' Str.AprExt{Cnt,1} '");'])
   eval(['set(itm' num2str(ItmCnt+Cnt) ', "DataType", "double");'])
end

Lay.log_Appros = {'CM_YD11D001B' 'PU12_MOTOR_SPEED'};

%% Appros Control
% Create the Item object - itm(ItmN-1)
eval(['itm' num2str(ItmN-1) '=additem(grp1,"OPC_INI!OPC_COMMAND");'])
eval(['set(itm' num2str(ItmN-1) ', "DataType", "char");'])

% Create the Item object - itm(ItmN)
eval(['itm' num2str(ItmN) '=additem(grp1,"ECCO!SIMULATION_CURRENT_TIME");'])
eval(['set(itm' num2str(ItmN) ', "DataType", "double");'])

%%
if nargout > 0
   out = [daobj1];
end
```

A.5.Initializing simulators (PD_Initializing.m))

```
function [St,Results] = PD_Initializing(Str,OPC_Host,CoSim)
% This Function initializ all Input/Output variables for all simulators

%% Simulink

% Initial Parameters
for Cnt = 1: size(Str.SimPar,1)
   eval([Str.SimPar{Cnt,1} '='  num2str(Str.SimPar{Cnt,2}) ';']);
end
```

```
% Initial Inputs
for Cnt = 1: size(Str.SimInp,1)
   eval([Str.SimInp{Cnt,1} '='  num2str(Str.SimInp{Cnt,2}) ';']);
end

% Initial Output
Results.SimTime = [];
Results.SimOut = [];
Results.SimExt = [];


%% Apros (OPC)

% Initial Inputs
ItmCnt = 0;
for Cnt = 1: size(Str.AprInp,1)
   write(OPC_Host.Group.Item(ItmCnt+Cnt),Str.AprInp{Cnt,2});
end

% Initial Output
ItmCnt = ItmCnt + Cnt;
for Cnt = 1: size(Str.AprOut,1)
   write(OPC_Host.Group.Item(ItmCnt+Cnt),Str.AprOut{Cnt,2});
end
Results.AprTime = [];
Results.AprOut = [];
Results.AprExt = [];

% Initial Time
% ItmN = size(Str.AprInp,1) + size(Str.AprOut,1) + size(Str.AprExt,1) + 2;
% write(OPC_Host.Group.Item(ItmN),CoSim.Tstart);
```

A.6. Data Exchange function (PD_DataExchange.m)

```
function [St,Results] =
PD_DataExchange(CoSim,Str,SimulinlkOut,OPCTemp,OPC_Host,St,Results)
% This Function exchange data between Simulink and OPC (Apros)
% and saves the results of Apros and Simulink in each Data exchange
% interval

%% From Simulink 2 Appros (Appros Inp)
eval(['tempT=SimulinlkOut.' Str.SimOut{1,1} '(:,1);']);
Results.SimTime = [Results.SimTime ...
   (tempT-CoSim.SteadyTime*ones(size(tempT)))'];
tempResult = [];
for Cnt = 1:size(Str.AprInp,1)
   eval(['temp=SimulinlkOut.' Str.SimOut{Cnt,1} '(:,2);']);
   write(OPC_Host.Group.Item(Cnt),mean(temp));  % temp(end) or mean(temp)
```

```
    tempResult = [tempResult;temp'];
end
Results.SimOut = [Results.SimOut tempResult];

%% saving Extra Simulink Results
tempResult = [];
for Cnt = 1:size(Str.SimExt,1)
    eval(['temp=SimulinlkOut.' Str.SimExt{Cnt,1} '(:,2);']);
    tempResult = [tempResult;temp'];
end
Results.SimExt = [Results.SimExt tempResult];

%% From Appros 2 Simulink (Appros Out)
Results.AprTime = [0 Results.AprTime] + CoSim.Ta;

tempResult = [];
SN_OPC = size(Str.AprInp,1); % Starting number of OPC
for Cnt = 1:size(Str.SimInp,1)
    eval([Str.SimInp{Cnt,1} '=' ...
        num2str(OPCTemp(SN_OPC+Cnt).Value) ';']);
    tempResult = [tempResult; OPCTemp(SN_OPC+Cnt).Value];
end
Results.AprOut = [Results.AprOut tempResult];

%% saving Extra Apros Results
tempResult = [];
SN_OPC = size(Str.AprInp,1) + size(Str.AprOut,1); % Starting number of OPC
for Cnt = 1:size(Str.AprExt,1)
    tempResult = [tempResult; OPCTemp(SN_OPC+Cnt).Value];
end
Results.AprExt = [Results.AprExt tempResult];
```

A.7.Report Progress (PD_ProgRep.m)

```
function [Prg] = PD_ProgRep(CrntStp,FrstStp,LstStp,DltStp,PrntStp,PrntMd)
% This function calculate the progrees and print if PrntMd ïs not equal 0

if PrntMd ~= 0
    StpN = floor((LstStp - FrstStp)/DltStp);
    PrnIndx = ceil(1:PrntStp*StpN/100:StpN);
    if PrnIndx(1,end) ~= StpN
        PrnIndx = [PrnIndx StpN];
    end

    Eps = 0.001*DltStp;
    CrntStpN = floor((CrntStp+Eps - FrstStp)/DltStp);

    [Lia,Locb] = ismember(CrntStpN,PrnIndx);
    if Lia
        disp([num2str(PrntStp*(Locb-1)) ' % Progress'])
    end
end

Prg = CrntStpN/StpN*100;
```

A.8. Plot results (PD_plot.m)

```
function PD_plot(DataStruc,Results)

%% Plot From Simulink
Fig = figure('InvertHardcopy','off','Color',[1 1 1]);
Axes = axes('Parent',Fig);
xlabel('Time (s)','FontWeight','bold')
ylabel('Parameters','FontWeight','bold')
set(Axes,'FontSize',12,'FontWeight','bold');
legend(Axes,'show');
title('From Simulink','FontSize',13,'FontWeight','bold')
hold on

for Cnt = 1:size(Results.SimOut,1)
    plot(Results.SimTime,Results.SimOut(Cnt,:),'DisplayName',...
            DataStruc.SimOut{Cnt,1},'LineWidth',2)
end

%% Plot To Simulink
Fig = figure('InvertHardcopy','off','Color',[1 1 1]);
Axes = axes('Parent',Fig);
xlabel('Time (s)','FontWeight','bold')
ylabel('Parameters','FontWeight','bold')
set(Axes,'FontSize',12,'FontWeight','bold');
legend(Axes,'show');
title('To Simulink','FontSize',13,'FontWeight','bold')
hold on

for Cnt = 1:size(Results.AprOut,1)
    plot(Results.AprTime,Results.AprOut(Cnt,:),'DisplayName',...
            DataStruc.AprOut{Cnt,1},'LineWidth',2)
end

%% Plot Extra Simulink
Fig = figure('InvertHardcopy','off','Color',[1 1 1]);
Axes = axes('Parent',Fig);
xlabel('Time (s)','FontWeight','bold')
ylabel('Parameters','FontWeight','bold')
set(Axes,'FontSize',12,'FontWeight','bold');
legend(Axes,'show');
title('Extra Simulink','FontSize',13,'FontWeight','bold')
```

```
hold on

for Cnt = 1:size(Results.SimExt,1)
   plot(Results.SimTime,Results.SimExt(Cnt,:),'DisplayName',...
         DataStruc.SimExt{Cnt,1},'LineWidth',2)
end

%% Plot Extra Apros
Fig = figure('InvertHardcopy','off','Color',[1 1 1]);
Axes = axes('Parent',Fig);
xlabel('Time (s)','FontWeight','bold')
ylabel('Parameters','FontWeight','bold')
set(Axes,'FontSize',12,'FontWeight','bold');
legend(Axes,'show');
title('Extra Apros','FontSize',13,'FontWeight','bold')
hold on

for Cnt = 1:size(Results.AprExt,1)
   plot(Results.AprTime,Results.AprExt(Cnt,:),'DisplayName',...
         DataStruc.AprExt{Cnt,1},'LineWidth',2)
end
```

Series title and number

VTT Technology 382

Title	**Description of the co-simulation platform for NPP**
Author(s)	Poria Divshali, Seppo Hänninen, Pasi Laakso, Timo Korvola, & Robert John Millar
Abstract	In the first year of the COSI project, WP1, the architecture of a co-simulation platform for power plants was designed. This architecture provides the possibility of studying the interaction among thermomechanical and automation processes, on-site electrical grids, and off-site transmission system for a power plant. Following the proposed architecture, in this year, WP1 starts to develop the first version of the co-simulation platform. In this regard, the architecture is developed further to solve some remaining issues, e.g. initialising, and then the first platform implemented using MATLAB/m-file. The initial survey of simulation tools in the COSI project shows that most of the nuclear power plants in Finland use Apros as the simulation tool for thermomechanical and automation processes. However, since Apros cannot simulate the detailed electrical system events, e.g. unsymmetrical faults such as a single phase-earth fault in the electric system, the detailed electrical power system models are simulated in different simulation tools while neglecting the interaction of them with thermomechanical and automation processes. To solve this challenge, the first version of the co-simulation platform includes a "Master Program" developed in MATLAB (m-file) environment to co-simulate Apros with power system simulators. Apros supports Open Platform Communications (OPC) and through this protocol, the Master Program can connect, read, write, and control Apros. Connecting to other power system simulators needs other appropriate protocols, which depend on the simulator's features. However, in the first version, Simulink will be the only power system simulator which is connected to the Master Program. The benefit of the co-simulation platform in different events will be studied in the third year of the project. In the case of a substantial benefit, the project will develop further this co-simulation platform to other power system simulator, e.g. PSCAD in the future. In order to prove the appropriate working of the architecture, small thermomechanical and electrical models have been developed. The preliminary results show that the co-simulation platform works as expected in the operation of APROS and power system simulators. The large-scale co-simulation using the exact NPP component models and off-site transmission power grids are done in WP2 and will be reported in the related deliverable.
ISBN, ISSN, URN	ISBN 978-951-38-8742-1 ISSN-L 2242-1211 ISSN 2242-122X (Online) DOI: 10.32040/2242-122X.2020.T382
Date	December 2020
Language	English
Pages	19 p. + app. 12 p.
Name of the project	COSI: Co-simulation model for safety and security of electric systems in flexible environment of NPP
Commissioned by	
Keywords	Co-simulation, Nuclear power, Electric systems, Thermomechanical system.
Publisher	VTT Technical Research Centre of Finland Ltd P.O. Box 1000, FI-02044 VTT, Finland, Tel. 020 722 111, https://www.vttresearch.com

Description of the co-simulation platform for NPP

In the first year of the COSI project, WP1, the architecture of a co-simulation platform for power plants was designed. This architecture provides the possibility of studying the interaction among thermomechanical and automation processes, on-site electrical grids, and off-site transmission system for a power plant. Following the proposed architecture, in this year, WP1 starts to develop the first version of the co-simulation platform. In this regard, the architecture is developed further to solve some remaining issues, e.g. initialising, and then the first platform implemented using MATLAB/m-file.

The initial survey of simulation tools in the COSI project shows that most of the nuclear power plants in Finland use Apros as the simulation tool for thermomechanical and automation processes. However, since Apros cannot simulate the detailed electrical system events, e.g. unsymmetrical faults such as a single phase-earth fault in the electric system, the detailed electrical power system models are simulated in different simulation tools while neglecting the interaction of them with thermomechanical and automation processes.

To solve this challenge, the first version of the co-simulation platform includes a "Master Program" developed in MATLAB (m-file) environment to co-simulate Apros with power system simulators. Apros supports Open Platform Communications (OPC) and through this protocol, the Master Program can connect, read, write, and control Apros. Connecting to other power system simulators needs other appropriate protocols, which depend on the simulator's features.

ISBN 978-951-38-8742-1
ISSN-L 2242-1211
ISSN 2242-122X (Online)
DOI: 10.32040/2242-122X.2020.T382