INL/EXT-21-62050
Revision 0

Development of the IES Plug-and-Play Framework

March | 2021

Konor L Frick
Andrea Alfonsi
Cristian Rabiti
Shannon Bragg-Sitton
Idaho National Laboratory

Integrated Energy Systems

INL/EXT-21-62050
Revision 0

Development of the IES Plug-and-Play Framework

Konor L Frick
Andrea Alfonsi
Cristian Rabiti
Shannon Bragg-Sitton
Idaho National Laboratory

March 2021

Idaho National Laboratory
Integrated Energy Systems
Idaho Falls, Idaho 83415

http://www.ies.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Science
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

Page intentionally left blank

ABSTRACT

Since early 2013, to accommodate the vast array of possibilities introduced by the concept of integrated energy parks that could incorporate multiple energy generation sources and multiple energy users, Idaho National Laboratory (INL) has been developing a library of high-fidelity process models in the Modelica modeling language. These models are a cornerstone of the analysis and optimization tools developed via the Department of Energy Office of Nuclear Energy (DOE-NE) Integrated Energy Systems (IES) program, led by Idaho National Laboratory (INL). Models are used to create and characterize system inertia, thermal losses, and the efficiency of integrated systems. These physical models help map physical performance into economic performance, allowing for system-level optimization. In addition, the models are used to test innovative system-level control strategies for interconnected thermal generators.

However, for real-world applications, it is not always practical to develop a model or rewrite an existing model in Modelica; instead, interoperability with Legacy FORTRAN, C, Python, or other codes is required. To accomplish this interoperability the IES Program is seeking to modify the existing suite of physical models, currently held in the HYBRID physical modeling repository, to be consistent with a "plug-and-play" approach in Modelica/Dymola models using Functional Mock-Up Units (FMUs), Functional Mock-Up Interfaces (FMIs), and machine-learning techniques. The models developed are held within the HYBRID repository that is part of the IES Framework for Optimization of ResourCes and Economics ecosystem (FORCE).

This report provides an overview of all the performed activities revolving around the deployment of methods, software infrastructures, guidelines, and a workflow for the construction and usage of models, as encapsulated using the FMI/FMU protocols and standards. The report is organized into three main macro-subjects, all of which are interconnected:

- FMI/FMU adaptors for Modelica models
- The HYBRID repository's new structure and open-source deployment
- RAVEN FMI/FMU exporting capabilities and artificial-intelligence (AI)-based analysis acceleration.

The first part of the report discusses the FMI/FMU adaptors created within the HYBRID repository to allow users to quickly export models such as FMUs. Several examples are given, highlighting the step-by-step process of converting an existing Modelica model into an FMU for use within the Dymola platform. Simulation results demonstrate that, though minor differences may occur, overall control, trends, and solution integrity are maintained between the standard Modelica simulation and FMU simulation results. However, it is worth noting that, for small systems, the FMU requires a longer simulation time than the Modelica-only simulation. Using this process, a company can provide external entities with models that contain proprietary information, without disclosing any model-related information that could be considered business sensitive. Such an ability would allow institutions to bypass the necessity of having "whitewashed" data.

In the second part of the report, the new structure of the HYBRID repository is discussed, with a major focus on the series of completed updates. These updates include the addition of Modelica system-level regression tests and software quality assurance (SQA) documentation to ensure that modifications to the Modelica models do not alter system-level model results.

The third and final part of the report documents the work performed for deploying methods and workflows to construct RAVEN AI-based models that are compliant with the FMI/FMU standard. Such work is key for deployment of the “flexible ecosystem” concept, since it allows for the replacement of high-fidelity Modelica models (or any other FMI/FMU-compliant model) with RAVEN-generated AI surrogate models.

Overall, extensive work has been completed in regard to developing FMUs and FMIs from existing models, understanding the requirements and limitations of FMUs, and open-sourcing the HYBRID repository with an integrated regression system for use within FORCE.

Page intentionally left blank

CONTENTS

FIGURES

TABLES

Page intentionally left blank

ACRONYMS

AI	Artificial Intelligence
API	Application Program Interface
CPU	Central Processing Unit
FMI	Functional Mock-Up Interface
FMU	Functional Mock-Up Unit
FORCE	Framework for Optimization of ResourCes and Economics ecosystem
FOM	Figure of merit
HTSE	High Temperature Steam Electrolysis
IES	Integrated Energy Systems
INL	Idaho National Laboratory
NHES	Nuclear Hybrid Energy Systems
RAVEN	Risk Analysis and Virtual Environment
ROM	Reduced-order model
SMR	Small Modular Reactor
SQA	Software Quality Assurance
V&V	Validation and Verification

Page intentionally left blank

1. INTRODUCTION

Grid demand variability is an inherent part of the modern dynamic lifestyle. The addition of renewable energy (e.g., wind and solar) technologies introduces variability into the grid supply. As renewable energy integration continues to grow, variability will further increase. The Department of Energy Office of Nuclear Energy (DOE-NE) Integrated Energy Systems (IES) Program, led by Idaho National Laboratory (INL), is researching the effects the impact of increasing variability on grid reliability and generator profitability, and is also investigating the complementary role of non-electric applications of these generators. IES involve the design, integration, and coordinated operation of several complex, traditionally standalone systems. The control algorithms involved are unique to each application and component design. IES architecture can include process steam applications, thermal energy storage, and the presence of intermittent energy sources such as wind and solar, as illustrated in Figure 1.

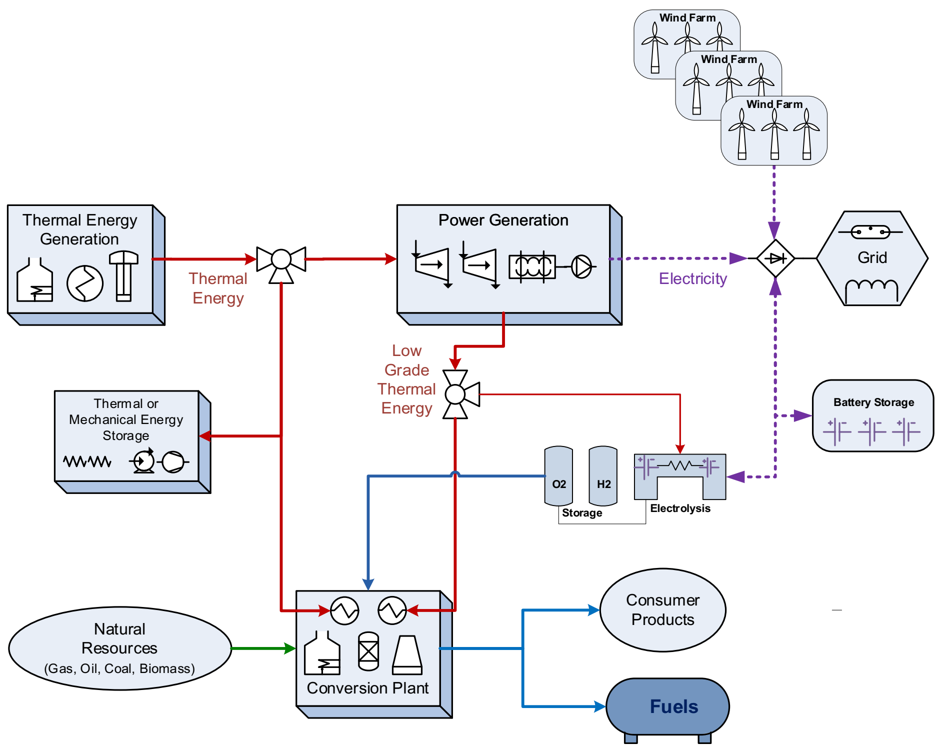


Figure 1. Example IES architecture, illustrating thermal and electrical interconnection to support hydrogen production and chemical conversion.

The goal of these systems is to operate as economically and efficiently as possible. For integrated energy parks that incorporate thermal storage, this means operating thermal generators at full power and storing excess energy during times of low total demand, then discharging that energy during times of high demand.

Since early 2013, to accommodate the vast array of possibilities introduced by integrated energy parks, the IES program team has been developing a library of high-fidelity process models in the Modelica modeling language [1]–[4]. Modelica is a non-proprietary, object-oriented, equation-based language for conveniently modeling complex physical systems. It is inherently time-dependent and enables the swift interconnection of independently developed models. As an equation-based modeling language that employs differential-algebraic equation solvers, Modelica allows users to focus on the physics of the problem rather than on the solving technique, thus enabling faster model generation and, ultimately, analysis. This feature, alongside system flexibility, has led to widespread use of Modelica for commercial applications throughout the industry. System interconnectivity and the ability to quickly develop novel control strategies while still encompassing overall system physics is why INL chose to develop the IES framework in the Modelica language.

The dynamic physical models created in Modelica are a cornerstone of the IES program. These models are used to create system architectures and characterize the system inertia, thermal losses, and the efficiency of integrated systems. These physical models help map physical performance into economic performance, allowing for system-level optimization. In addition, the models are used to test innovative system-level control strategies for interconnected thermal generators. However, it is noted that, for real-world applications, it is not always practical to rewrite a model in Modelica; instead, interoperability with Legacy FORTRAN, C, *Python*, or other codes may be required.

To accomplish this, the IES Program is seeking to modify HYBRID, the existing physical modeling repository, to be consistent with the "plug-and-play" approach in Modelica/Dymola models using Functional Mock-Up Units (FMUs), Functional Mock-Up Interfaces (FMIs), and machine-learning techniques (see Figure 2). The final product will greatly enhance the physical modeling interoperability within INL's Framework for Optimization of ResourCes and Economics ecosystem (FORCE) that is used to solve system/grid level optimization problems [5],[6].

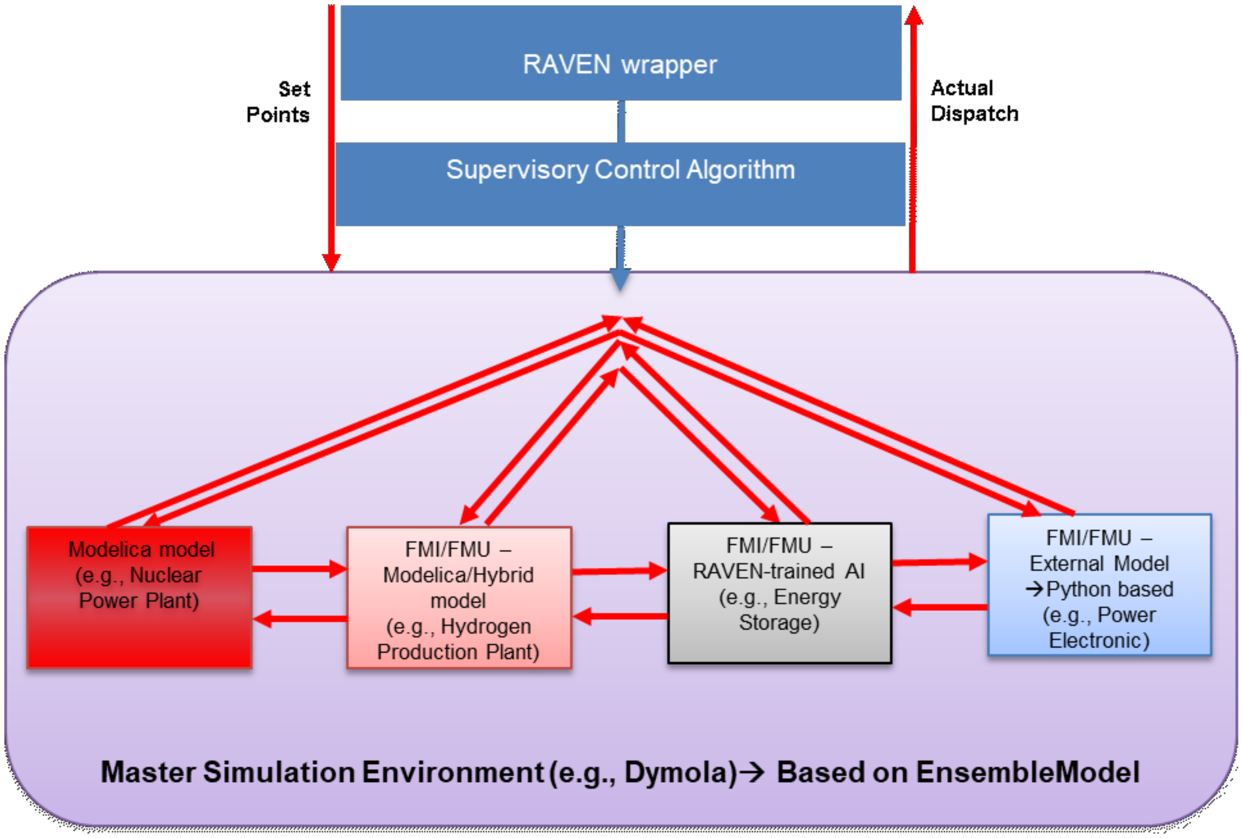


Figure 2. Plug-and-play framework environment.

This report summarizes the fiscal year (FY) 2020 efforts to create a plug-and-play repository of process models using the existing HYBRID repository, FMIs, FMUs [4], and the newly developed capabilities in the Risk Analysis and Virtual Environment (RAVEN) software for exporting artificial intelligence (AI)-based FMI/FMU models. The document characterizes and demonstrates the capabilities and improvements made to the previously-developed HYBRID repository of Modelica models for use as a software-quality-assured (SQA) plug-and-play system within FORCE.

The infrastructure of the GitHub repository that hosts the HYBRID repository was also enhanced. These improvements, described later in full detail, include the development (using the RAVEN-based ROOK regression system) of a Dymola output “differ” script for use with the commercially available Modelica-based modeling and simulation environment (i.e., a Dynamic Modeling Laboratory [Dymola] version 2021 FD01 [7]), inclusion of the Oak Ridge National Laboratory (ORNL) TRANSFORM library as an automatic submodule [8], creation of a user manual [9], and development of component-level regression tests for each Modelica model.

Extensive work was carried out on the deployment of methods for constructing RAVEN AI-based models compliant with the FMI/FMU standard. Such work represents the necessary initial development for deploying the “flexible ecosystem” (plug-and-play) concept, since it allows for replacement of high-fidelity Modelica models (or any other FMI/FMU-compliant model) with RAVEN-generated AI surrogate models. This capability enables the deployment of acceleration schemes for analyzing IES.

The Conclusions section of this report highlights the high flexibility achieved via the plug-and-play framework, possible shortcomings of the approach, and areas for further enhancement.

2. FUNCTIONAL MOCK-UP INTERFACES AND UNITS

This section briefly describes the FMIs and FMUs. As per the Modelon website (https://www.modelon.com): "FMI is an open standard for exchanging dynamical simulation models between different tools in a standardized format."

FMIs were first introduced by Dassault Systems under the name MODELISAR in 2008. FMIs define a standardized interface for use in computer simulations to develop complex cyber-physical systems. Additionally, FMIs/FMUs can be exported as binary files, enabling industry partners to exchange and simulate proprietary information safely and securely, without potential information leakage.

The FMI standard describes an open format for exporting and importing simulation models using a common data exchange nomenclature. In other words, the FMI standard allows the user to retain the same model while selecting the tools best suited for each type of analysis.

In order to be executed, an FMI is always "shipped" with an FMU. An FMU is the executable that implements the FMI. During exportation of an FMU, an FMU archive is generated from a systems model, whereas during an FMU import, a systems model is generated from an FMU archive.

FMUs contain the following:

- ***A model description XML file***: This file contains information about the model (e.g., variable definitions: type, unit, description, etc.) and other more general model information, such as model name, generation tool, and FMI version.
- ***Model equations***: A model can be described using ordinary differential equations, algebraic relations, and discrete equations—including time, state, and step events. These equations can in turn be represented by a small set of *C* functions. The *C* code is then distributed in the FMU in source and/or binary form, and one FMU can contain binaries for more than one platform and/or platform version.
- ***Optional resource files***: Other optional files might be included in the FMU, such as documentation files (HTML), model icons (bitmap files), maps and tables, and other libraries or dynamic link libraries (DLLs) used in the model.

The FMI/FMU standard currently specifies two types of protocols:

- FMI/FMU for model exchange (import and export)
- FMI/FMU for co-simulation (master and slave).

The main difference between these two protocols is that, in model exchange, the FMU is simulated using the importing tool's solver, whereas in co-simulation, the FMU is shipped with its own solver.

The FMI for model exchange allows FMUs to be used in offline or online simulation—with several FMUs potentially being connected—or in embedded control systems on microprocessors.

2.1 Co-simulation

Figure 3 shows the information flow and scheme of FMIs/FMUs in a co-simulation configuration. The co-simulation (CS) configuration is characterized by:

- Standalone black-box simulation components
- Data exchange being restricted to discrete communication "checkpoints"
- Between two consecutive communication checkpoints, the system model is solved by its internal solver.

In summary, the goal of a co-simulation operation is to individually compute the solution of time-dependent coupled systems and have them communicate back and forth at predetermined time steps, Δ**t**, known as communication steps (or checkpoints). The simulation is independently performed between all the subsystems, and at each Δ**t** there is a communication and transfer of boundary conditions between subsystems. Because of the independent nature of these subsystems, an FMI for co-simulation is the easiest method to implement. However, due to the different solver types and the need to specify Δ**t,** the scheme between systems becomes fully explicit. Being fully explicit, it is crucial to identify a small enough Δ**t** to ensure system stability. This step size limitation ultimately reduces the simulation speed.

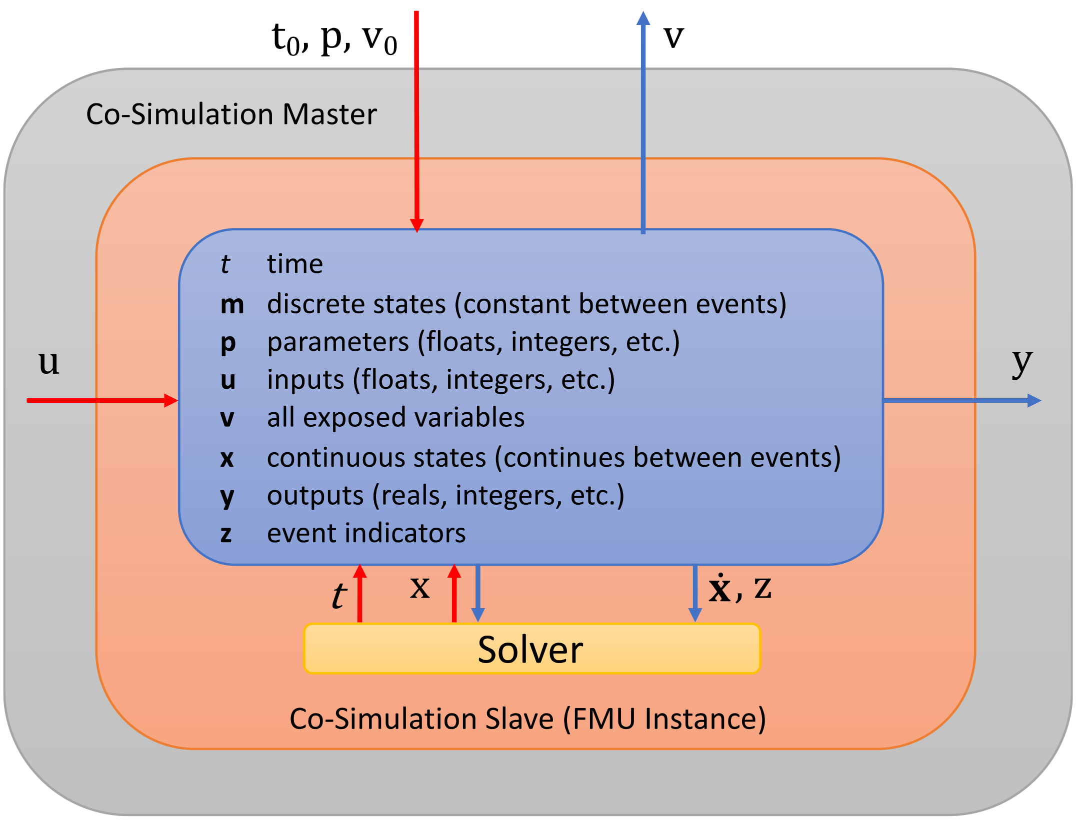


Figure 3. Co-simulation FMI/FMU scheme.

2.2 Model Exchange

Figure 4 shows the information flow and scheme of an FMI/FMU in a model exchange (ME) configuration. As shown in the figure, the model exchange configuration can be described as having the following characteristics:

- Standardized access to model equations
- Models described by algebraic, differential, and discrete equations
- Monitoring of time, state, and step events
- Models that must be solved using solvers provided by the embedding environment.

In summary, in a model exchange FMI/FMU, the numerical solver is supplied by the importing tool. The FMU provides functions to set the state/inputs and compute the state derivatives. The solver in the importing tool will determine what time steps to use and how to compute the state at each subsequent time step.

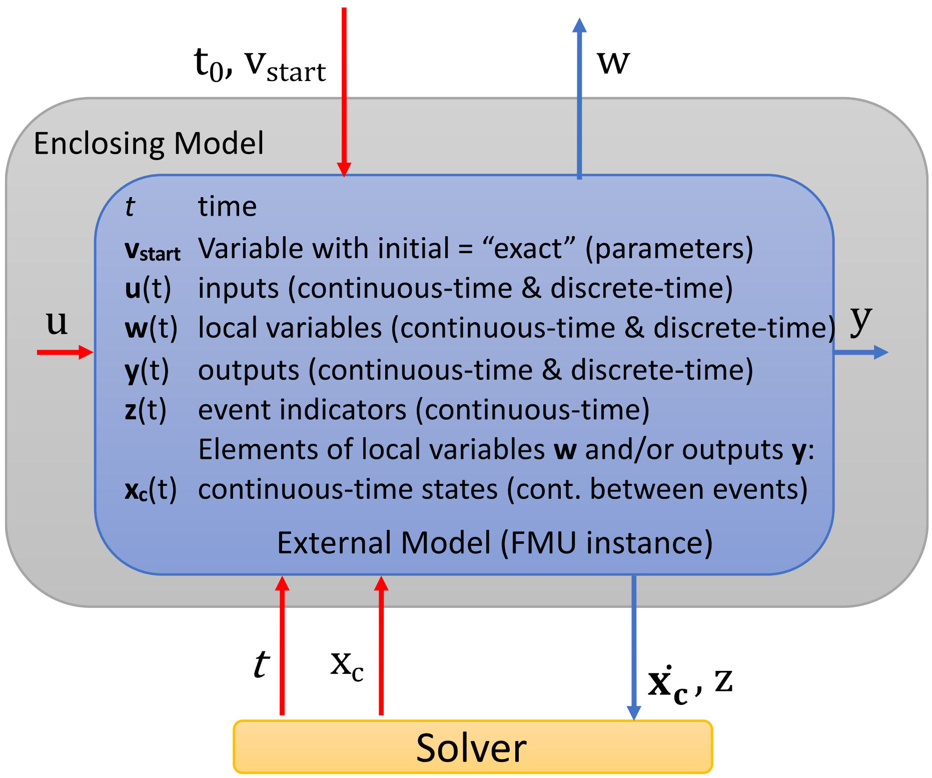


Figure 4. Model exchange FMI/FMU scheme.

2.3 Advantages of Each Protocol

Each of the two protocols described in the previous section, namely CS and ME, offer certain advantages.

Co-simulation

1. Not all tools support both protocol types. Support for CS is more common than for ME.
2. The numerics of the model may require a specific solver available in the exporting tool but not in the importing tool.

3. The FMU may represent a sampled data system (e.g., signal processing or control algorithms) not governed by differential equations and therefore more naturally expressed as a co-simulation FMU.
4. The exporting tool may have a more efficient implementation of the solver than the importing tool.

Model exchange

1. An explicit scheme is avoided, since the entire solve is done simultaneously.
2. Dynamic time stepping is allowed.
3. The importing tool could have a more efficient implementation of the solver than the exporting tool.

3. MODELICA TO FMU ADAPTATION

Modelica is a physical modeling language that relies on an acausal (rather than causal) assignment of equations. This means that an equation can only appear once, and that the translator and system solvers will determine the proper way to assign the flow of information. In addition, since Modelica is a physical modeling language, there are the assignments of special variable containers "flow" and "stream" that have an inherent physical representation in the code. Flow variables have a direction and must sum to zero in a "connection." The "stream" qualifier is used to qualify when a given element in a connection has an intensive property flowing through a connector. These "connectors" include a singular flow variable with several stream variables alongside it. For example, a "fluid port" is a connector that has the mass flow rate as the "flow" variable and enthalpy as the "stream" variable. Mass flow is what physically goes through the connector, while enthalpy is a property of the mass flow. This nuance in variable types is particularly important when considering the translation of Modelica models into FMIs and FMUs. FMIs can only import and export real input/output signals. These signals cannot retain the physical properties seen in Modelica, thus requiring special adaptors to translate them back into physical values for use in other Modelica models.

3.1 Adaptors

For connections between FMIs and other Modelica models within the Dymola platform, a set of standardized variables and adapters are needed to properly transmit energy values among subsystems. This is particularly true if the interconnection is between two physical models, such as a nuclear power plant and a turbine. This is because the physical models contain "ports," as shown in Figure 5.

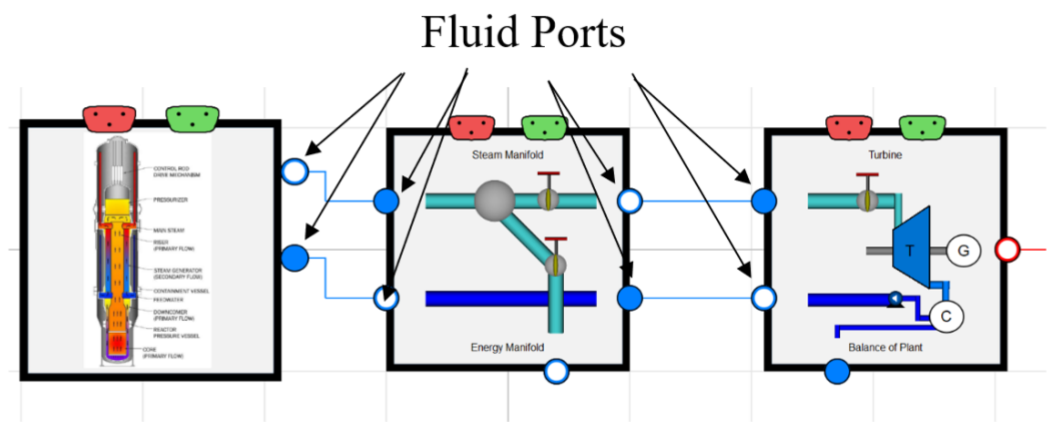


Figure 5. Fluid ports (note that "ports" are a container method in Modelica used to transfer several pieces of physics-based information within a single "connector").

Each fluid port contains:

- Mass flow (flow variable), m_flow
- Conditional enthalpy (stream variable), h_outflow
- Pressure, P
- Trace substance fraction (stream variable), Ci
- Mass fraction (stream variable), Xi.

Each electric port contains:

- Power (flow variable), W
- Frequency, f

To properly transition from ports to input and output signals, the individual components of the ports must be separated out and assumed to be either an input or an output. This is illustrated in Figure 6, with each fluid port being separated into its five constituent pieces (mass flow, enthalpy, pressure, mass fraction, trace substance fraction), and the electric port being separated into its two constituent parts (power and frequency).

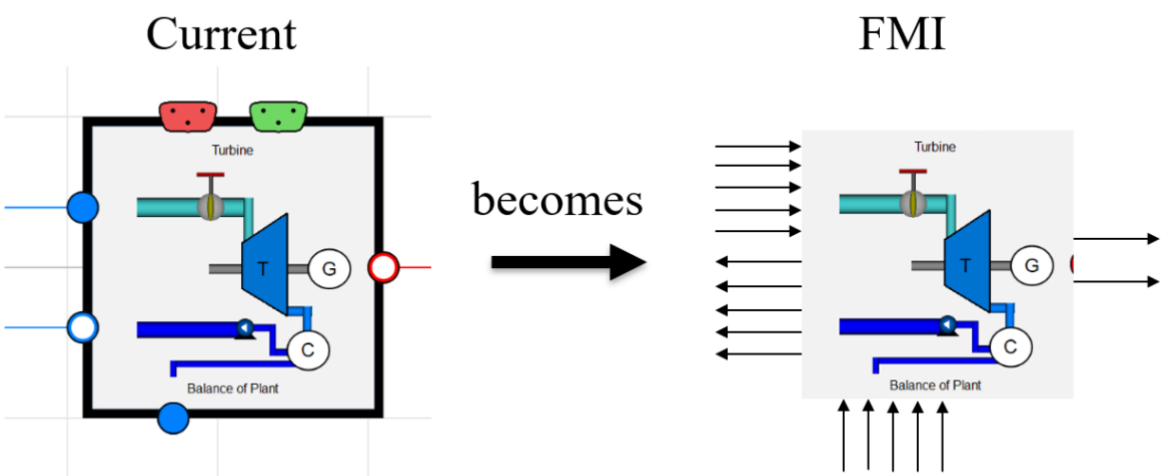


Figure 6. Transition from a Modelica physical model into an FMU.

In the HYBRID repository package structure, a set of adaptors was created and added to the utility folder to enable users to convert an existing Modelica model into a model ready for export as a FMU. The package placement is seen in Figure 7. Further details on each FMI template and interface are outlined in the next section.

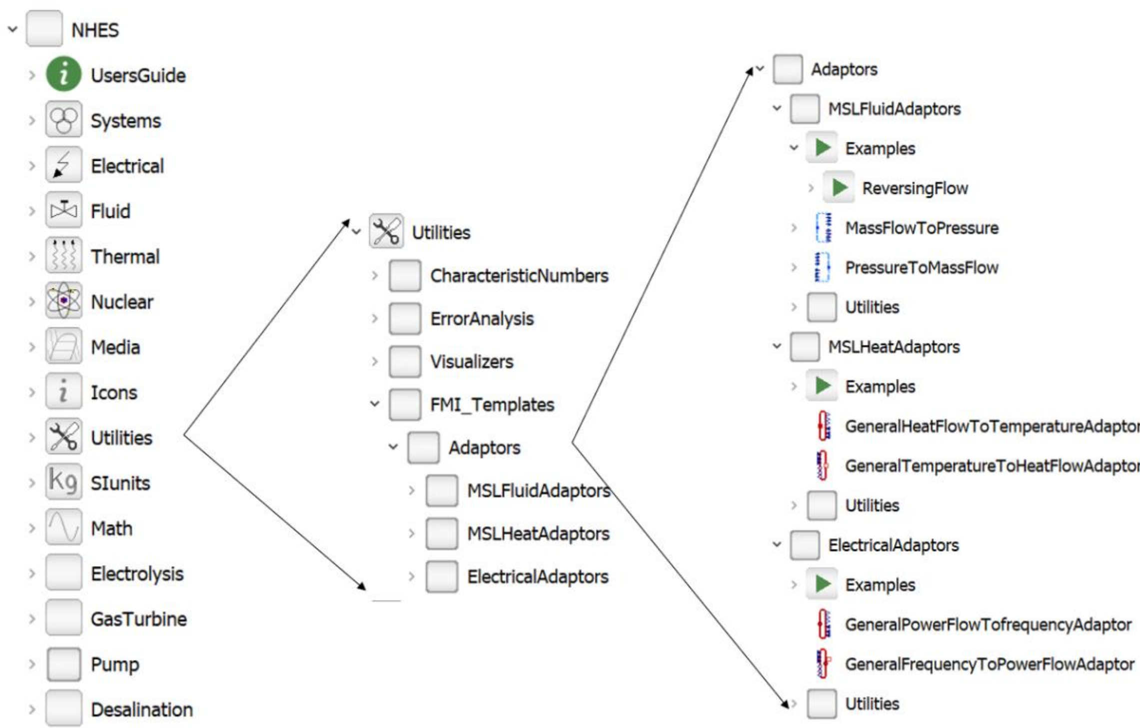


Figure 7. FMU template folder location within the larger Nuclear Hybrid Energy Systems (NHES) folder as part of the HYBRID Repository.

Fluid Port Adaptors

Within the Utility.FMI_Templates folder is an adaptor package created specifically for Modelica standard library fluid adaptors. This package is called MSLFluidAdaptors, and it models acausal to causal adaptors. This folder was created in unison with Modelon. Within this folder are two adaptors, shown in Figure 8. One is a “pressure to mass flow” adaptor, aptly named PressuretoMassFlow. This adaptor’s fluid port is best connected to a flow port of some sort (e.g., valves, resistance, pipe model). The inputs to this model are the pressure at the interface, upstream enthalpy from the causal side, upstream mass fraction from the causal side, and upstream trace composition from the causal side. The outputs are the acausal mass-flow rate, upstream enthalpy from the acausal side, upstream mass fraction from the acausal side, and upstream trace composition from the acausal side.

The second adaptor, called the MassFlowtoPressure adaptor, is a “mass flow to pressure” adaptor. This adaptor’s fluid port is best connected to a volume port (e.g., pressure sink, tank model). The inputs to this model are the causal mass-flow rate, upstream enthalpy from the causal side, upstream mass fraction from the causal side, and upstream trace composition from the causal side. The outputs are the pressure at the interface, upstream enthalpy from the acausal side, upstream mass fraction from the acausal side, and upstream trace composition from the acausal side.

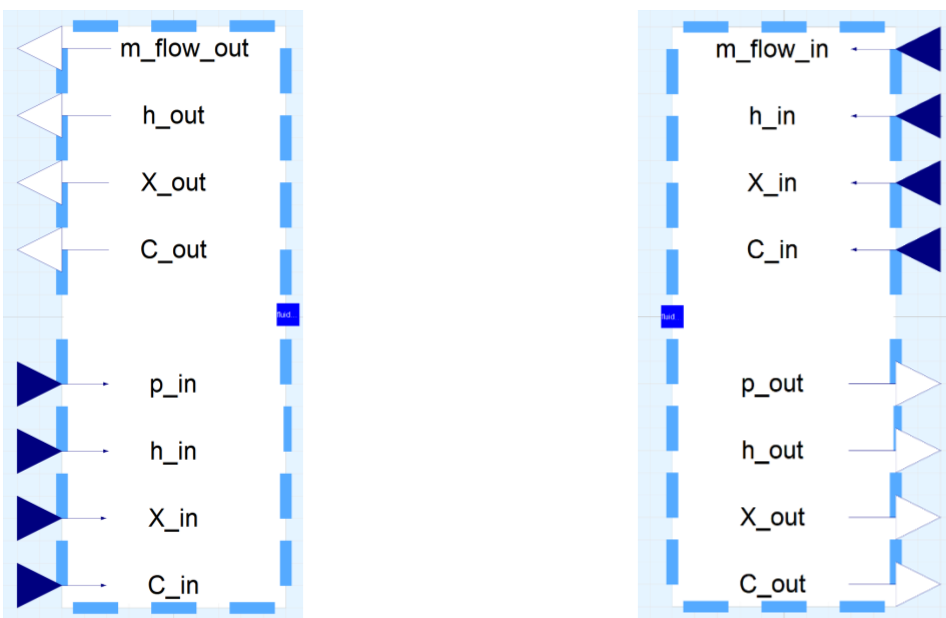


Figure 8. (Left) PressuretoMassFlow adaptor. This adaptor is best connected to a resistance port able to set the output mass flow rate. (Right) MassFlowtoPressure adaptor. This adaptor is best connected to a volume port able to set the output pressure. Note: Both of these adaptors were created by Modelon for use in the INL plug-and-play framework as part of an FMI/FMU course subcontract.

Figure 9 illustrates the usage of the two adaptors on a single model involving reversible flow. The model is of a series of two fully open valves connected to a volume source positioned between them, and a pressure source on either side of the valves. The system fluid is moist air from the Modelica standard library. The pressure source is then subjected to a 1 Hz oscillatory frequency on the pressure system, as would be present in a fast-moving pressure chamber, while the pressure sink remains at a constant pressure. In normal operations, this system will have a reversible flow, as the pressure of the source oscillates about the pressure sink's pressure. Such scenarios have been challenging to meet with FMIs and FMUs, due to the reversible nature of the mass flow. With the new adaptors, this reversible flow issue can be met.

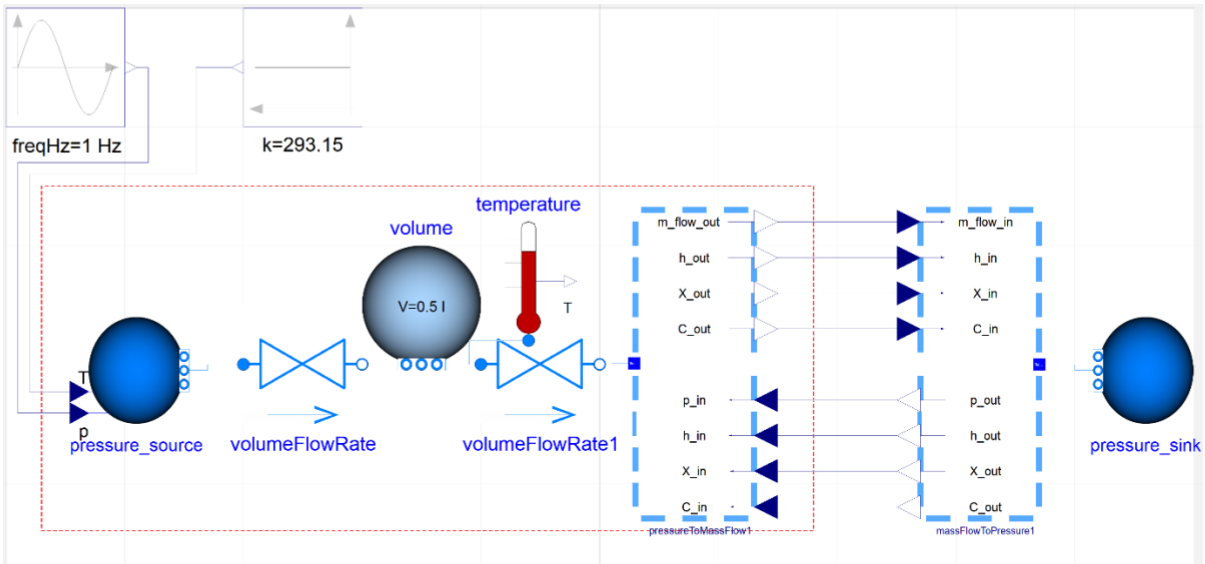


Figure 9. An example (using adaptors) involving two pressure sources using moist air, one of which oscillates in pressure, causing a mass flow reversal. The unit in the red box will become an FMU. (k=Temperature input to pressure flow boundary; freqHz = pressure frequency oscillation placed on pressure_source).

The unit inside the red box in Figure 9 was exported as both a model exchange and co-simulation FMU, as shown in Figure 10 and Figure 11. All systems were then run for 10 seconds of simulation time. The results are depicted in Figure 12.

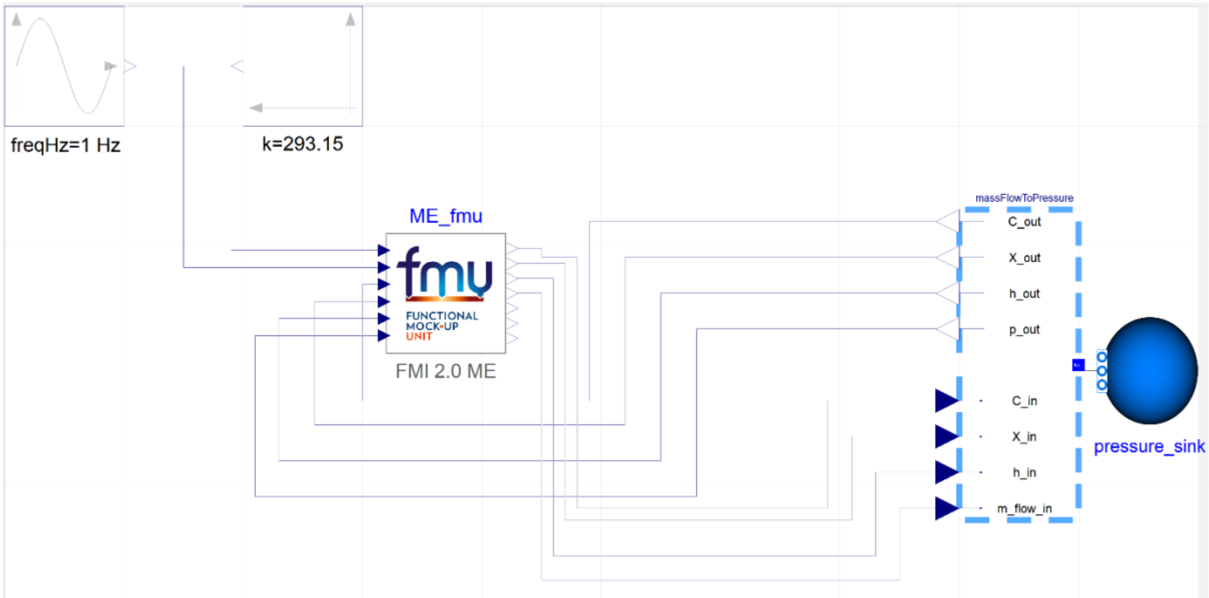


Figure 10. Example of a reversible flow using two pressure sources, moist air, and a model exchange FMU. (k=Temperature input to pressure flow boundary; freqHz = pressure frequency oscillation placed on pressure_source).

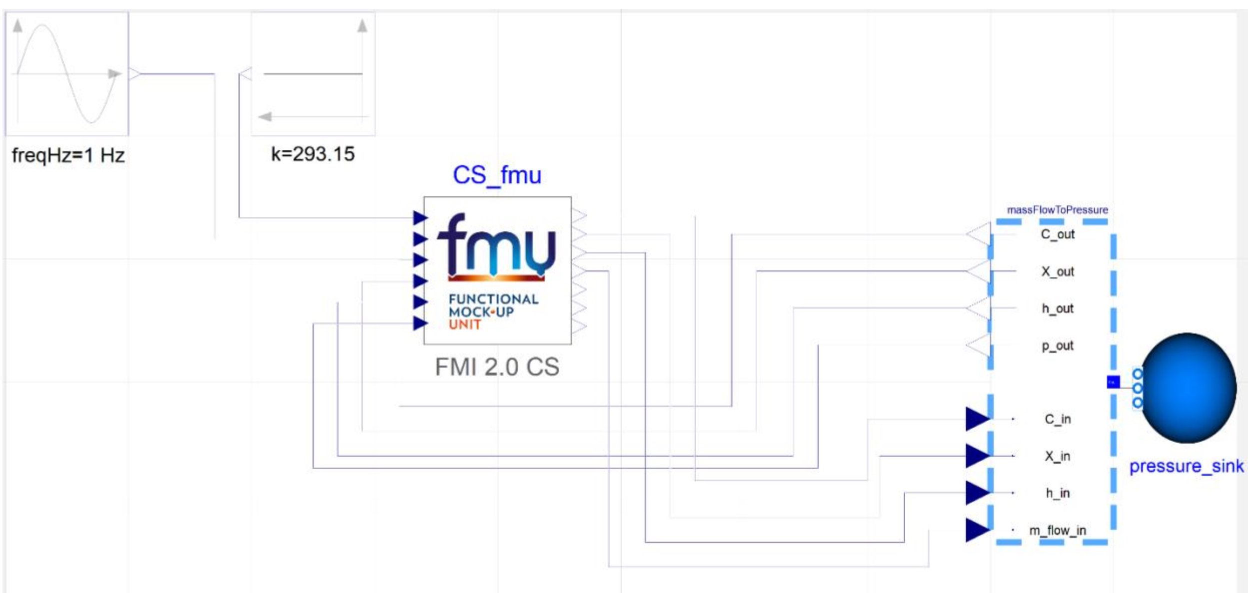


Figure 11. Example of a reversible flow using two pressure sources, moist air, and a co-simulation FMU. (k=Temperature input to pressure flow boundary; freqHz = pressure frequency oscillation placed on pressure_source).

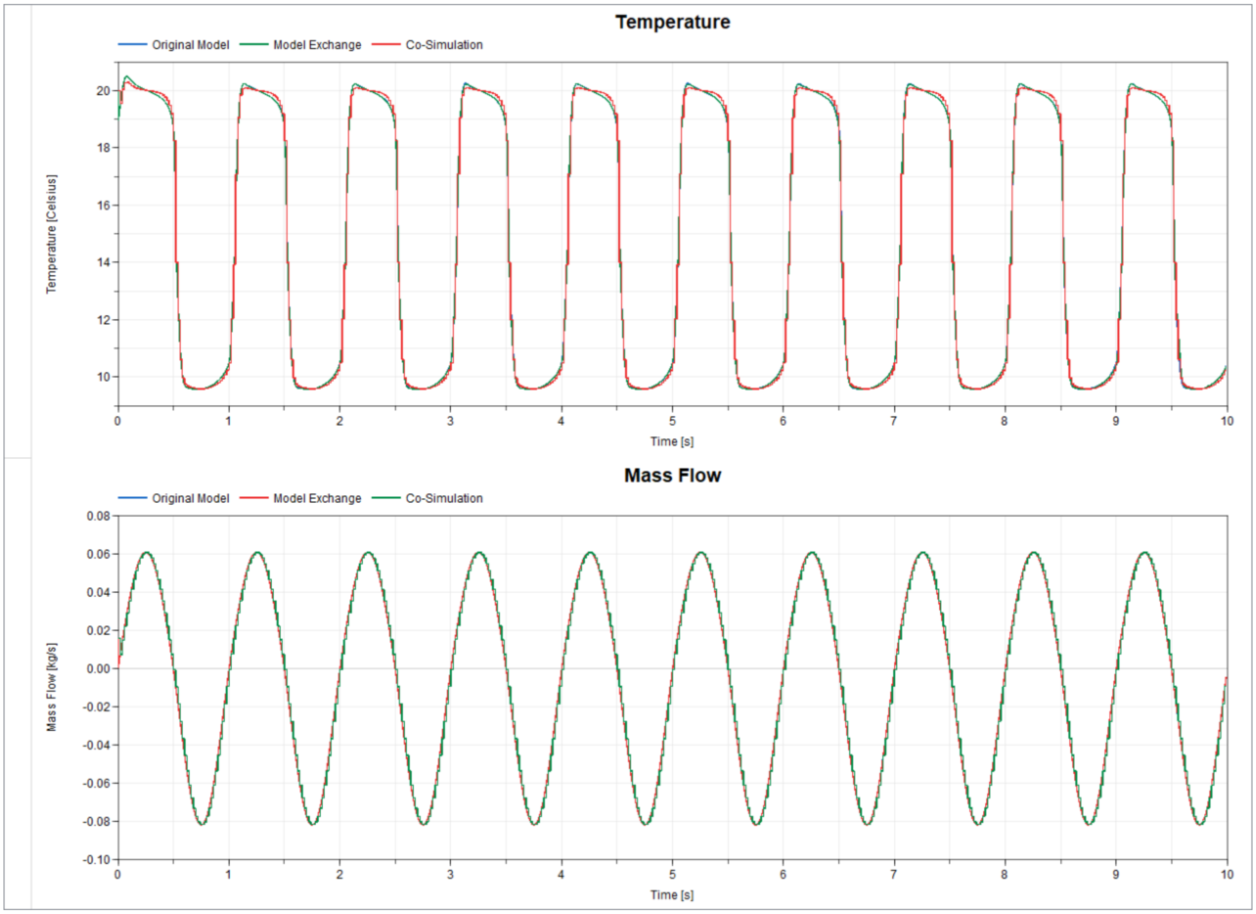


Figure 12. Comparison of mass flow to the pressure sink across the original model, model exchange FMU, and co-simulation FMU (timestep = 0.02 seconds).

The results showcase that, by utilizing the fluid port adaptors, reversible flow is achievable in both the model exchange and co-simulation FMIs/FMUs. However, these capabilities carry additional overhead in regard to central processing unit (CPU) time. Model exchange for this particular model increases the simulation time from 2.364 to 6.749 seconds. Co-simulation with a 0.02-second communication interval took 10.795 seconds. Even so, co-simulation still shows the largest error, due to co-simulation models inherently being an explicit solve. However, given a sufficiently small communication interval, and depending on the dynamics of the model, an acceptable solution can be achieved.

Thermal Port Adaptors

Within the Utility.FMI_Templates folder is an adaptor package created specifically for Modelica standard library thermal adaptors. This package is called MSLHeatAdaptors, and it models acausal to causal adaptors. These models were initially made available in the Modelica standard library and have been augmented with additional examples and placed within the NHES package for ease of access relative to other FMI adaptors. Two adaptors are included, one being the GeneralHeatFlowToTemperature adaptor. The inputs to this adaptor are the acausal heat flow port, causal heat flow, and optional causal first and second derivatives of heat flow. The outputs are the temperature and the optional first and second derivatives of temperature.

The second adaptor is the GeneralTemperaturetoHeatFlow adaptor. The inputs to this adaptor are the acausal heat flow port, causal temperature, and optional causal first and second derivatives of temperature. The outputs are the heat flow and the optional first and second derivatives of heat flow.

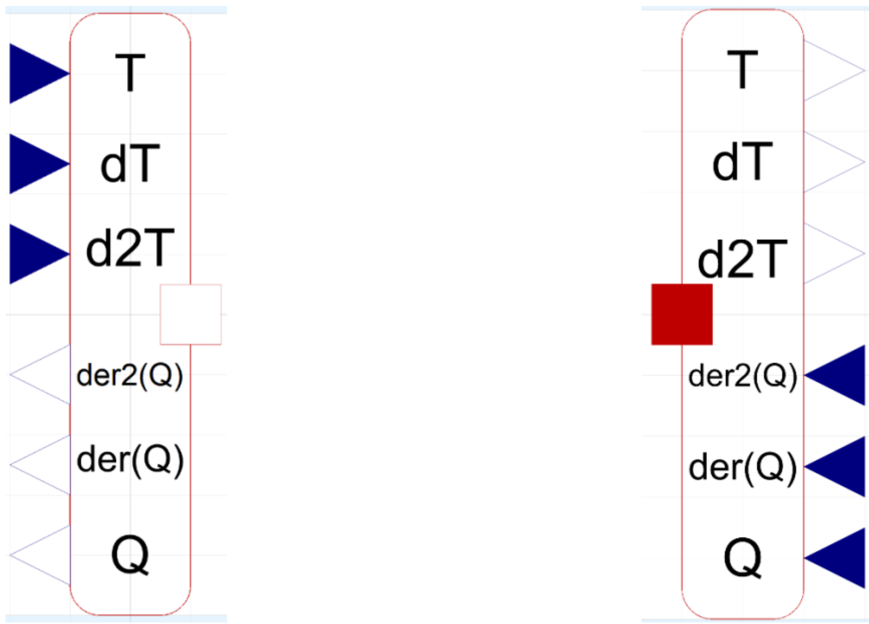


Figure 13. (Left) GeneralTemperatureToHeatFlow adaptor for use in the INL plug-and-play framework. (Right) GeneralHeatFlowToTemperature adaptor for use in the INL plug-and-play framework. (T=temperature, dT = first derivative of temperature, d2T = second derivative of temperature, Q = heat flow, der(Q) = first derivative of heat flow, der2(Q) = second derivative of heat flow.) Note: only T and Q are required the derivative values are optional for stability.

Figure 14 illustrates the usage of the two adaptors in a single model involving two methods of heat port usage. The upper model demonstrates how to export two heat capacitors and connect them together in a target system. This requires that one of the capacitors (here, DirectCapacity) be defined to have states, and that the temperature and derivatives of the temperature are provided in the interface. The other capacitor (here: InverseCapacity) requires a heat flow in accordance with the provided input temperature and derivative of temperature. The lower part demonstrates how to export a conduction element that only requires temperatures for its conduction law, and connects this conduction law to both the heat capacitors in a target system. Both models will be translated into a model exchange and co-simulation model, as shown in Figure 15, Figure 16, Figure 17, and Figure 18. The results are compared in Figure 19 and Figure 20.

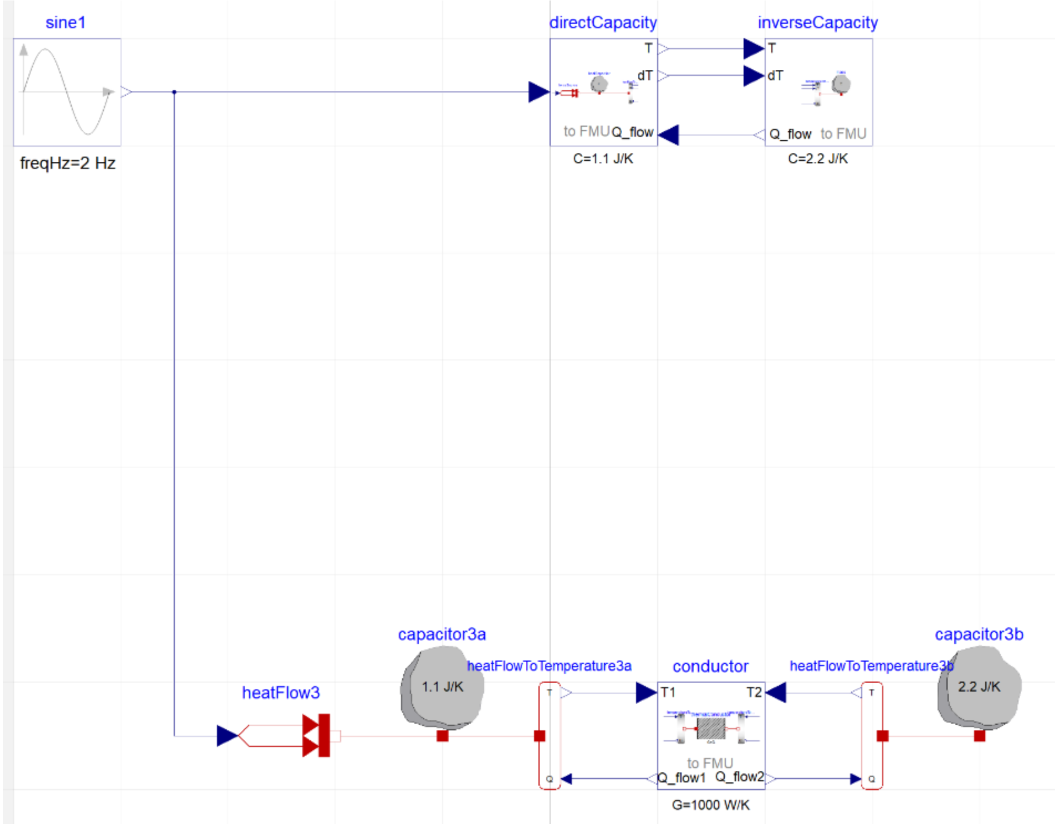


Figure 14. Example meant to demonstrate the FMU variants available with the thermal FMU adaptors. The upper part demonstrates how to export two heat capacitors and connect them together in a target system. The lower part demonstrates how to export a conduction element that only requires temperatures for its conduction law, and connects this conduction law to both heat capacitors in a target system.

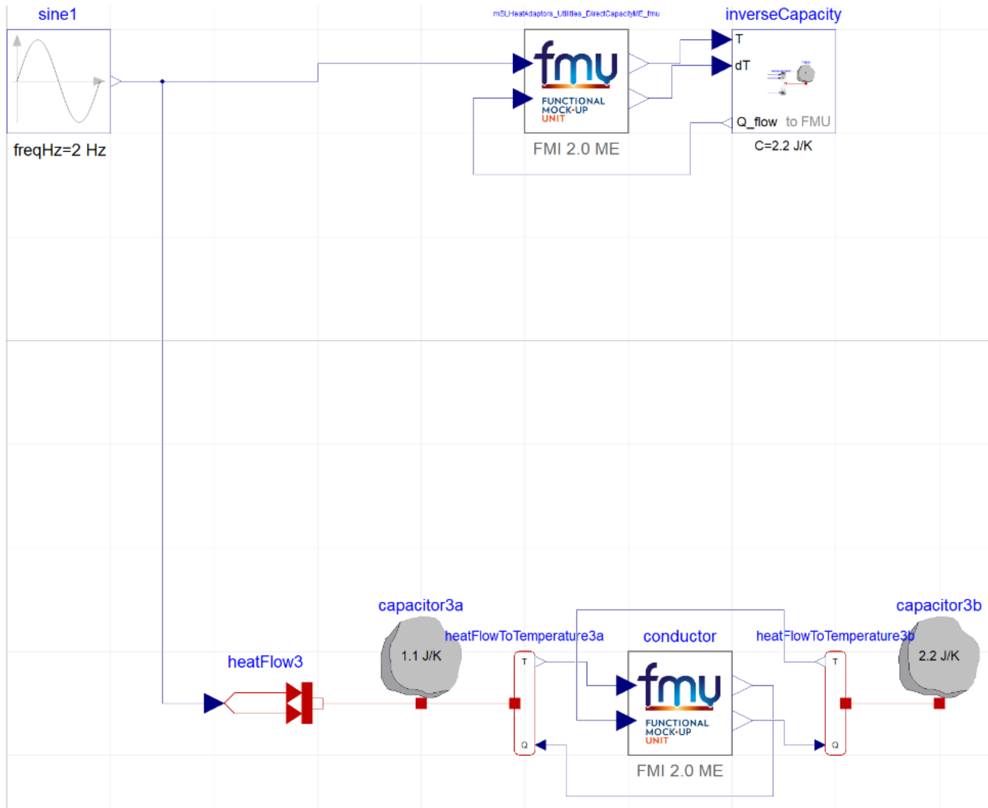


Figure 15. Demonstration of an FMU variant example that uses model exchange FMUs for the thermal heat port adaptors.

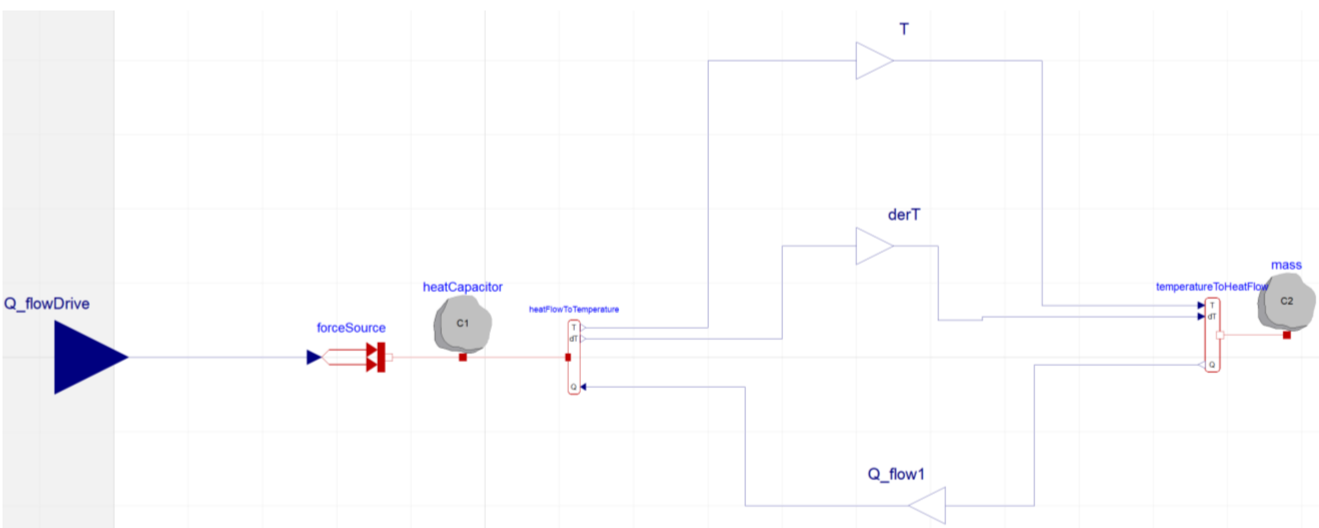


Figure 16. Collapse of the upper part of Figure 14 into a single FMU for co-simulation. This is required because the frequency between the direct and inverse conduction problem is so fast that a single cut between the two could not be made without instabilities occurring.

Figure 17. Upper model of Figure 14 connected with the combined direct/inverse co-simulation FMU.

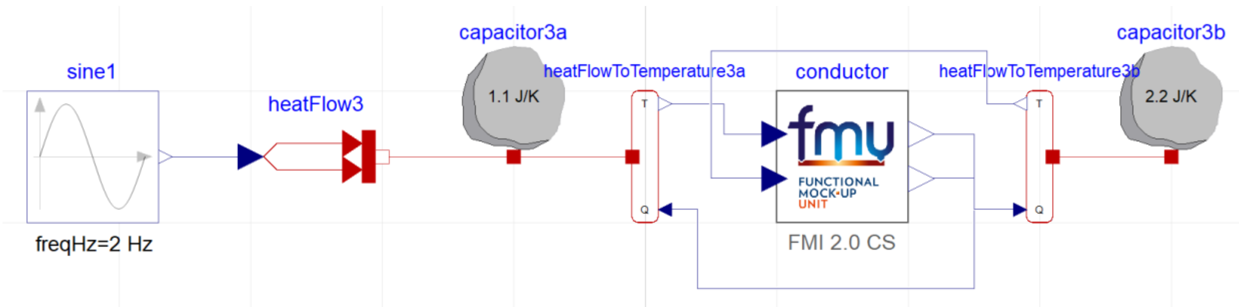


Figure 18. Lower model of Figure 14, co-simulation FMU.

The results showcase that, by utilizing the thermal adaptors, acceptable results in terms of the heat flow between models can be achieved via both model exchange and co-simulation FMIs/FMUs. However, these capabilities carry additional overhead in regard to CPU time, as was the case in the fluid port scenario. The co-simulation mode, though theoretically easier to export to external codes thanks to its inclusion of a solver, required the most augmentation, due to the fast system dynamics. This limitation required the FMU to include both the direct and inverse capacitors within a singular model, as shown in Figure 16, otherwise a divergent solution was acquired. Even with this additional step, the co-simulation solve still showed the largest error, as depicted in Figure 19 and Figure 20. This is because co-simulation models are inherently an explicit solve. However, given a sufficiently small communication interval and depending on the dynamics of the model, an acceptable solution can be achieved.

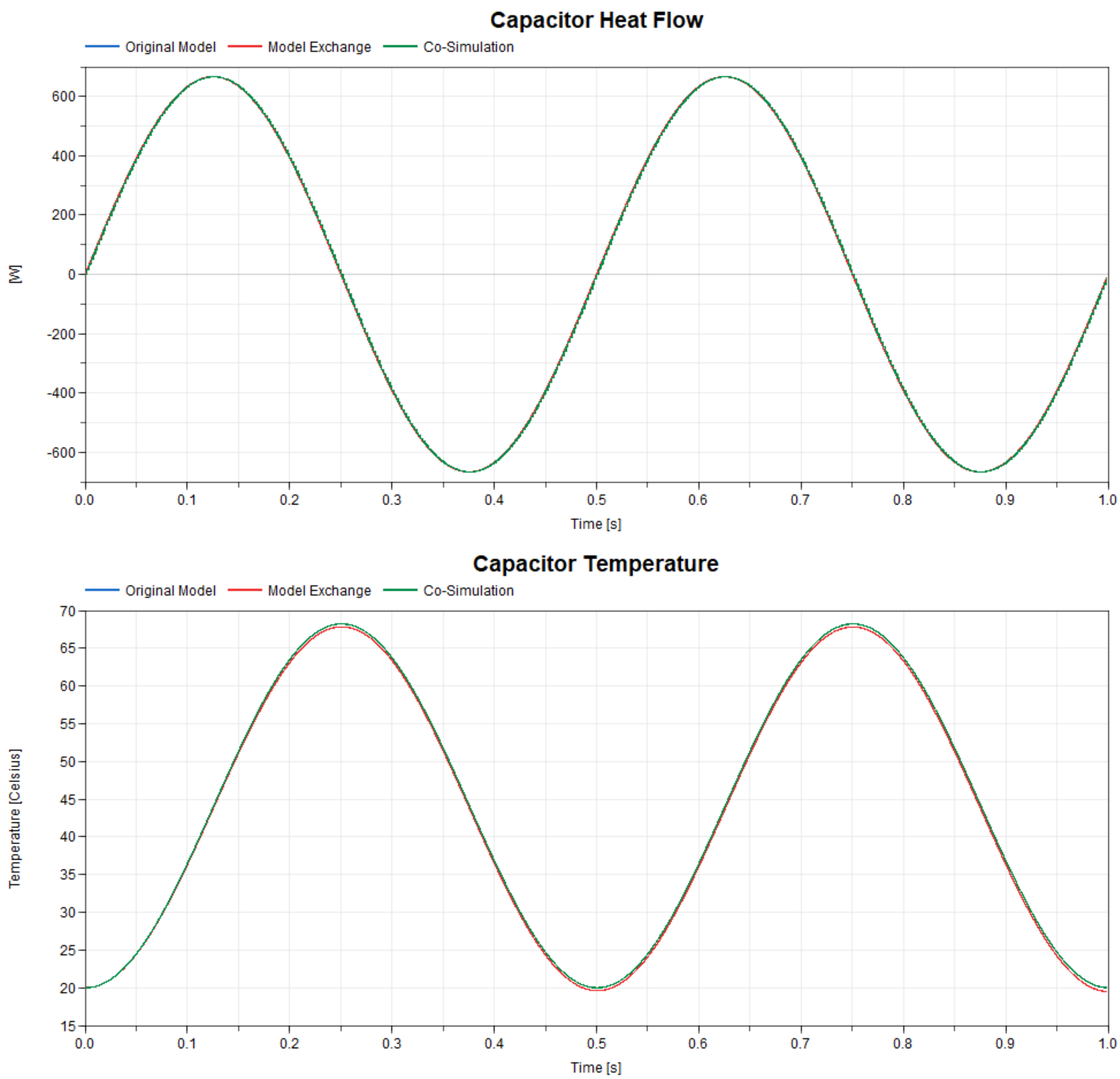


Figure 19. Direct/inverse simulation results for the original, model exchange, and co-simulation (communication interval: 0.002 seconds) FMU runs. (Top) Heat flow into the capacitor. (Bottom) Capacitor 3b temperature.

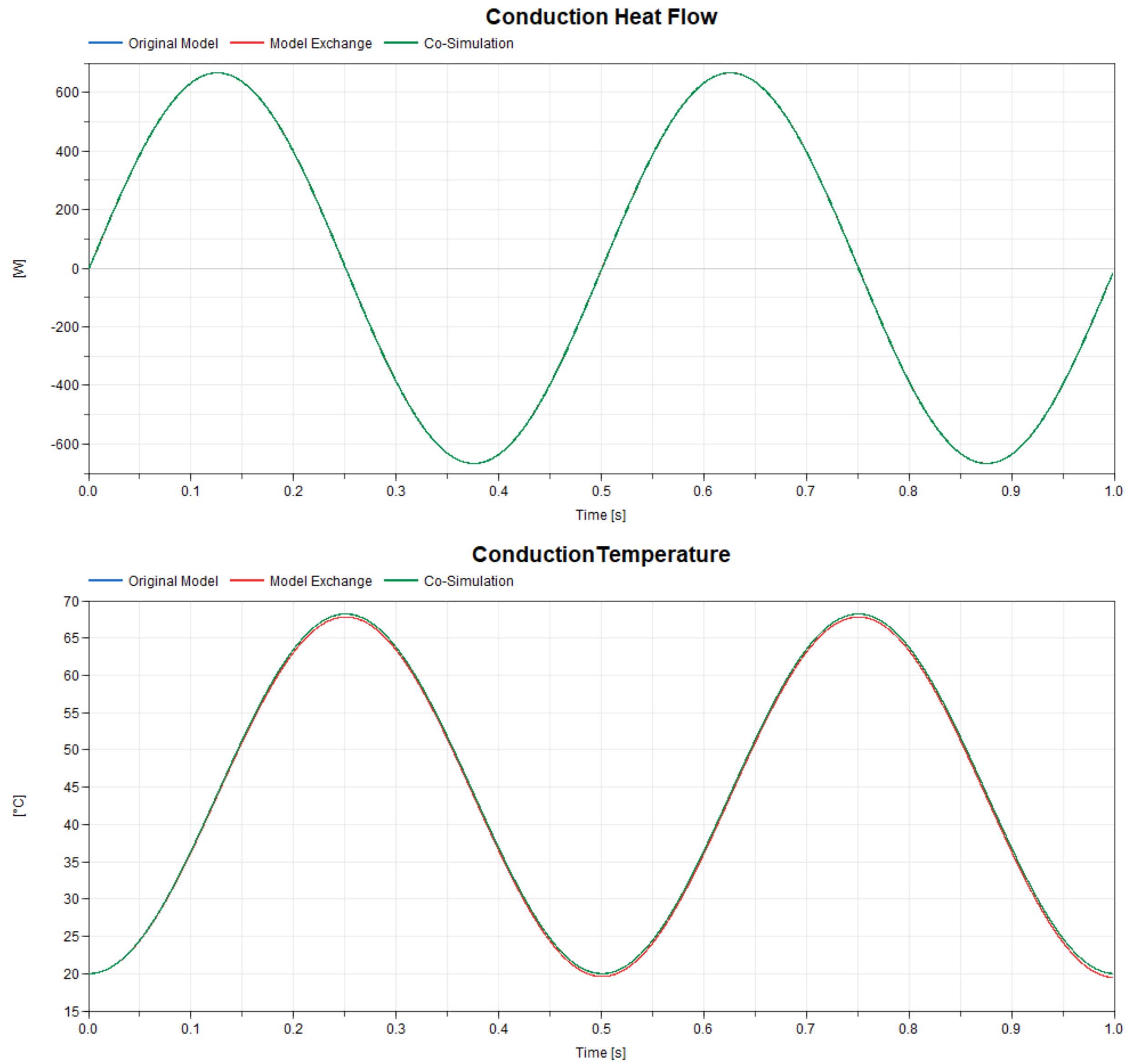


Figure 20. Conduction (lower model) simulation results for the original, model exchange, and co-simulation (communication interval: 0.002 seconds) FMU runs. (Top) Heat flow into the capacitor. (Bottom) Capacitor 3b temperature.

Electrical Port Adaptors

Within the Utility.FMI_Templates folder is a package created specifically for electrical adaptors. This package is called ElectricalAdaptors, and it models acausal to causal adaptors. Two adaptors are included (see Figure 21), one being the GeneraPowerFlowToFrequency adaptor. The inputs to this adaptor are the acausal electrical port, causal power, and optional causal first and second derivatives of power. The outputs are the frequency and the optional first and second derivatives of frequency.

The second adaptor is the GeneralFrequencyToPowerFlow adaptor. The inputs to this adaptor are the acausal electrical port, causal frequency, and optional causal first and second derivatives of frequency. The outputs are the power flow and the optional first and second derivatives of power flow.

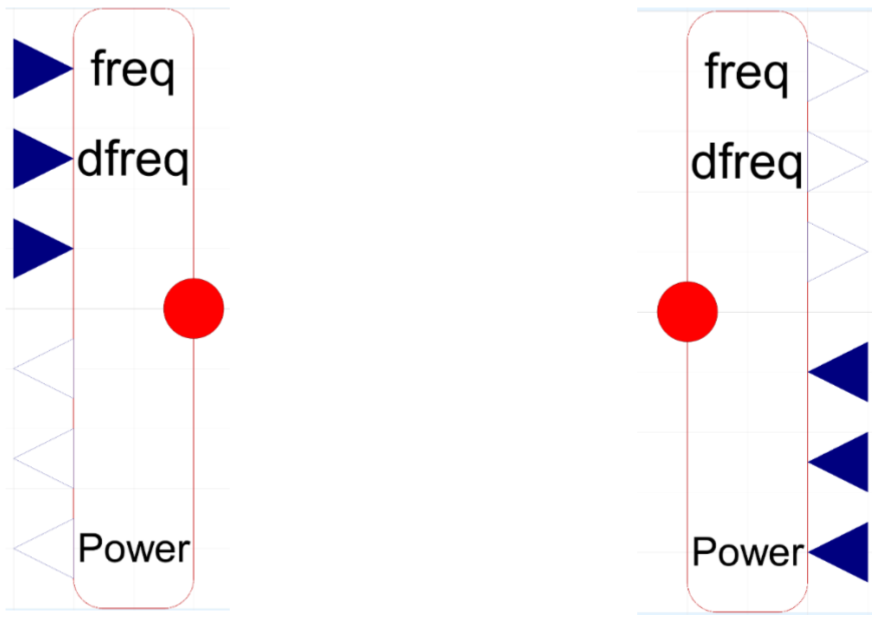


Figure 21. (Left) GeneralFrequencyToPowerFlow adaptor for use in the INL plug-and-play framework. (Right) GeneralPowerFlowToFrequency adaptor for use in the INL plug-and-play framework. (Red circle represents the electrical port, with inputs and outputs equal to the aforementioned variables in the section).

3.2 FMI Construction Guide

To properly create and utilize a model as an FMI/FMU, the following five steps must be accomplished.

1. Model Preparation
2. Adaptors
3. Export
4. Import
5. Simulation

This section seeks to provide step-by-step guidance on how each of these steps can be accomplished.

Model Preparation

For a model to become a usable FMU, it must contain all the required input/output variables within itself, aside from those designated to come from an outside model. This requirement means that, for units featuring interchangeable control systems, a particular control system must be declared via a top-level declaration in an example style file. An example of both an incorrect and a correct file format for a natural gas peaking turbine are shown in Figure 22 and Figure 23, respectively. In Figure 22 the natural gas turbine model includes the basic constituent parts for its simulation (compressor, turbine, combustion chamber, inertial generator shaft, generator, fuel controllers, and geometrical data assumptions). But in Dymola if this model is run it is missing a selected control system for the sensor and actuator bus as this is a "replaceable" component. Meaning the model needs to be imported within a new model to allow us to select the control system.

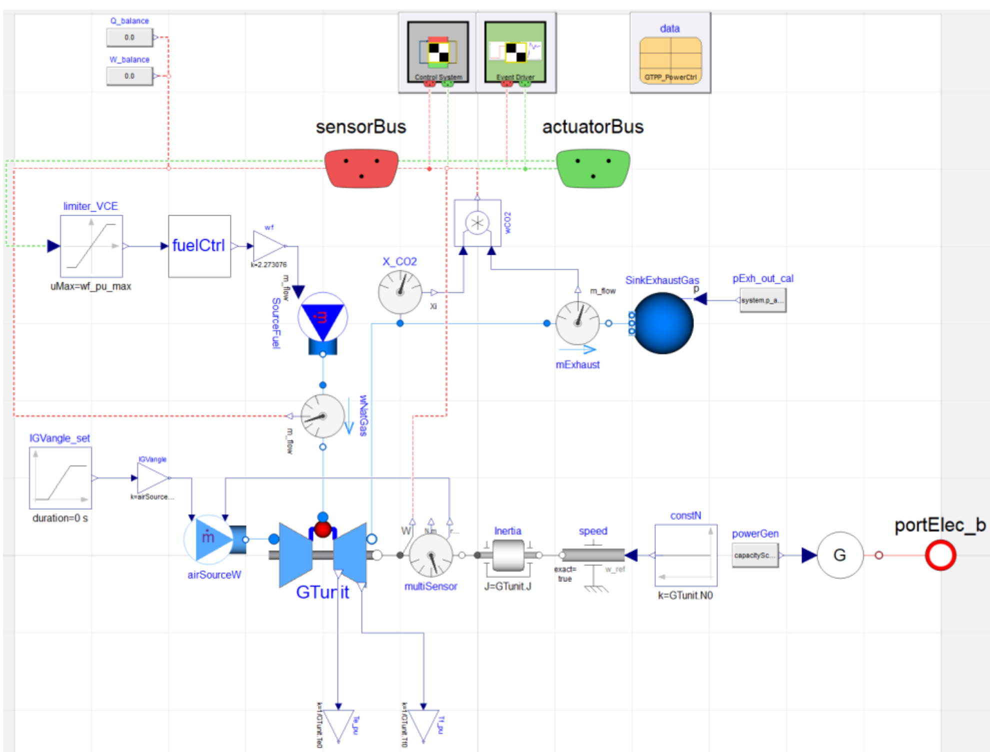


Figure 22. Incorrect level for proper export as FMI/FMU. Control system has not been declared and is replaceable from a higher level within the HYBRID repository.

This placement within a new model is shown in Figure 23. In this case the model from Figure 22 is placed within a new model, the electric port is attached to a frequency boundary condition and if we double click the natural gas turbine icon the table on the left pulls up where the control system can be selected and values can be imported for system size and maximum power output. Once this control system is selected the model is now ready to begin model preparation for FMI/FMU exportation. Note: the control system selected will be the control system exported with the FMI/FMU.

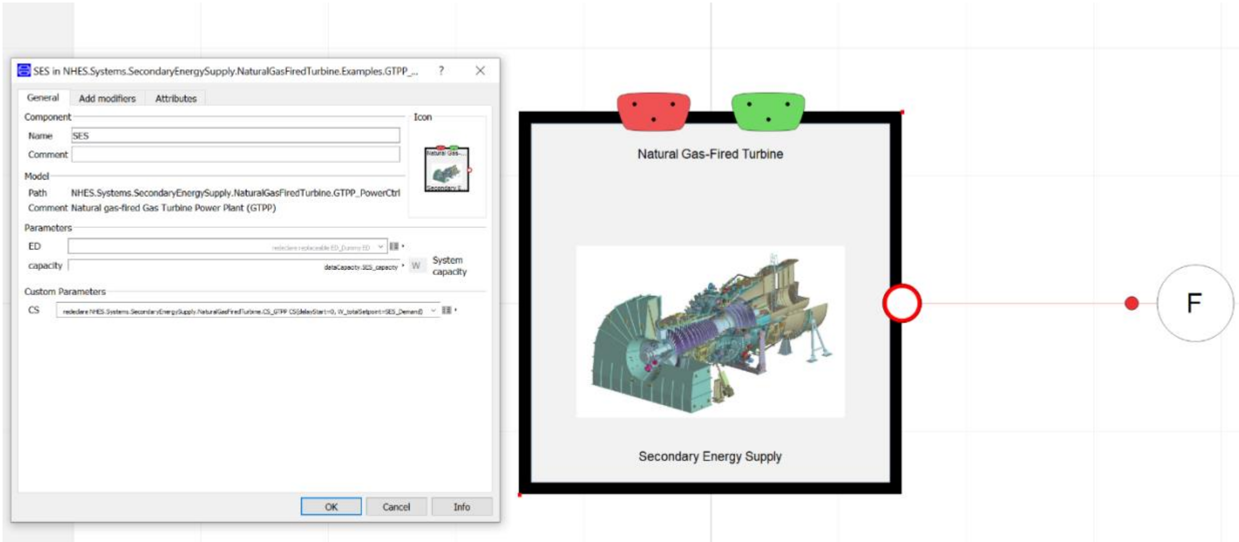


Figure 23. Correct level from which to begin FMI/FMU preparation. Control system has been selected via the drop-down menu available in the custom parameters section, shown on the left. (Red dots are electrical flow ports).

Adaptors

Now that the proper model has been created, the variables designated to come from outside the FMU must be declared as a real "input" or "output" variables, as demonstrated in Figure 6. To accomplish this, the adaptors can be employed in the manner previously outlined. For the natural gas turbine example illustrated in Figure 23, the electric port must be converted into real inputs/outputs using the PowerFlowToFrequency adaptor described in the previous section. In addition, the control system of the natural gas turbine requires a top-level demand signal to communicate the grid demand at each time interval. To implement such communication into the model, an additional real input variable, "SES_Demand," was created. With the adaptor and new input signal created, the model took the form depicted in Figure 24, and is ready for export as an FMU. This procedure of using an adaptor to transform ports into their real input/output components, and creating additional inputs/outputs for declared variables, works well for simple models and models intended for use in model exchange mode. For complex models planned for simulation in co-simulation mode, use of adaptors may prove challenging if the initialization of the models is not well-defined. This is due to the explicit nature of co-simulation modeling. Further details on this will be given in later sections of this report.

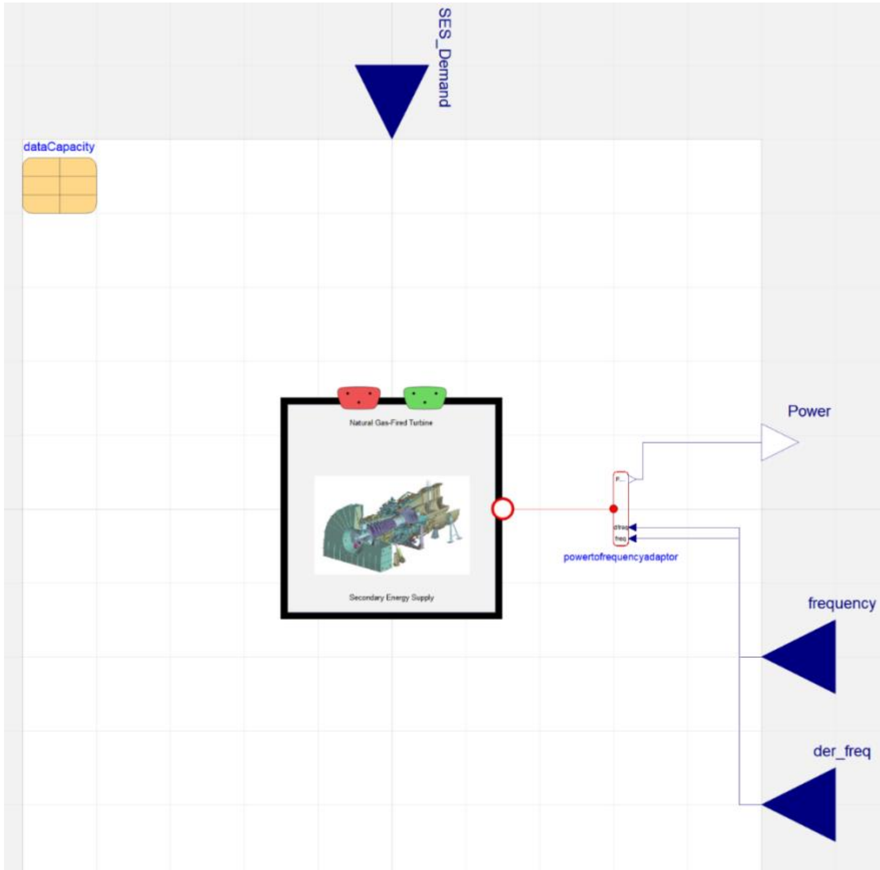


Figure 24. Preparing a natural gas turbine to be converted into an FMU. The inputs into the system are the peaking demand and the connection points for electricity backflow into the turbine model. The output is the electrical power as a real value.

Export

Dymola offers several ways to export a model as an FMU, as shown in Figure 25. The FMU can include three different types of export: model exchange, co-simulation using the CVode solver, and co-simulation using various Dymola solvers.

In model exchange, the component model will be exported without a solver, as it is assumed that the importing tool will provide the solver. For co-simulation models, CVode and Dymola solvers can be exported with the component model for use within other models. In general, CVode solvers are sophisticated enough for most models, and export can be selected in either C-code or binary code, depending on the purchased Dymola license. In the event a particular Dymola solver is required to compile a component model, the co-simulation export can only be accomplished as a binary, thus protecting the proprietary solver information held by Dassault systems. However, binaries are operating-system dependent, so care must be taken to ensure that export of binary FMUs is conducted on the same operating system as the planned importing tool.

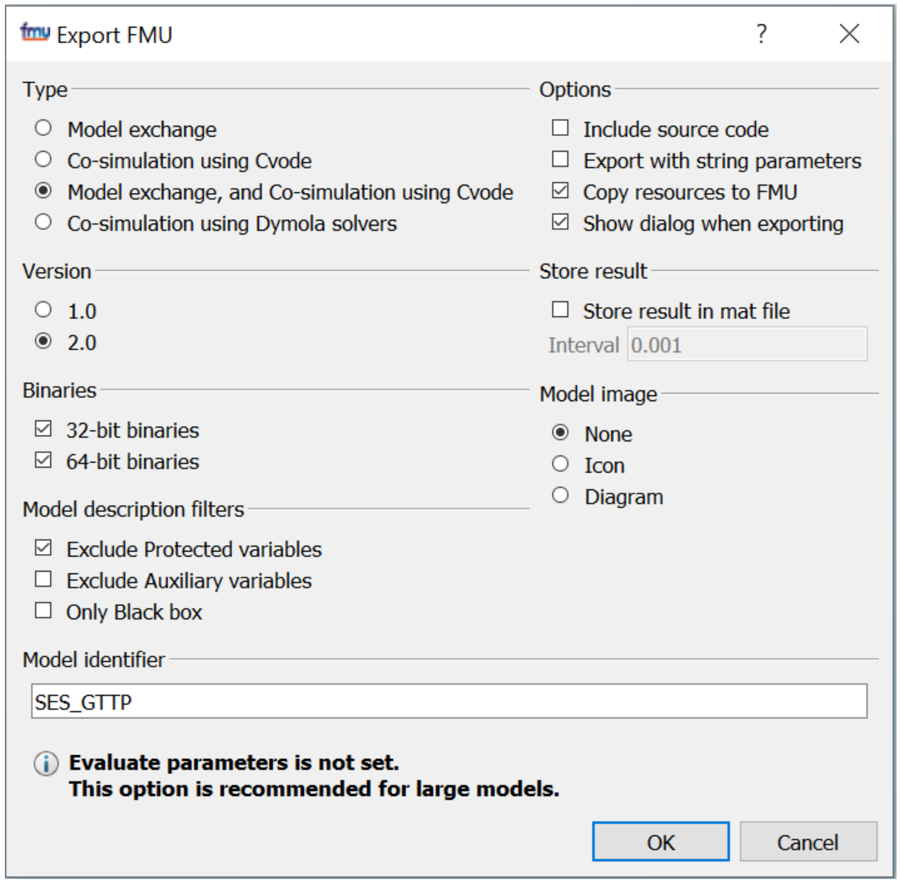


Figure 25. Export settings from Dymola 2021x.

Import

Once the model has been exported and an FMU created, the model will be present as an .fmu file. In the case of the natural gas turbine, it will be called "SES_GTTP.fmu." To import this file in Dymola, click File → Open → Import FMU, as shown in Figure 26.

The FMU can be imported in either model exchange or co-simulation mode, as per Figure 27. This selection should be consistent with the export options included in the FMU. If the desired import mode is different than the model of the original FMU, the imported FMU will fail.

Including the "structured declaration of variables" option retains the structured file tree of variables that were present in the original model, enabling the user to look inside the FMU as though it were the original Dymola model. If this option is not selected, a single large list featuring all the variables available for access will be made available to the user.

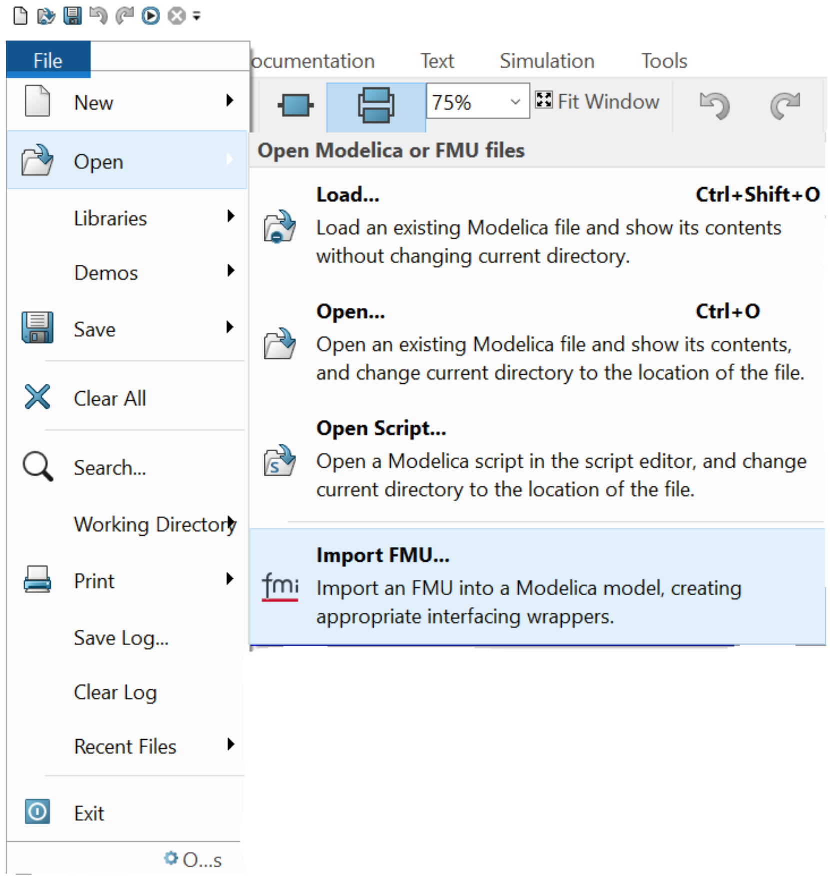


Figure 26. Importing FMU steps in Dymola 2021x.

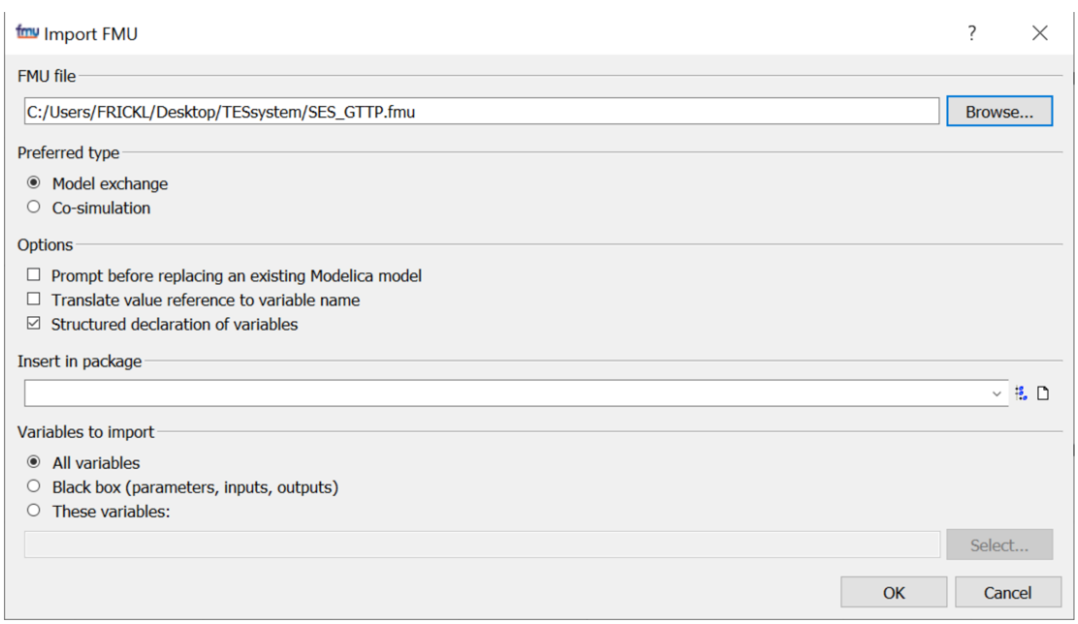


Figure 27. Import settings from Dymola 2021x.

Simulation

After the import step, the model can be used in place of the main component, as shown in Figure 28. In the system, it is important to ensure that all materials, initial conditions, nominal conditions, and parameter setpoints are consistent across the boundaries between the FMU and the rest of the model. This is particularly important because FMUs take real inputs and provide the surrounding model with outputs that have no physical constraint placed upon them. This reduces the number of checkpoints that the underlying application program interface (API)s has in order to ensure a consistent model. This places more onus on the engineers/researchers.

When using model exchange, the model will act similarly to the primary model, as the equation set remains exposed to the underlying import tool solvers. Conversely, in co-simulation mode, a specified "communication step" size must be selected, at which point the models will export results for communication with the surrounding external models. Selecting a small enough communication step to ensure that all the dynamics are captured is critical, but selecting a time-step communication interval that is too small greatly reduces the system's simulation speed.

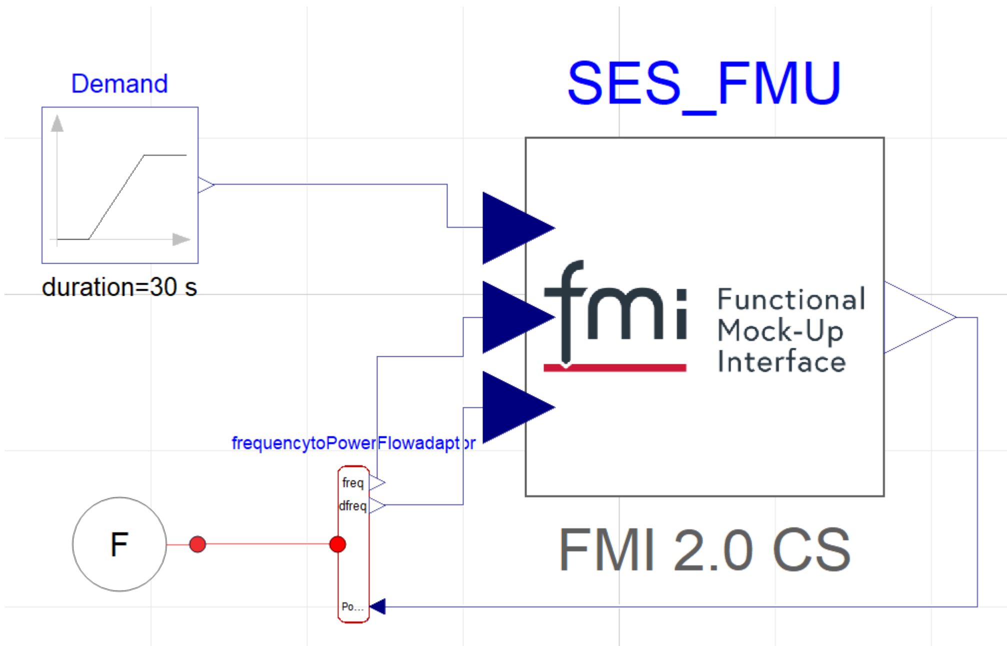


Figure 28. Proper import and use of a co-simulation FMU in Dymola.

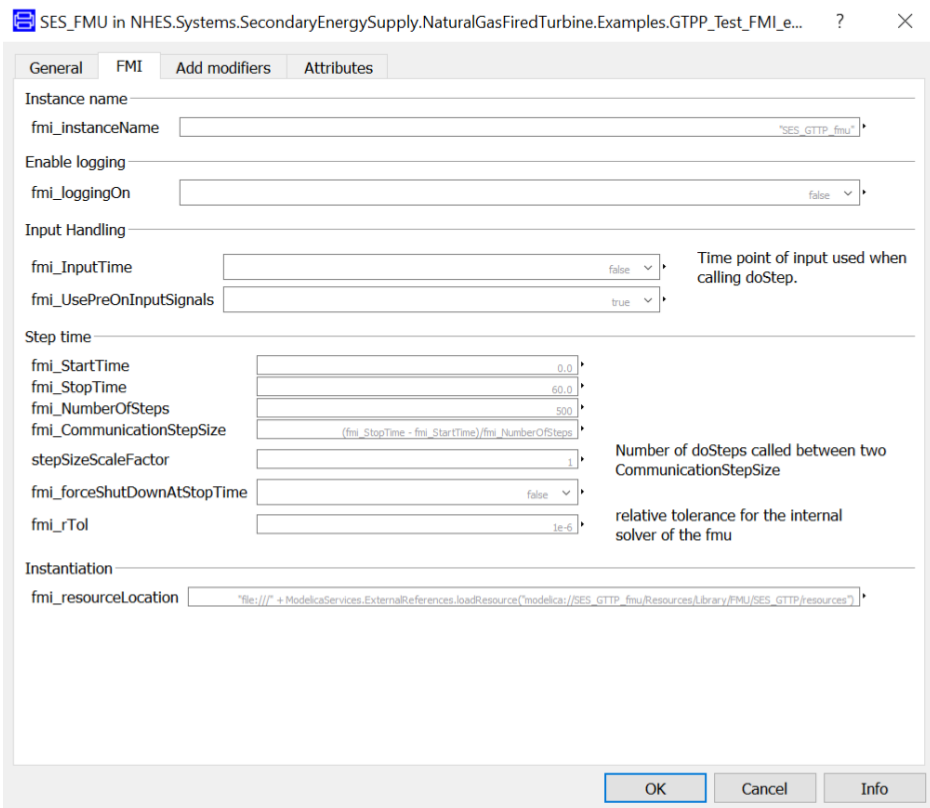


Figure 29. FMI settings for the natural gas turbine FMI/FMU in co-simulation mode. The communication interval was every 0.12 seconds, with an internal solver tolerance of 1e-6. The internal solver was the Dymola specific DASSL solver.

To test the FMU, the physical model was run in co-simulation, model exchange, and normal Modelica-only mode. The resulting turbine output is illustrated in Figure 30. For the three aforementioned modes, all the models converged to the same solution over the 60-second simulation time, with real-time simulation speeds of 1.316, 0.147, and 0.064 seconds, respectively. The co-simulation FMI settings are shown in Figure 29. In all cases, the simulation speeds are slower for FMU representations. This can be attributed to the increased overall number of variables that must be simulated due to the need for additional boundary blocks to accommodate real inputs/outputs. In addition, for co-simulation, the limiter on simulation speed is directly impacted by the communication step size and the nonlinearity of the coupled system. For example, increasing the communication step size from every 0.12 seconds to every second reduces simulation time from 1.316 seconds to 0.514 seconds. However, as demonstrated in Figure 30, this comes at the price of accuracy. Therefore, it is essential that, for co-simulation models, the communication step occur at points with slow-moving physics in order to allow the system a larger communication step size.

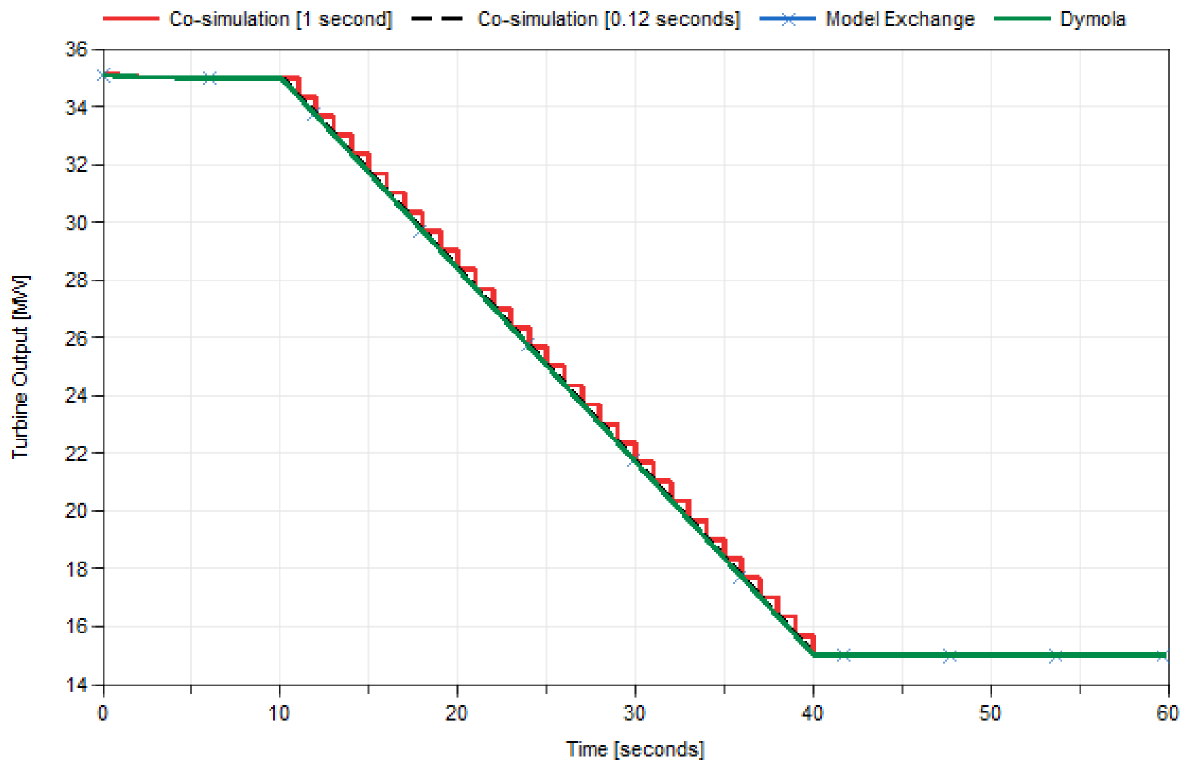


Figure 30. Comparison of Dymola model results to co-simulation and model exchange FMU results. Communication intervals for co-simulation = 0.12 seconds and 1 second.

3.3 Turbine Replacement Example

The creation of FMUs makes it possible to take a model from one coding language and encapsulate it in a standardized format for use within another coding language. To test this functionality with the more complicated fluid equation set of water, a natural circulation small modular reactor (SMR) set was chosen. The modeling set, shown in Figure 32, includes the reactor, energy manifold, turbine generator, and electric grid—all modeled in the Modelica language. The turbine generator set was then converted from a Modelica model into an FMU to ensure that all the proper data were input into and transferred between the models. The initial step was to implement the adaptors (discussed in the previous section) that transform the fluid ports into constituent real outputs, as shown in Figure 31. The progression of translation is shown in Figure 32, going from the Modelica-only model to a model exchange FMU that is then included in the model.

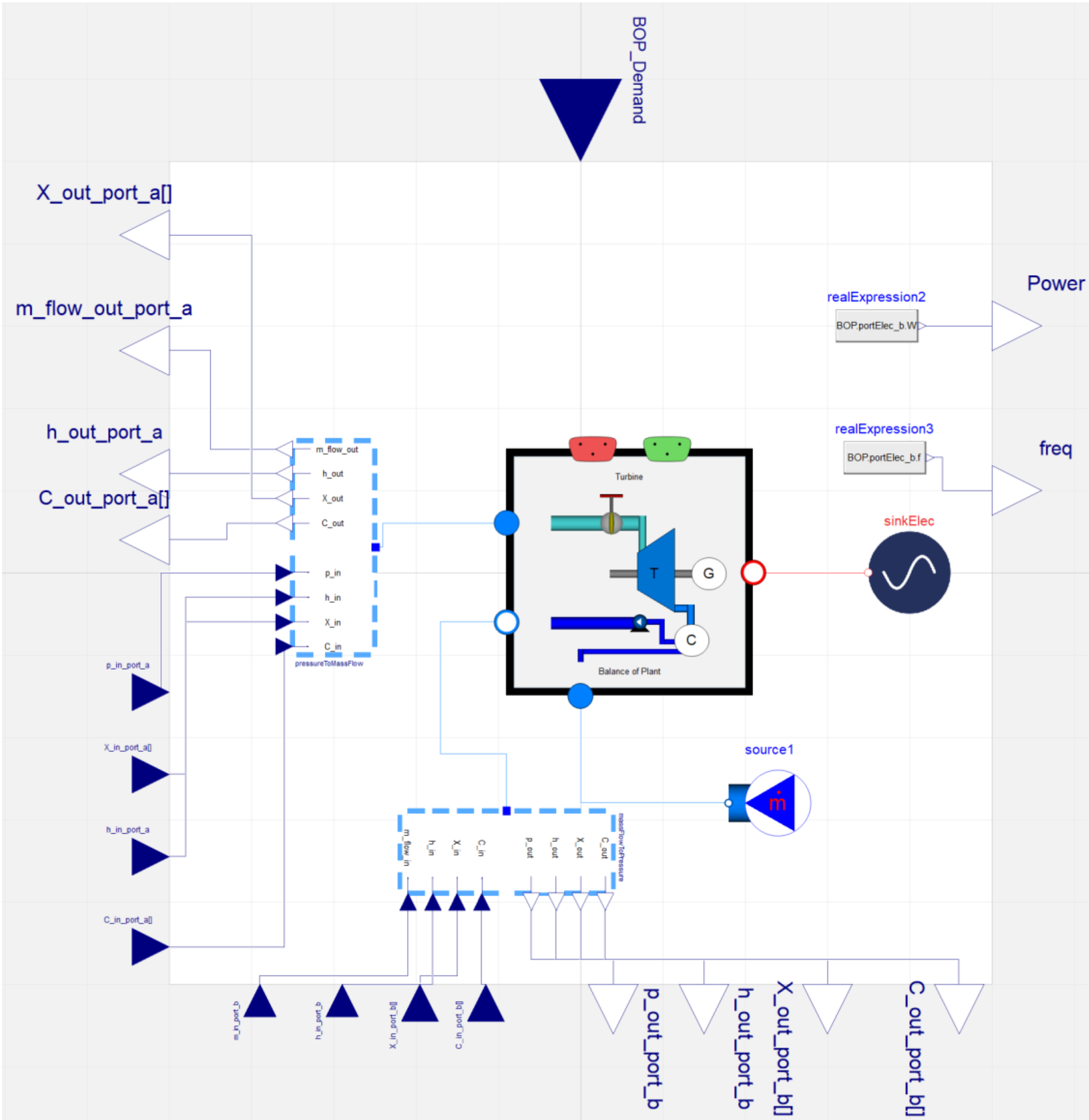


Figure 31. Translation of the Modelica turbine generator model into an FMU-ready design.

The control system within the turbine generator model is maintained through the translation process and can fulfill the desired setpoints within the turbine model. Then, the model is exported into a model exchange FMU and reimported into the Modelica framework. A comparison of the turbine power output is depicted in Figure 33, showing that the different versions of the model are in close agreement with each other. The differences can be attributed to minute variations in initialization subroutines that occur in the initialization phase of the run. The FMU-based results and input-based Modelica results are nearly identical, and both simulations were able to meet the turbine demand setpoints. It is worth noting that a version using co-simulation was attempted, but instabilities arising from the explicit time-stepping scheme could not be overcome; thus, the co-simulation was deemed unsolvable. Such scenarios become more common as the complexity of the models increases. While co-simulation is the easiest version of FMI to implement, instabilities such as these also increase the possibility of simulation roadblocks.

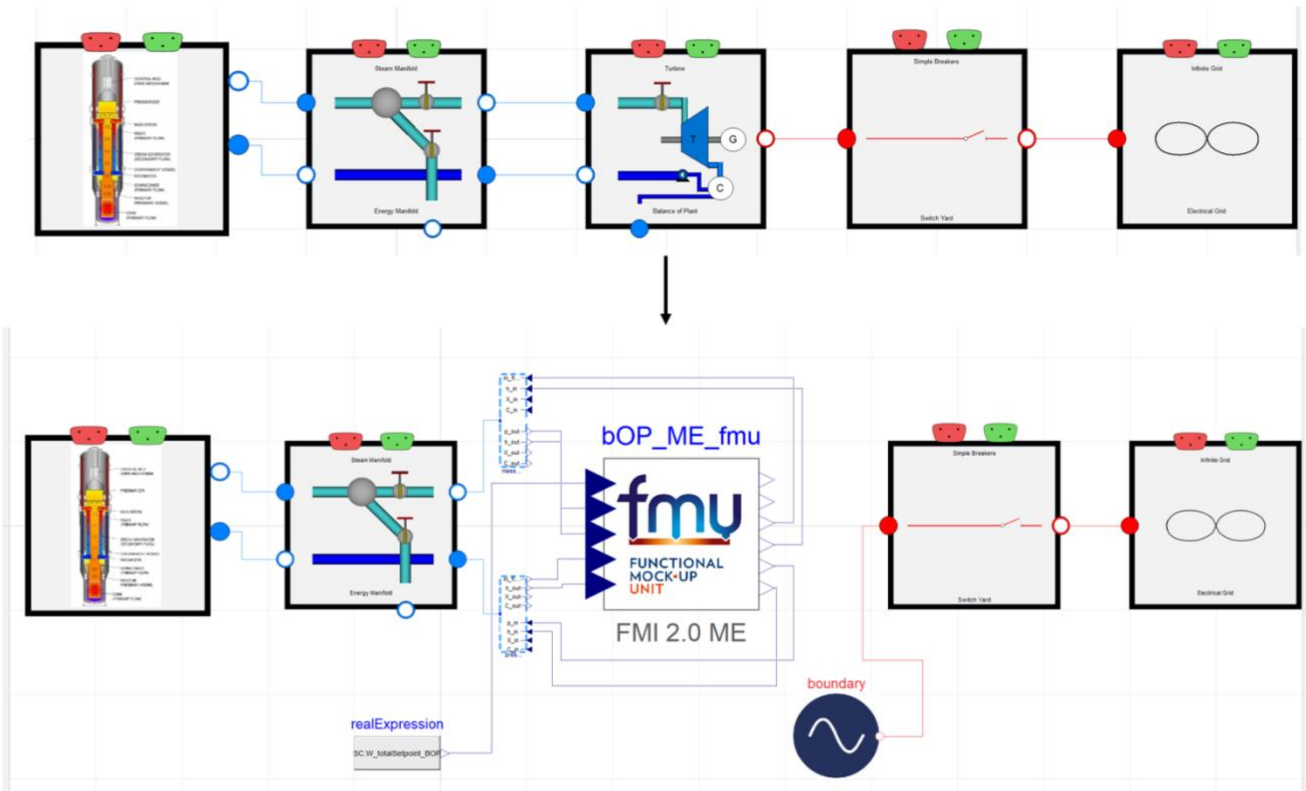


Figure 32. Transition from a Modelica model to an FMI-based simulation.

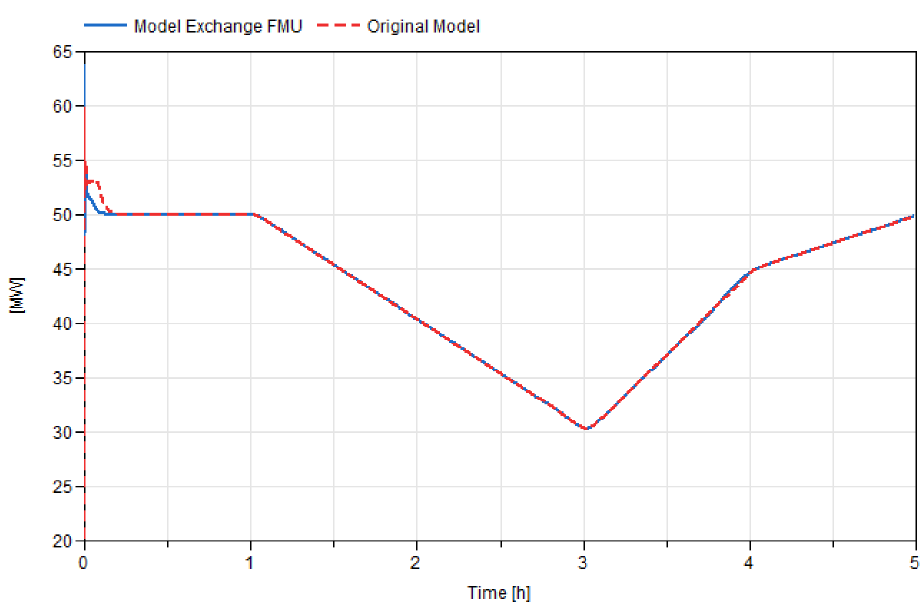


Figure 33. Comparison of turbine output results between the original model and model exchange FMU.

4. HYBRID REPOSITORY

At the beginning of the IES Project, a version control repository was delivered in order to provide a common location for the deployment of system and component models and analyses developed and constructed with Modelica/Dymola and RAVEN. To initiate the construction of a flexible plug-and-play Modelica/RAVEN framework for IES analysis, a restructuring of the version control repository (HYBRID, available at https://hpcgitlab.inl.gov/hybrid/hybrid and at the open-source repository location https://github.com/idaholab/HYBRID) was performed.

The following main tasks were performed for this specific activity:

- Usage of the RAVEN regression test system (named ROOK) for deployment of a single, integrated testing platform for both Modelica and Dymola models/analysis and RAVEN workflows. The testing system was linked with the automatic continuous integration tool for the automatic testing of the models and analyses when new modifications are added in the repository.
- Folder structure optimization for easier browsing and usage of the version control repository.

Figure 34 shows the new repository structure, with the following main folders identifiable:

- ***Models***: contains the Modelica and Dymola models
- ***archive:*** where old examples and analyses (i.e., documents, models, input files, etc.) are archived and stored to guarantee reproducibility of published results
- ***developer_tools:*** contains utility scripts, methods, and files required for the automation, deployment, and verification of the tools and software products of the HYBRID repository. This folder contains all the scripts for the automatic generation of software quality assurance (SQA) documentation (e.g., requirements, traceability matrix, etc.).
- ***scripts:*** contains scripts for installing the HYBRID repository (e.g., scripts to create the HYBRID configuration file). It also contains specialized classes and scripts for the automatic regression testing system (e.g., output checkers) and *Python*-based launchers for Dymola models (*dymola_launcher)*.
- ***tests***: contains all the tests that are automatically executed by the continuous integration system and are locally executable by running the command "run_tests."
- ***TRANSFORM-library***: submodule of the Oak Ridge National Laboratory based TRANSFORM library that provides base models for many of the integrated energy systems models
- ***raven***: links to the RAVEN repository.

Name	Last commit	Last update
.gitlab	issue and MR templates added	3 years ago
Models	Cleanup of File Tree and removal of broken ...	2 weeks ago
archive	added licens	1 month ago
developer_tools	added RTR scripts	3 weeks ago
doc	Update manual	2 weeks ago
scripts	added headers	1 month ago
tests	added rts+srs + modified tests file to add re...	3 weeks ago
TRANSFORM-Library @ d735ccc3	update of submodules	2 months ago
raven @ cd67dba1	update of submodules	2 months ago
.gitignore	Changing the .gitignore file to not ignore .m...	7 months ago
.gitmodules	added transform submodule	9 months ago
00README.txt	Civet test dirtectory set-up	4 years ago
LICENSE.txt	added licens	1 month ago
NOTICE.txt	added licens	1 month ago
README.md	added licens	1 month ago
run_tests	added headers	1 month ago

Figure 34. New structure of the repository.

Furthermore, a series of Modelica tests has been added to test the system-level interactions in the NHES Modelica repository. An example output of the regression system is shown in Figure 35.

Table 1. Synopsis of Modelica test cases.

Test	Description
Bouncing Ball	Simple test that models a bouncing ball hitting the ground.
BOP Boundaries Test A	Balance of plant system based on pressure difference
BOP Boundaries Test B	Balance of plant system based on forced mass-flow rate
Desalination 1 Pass	Single-stage reverse osmosis component check
Desalination 2 Pass	Second stage reverse osmosis component check
Desalination 2 Pass Mixing	Two-stage reverse osmosis with mixing

Desalination Reverse Osmosis Module	Fully encapsulated two-stage reverse osmosis with mixing
Desalination NHES Basic	Controlled desalination NHES system
Desalination NHES Complex	Controlled via signal bus NHES RO system with parallel osmosis units
FMI Fluid CS	Test of the fluid adaptors in a small problem in co-simulation mode
FMI Fluid CS	Test of the fluid adaptors in a small problem, using model exchange
FMI Heat CS Capacity	Test of the thermal adaptors in a small problem in co-simulation mode, using a thermal capacitance model
FMI Heat CS Conduction	Test of the thermal adaptors in a small problem in co-simulation mode, using a heat conduction model
FMI Heat ME	Test of the thermal adaptors in a small problem in model exchange (solving both the conduction and capacitance models simultaneously)
Generic Modular PWR	SMR of a NuScale size system with a pump
GTTP_Test	Gas turbine load follow test – 60-second electric demand oscillation
HTSE Power Test	High Temperature Steam Electrolysis (HTSE) NHES system based on power input control
HTSE Steam Test	HTSE NHES system based on steam and power input control
Hydrogen Turbine Test	Hydrogen turbine load follow test – 60-second electric demand oscillation
NSSS_test	Westinghouse-style four loop PWR test – 10,000 seconds at nominal power
Simple_Breakers_Test	Test of electrical breakers on an infinite grid
SMR_4Loop	Test of load following a natural-circulation SMR – 5-hour load follow simulation
SMR Primary Test	Test of the primary loop of a natural-circulation SMR loop
SMR Nominal Test	Addition of nominal power test for a natural-circulation SMR reactor

Step-Down Turbines	Basic set of step-down turbines
Step-Down Turbines Complex	Test of a more complex step-down turbine system
Supervisory Control Test	Test of the supervisory control system for receiving input from external files
Test_Battery_Storage	Test of a simple electrical battery system – logical power flow simulation
Test_Thermal_Storage	Test of a Therminol-66 thermal energy storage facility through both charge and discharge cycles
TightlyCoupled_FY18_Battery	Complex system of systems from the 2018 case (including electric battery storage)
Tightly Coupled_FY18_TES	Complex system of systems from the 2018 case (including thermal energy storage [two-tank sensible heat])
Thermocline Cycling Test	Test of the hourly cycling of a single-tank packed-bed thermocline system
Thermocline Insulation Test	Test of the insulation heat loss through the tank walls of a single-tank packed-bed thermocline system

While these tests are not exhaustive of the Modelica repository system, they provide a systems-level understanding of the repository model state. Other tests will be added on an as-needed basis.

```
scripts/testers\DymolaMatDiffer.py:267: RuntimeWarning: invalid value encountered in subtract
  diffMatrix = np.abs(varTrajectories - varTrajectoriesgold)
rook: loading init file "C:/msys64/home/FRICKL/FORCE_Hybrid/hybrid/scripts/rook.ini"
rook: ... loaded setting "add_non_default_run_types = dymola,raven"
rook: ... loaded setting "add_run_types = dymola,raven"
rook: ... loaded setting "test_dir = tests"
rook: ... loaded setting "testers_dirs = scripts/testers,raven/scripts/TestHarness/testers/"
rook: found 34 test dirs under "tests" ...
rook: loading init file "C:/msys64/home/FRICKL/FORCE_Hybrid/hybrid/scripts/rook.ini"
rook: ... loaded setting "add_non_default_run_types = dymola,raven"
rook: ... loaded setting "add_run_types = dymola,raven"
rook: ... loaded setting "test_dir = tests"
rook: ... loaded setting "testers_dirs = scripts/testers,raven/scripts/TestHarness/testers/"
(1/34) Success( 20.32sec)tests\dymola_tests\Bouncing_Ball\
(2/34) Success( 21.43sec)tests\dymola_tests\Desalination_1_pass\
(3/34) Success( 25.44sec)tests\dymola_tests\BOP_L1_Boundaries_a_Test\
(4/34) Success( 26.46sec)tests\dymola_tests\BOP_L1_Boundaries_b_Test\
(5/34) Success( 15.22sec)tests\dymola_tests\Desalination_2_pass\
(6/34) Success( 17.50sec)tests\dymola_tests\Desalination_2pass_mixing\
(7/34) Success( 25.76sec)tests\dymola_tests\Desalination_NHES_basic\
(8/34) Success( 22.70sec)tests\dymola_tests\Desalination_ROmodule\
(9/34) Success( 26.41sec)tests\dymola_tests\FMI_Fluid_CS\
(10/34) Success( 44.89sec)tests\dymola_tests\Desalination_NHES_complex\
(11/34) Success( 13.93sec)tests\dymola_tests\FMI_heat_CS_capacity\
(12/34) Success( 26.64sec)tests\dymola_tests\FMI_Fluid_ME\
(13/34) Success( 13.90sec)tests\dymola_tests\FMI_heat_CS_conduction\
(14/34) Success( 15.00sec)tests\dymola_tests\FMI_Heat_ME\
(15/34) Success( 18.33sec)tests\dymola_tests\GTTP_Test\
(16/34) Success( 24.72sec)tests\dymola_tests\HTSE_Power_Test\
(17/34) Success( 33.96sec)tests\dymola_tests\Generic_Modular_PWR\
(18/34) Success( 17.87sec)tests\dymola_tests\Hydrogen_Test\
(19/34) Success( 15.92sec)tests\dymola_tests\Simple_Breakers_Test\
(20/34) Success( 39.00sec)tests\dymola_tests\HTSE_Steam_Test\
(21/34) Success( 41.62sec)tests\dymola_tests\NSSS_test\
(22/34) Success( 26.36sec)tests\dymola_tests\SMR_primary_test\
(23/34) Success( 36.63sec)tests\dymola_tests\SMR_Nominal_Test\
(24/34) Success( 16.67sec)tests\dymola_tests\StepDownTurbines\
(25/34) Success( 16.47sec)tests\dymola_tests\StepDownTurbines_complex\
(26/34) Success( 56.81sec)tests\dymola_tests\SMR_4Loop\
(27/34) Success( 14.09sec)tests\dymola_tests\Supervisory_Control_Test\
(28/34) Success( 13.83sec)tests\dymola_tests\Test_Battery_Storage\
"failing"
(29/34) Skipped(  None!  )tests\dymola_tests\TightlyCoupled_FY18_Battery\
"failing"
(30/34) Skipped(  None!  )tests\dymola_tests\TightlyCoupled_FY18_TES\
(31/34) Success(  7.77sec)tests\raven_tests\train\TrainArmaOnData
(32/34) Success( 33.35sec)tests\dymola_tests\Test_Thermal_Storage\
(33/34) Success( 27.62sec)tests\dymola_tests\Thermocline_Insulation\
(34/34) Success( 39.62sec)tests\dymola_tests\Thermocline_Cycling\

PASSED: 32
SKIPPED: 2
FAILED: 0
```

Figure 35. An example of tests run in the ROOK regression system.

Other capabilities besides tests were added to the regression system in order to allow for smoother cross-platform and cross-machine compatibility. These capabilities were necessary because the commercial platform Dymola by Dassault systems has a series of settings that control the type of outputs sent to the final solution file. Ensuring that every user has the same flags turned on/off is unrealistic, since some of the flags are global settings turned on for every simulation loaded into their particular instantiation of Dymola. To get around this, the ROOK testing system added the capability to only look at those time steps or time intervals guaranteed to be included in each simulation of the model, regardless of the flags automatically loaded by Dymola. To accomplish this, an extra option (either “numberOfIntervals” or “OutputInterval”) is required in the simulateModel command in the regression system. The option numberOfIntervals tells Dymola how many output intervals to make, whereas OutputInterval tells Dymola at what time-step interval an output should be present for comparison. These can be selected in the Simulation Setup tab of the Dymola graphical user interface (GUI).

Further, a restart file loading capability was added to the Modelica regression system. This was included because, for complex models, the initialization phase of a simulation can require the Modelica solvers to spend a significant amount of time finding an initialization point. This is due to the highly nonlinear nature of the underlying physical equations. One way to avoid such situations is to provide a restart file to bypass the initialization phase of the simulation. A restart file is automatically created at the end of each simulation; this is the dsfin.txt file created in the folder from which the simulation was run. This file includes the final values of the previous simulation, from which the new model can restart. Moving this file to the tests/reference folder and loading it into the regression system can save a substantial amount of time in regression testing and provide a consistent starting point for each test, rather than relying on the same initialization point being found during each regression testing cycle. Full details on how to utilize and create new regression tests can be found on the HYBRID wiki at https://hpcgitlab.inl.gov/hybrid/hybrid/wiki or at its open-source location, https://github.com/idaholab/HYBRID/wiki).

The work covered in this report was propaedeutic for releasing the modeling framework in the open-source community. Several activities were deployed for open-sourcing of the software:

- User documentation:
  - Development of an extensive user manual [9], providing a detailed description of the models (Modelica and Dymola) and instructions on how to execute them
- SQA documentation (see Figure 36), available both in the INL internal Electronic Document Management System (EDMS) and the GITHUB website under "./doc/sqa/". Such documentation is aimed at collecting the following information:
  - Project planning information
  - High-level overview touching on our entire project and software development activities.
  - Roles and responsibilities
  - Merge request workflow (e.g. code change requests)
  - Workflow diagram
  - Software development plan
  - Documentation of references to other relevant plans and procedures
  - Information about the software safety and quality level determinations
  - Definitions of software validation and verification
  - Methods and procedures for software validation and verification.

And it is composed of the following set of documents:

- HYBRID Software Quality Assurance Plan (PLN-6274) (detailing the SQA procedures adopted for the development and lifecycle of the HYBRID software framework)
- HYBRID Software Configuration Management Plan (PLN-6274)
- HYBRID Software Test Plan (PLN-6274)
- HYBRID IT Asset Maintenance Plan (PLN-6274)
- HYBRID Verification and Validation Plan (PLN-6274)
- HYBRID Software Design Description (SDD-561)
- HYBRID Software Requirement Specification (SPC-2990)

- HYBRID Traceability Matrix (SPC-2990)
- HYBRID Configuration Item List (LST-1296).

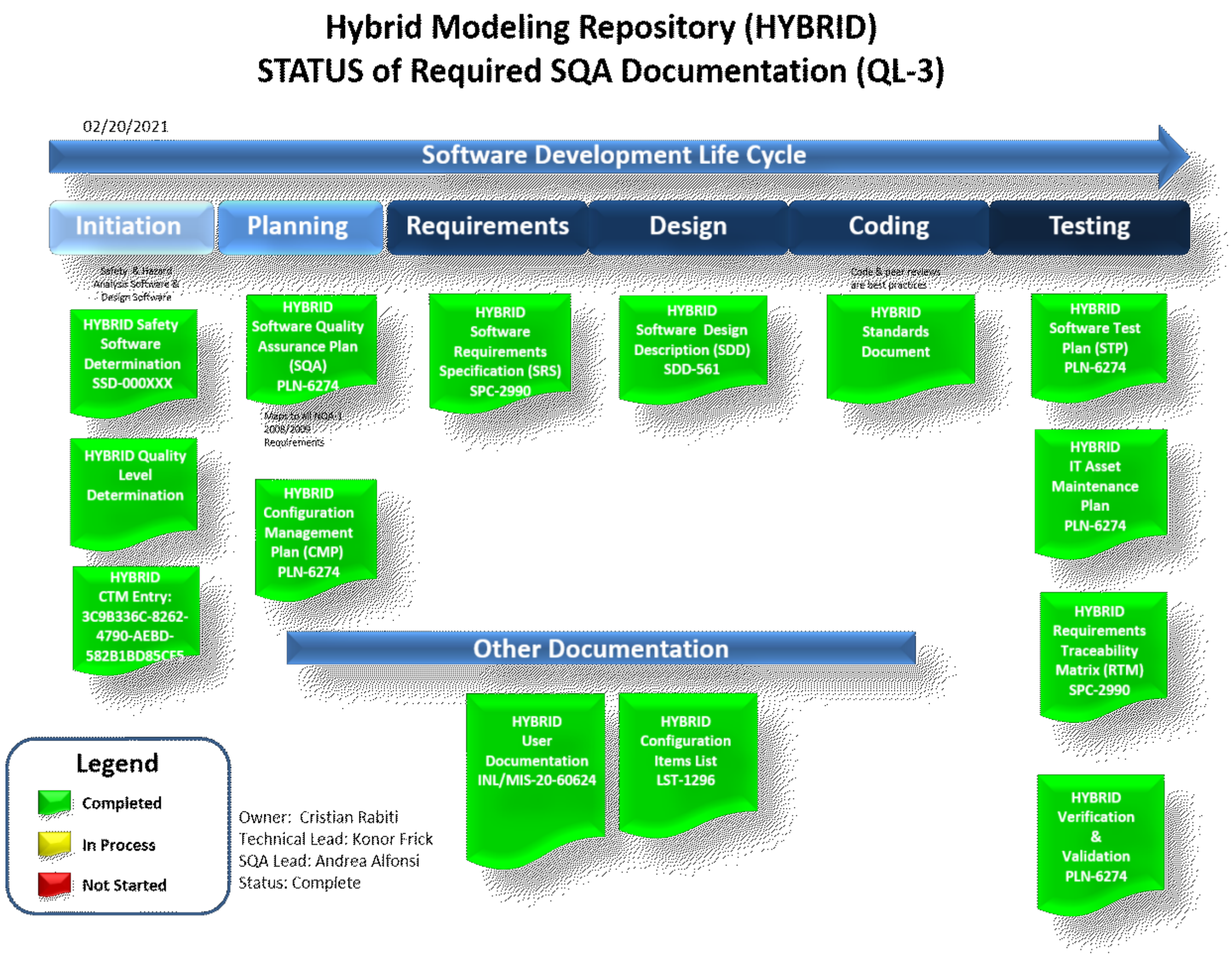


Figure 36. Status of the required SQA documentation for the HYBRID modeling repository.

5. DEPLOYMENT OF A RAVEN FMI/FMU DRIVER

Previous milestone reports [10],[11] demonstrated the successful execution of the FMIs and FMUs using external *Python*-based frameworks (FMPy [12] and PyFMI [13]). Such showcasing provided the basis for implementing the FMI and FMU interfaces within the RAVEN framework. The following sections offer a brief overview of the RAVEN code and the implementation of the driver for FMI/FMU-based models.

5.1 RAVEN Introduction

RAVEN is designed to perform parametric and probabilistic analyses based on the response of complex system codes. RAVEN can be used to investigate the system response—as well as the input space—using Monte Carlo, grid, or Latin hypercube sampling schemes, but its strength lies in the discovery of system features, such as limit surfaces, identifying and separating regions of the input space leading to system failure, and using dynamic supervised learning techniques. RAVEN includes the following major capabilities:

- Sampling of codes for uncertainty quantification and reliability analyses
- Generation and use of reduced-order models (ROMs) (also known as surrogate models)
- Data post-processing (time-dependent and steady-state)
- Time-dependent and steady-state statistical estimation and sensitivity analysis (mean, variance, sensitivity coefficients, etc.).

The RAVEN statistical analysis framework can be employed for several types of applications:

- Uncertainty Quantification
- Sensitivity/Regression Analysis
- Probabilistic Risk and Reliability Analysis
- Data Mining Analysis
- Model Optimization.

RAVEN provides a set of basic and advanced capabilities that range from data generation to data processing and data visualization. Its mission is to provide a framework/container of capabilities that engineers and scientists can use to analyze system responses, physics, and multi-physics by employing advanced numerical techniques and algorithms.

RAVEN was conceived with two major objectives in mind:

- To be as easy and straightforward as possible for scientists and engineers to use
- To allow for straightforward expansion of itself by providing clear and modular APIs (Application Programming Interfaces) to developers.

The RAVEN software is meant to be approachable by any type of user (computational scientists, engineers, or analysts). Every aspect of RAVEN was driven by this singular principle, from the build system to the APIs to the software development cycle and input syntax.

The main idea behind the RAVEN software design remains the creation of a multi-purpose framework characterized by high flexibility with respect to the possible performable types of analyses. The framework must be able to construct the analysis/calculation flow at run-time, interpret the user-defined instructions, and assemble the different analysis tasks following a user-specified scheme.

5.2 RAVEN Models

In RAVEN, coupling of the system to physical models is performed by the model entity API. The model entity represents a “connection pipeline” between the input and output spaces. The RAVEN framework (see Figure 37) provides APIs for the main model categories described below.

- Codes: The *Code* model represents the communication pipe between the RAVEN framework and any system and/or physical code/model. The communication between RAVEN and any driven code is performed through the implementation of interfaces directly operated by the framework. The procedure for coupling a new code/application with RAVEN is a straightforward process. The coupling is performed through a *Python* interface that interprets the information coming from RAVEN and translates them to the input of the driven code. The coupling procedure does not require modifying RAVEN itself. Instead, the developer creates a new *Python* interface that will be embedded in RAVEN at run-time (no need to introduce hard-coded coupling statements). If the coupled code is parallelized and/or multi-threaded, RAVEN will manage the system in order to optimize the computational resources of both the workstations and High-Performance Computing systems.

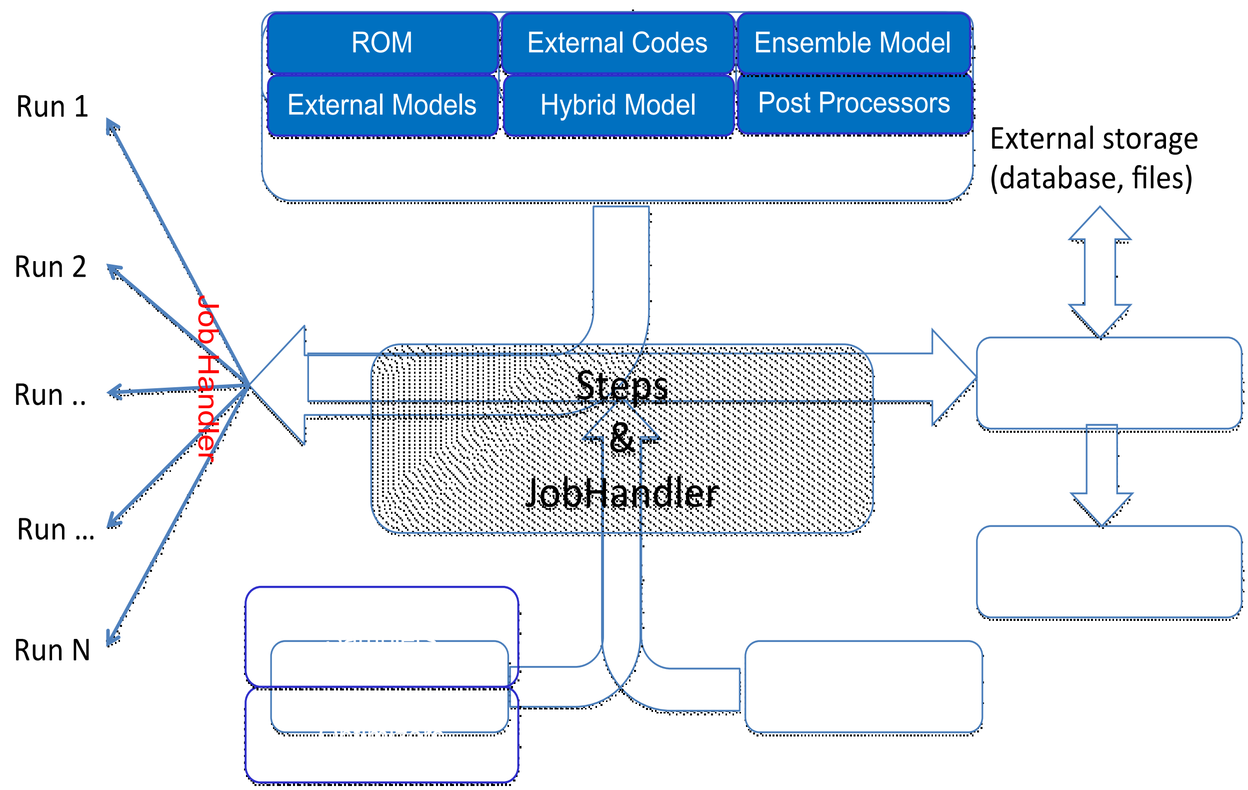


Figure 37. RAVEN framework scheme.

- Externals: The *External* model allows the user to create, in a *Python* file (imported at run-time into the RAVEN framework), its own model (e.g., set of equations representing a physical model, connection to another code, and control logic). This model will be interpreted/used by the framework and, at run-time, will become part of RAVEN itself.
- Reduced Order Models (ROMs): Reduced order, AI-based surrogate models, are a mathematical representation of a system, used to predict a physical system's selected output space. The "training" is a process that uses sampling of the physical model to improve the ROM's prediction capability (i.e., the capability to predict the status of the system given a realization of the input space). More specifically, in RAVEN, the ROM is trained to emulate a high-fidelity numerical representation (system codes) of the physical system.

- Hybrid models: The *HybridModel* can combine ROMs with any other high-fidelity model (e.g., *Code* or *ExternalModel*). The ROM will be "trained" based on the results from the high-fidelity model. The accuracy of the ROM will be evaluated based on the cross-validation scores, and the validity of the ROM will be determined via local validation metrics. After the ROM is trained, the *HybridModel* can decide which model (i.e., the ROMs or high-fidelity model) to execute, based on the accuracy and validity of the ROMs in a particular operating region.
- Ensemble models: The *EnsembleModel* is used to create a chain of Models whose execution order is determined by the Input/Output relationships among them. If the relationships among the models evolve in a non-linear system, a Picard's Iteration scheme is employed.
- Postprocessors: The Post-Processor model represents the container of all the data analysis capabilities in the RAVEN code. This model is used to process the data (e.g., derived from sampling of a physical code) in order to identify representative Figures of Merit. For example, RAVEN uses Post-Processors to perform statistical and regression/correlation analysis, data mining and clustering, reliability evaluation, topological decomposition, etc.
- RAVEN FMI/FMU Driving System Development.

Development of the FMI/FMU driving system is based on the *ExternalModel* entity in RAVEN. As briefly reported in the previous section, the external model (see Figure 38) enables developers to create, in a *Python* module or platform, a direct coupling with a model coded in *Python* (e.g., a set of equations representing a physical model, connection to another code, and control logic). Once the external model is constructed, it is interpreted and used by RAVEN, ultimately becoming, at run-time, part of RAVEN itself.

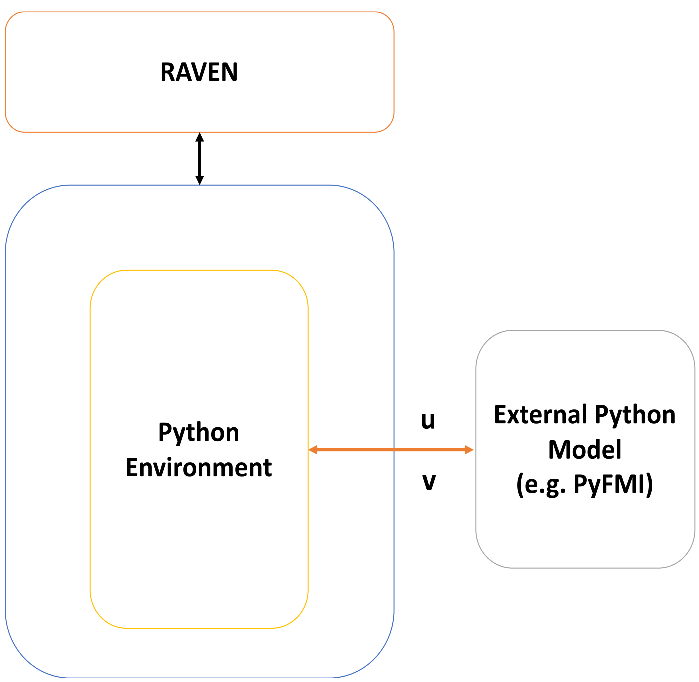


Figure 38. External model API.

`class FMIFMU(ExternalModelPluginBase):`	
`def run(self,container,inputs)`	Required
`def readExtInput(self,container,xmlNode)`	Optional
`def initialize(self,container,runInfo,oinputs)`	Optional
`def createNewInput(self,container,inputs,oinputs,samplerType,**Kwargs)`	Optional

Figure 39. FMI/FMU model skeleton in RAVEN.

The *ExternalModel* API (*ExternalModel* plugin) was used to develop, in RAVEN, a native driver for models using the FMI/FMU protocol. Figure 39 shows a snapshot of the "wrapper" that was developed. The "FMIFMU" RAVEN model implements a generalized method—based on the RAVEN API and syntax—to import, execute, and process the results of any model compatible with the FMI/FMU standard. The model consists of the following methods:

- ***run***: The run method (the only required method in the API) aims to execute the FMU (FMI) for a given input coordinate (or input perturbation). The ***run*** method represents the pipeline between RAVEN and the FMI/FMU model. The method both executes and collects the results that will be then stored in the object "container," ready for processing by RAVEN.
- ***readExtInput***: This method is in charge of reading the user-define input for the FMI/FMU that needs to be driven. It collects the following information (expandable in the future, if needed):
    - ***startTime***: The start time of the driven FMU (e.g., 0.0 seconds)
    - ***stopTime***: The stop time of the driven FMU (e.g., 60 seconds)
    - ***stepSize***: The time step size to use for the calculation (e.g., 1.e-2 seconds)
    - ***inputVariables***: A list of the input variables (e.g., demand)
    - ***outputVariables***: A list of the output variables (e.g., power level)
    - ***fmuFile***: The FMU location (e.g., */path/to/myFmu.fmu*)
- ***initialize***: This method is invoked right before the model is executed. This method aims to load the FMI/FMU, instantiate the class, and initialize its settings.

  This method is also in charge of performing error checking of the user-defined settings/options.
- ***createNewInput***: This method, in case of a sampling strategy, is responsible for translating" the RAVEN info (e.g., the values of sampled variables) into the FMI/FMU syntax.

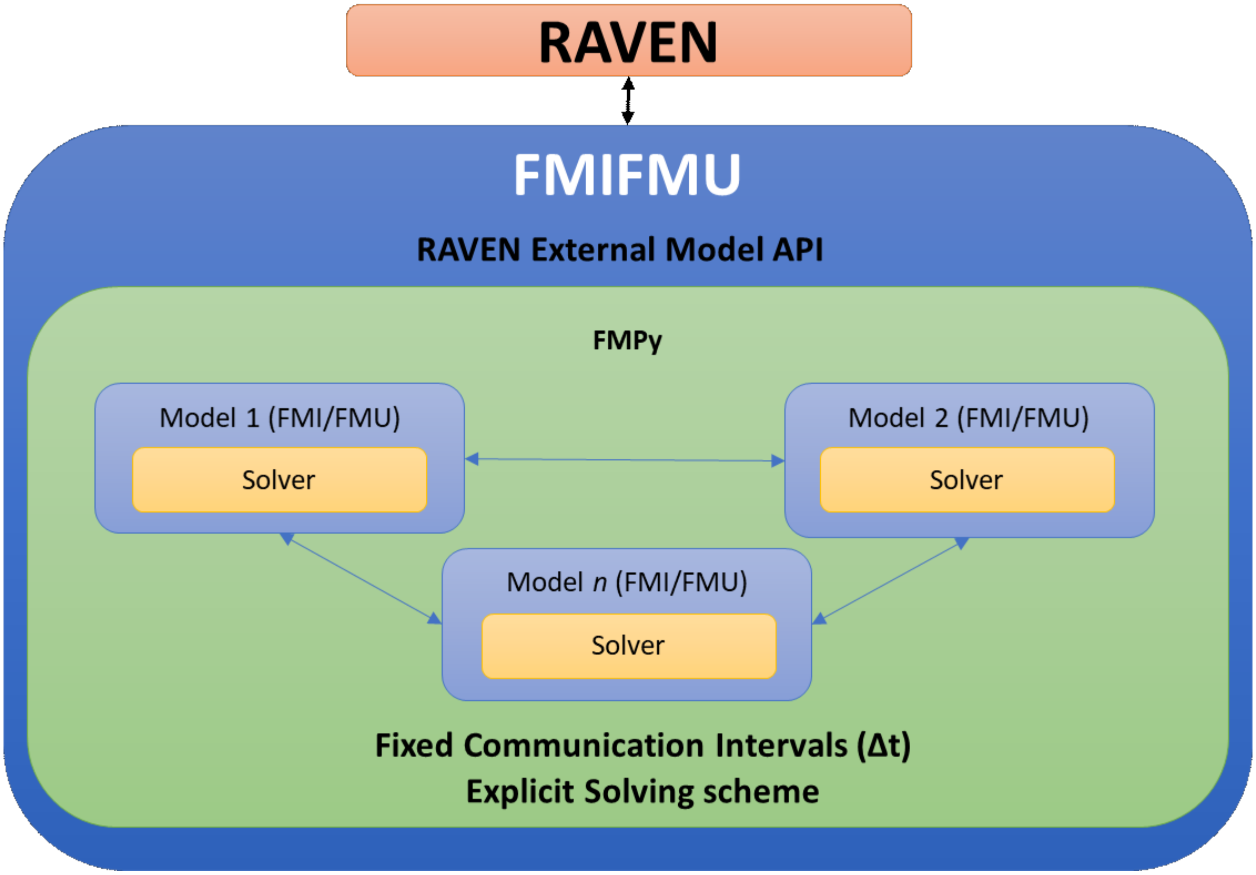


Figure 40. FMI/FMU co-simulation protocol coupled with RAVEN.

Depending on the type of protocol for the FMI or FMU of interest, two coupling schemes in the FMIFMU wrapper were developed. Both schemes are encapsulated in the same wrapper and are executable via the model API in RAVEN.

Figure 40 shows the coupling scheme for FMIs/FMUs when the co-simulation protocol must be used; RAVEN interacts with the different models via the FMIFMU wrapper that uses FMPy to import and interact with the FMUs. In this coupling scheme, RAVEN "perceives" the models imported via FMIs/FMUs just as it would any other external model or code. This protocol is indicated when the models to connect are loosely coupled (multi-physics feedbacks are not strong and/or the physics dynamic of the different models act on different time scales, e.g., seconds vs. hours or days).

On the other end, Figure 41 shows the coupling scheme for FMIs/FMUs when the model exchange protocol is used; in this configuration, RAVEN can directly interact with the universal solver that aims to solve all the models (compatible with the FMI/FMU protocol, in this case Dymola). This coupling scheme is preferrable when the models are highly nonlinear and the models are tightly coupled with fast moving dynamics.

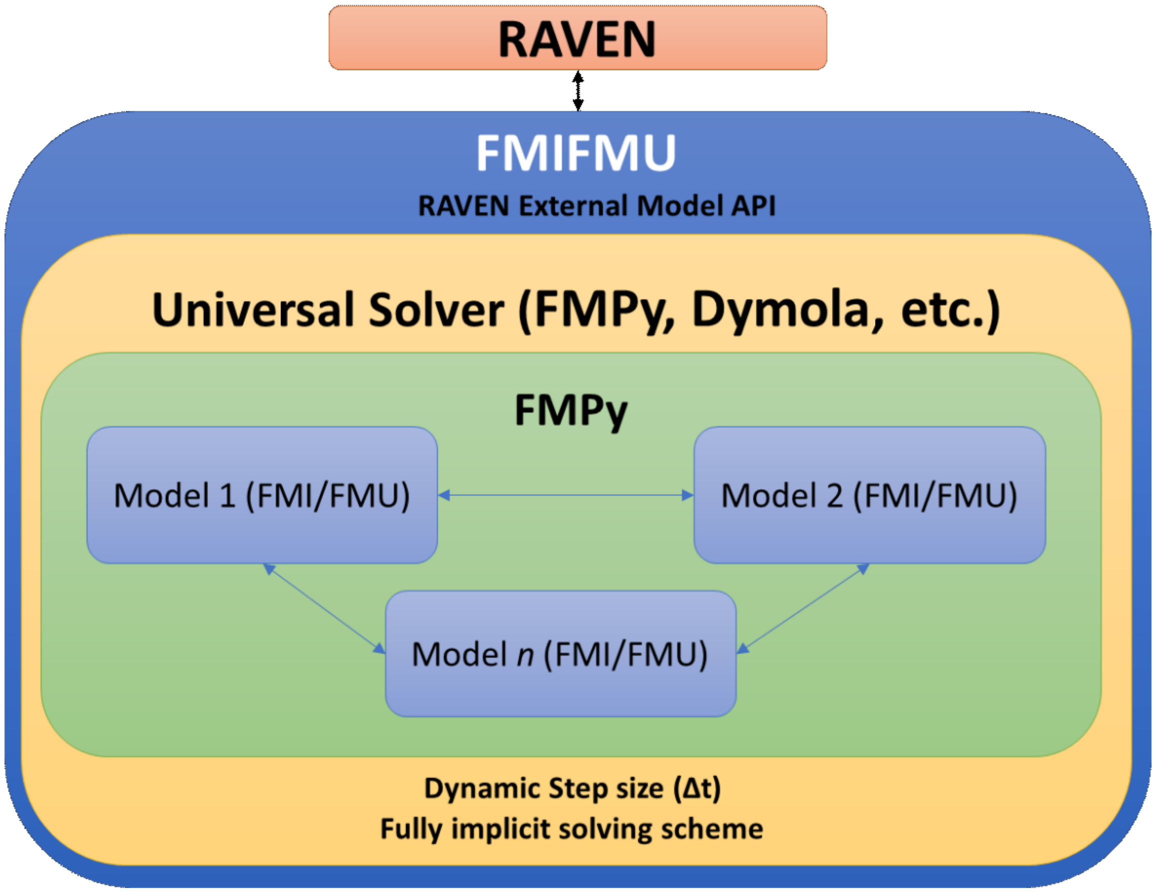


Figure 41. FMI/FMU model exchange protocol coupled with RAVEN.

```
<Models>
  <ExternalModel name="fmu" subType="FMIFMU">
    <variables> demand, time, SES_generator_P_flow</variables>
    <startTime> 0.0 </startTime>
    <stopTime> 14400</stopTime>
    <stepSize> 1.0 </stepSize>
    <inputVariables> demand </inputVariables>
    <outputVariables> SES_generator_P_flow</outputVariables>
    <fmuFile> GTTPfmu.fmu </fmuFile>
  </ExternalModel>
</Models>
```

Figure 42. External model FMIFMU example RAVEN input file.

Figure 42 shows an example of the portion (in XML) of the RAVEN input file required to use the FMIFMU wrapper. This XML block is the one processed by the previously-described method "*readExtInput*.". Independently on the type of FMI/FMU that the model will import and use, the input file specifications do not change; the FMIFMU wrapper will collect the information (co-simulation or model exchange) directly from the FMU (i.e., the **<fmuFile>**) after loading.

6. DEPLOYMENT OF RAVEN FMI/FMU EXPORTER FOR AI-BASED ANALYSIS ACCELLERATIONS

As described in the previous section, the RAVEN framework provides APIs for different model categories, among which are the ROM, AI-based algorithms. In order to deploy the acceleration of IES analysis, the ROM (AI) entity is key. Indeed, the ROM is aimed at higher fidelity surrogate and system simulator-based models (for specific and limited operational domain) with a set of faster-execution equations that allow for the prediction of Figures of Merit (of interest) in a span of milliseconds.

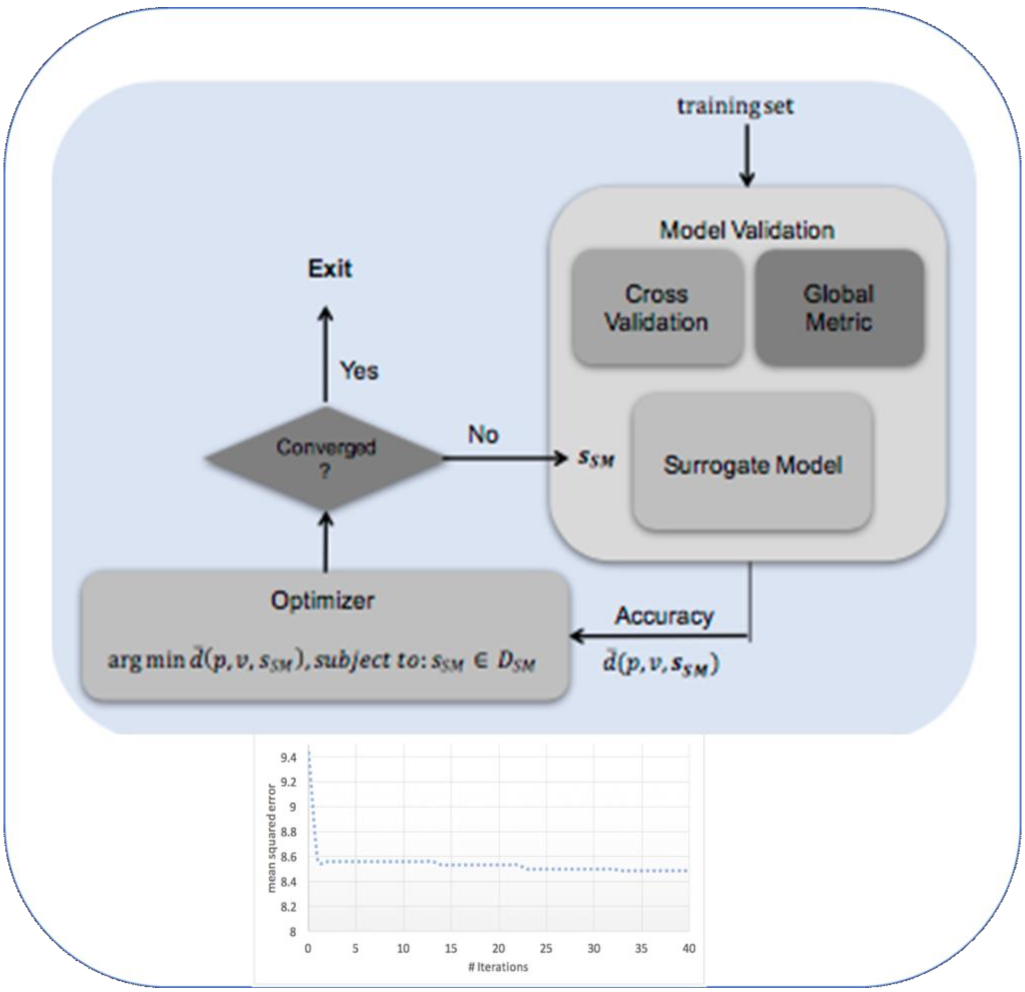


Figure 43. Construction process for surrogate models in RAVEN.

6.1 RAVEN AI construction

Figure 43 illustrates the standard process of constructing (via optimization) RAVEN surrogate (AI) models. The surrogate model of interest is trained on a dataset, and its hyper-parameters (i.e. parameters and characteristics of the surrogate model of interest) are tuned to maximize the accuracy in predicting the figure(s) of merit (FOMs) of interest. As shown in Figure 44, the accuracy is assessed by applying statistical methodologies (i.e., cross-validation), which consists of randomly portioning the dataset into "training" and "testing" datasets. The "training" dataset is used for constructing the surrogate model, and its prediction is compared with the "testing" dataset. The prediction accuracy is then assessed using distance metrics (e.g., R2 score) between the surrogate model and the testing dataset.

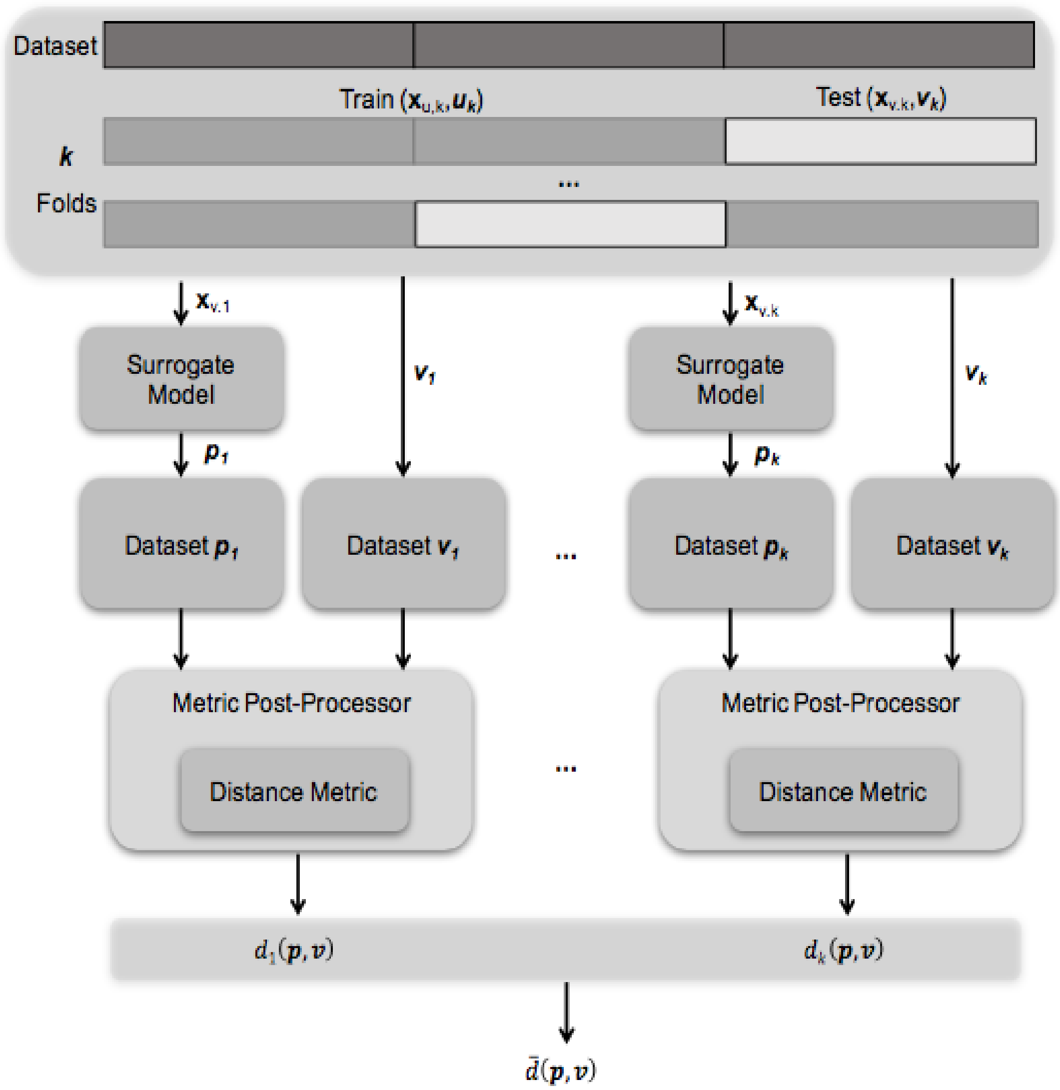


Figure 44. RAVEN ROM cross-validation scheme.

The so-constructed surrogate models allow for fast evaluation of the dynamics (or steady state) of the FOMs of interest. Therefore, such models can accelerate analyses (greatly reduce the computational time), by replacing high-fidelity physical models with a ROM representation.

6.2 Development of FMI/FMU exporting capabilities for RAVEN AI

To exploit RAVEN AI capabilities, a workflow to export trained (constructed) ROMs using the FMI/FMU protocol was developed in RAVEN.

The exporting of RAVEN AI is performed according to the following two steps:

1) Exploit the native RAVEN serialization system, which is responsible for serializing (i.e., saving in a binary file) already-trained surrogate models that can be loaded in external (*Python*-based) packages (outside RAVEN).
2) Use and extend the PythonFMU library (https://github.com/NTNU-IHB/PythonFMU), which is a lightweight framework that enables the packaging of *Python* 3 code as co-simulation FMUs (following FMI version 2.0).

To deploy any model in an FMI/FMU-compatible framework, that model (i.e., ROM) must be able to be inquired at each "time step," meaning that the model must allow for execution as an integrated model and not as a "black-box simulation". To achieve this goal, the RAVEN ROM APIs were upgraded by implementing a "method" to solve the surrogate model at each time step. This modification, in conjunction with the two steps reported above, allows for RAVEN ROM models to be exportable as FMI/FMUs.

Once the RAVEN AI is trained following the standard process reported in section 6.1, it can be finally exported following the steps reported in Figure 45. An example of the RAVEN input blocks is reported in Figure 46, where:

- In the ***<Models>*** node, the RAVEN ROM (AI) is shown.
- In the ***<Files>*** node, the output FMI/FMU filename is specified.
- In the ***< Steps>*** node, the trained ROM (input) is exported as FMI/FMU (output).

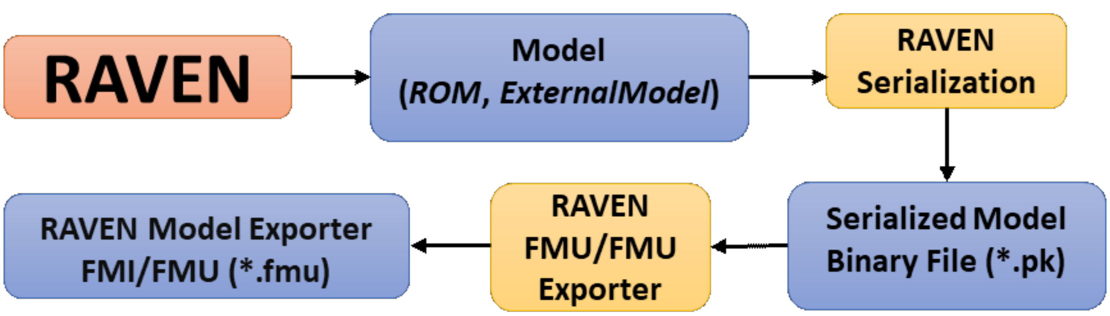


Figure 45. RAVEN AI FMI/FMU exporting process.

```
<Models>
  <ROM name="GTTProm" subType="SciKitLearn">
    ...
  </ROM>
</Models>
<Files>
  <Input name="FMU" type="fmu"> GTTProm.fmu </Input>
</Files>
```

```
<Steps>
  <IOStep name="romDump">
    <Input class="Models" type="ROM"> GTTProm </Input>
    <Output class="Files" type=""   > FMU      </Output>
  </IOStep>
</Steps>
```

Figure 46. Example RAVEN input file to export AI as FMIs/FMUs.

The FMI/FMU exporting capability allows for the deployment of the scheme reported in Figure 47, where the RAVEN models can be used, as FMI/FMUs, in tandem with any Dymola/Modelica (in general) and HYBRID (in particular) physical models.

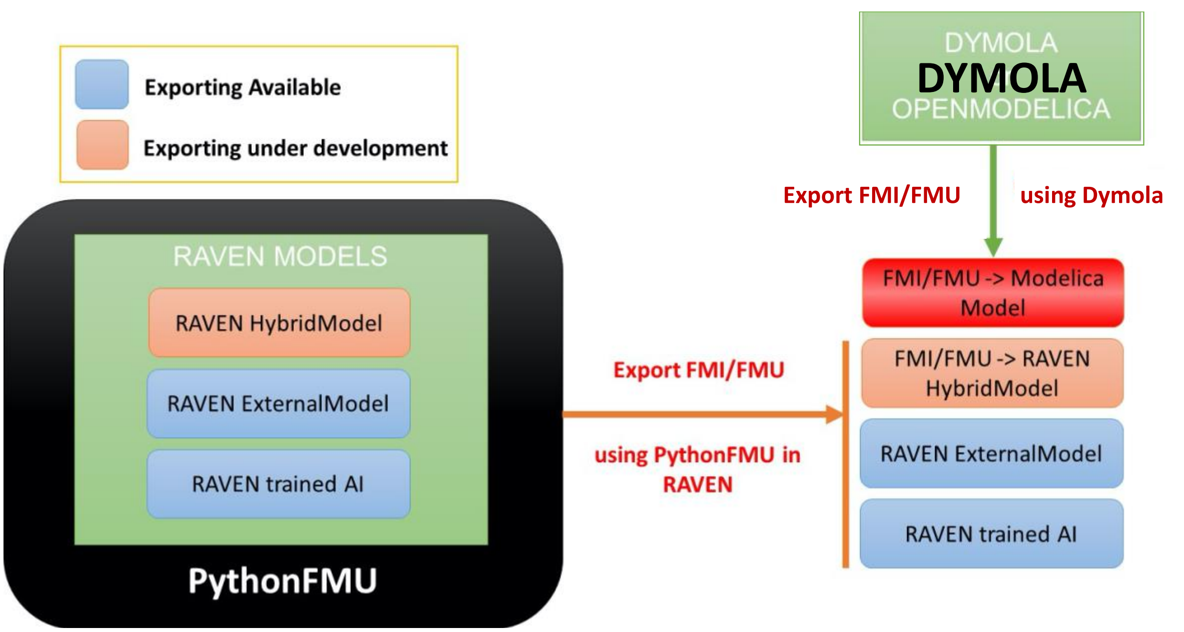


Figure 47. RAVEN’s current FMI/FMU exporting capabilities.

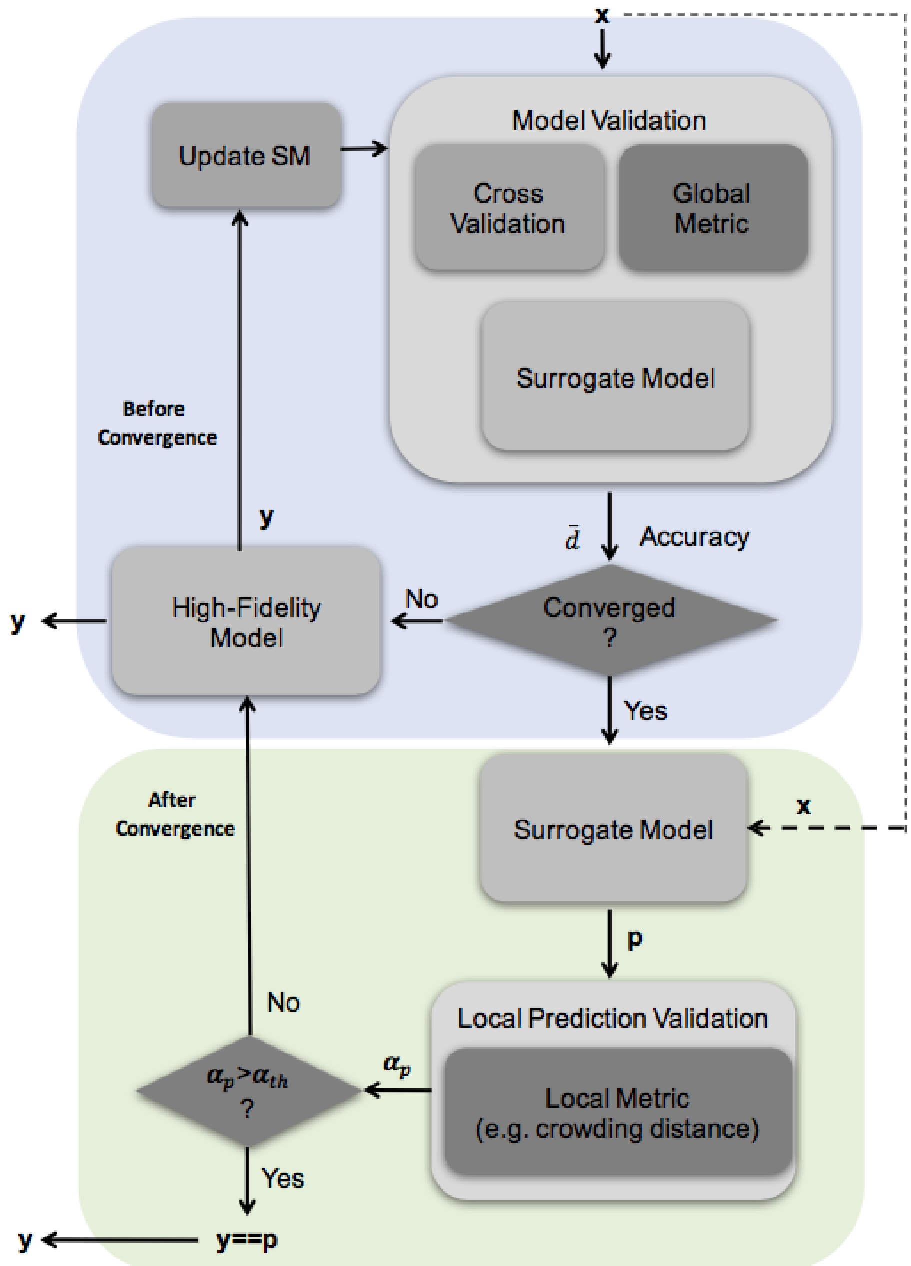


Figure 48. RAVEN hybrid model scheme.

6.3 Future Extension of the FMI/FMU Exporter to the RAVEN Hybrid Model

In previous sections, the creation and export of RAVEN AI was discussed. ROM usage is an approach that can drastically reduce the computational time of analyses and accelerate deployment of models. To obtain the optimal prediction capability, the ROM must be constructed and applied only within the domain of its training set; in other words, the ROM can guarantee valid predictions only within (or slightly outside) the boundaries of its training set. For example, if a ROM is trained by perturbing a temperature between 500 and 600 K, the ROM should not be used for predicting a system response at 1000 K.

RAVEN includes an advanced capability called the “hybrid model” to tackle this problem. Indeed, this model is a special class of algorithms aimed to couple in-tandem, high-fidelity physical and mathematical models (e.g., FMI/FMU Dymola models) and AI algorithms (e.g., ROM, AI). The AI is trained based on the results from the high-fidelity model. The global accuracy of the AI is evaluated based on cross-validation scores, and the local (e.g., prediction) validity is determined via certain local validation metrics (i.e., metrics aimed to assess the confidence of the AI predictions). Once the AI is trained, the hybrid model can decide which model (i.e., the AI or high-fidelity model) to execute, based on the aforementioned accuracy and validation metrics. Figure 48 shows the scheme behind the hybrid model formulation. Since the predictions of the surrogate model are assessed in terms of accuracy, this algorithm discards ROM predictions if they fall outside its training set boundaries or the response confidence is too low. In such cases, the high-fidelity model is used and the ROM training set updated.

In the next steps for this program, the “hybrid model” capability will be leveraged in tandem with the FMI/FMU exporting protocol in order to accelerate the execution of systems that include multiple FMI/FMUs, allowing for the deployment of models that are able to autonomously switch between RAVEN AI and Dymola models during analyses. Each FMI/FMU will be coupled in a hybrid model configuration, resulting in accurate modeling and CPU time saving.

7. Integrated Energy Park Demonstration Case

To demonstrate the full range of capabilities described in this report, a final test case on an integrated energy park was conducted. The integrated energy park, shown in Figure 49, consists of a nuclear reactor, electric batteries, and a natural gas turbine. The natural gas turbine is the component to be exported as an FMU. The natural gas peaking turbine will then be replaced with its own FMU from three different sources: the Dymola FMU in both model exchange and co-simulation, then a RAVEN-based surrogate using co-simulation mode.

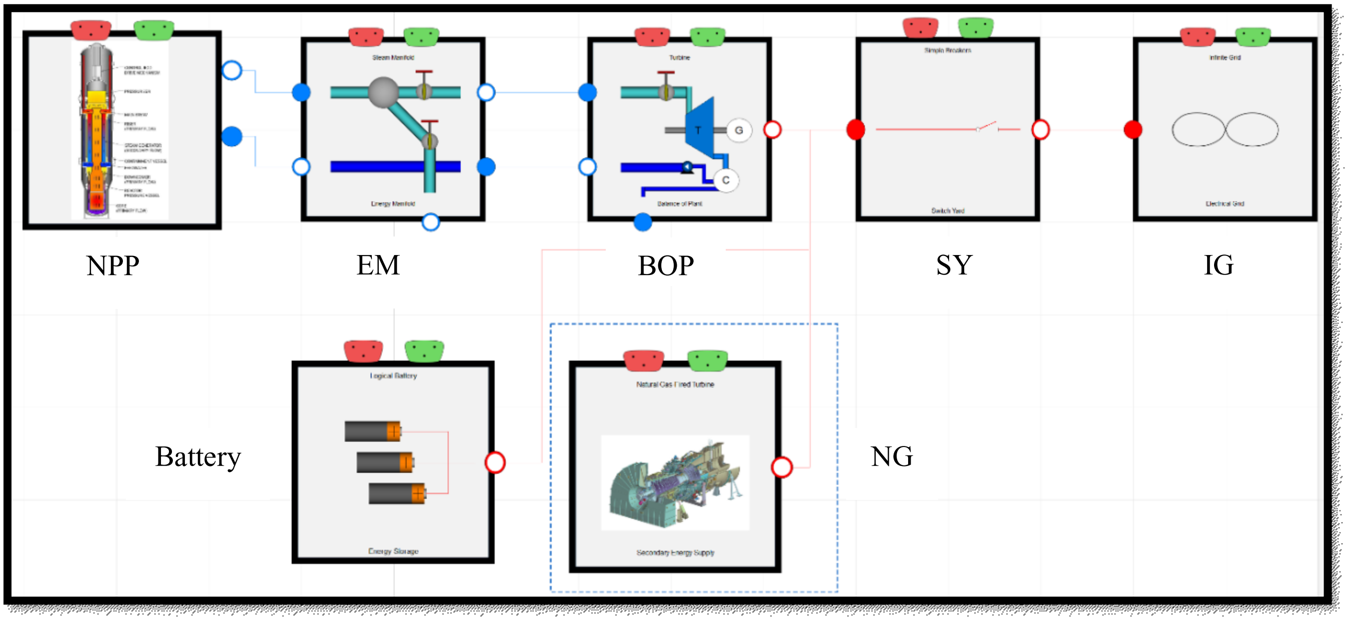


Figure 49. Integrated energy park consisting of a nuclear reactor (NPP), Energy Manifold (EM), Balance of Plant (BOP), Switch Yard (SY), Electric Batteries (Battery), Infinite Grid (IG), and a Natural Gas turbine (NG). The natural gas turbine is to be exported as an FMU.

7.1 FMI/FMU Creation and Use within Dymola

As outlined in earlier sections of this report, the natural gas turbine model needs to be modified with an electric power adaptor and an input demand signal in order to ensure that all the variables contained in the flow ports are realigned into real input/output variables.

The adaptors outlined in the previous sections can be used to accomplish these modifications. For the natural gas turbine example, illustrated in Figure 50, the electric port must be converted into real inputs/outputs using the PowerFlowToFrequency adaptor previously described. In addition, the control system of the natural gas turbine requires a top-level demand signal to communicate the grid demand at each time interval. To implement this communication into the model, an additional real input variable, "SES_Demand," was created. With the adaptor and the new input signal created, the model is ready to be exported as an FMU.

This procedure of using an adaptor to transform ports into their real components and creating additional inputs/outputs for declared variables works well for simple models and models intended to be used in model exchange mode. For complex models planned for simulation in co-simulation mode, use of adaptors may prove challenging if the initialization of the models is not well-defined. This is due to the explicit nature of co-simulation modeling.

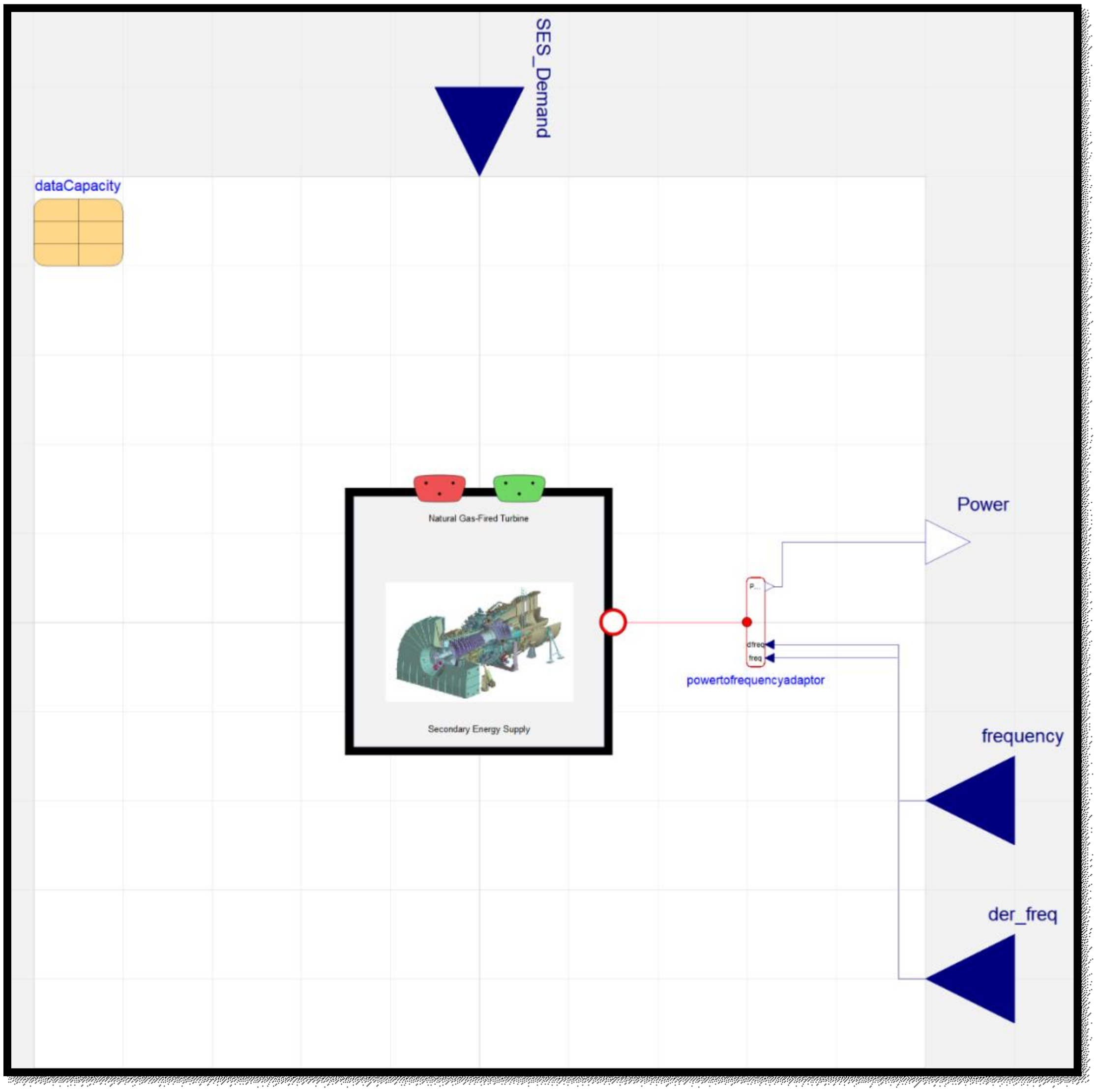


Figure 50. Preparing the natural gas turbine for conversion into an FMU. The inputs into the system are the peaking demand and connection points for electricity backflow into the turbine model. The output is the electrical power as a real value.

Once the model has been exported using the Dymola interface, it can then be re-imported into the program and can replace the natural gas turbine model. Since the FMI consists of three inputs and one output, the three inputs must be specified by the user. To accomplish this, the FrequencytoPowerFlow adaptor was placed in the Modelica model along with a “real” expression to connect the turbine demand to the FMI/FMU, as shown in Figure 51.

Using this version of the FMI/FMU, three separate 5-hour simulations were run: one with Modelica-only input, one with a co-simulation version of the gas turbine, and one in model exchange mode. The results of this simulation set are depicted in Figure 52. Over the course of the full 5-hour simulation, the results are all in near-perfect agreement with the setpoints, with the model exchange and Dymola results being basically identical, and co-simulation being only as accurate as the communication step of 1 second would allow. However, of note is that, while the Dymola and model exchange versions of the model completed in 121.3 and 156 seconds, respectively, the co-simulation model took far longer to solve (a total of 642 seconds). This increased simulation time can be attributed to the additional communication time between the models as well as the additional initialization routine required by the solvers.

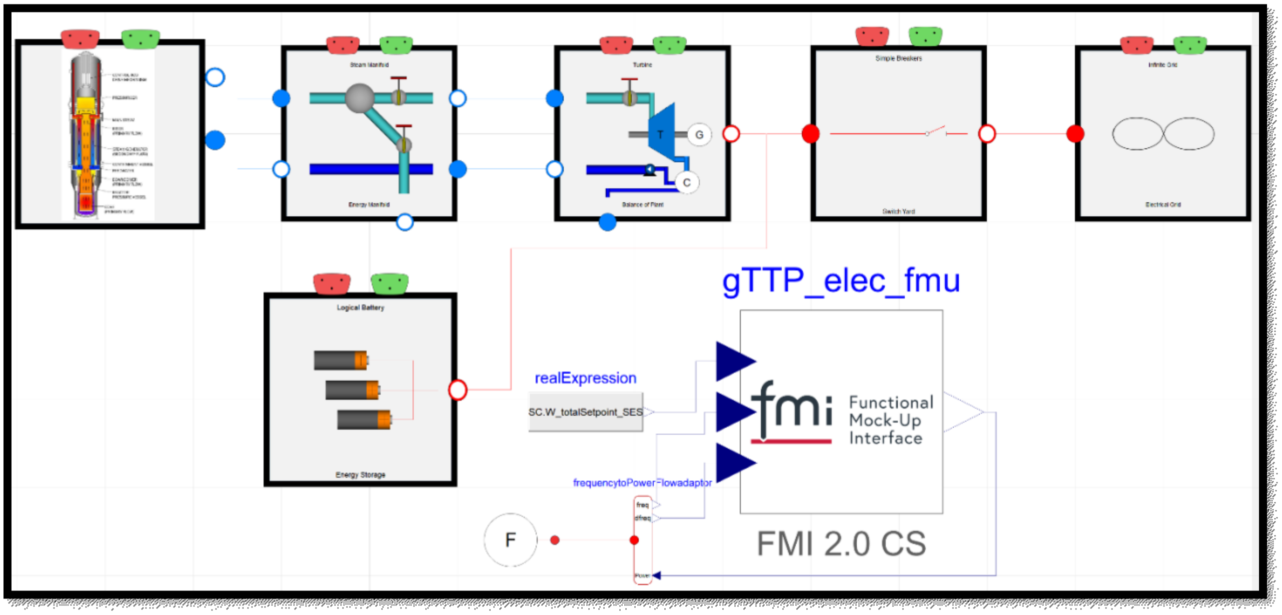
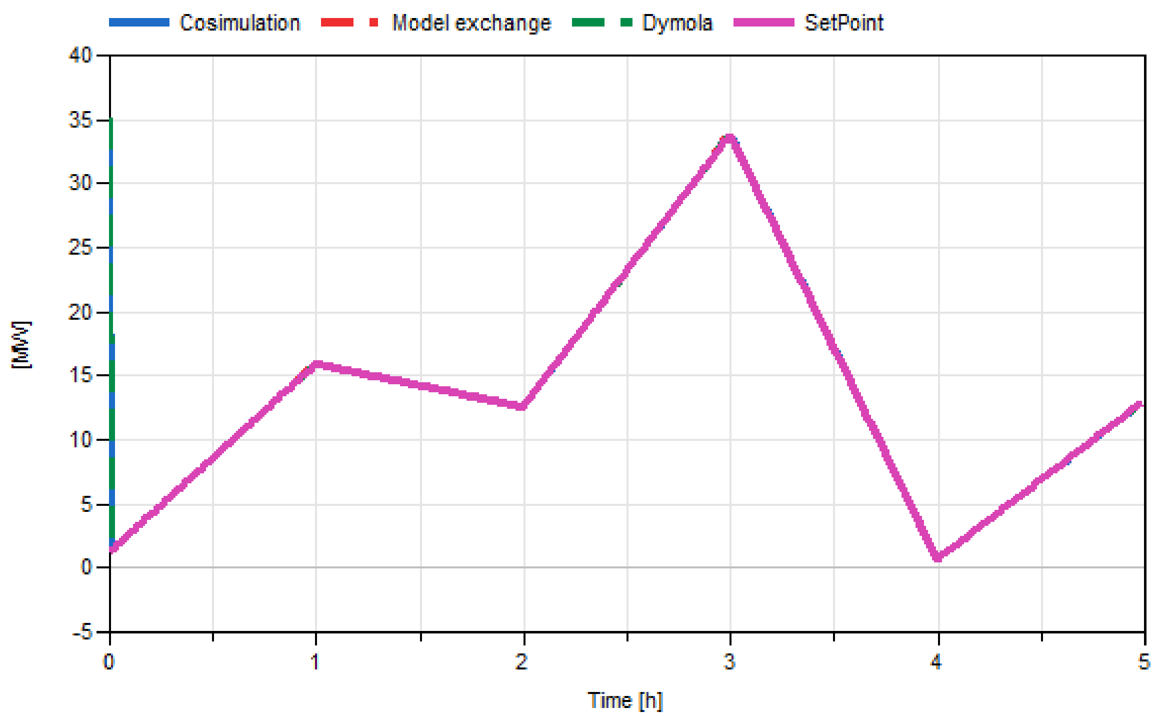


Figure 51. Integrated energy park consisting of a nuclear reactor, electric batteries, and a natural gas turbine replaced by a co-simulation FMU.



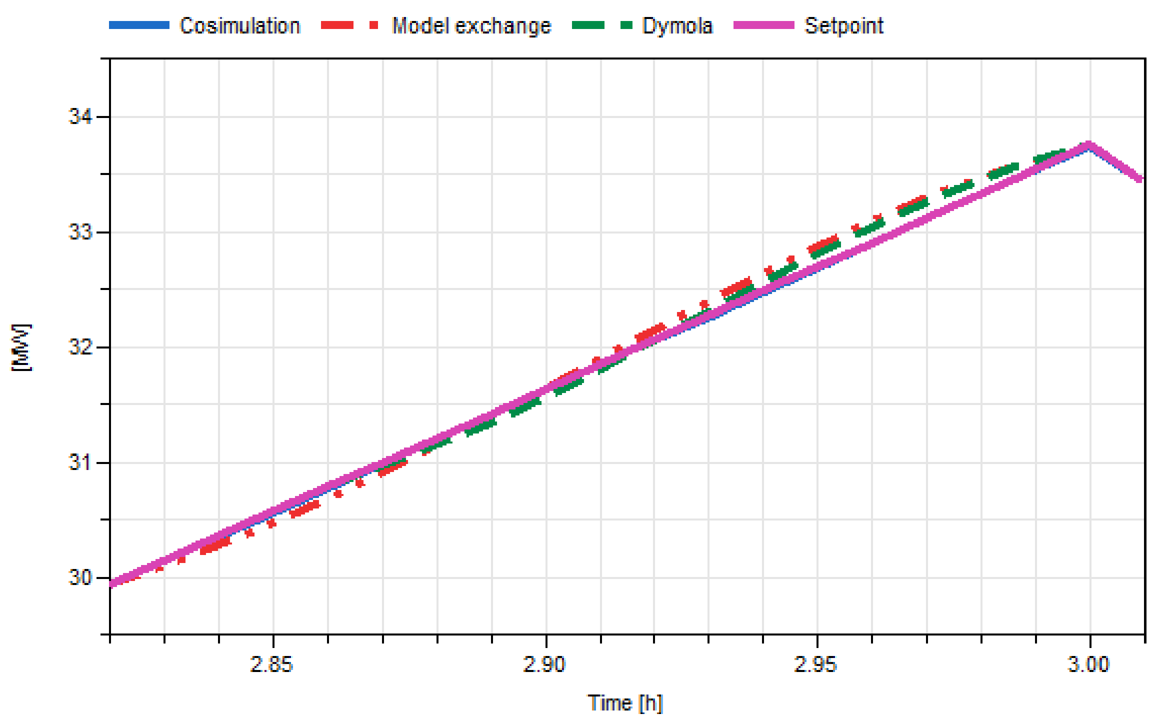


Figure 52. Top) Five-hour simulation of the natural gas turbine power vs. setpoint demand for the integrated energy park in regard to Modelica-only model, co-simulation FMI, and model exchange FMU. Bottom) Closeup shot of the turbine demand vs. turbine output for the different FMI versions. Note that all agree reasonably well. Co-simulation communication interval = 1 second.

7.2 Creation of Surrogate Using RAVEN

Due to the large increase in simulation time, the relative issues with co-simulation initialization routines, and the fact that there is no feedback to the rest of the grid, the FMI created for use in the RAVEN surrogate training was reduced to having only a single input (turbine demand) with no connected outputs. This setup allows the natural gas turbine to keep all the initialization pieces of the "infinite" grid self-contained, thus drastically improving the initialization routine and system robustness. Since the turbine power is a variable given by the FMU, and no feedback is used in other units' control systems, the turbine power was not required to be an external variable for the initial export. The FMI/FMU (GTTP.fmu) exported to RAVEN is shown in Figure 53.

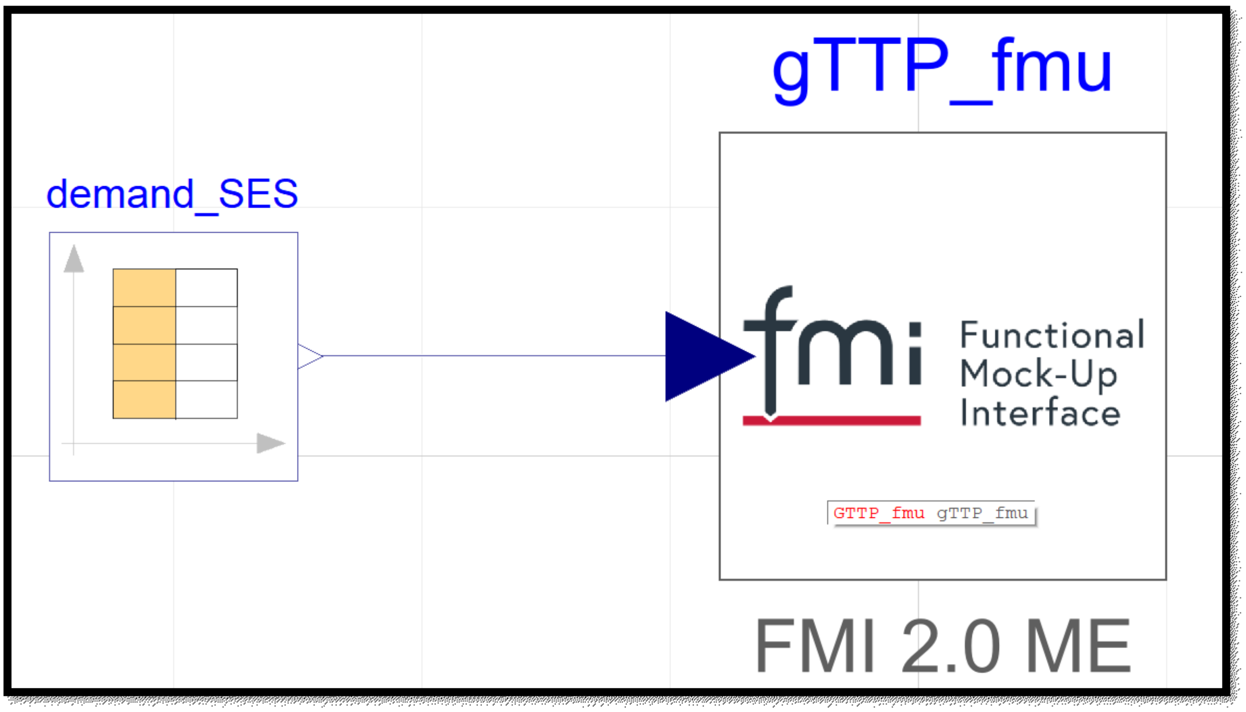


Figure 53. Simplified model of the FMI for RAVEN surrogation.

To construct a RAVEN-based AI to surrogate the response of the turbine component, the FMIFMU RAVEN importer described in Section was used to drive the Dymola-exported FMI/FMU model.

Since the turbine's response to changes in the demand is very quick (very limited inertia) and almost perfectly linear, a Support Vector Regressor with linear kernel **Error! Reference source not found.** was selected for surrogating the response. The turbine FMI/FMU GTTP.fmu was loaded via the FMIFMU RAVEN importer and its demand sampled (1,000 Monte Carlo samples) between 0 and 35 MW to capture the model's full domain of variability. Finally, the Support Vector Regressor was trained (constructed) and exported to a "brand-new" FMI/FMU (GTTProm.fmu) by the RAVEN FMI/FMU exporter (co-simulation), as described in Section 6.2.

To validate the RAVEN AI FMI/FMU, a cross-validation assessment was performed in RAVEN, and, due to the pure linearity of both the turbine and AI models, its average R2 score was >0.99. This is further demonstrated in Figure 54 and Figure 55 which show a comparison of the Dymola-generated turbine FMI/FMU and the RAVEN AI FMI/FMU, with the models demonstrating good agreement.

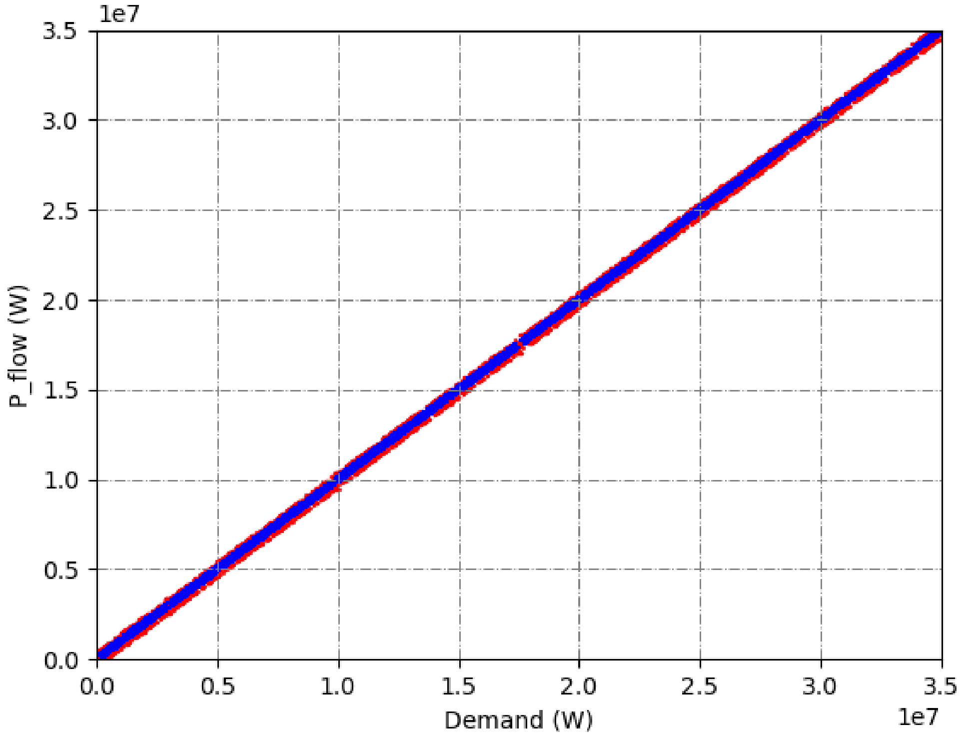


Figure 54. Comparison of the Modelica/Dymola GTTP.fmu response (P_flow) and the RAVEN AI-based GTTProm.fmu.

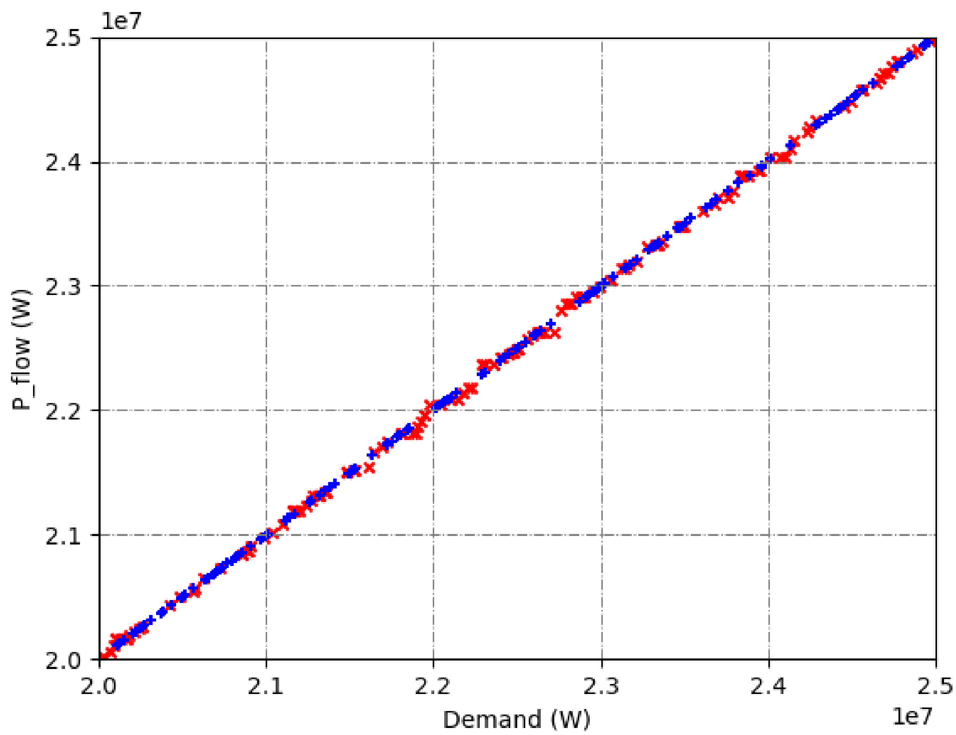


Figure 55. Closeup of the comparison of the Modelica/Dymola GTTP.fmu response (P_flow) and the RAVEN AI-based GTTProm.fmu.

7.3 Comparison of Results

Section 7.2 described the process of constructing an AI model exported in an FMI/FMU from RAVEN. To demonstrate the concept of the “plug-and-play” framework, along with the usage of AI for accelerated analysis, the integrated energy park model was simulated, both using the original Dymola model (FMI/FMU) for the gas turbine and using the RAVEN AI-based model.

Figure 56 shows the integrated energy park FMI/FMU exported via Dymola. Among the different variables and outputs is the model fulfillment of the gas turbine model’s demand. Such output represents the link between the IES park, and the turbine chosen for the demonstration.

Figure 57 and Figure 58 show the setup of the integrated energy park along with the detailed Dymola FMI/FMU and the RAVEN AI-based FMI/FMU, respectively. Both models were simulated in an ad-hoc *Python* code (master simulator) using the FMPy package.

Using the above-mentioned FMI/FMU setup, the two 5-hour simulations were run in the master simulator (*Python* code using FMPy). Since the RAVEN AI-based FMI/FMU can be evaluated in mere milliseconds, the simulation of the setup with the AI was much faster (~20%) to complete, making the computation time for the turbine evaluation completely negligible; indeed, the AI FMI/FMU almost zeroed out the CPU time for the turbine simulation, and the totality of the CPU time was used to simulate the remaining systems in the integrated energy park, which were more complex and computationally intensive.

Figure 59 and Figure 60 show a comparison of the turbine responses in the integrated energy park using the Dymola FMI/FMU and the RAVEN AI-based FMI/FMU. Over the course of the full five-hour simulation, all the results were in near-perfect agreement with the setpoints. The results show that the setup using the RAVEN AI FMI/FMU outperformed (in terms of speed) the Dymola model, with no loss of accuracy.

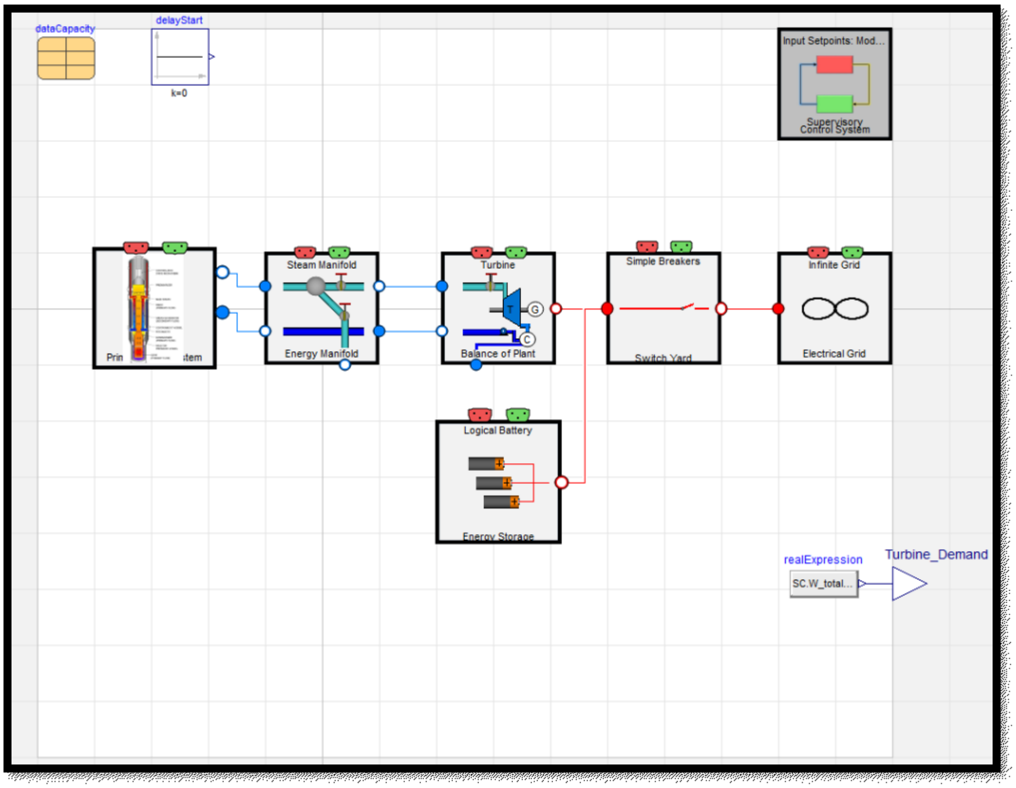


Figure 56. Integrated energy park (excluding the turbine) FMI/FMU generated with Dymola.

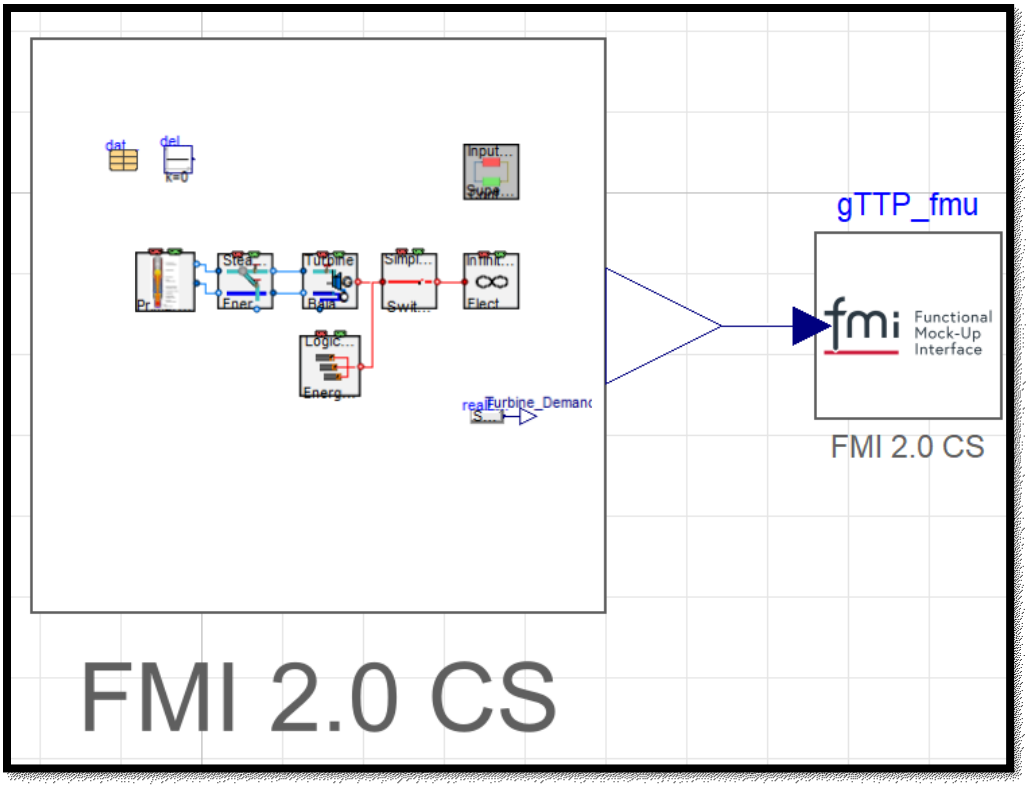


Figure 57. Integrated energy park FMI/FMU, including the Dymola GTTP model.

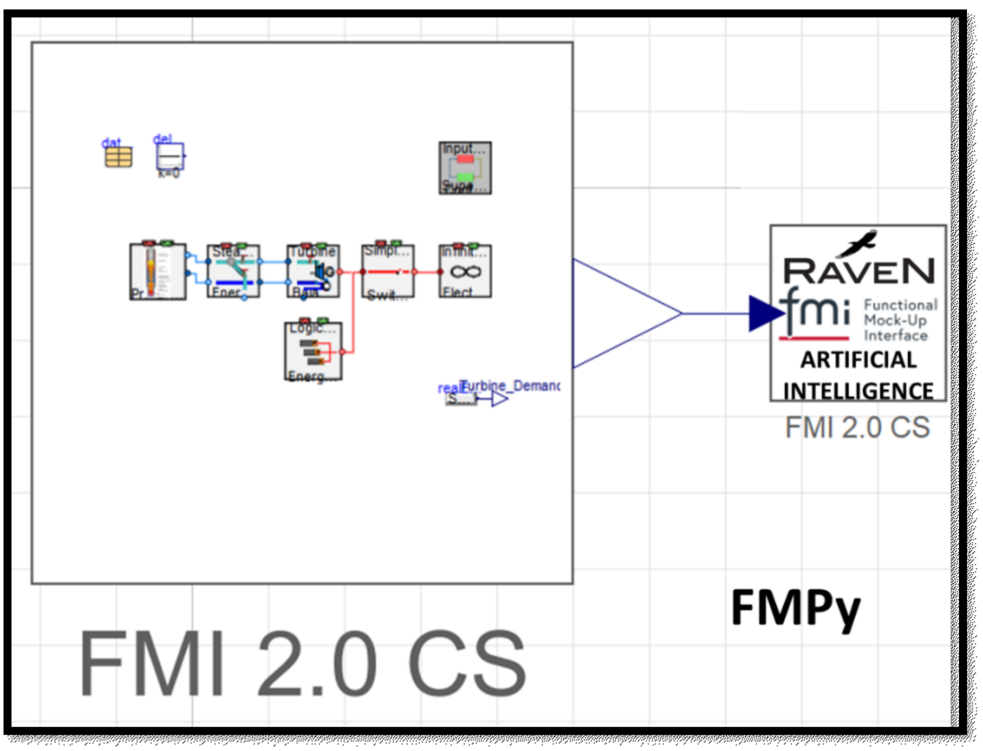


Figure 58. Integrated energy park FMI/FMU, replacing the Dymola GTTP model with the RAVEN AI-based FMI/FMU.

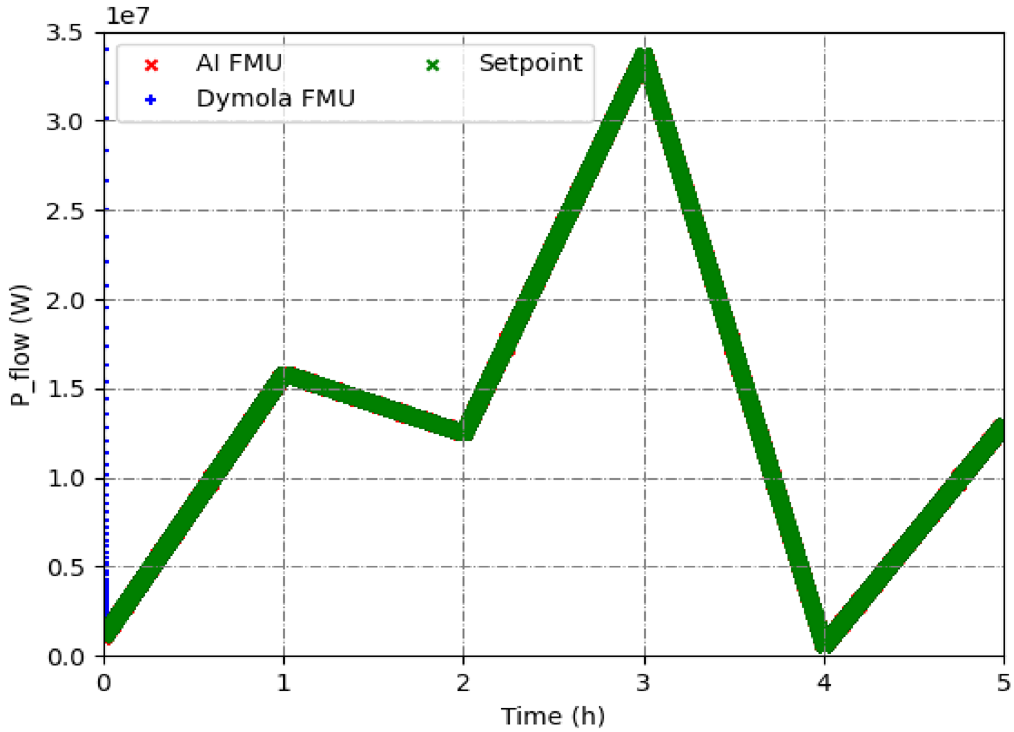


Figure 59. Co-Simulation FMI/FMU (Dymola) vs. RAVEN AI-based FMI/FMU for a 5-hour simulation of the turbine power vs. setpoint demand for the integrated energy park.

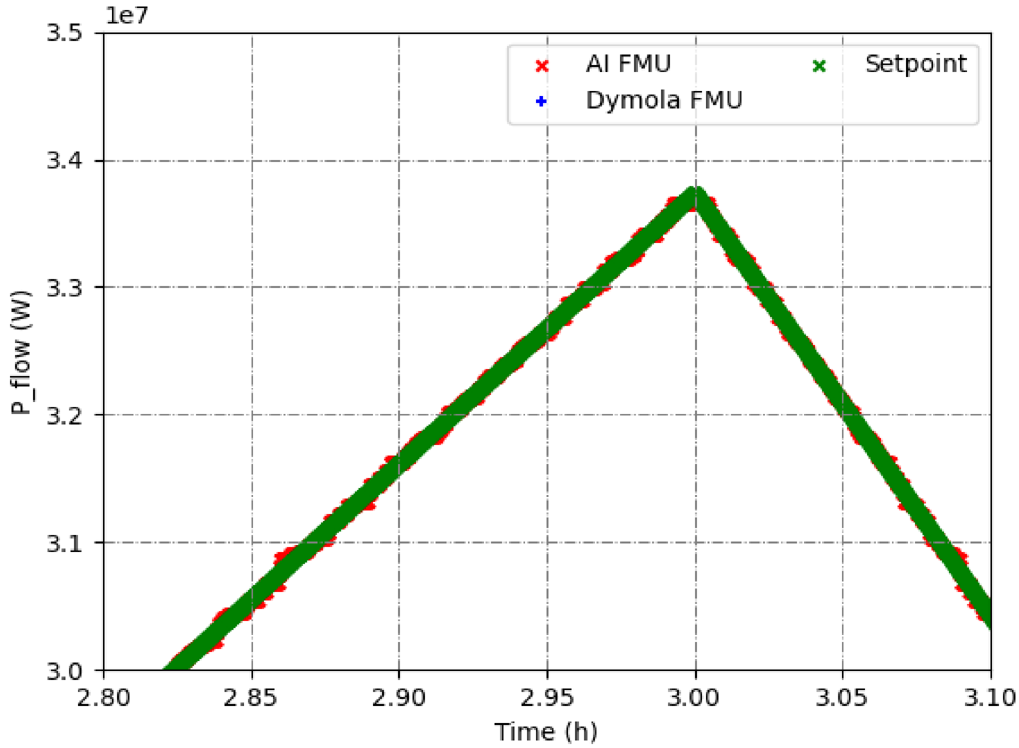


Figure 60. Co-Simulation FMI/FMU (Dymola) vs. RAVEN AI-based FMI/FMU closeup shot of turbine demand vs turbine output.

8. CONCLUSION

This report describes the status of the flexible plug-and-play framework development for design, analysis, and optimization of integrated energy systems. This framework seeks to integrate Modelica and Dymola with RAVEN in terms of both FMI/FMU construction and repository structures intended to simplify model sharing and simulation of complex dynamic systems.

The report provides an in-depth look at the alterations needed to modify existing system-level models for exportation as FMUs. These alterations include modifying specialty "port" variables into their constituent parts as real variables via a new FMI adaptor package added to the existing HYBRID repository. This package includes new adaptors for electrical, fluid, and heat ports for export into the FMIs/FMUs. Examples were included within the FMU adaptor package, illustrating how to properly utilize the system. Several of these examples are discussed in Section 2 of this report.

Simulation results demonstrate that, while minor differences may occur, the overall control, trends, and solution integrity is maintained between the standard Modelica simulation and FMU simulation results. However, it is worth noting that, for small systems, the FMU results have a slower simulation time than the Modelica-only simulation. While this step-by-step process does require several levels of checks, it provides a degree of system flexibility never before experienced. Using this process, a company can provide models that contain proprietary information to separate entities, without disclosing any information about the model that could be considered business sensitive. Such a capability would allow institutions to bypass the necessity of having to "whitewash" data.

In addition to the investigative work being conducted on FMUs and FMIs, a series of updates to the HYBRID repository regression system was completed to ready the repository for open-sourcing. These updates include additional system-level tests for components in the HYBRID repository, as well as increasing the testing level from a mere six tests to 32 and counting. Further, new features have been included in the testing system, such as an initialization subroutine for Dymola models that helps highly nonlinear complex systems initiate their regression test. Additionally, the output keys "numberOfIntervals" and "OutputInterval" were added to the regression system, allowing for consistent comparison points between the reference file and the simulation results between machines. This step is necessary because the commercial Modelica platform Dymola has a series of global output flags that are rarely consistently utilized from one organization to another, yet do not change the trajectories of the solution.

Finally, the work that was deployed to simulate, export, and use FMI/FMU in conjunction with AI algorithms in RAVEN represents a significant step forward in regard to delivering a streamlined process to accelerate simulations and analysis by leveraging RAVEN advanced algorithms. The possibility of using AI exported in FMI/FMU in any FMI/FMU-compatible framework (e.g., FMPy and Dymola) is unique to this framework, posing the basis for deployment of fast simulation, modeling, and analysis accelerations.

Overall, extensive work was completed to develop FMUs and FMIs from existing models and gaining greater understanding of the requirements and limitations of FMI/FMUs.

9. FUTURE WORK

The activities described in this report show the potential of the concept of a "flexible plug-and-play ecosystem" being developed within the IES program and deployed via the creation of FORCE. In order to fulfill the promises of FORCE, several tasks are planned to be carried out in the future of the program:

1) Master Simulator development in RAVEN: in order to automate the deployment of models in a system that is compatible with any FMI/FMU interface, an entity (Master Simulator) needs to be developed within RAVEN. Such development will allow for the simulation of FMI/FMU models (AI, Dymola, etc.) directly within the RAVEN framework allowing for the integration of such models in any RAVEN workflow, in general, and in IES technoeconomic analysis, in particular. The Master Simulator in RAVEN will be based on the EnsembleModel entity (see sec. 5.2), in conjunction with the FMPy library. The Master Simulator is shown in Figure 61.

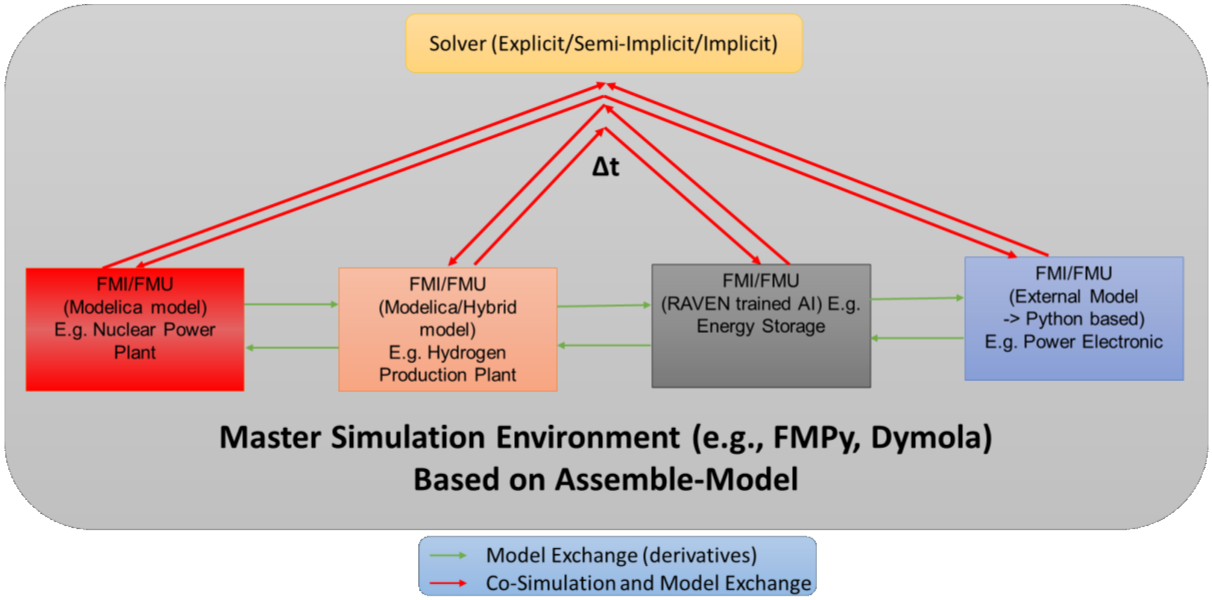


Figure 61. Proposed master simulator within RAVEN.

2) Model Exchange for RAVEN-based models: in section 6.2 the deployment of a system for exporting RAVEN-based AI as FMI/FMU leveraging, the PythonFMU library has been shown. However, the current library only supports FMI/FMU in co-simulation, useful for loosely coupled models but inadequate for tightly coupled systems. To allow for exporting of nonlinear models (e.g. Nuclear Reactor Balance of Plant, Storage, etc.), the PythonFMU library needs to be upgraded to allow for exporting models in model exchange and, consequentially, leverage the capability of RAVEN AI to provide first and second order derivative information.

3) Integration of the FARM supervisory control model: Argonne National Laboratory, in collaboration with Idaho National Laboratory, recently released a RAVEN plugin called Feasible Actuator Range Modifier (FARM) [14],[15]. This plugin oversees deploying supervisory bounding control for dynamic models to ensure physical limitations of the model are not exceeded. This is an additional layer of control on top of the existing physical modeling control systems. While control is still imposed for each individual process, FARM can identify demand signals that cannot be met within safety limits and augments the demand to meet safety specifications. For the FORCE framework to deploy these supervisory controllers the model needs to be exported as FMI/FMU and integrated into the plug-and-play framework.

4) Integration of the HERON plugin: INL has been developing the Holistic Energy Resource Optimization Network (HERON) plugin to construct workflows for solving resource allocation problems inherent to the electrical grid. This plugin oversees the allocation of energy resources within integrated energy systems. The idea of FORCE is to connect HERON with FARM, RAVEN, and HYBRID to solve real world energy allocation problems. With the work completed in FY 2020 the next step is to develop the

interconnection between these different platforms and ensure simulation speed is capable of solving real world problems.

Additional investigative work is planned in order to expand the FMU capabilities within the existing HYBRID repository framework.

10. REFERENCES

[1] C. Rabiti, A.S. Epiney, P. Talbot, J.S. Kim, S. Bragg-Sitton, A. Alfonsi, A. Yigitoglu, S. Greenwood, S.M. Cetiner, F. Ganda, G. Maronati. September 2017. "Status Report on Modeling and Simulation Capabilities for Nuclear-Renewable Hybrid Energy Systems." INL/EXT-17-43441, Idaho National Laboratory.

[2] J.S. Kim, M. McKellar, S. Bragg-Sitton, R. Boardman. October 2016. "Status Report on the Component Models Developed in the Modelica Framework: High-Temperature Steam Electrolysis & Gas Turbine Power Plant." INL/EXT-16-40305, Idaho National Laboratory.

[3] J.S. Kim, K.L. Frick. May 2018. "Status Report on the Component Models Developed in the Modelica Framework: Reverse Osmosis Desalination Plant & Thermal Energy Storage." INL/EXT-18-45505, Idaho National Laboratory.

[4] K.L. Frick. August 2019. "Status Report on the NuScale Module Development in the Modelica Framework." INL/EXT-19-55520, Idaho National Laboratory.

[5] A. Alfonsi, C. Rabiti, D. Mandelli, J. Cogliati, C. Wang, P.W. Talbot, D.P. Maljovec, C. Smith. 2016. "RAVEN Theory Manual and User Guide." INL/EXT-16-38178, Idaho National Laboratory.

[6] C. Rabiti, A. Alfonsi, J. Cogliati, D. Mandelli, R. Kinoshita, S. Sen, C. Wang, J. Chen. 2017. "RAVEN User Manual." INL/EXT-15-34123, Idaho National Laboratory.

[7] Dassault Systems. "DYMOLA Systems Engineering: Multi-Engineering Modeling and Simulation Based on Modelica and FMI." Accessed July 24, 2020. https://www.3ds.com/products-services/catia/products/dymola/.

[8] M.S. Greenwood: TRANSFORM - TRANsient Simulation Framework of Reconfigurable Models. Computer Software. https://github.com/ORNL-Modelica/TRANSFORM-Library. 07 Nov. 2017. Web. Oak Ridge National Laboratory. doi:10.11578/dc.20171109.1. Available: https://github.com/ORNL-Modelica/TRANSFORM-Library.

[9] K. Frick, A. Alfonsi, C. Rabiti. 2020. "Hybrid User Manual." INL/MIS-20-60624, Idaho National Laboratory.

[10] A. Alfonsi, K. Frick, S. Greenwood, C. Rabiti. 2020. "Status on the Development of the Infrastructure for a Flexible Modelica/RAVEN Framework for IES." INL/EXT-20-00160, Idaho National Laboratory.

[11] K. Frick, A. Alfonsi, C. Rabiti. 2020. "Flexible Modelica/RAVEN Framework for IES." INL/EXT-20-00419, Idaho National Laboratory.

[12] FMPy. 2020. "FMPy 0.2.21." Accessed July 8, 2020. https://pypi.org/project/FMPy/.

[13] PyFMI. 2018. "PyFMI 2.5." Accessed March 25, 2020. https://pypi.org/project/PyFMI/.

[14] H. Wang, R. Ponciroli, A. Alfonsi. Feasible Actuator Range Modifier (FARM). https://github.com/Argonne-National-Laboratory/FARM

[15] H. Wang, R. Ponciroli, R. Vilim, A. Alfonsi, C. Rabiti. "A Recursive Data-Driven Approach to State Variable Selection and Digital Twin Derivation." in *12th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies. (NPIC&HMIT 2021)*. June 2021.

APPENDIX A – HYBRID USER MANUAL

MANUAL

INL/MIS-20-60624
Revision 0
Printed March 30, 2021

HYBRID User Manual

Konor Frick, Andrea Alfonsi, Cristian Rabiti

Prepared by
Idaho National Laboratory
Idaho Falls, Idaho 83415

The Idaho National Laboratory is a multiprogram laboratory operated by
Battelle Energy Alliance for the United States Department of Energy
under DOE Idaho Operations Office. Contract DE-AC07-05ID14517.

Idaho National Laboratory

Issued by the Idaho National Laboratory, operated for the United States Department of Energy by Battelle Energy Alliance.

INL/MIS-20-60624
Revision 0
Printed March 30, 2021

HYBRID User Manual

Konor Frick

Andrea Alfonsi

Cristian Rabiti

Contents

1 Introduction

One of the goals of the HYBRID modeling and simulation project is to assess the economic viability of hybrid systems in a market that contains renewable energy sources like wind. The hybrid system would be a nuclear reactor that not only generates electricity, but also provides heat to another plant that produces by-products, like hydrogen or desalinated water. The idea is that the possibility of selling heat to a heat user absorbs (at least part of) the market volatility introduced by the renewable energy sources.

The system that is studied is modular and made of an assembly of components. For example, a system could contain a hybrid nuclear reactor, a gas turbine, a battery and some renewables. This system would correspond to the size of a balance area, but in theory any size of system is imaginable. The system is modeled in the 'Modelica/Dymola' language. To assess the economics of the system, an optimization procedure is varying different parameters of the system and tries to find the minimal cost of electricity production.

1.1 Modelica Models

Idaho National Laboratory (INL) has been developing the NHES package, a library of high-fidelity process models in the commercial Modelica language platform Dymola since early 2013 [1], [2], [3], [4]. The Modelica language is a non-proprietary, object oriented, equation-based language that is used to conveniently model complex, physical systems. Modelica is an inherently time-dependent modeling language that allows the swift interconnection of independently developed models. Being an equation-based modeling language that employs differential algebraic equation (DAE) solvers, users can focus on the physics of the problem rather than the solving technique used, allowing faster model generation and ultimately analysis. This feature alongside system flexibility has led to the widespread use of the Modelica language across industry for commercial applications. System interconnectivity and the ability to quickly develop novel control strategies while still encompassing overall system physics is why INL has chosen to develop the Integrated Energy Systems (IES) framework in the Modelica language.

1.2 Individual Components

The current version of the NHES library employs both third party components from the Modelica Standard Library [5] and TRANSFORM [6] and components developed internal to the project for specific subsystems. For example, the NHES library contains a large variety of models for the development of a high-temperature steam electrolysis plant, a gas turbine, a basic Rankine cycle balance of plant, and a light water nuclear reactor. Components included in the library that support the development of these systems include 1-D pipes, pressurizers, condensers, turbines (steam and gas), heat exchangers, a simple logic-based battery, a nuclear fuel subchannel, etc. Third party models include numerous additional models including source/sink components (e.g., fluid boundary conditions), additional heat exchanger models, logical components for control system development, multi-body components, additional supporting functions (e.g., LAPACK, interpolation, smoothing), etc. Please see the specific libraries for additional information.

1.3 Hybrid Requirements

The repository itself can be found here: `https://hpcgitlab.inl.gov/hybrid/hybrid`

Software requirements are as follows:

1. Commercial Modelica platform Dymola – `https://www.3ds.com/products-services/catia/products/dymola/latest-release/`.
2. Risk Analysis and Virtual ENviroment (RAVEN) – `https://raven.inl.gov/SitePages/Software%20Infrastructure.aspx`
3. Python 3 – `https://docs.conda.io/en/latest/miniconda.html`
4. Microsoft Visual Studio Community Edition. – `https://visualstudio.microsoft.com/downloads/`

Note: Steps 3 and 4 can be accomplished by following the RAVEN installation instructions in step two. The installation procedure will be outlined below. All physical models are run within the Dymola simulation framework graphical user interface (GUI). Background information on the Modelica as a language as well as good general guidance on coding practices can be found at the two references shown below.

1. `https://webref.modelica.university/`
2. `https://mbe.modelica.university/`

2 HYBRID Installation Procedure

2.1 Overview

The installation of the HYBRID repository is a straightforward procedure; depending on the usage purpose and machine architecture, the installation process slightly differs.

In the following sections, the recommended installation procedure is outlined. For alternatives, we encourage checking the RAVEN wiki. The windows 10 machine on which HYBRID is tested and developed, uses the standard installation procedures outlined below.

The installation process will involve four steps:

- Installing prerequisites, which depends on your operating system;
- Installing conda;
- Installing RAVEN.
- Installing HYBRID.

2.2 Cloning the Hybrid Repository

The first step in installing the package is to clone the HYBRID repository. To do this, use

```
git clone git@hpcgitlab.inl.gov:hybrid/hybrid.git
```

This will download the repository into a folder called 'hybrid'. To go inside the folder, use

```
cd hybrid
```

Note: This only works if you have access to the HYBRID repository.

Note2: If you are outside INL, be sure you have the ssh tunnel to INL set-up. For instructions, please check the HPC GitLab Connectivity page.

Note3: Be sure to have the proxy settings correct. See RAVEN wiki .

Note4: An ssh key needs to be registered for hpcgitlab.inl.gov. This key should be good for both, the HYBRID repository as well as the CashFlow plugin. Instructions to generate and register ssh keys can be found here. If you have troubles accessing the repository, see Installation trouble shooting.

2.2.1 Install RAVEN and its plugins as a sub-module

The next step is to download and install RAVEN and the submodule (e.g. TEAL, HERON) plugins as a sub-module of the HYBRID repository.

A submodule allows you to keep another Git repository in a subdirectory of your repository. The other repository has its own history, which does not interfere with the history of the current repository. This can be used to have external dependencies such as third party libraries for example.

In order to get RAVEN do the following in the hybrid folder

```
git checkout devel
```

Update the Branch

```
git pull
```

to add RAVEN as a submodule

```
git submodule update --init --recursive
```

Install and Compile RAVEN. Once you have downloaded RAVEN as a sub-module, you have to install it. go to the RAVEN Wiki for information about how to install it. Run all the tests outlined in the RAVEN wiki.

2.2.2 Inform the Framework Paths

In order to set up the hybrid repository, you must inform the framework about the location of the Dymola python interface. For doing so, navigate to the hybrid directory:

to add RAVEN as a submodule

```
cd <path to your hybrid repository>/hybrid
```

Run the following command:

```
./scripts/write_hybridrc.py -p DYMOLA_PATH
```

Where DYMOLAPATH is the path to the python interface egg folder in the DYMOLA installation locally. For example:

```
./scripts/write_hybridrc.py -p
        "/c/Program\␣Files/Dymola\␣2020x/Modelica/Library/
␣␣␣␣␣␣␣␣python_interface/dymola.egg"
```

2.3 Setup of Dymola for the Regression Testing System

To properly setup the Hybrid repository to work with the regression system one needs to download Dymola as mentioned above and activate the license. Once those two steps are complete the dymola.mos file needs to be edited. The dymola.mos file is the file that tells Dymola what libraries to load when the application is opening and where the working directory is located. For the automatic regression test system to properly test the downloaded library the proper NHES library must be loaded automatically by dymola in the dymola.mos file located at C:/Program Files/Dymola 2020/insert/dymola.mos for example. To properly run the tests the NHES and the TRANSFORM libraries need to be automatically loaded by Dymola upon startup. This can be accomplished by adding to the Dymola.mos until it looks something like:

```
RunScript("$DYMOLA/insert/displayunit.mos", true);
definePostProcessing("SDF output", "Convert result file to SDF format",
"Modelica.Utilities.System.command(\"\\\"%DYMOLA%/bin/dsres2sdf\\\"
%RESULTFILE%.mat %RESULTFILE%.sdf\")");

openModel("C:\Users\FRICKL\Desktop\TRANSFORM3_20_2020\TRANSFORM-
Library\TRANSFORM\package.mo"); //Loads Transform package.mo

openModel("C:\msys64\home\FRICKL\hybrid_devel\hybrid\models
 \NHES\package.mo");//Loads NHES package from hybrid directory

cd("C:\Users\FRICKL\Desktop\TESsystem");
//Place where all the Dymola runs will occur.
```

Having the Dymola.mos file written like this will allow Dymola to automatically load all the needed libraries for Regression testing when conducted using the Regression Test Harness. It should be noted that typically the dymola.mos file will need its permissions to be changed to allow a user to write this. On a Windows machine this is done by running as an administrator on the system and changing the properties of the file to include “read and write” rights, as opposed to “read-only” rights.

2.4 Run Regression tests related to the HYBRID project

Now that the dymola.mos file has been edited to automatically load the TRANSFORM and NHES packages one can run all tests associated with the HYBRID repository. To do this follow the instructions below.

```
cd <path to your hybrid repository>/hybrid
./run_tests -jX -lY --only-run-types ZZ
```

where:

- ”X” is the number of processors to use for testing
- ”Y” is the maximum load to limit to execute tests
- ”ZZ” is the subtype of tests to be run. Currently only ”raven” and ”dymola” are available.

If all tests need to be run, just execute the following command.

```
cd <path to your hybrid repository>/hybrid
./run_tests -jX -lY
```

The output will look like the following:

```
######################################################################
#
#  Testing of Hybrid RAVEN Modules  (./run_tests)
#
######################################################################
/c/msys64/home/FRICKL/cleaning_hybrid/hybrid
Found $DYMOLA_PATH and set to C:/Program Files/Dymola 2021/Modelica/Library/python_interface/dymola.egg
Loading raven_libraries conda environment ...
CONDA
... Run Options:
... Mode: 1
... Verbosity: 0
... Clean: 0
... Mode: CONDA
... Conda Defs:
... Loading RAVEN libraries ...
... Detected OS as --os windows ...
... Using Python command python
... $RAVEN_LIBS_NAME set through raven/.ravenrc to raven_libraries
... >> If this is not desired, then remove it from the ravenrc file before running.
... >> RAVEN environment is named "raven_libraries"
... found conda path in ravenrc: C:/Users/FRICKL/AppData/Local/Continuum/miniconda3/etc/profile.d/conda.sh
... >> If this is not the desirable path, rerun with argument --conda-defs [path] or remove the entry from raven/.ravenrc file.
... Found conda definitions at C:/Users/FRICKL/AppData/Local/Continuum/miniconda3/etc/profile.d/conda.sh
conda 4.8.3
raven_libraries          C:\Users\FRICKL\AppData\Local\Continuum\miniconda3\envs\raven_libraries
... Found library environment ...
... Activating environment ...
... Activating environment ...
... done!
rook: loading init file "C:/msys64/home/FRICKL/cleaning_hybrid/hybrid/scripts/rook.ini"
rook: ... loaded setting "add_non_default_run_types = dymola,raven"
rook: ... loaded setting "add_run_types = dymola,raven"
rook: ... loaded setting "test_dir = tests"
rook: ... loaded setting "testers_dirs = scripts/testers,raven/scripts/TestHarness/testers/"
rook: found 27 test dirs under "tests" ...
rook: loading init file "C:/msys64/home/FRICKL/cleaning_hybrid/hybrid/scripts/rook.ini"
rook: ... loaded setting "add_non_default_run_types = dymola,raven"
rook: ... loaded setting "add_run_types = dymola,raven"
rook: ... loaded setting "test_dir = tests"
rook: ... loaded setting "testers_dirs = scripts/testers,raven/scripts/TestHarness/testers/"
(1/27) Success( 40.44sec)tests\dymola_tests\BOP_L1_Boundaries_a_Test\
(2/27) Success( 41.22sec)tests\dymola_tests\BOP_L1_Boundaries_b_Test\
(3/27) Success( 15.27sec)tests\dymola_tests\Desalination_1_pass\
(4/27) Success( 15.84sec)tests\dymola_tests\Desalination_2pass_mixing\
(5/27) Success( 14.17sec)tests\dymola_tests\Desalination_2_pass\
(6/27) Success( 24.64sec)tests\dymola_tests\Desalination_NHES_basic\
(7/27) Success( 22.12sec)tests\dymola_tests\Desalination_ROmodule\
(8/27) Success( 42.10sec)tests\dymola_tests\Desalination_NHES_complex\
(9/27) Success( 17.21sec)tests\dymola_tests\GTTP_Test\
(10/27) Success( 36.21sec)tests\dymola_tests\Generic_Modular_PWR\
(11/27) Success( 23.93sec)tests\dymola_tests\HTSE_Power_Test\
(12/27) Success( 32.00sec)tests\dymola_tests\HTSE_Steam_Test\
(13/27) Success( 39.60sec)tests\dymola_tests\NSSS_test\
(14/27) Success( 55.31sec)tests\dymola_tests\NuScale_4Loop\
(15/27) Success( 36.51sec)tests\dymola_tests\NuScale_Nominal_Test\
(16/27) Success( 15.14sec)tests\dymola_tests\Simple_Breakers_Test\
(17/27) Success( 26.16sec)tests\dymola_tests\NuScale_primary_test\
(18/27) Success( 15.70sec)tests\dymola_tests\StepDownTurbines\
```

```
(19/27) Success( 16.61sec)tests\dymola_tests\StepDownTurbines_complex\
(20/27) Success( 14.34sec)tests\dymola_tests\Supervisory_Control_Test\
(21/27) Success( 14.31sec)tests\dymola_tests\Test_Battery_Storage\
(22/27) Success( 34.01sec)tests\dymola_tests\Test_Thermal_Storage\
(23/27) Success( 37.58sec)tests\dymola_tests\Thermocline_Cycling\
"failing"
(24/27) Skipped(  None!  )tests\dymola_tests\TightlyCoupled_FY18_Battery\
"failing"
(25/27) Skipped(  None!  )tests\dymola_tests\TightlyCoupled_FY18_TES\
(26/27) Success( 25.44sec)tests\dymola_tests\Thermocline_Insulation\
(27/27) Success( 31.37sec)tests\raven_tests\train\TrainArmaOnData

PASSED: 25
SKIPPED: 2
FAILED: 0
```

7

3 Running and Creating New Code

The physical modelica models are the cornerstone of the Hybrid repository. They are designed to represent physical industrial processes that can be configured into different potential integrated energy systems (IES). Table 1 gives an overview of the main types of integrated energy systems, along with models currently incorporated in the hybrid repository.

3.1 Understanding and Running Existing Models

The hybrid repository is broken down using the templating system shown in Figure 1. The top level is the overall system package which incorporates all of the Modelica models contained within the NHES package. Then inside of the NHES package are the different subpackages (Systems, Electrical, Thermal, etc. . .). Within each of the subpackages are further subpackages as seen in the Systems package. Within the Systems package there are further subpackages called *SubSystem Category* (Examples, PrimaryHeatSystem, EnergyStorage, etc. . .). Then within these SubSystem Categories there is yet another level of subpackage that is called *SubSystem_Specific*. Within the *SubSystem_Specific* category is where development takes place and potential configurations of the different processes take shape. Inside each SubSystem_Specific there is a template that includes *Examples*, *Subsystem Dummy*, *CS_Dummy*, *ED_Dummy*, Data, BaseClasses, and usually a Components folder. For existing systems the Examples folder contains a runnable example the user can execute to see how the code runs at a top level and what scenarios it is capable of running. An example of which is depicted in Figure 2. For each example the user can double click on the main system which will open the table in the upper left hand corner of Figure 2 which provides inputs for the user to change parameters about the system. Then if the user wishes to modify the control system utilized they can either choose from the drop-down menu, or click the button at the right of the "CS" line to open the table in the lower left hand section which provides options to delay when different control systems come online.

These example tests provide a good way for the user to become associated with the large subsystem in terms of how they work and the different parameters that can be utilized to tune and interact with the models. In addition to the examples file a deeper understanding of the model can be realized by looking into the component structure of the model. This is typically accomplished through looking at the filled-out Subsystem Dummy section. For the Westinghouse 4-Loop plant this can be seen in Figure 3. This model includes several subcomponents connected into a singular model. Each model with its' own set of parameters. Using this version of the model it is possible to discern the inner workings of the model in terms of sensors, physical descriptions of the code, inlet and outlet conditions, and system dependencies. In addition to the SubSystem Dummy section, large process models typically include a control system section which is created from the CS_Dummy file in the branch. These control system files can be added as a control system for the Subsystem to control different valves, pumps, and control drives within the process from the drop-

Table 1: Examples of large-Scale Systems within the Hybrid repository used in the creation of Integrated Energy Systems.

Category	Description	Specific Example
Primary Heat System	Provides base load heat and power	Nuclear Reactor
Energy Manifold	Distributes thermal energy among subsystems	Steam Manifold
Balance of Plant	Serves as primary electricity supply from energy not used in other subsystems	Turbine, condenser, and feed-train
Industrial Process	Generates high value product(s) using heat and electricity from other systems	Steam Electrolysis, gas to liquids, reverse osmosis
Energy Storage	Serves as energy buffer to increase overall system robustness and system that can increase profits during highly fluctuating energy prices	Electric Batteries, Two-Tank Sensible Heat Storage, Thermocline
Secondary Energy Source	Delivers small amounts of topping heat required by industrial processes or rapid dynamics in grid demand that cannot be met the remainder of the system	Natural Gas Turbine
Switch Yard	Distributes electricity among subsystems, including the grid	Electricity Distribution
Electrical Grid	Sets the behavior of the grid connected to the IES	Large Grid Behavior
Control Center	Provides proper system control and test scenarios	Control System/ Supervisory Control System

System.png

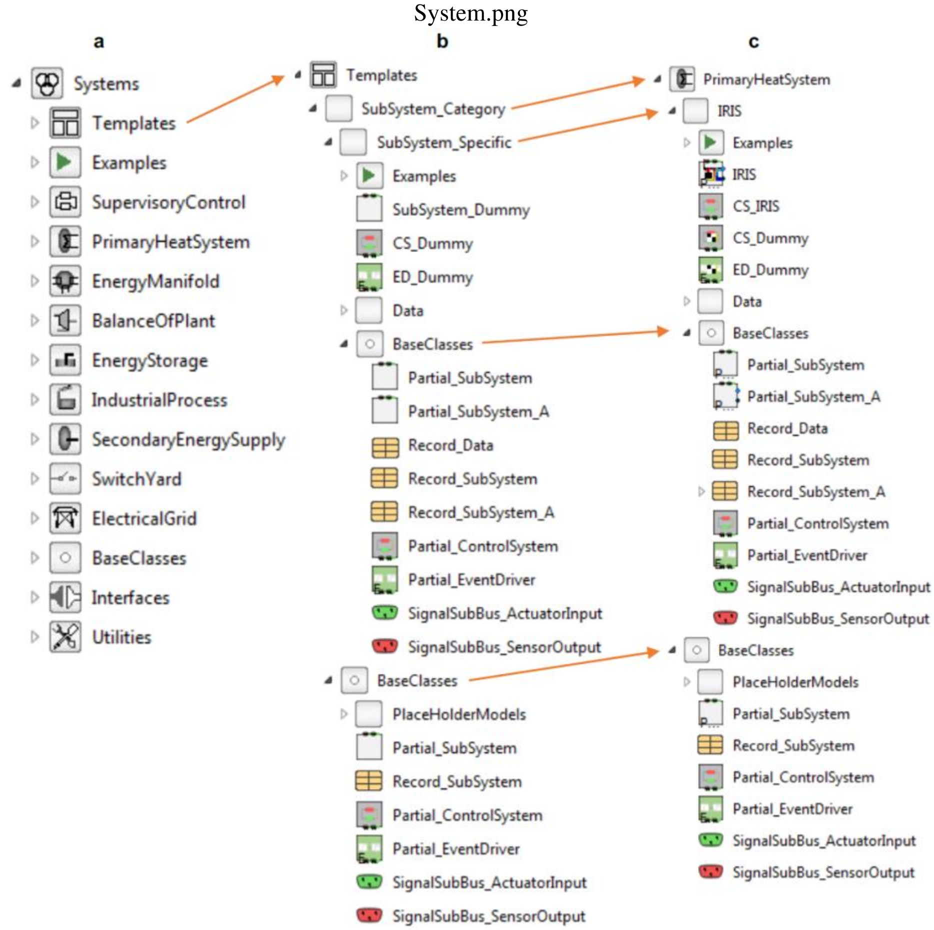


Figure 1: a) Overall Modelica package, b) template structure for creating new subsystem categories and specific subsystem models within a category, c) example of a specific implementation of a primary heat system using the template approach.

10

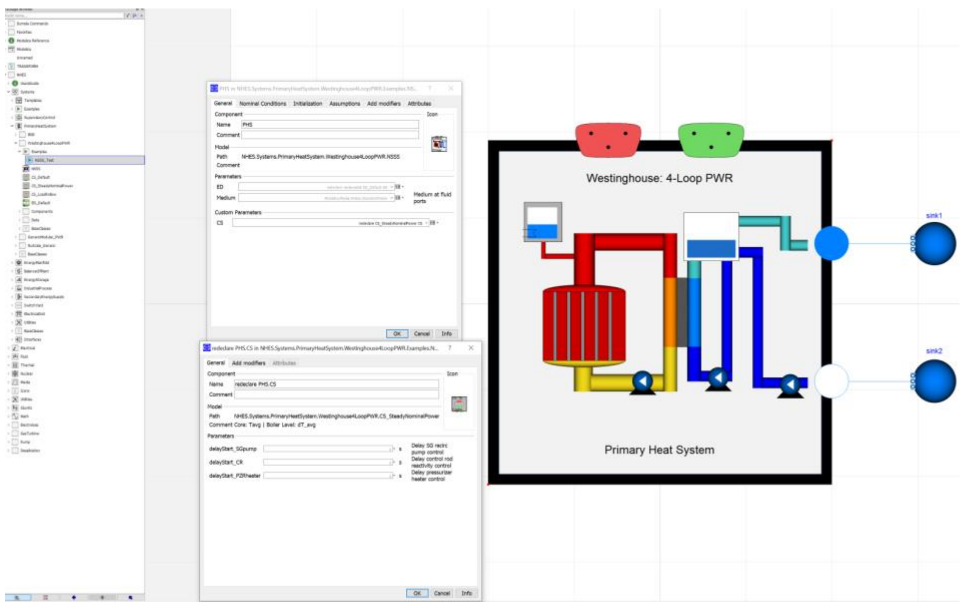


Figure 2: Exploded view of the NSSS_Test example within the NHES library with control system options opened up

11

down menu in the “CS” section seen in Figure 2. large process systems may have several different potential control systems based upon what type of Integrated Energy System they are operating within. An illustration of one of the Westinghouse Control systems can be seen in Figure 4.

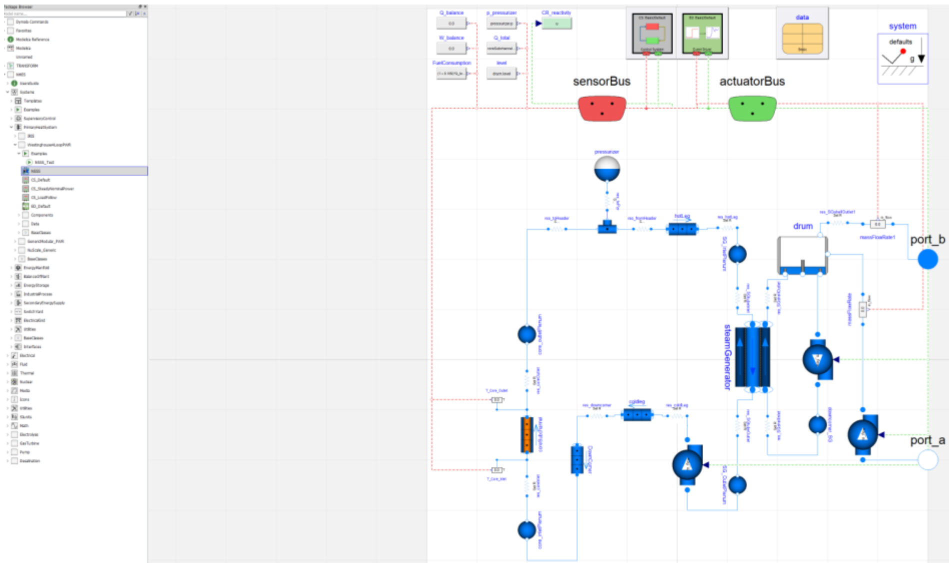


Figure 3: Subsystem for the Westinghouse-4 Loop model.

Assuming the user is creating a new package with new components specific to the model it is best to include those models with a “components” folder in the subpackage containing the “BaseClasses” folder. The Data folder is typically where the main data structures in terms of “records” of kept for the process model. Records are files that are intended to be used as an input deck to the main model for use as a set of “parameters” the components will read from. The ED_dummy file within the *Subsystem_Specific* category is the Event Driver file and is rarely used and can be ignored from a user perspective.

3.1.1 Modifying Existing Models for Specific Runs

A starting point from which a user can begin model development and analysis is from an existing Example model. To properly edit the *Examples* within the hybrid repository while still maintaining the regression system one needs to create a duplicate model of the example file that is to be edited. This can be done by right clicking on the file as shown below and creating a duplicate class.

This file will the be placed in the Examples folder where edits can be made to it for new and

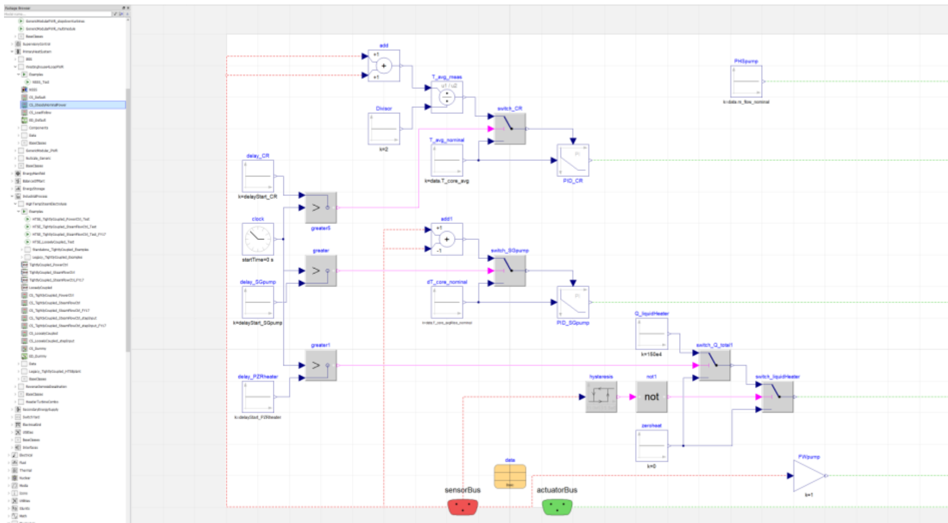


Figure 4: Control System (CS) for the Westinghouse 4 Loop model

unique runs. This includes things such as new control schemes, sizing, timeruns, etc..

3.2 Configuring Existing Models into Integrated Energy Systems

Each subsystem of the Integrated Energy Systems is inherently interesting on its own and large spans of time can be spent researching and fine tuning them independently. However, the developer team is aware that in the evolving energy landscape, and to the extent users will come across this repository, that integrated energy systems are the primary focus.

This focus includes systems that involve the distribution of heat and electrical energy among several subsystems and the control schemes utilized to accomplish this. Therefore, this section seeks to provide an introductory understanding of how to connect subsystems together within the Hybrid repository. To accomplish this the NuScale_Coupling_Test Example will be created starting from the GenericModularPWR_park system.

The first step is to take a similar example that has the Supervisory Control System in the top level. In this case the GenericModularPWR_park was used. A duplicate class was created and all the components aside the Steam Manifold, Turbine, Simple Breakers, infinite grid, supervisory control system, delay start, and data capacity were removed. See below.

Then the primary side of the NuScale was added in this case the *NuScale_Taveprogram* version

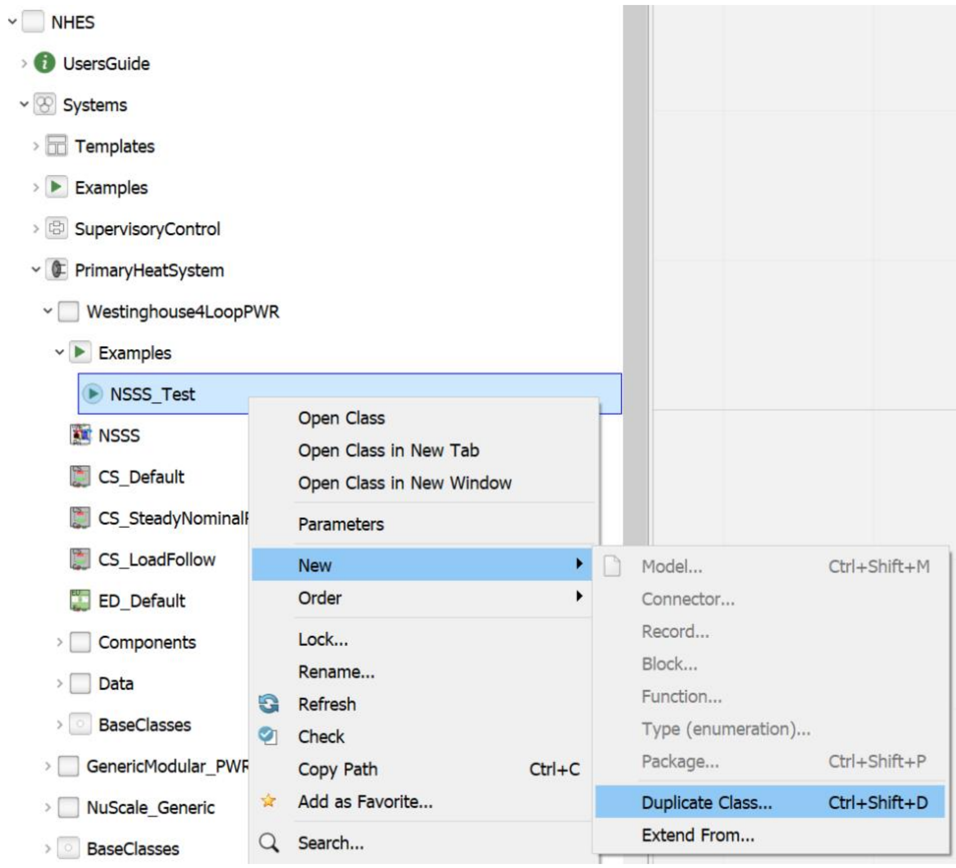


Figure 5: Creating a duplicate class for model runs.

14

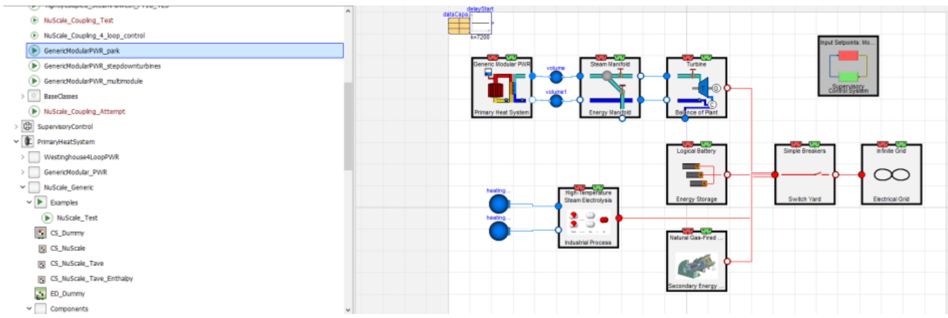


Figure 6: Initial Integrated Energy System Starting Point

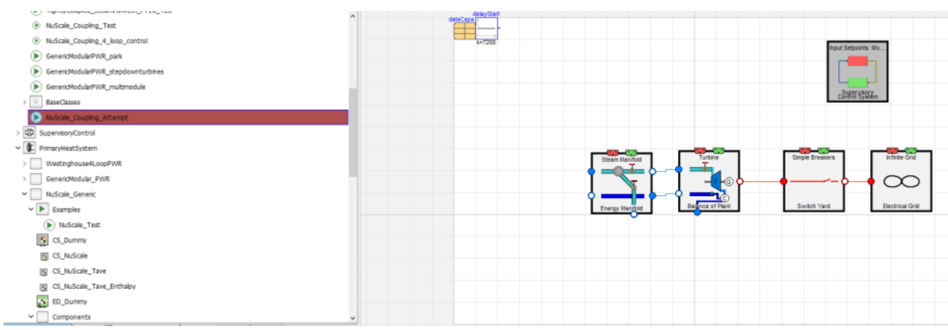


Figure 7: Rearrangement of Initial Energy System

of the NuScale primary unit.

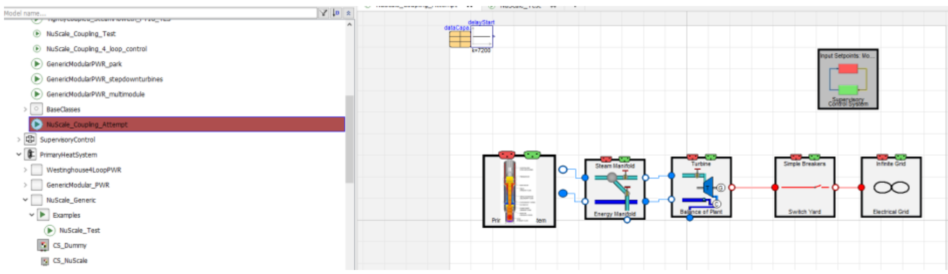


Figure 8: Creation of NuScale Energy System

Then from this point it is a matter of telling the systems what control schemes to use. For this system the reactor operates to meet a certain primary system average temperature in accordance with the turbine output. To input this the control system: PrimaryHeatSystem.NuScaleGeneric. CS_NuScale_Tave was used with input: W_turbine = BOP.powerSensor.power and W_Setpoint = SC.W_totalSetpoint_BOP, see Figure 9. And the turbine control scheme is modified to reflect a once through system type control strategy where the turbine control valve operates to meet a constant pressure in the turbine, Figure 10. While it is noted that is not the official control strategy strictly speaking for the NuScale system nor is it the one used in load following scenarios in the hybrid repository, it does provide a baseline for which to control the system and modifications can be made from this point. The power setpoints in the BalanceOfPlant.Turbine.CS_OTSG_Pressure control module are 160MW for both Reactor_Power and Nominal_Power while p_nominal parameter is set to BOP.port_a_nominal.p to ensure a single parameter value is carried throughout the system. Additionally, W_totalSetpoint is set to SC.W_totalSetpoint_BOP, Figure 10.

To complete the construction of the model the systems need to match on the boundaries. To do this the values from the primary heat system need to be transferred to the Steam Manifold under the nominal values tab, Figure 11 and 12.

3.3 Test Creation

To create a regression test once a user develops an example test in the Dymola NHES library can be accomplished through a couple of settings. In the Dymola simulation setup tab in the output tab uncheck the store at variable events box. Then click store in model button and check the output box, click ok, then click ok again, then Save the model. Example settings are shown in Figure 13.

In the simulateModel command one of the following two flags is required. Either "*numberOfIntervals*" or "*OutputInterval*". numberOfIntervals tells dymola how many output intervals to make. OutputInterval tells dymola at what timestep interval should an output be present for comparison.

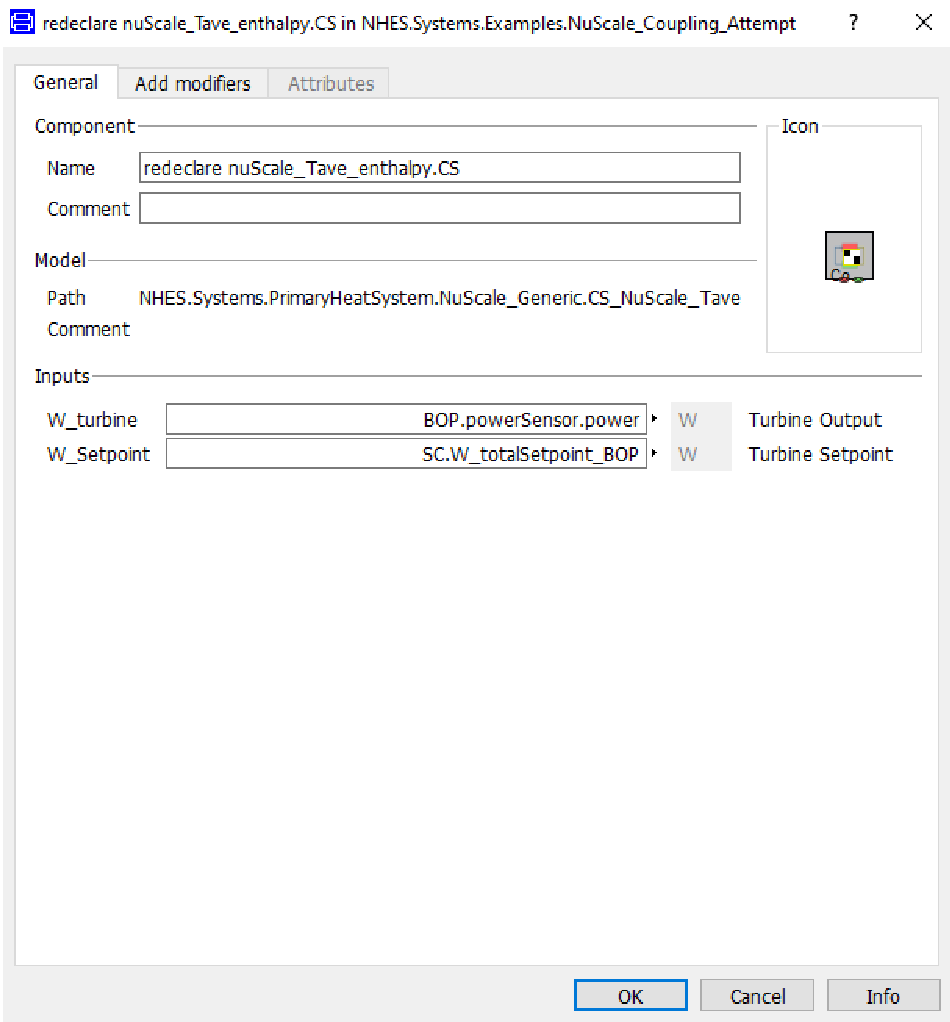


Figure 9: Primary System Controller Settings

17

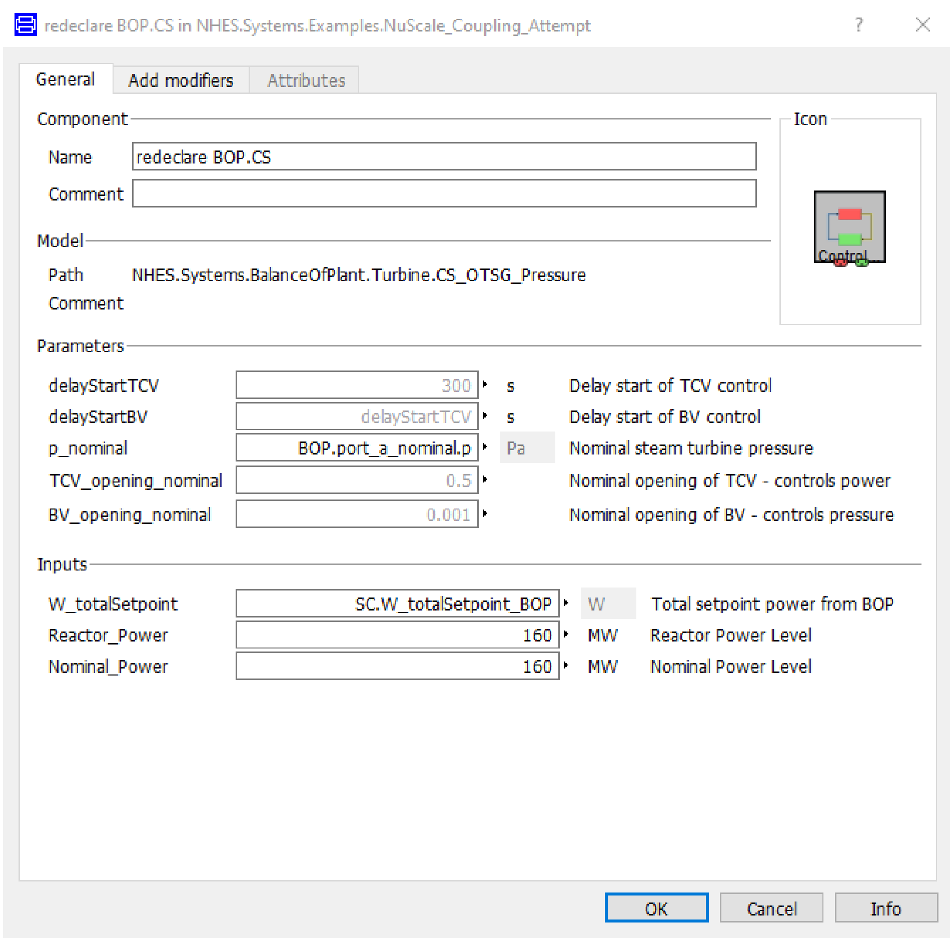


Figure 10: Turbine Control Settings

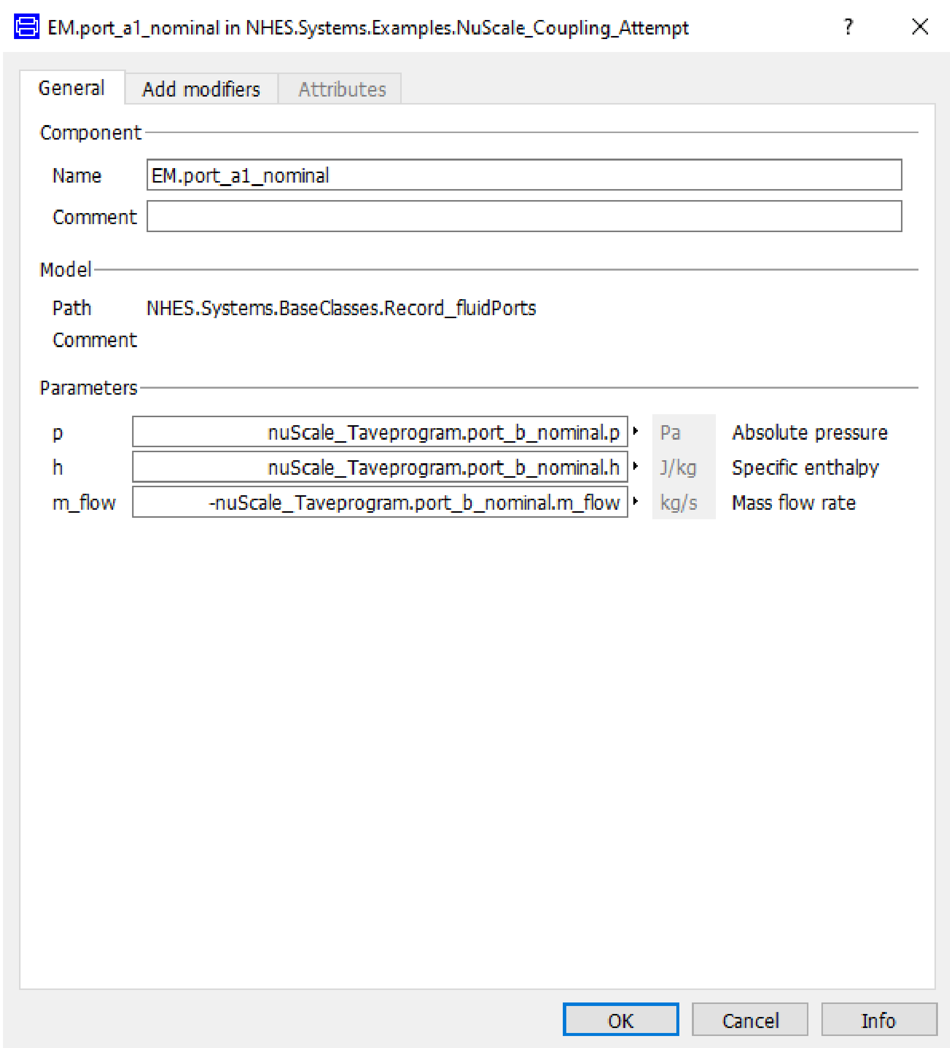


Figure 11: Port a Boundary Values of the Energy Manifold

19

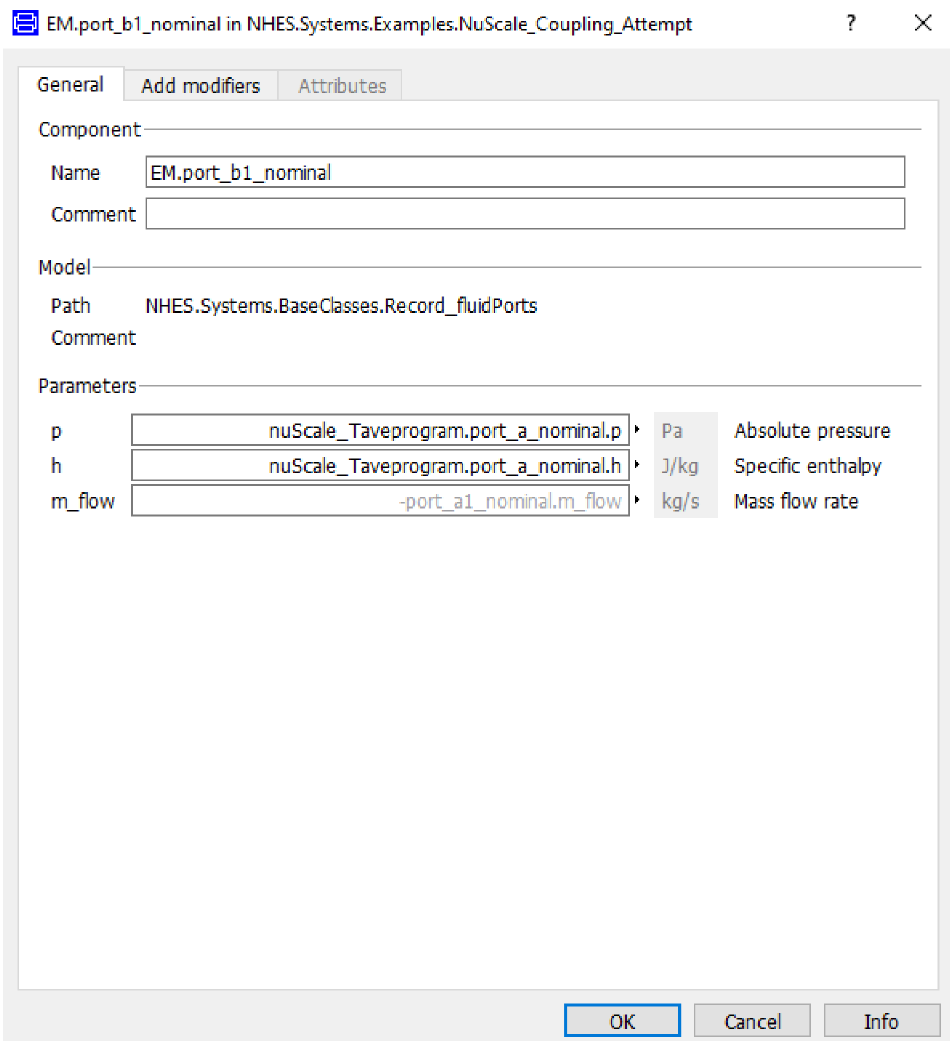


Figure 12: Port b Nominal Values of the Energy Manifold

20

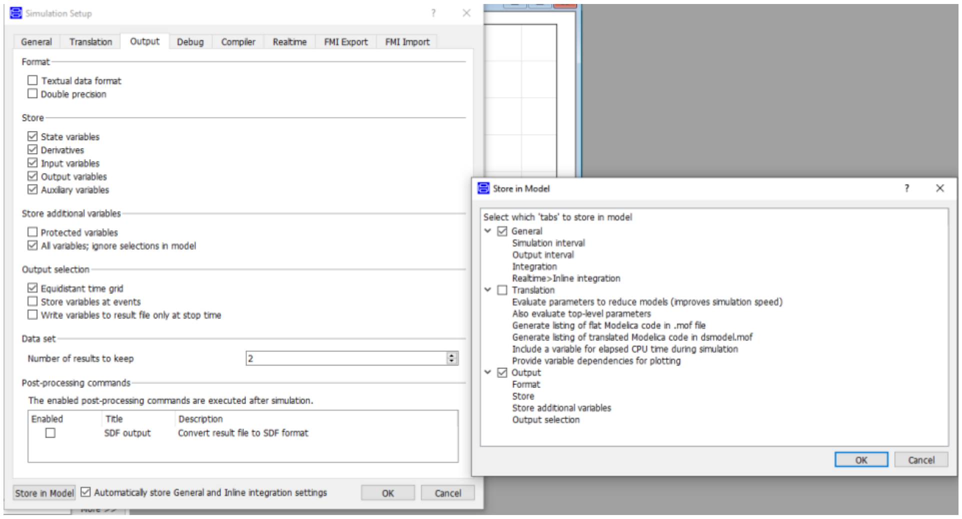


Figure 13: Settings to Create a proper mat file for a gold folder test

The .mat file in the gold folder will need to be run using the same simulateModel command that is present in the .mos file being created.

These can be selected in the Simulation Setup tab of the Dymola GUI, Figure 14, and should carry down to the command you copy and paste in the .mos file. An example is shown below of the simulation setup tab.

Then run the simulation, (ideally a test should take less than 100 seconds). On the simulation tab in the command line copy the simulation command. Example below:

```
simulateModel("NHES.Systems.EnergyManifold.SteamManifold.
Examples.SteamManifold_Test", stopTime=100, numberOfIntervals=100,
method="Esdirk45a", resultFile="SteamManifold_Test");
```

This command should then be added to a file and named something like Test_Example.mos. The command can be found in the Simulation Setup tab of the Dymola GUI once you hit simulate

Then in folder /path/to/hybrid/hybrid/tests/dymola_tests create a folder named Test_YourModel.

Create a *gold* folder in the new folder, drop the .mat file from your simulation that is named resultFile="SteamManifold_Test" from your simulateModel command into the gold folder. The .mat file is created in your working directory in Dymola. Then in the main Test_YourModel folder drop the Test_Example.mos file and create a tests file open it up and place the following in it:

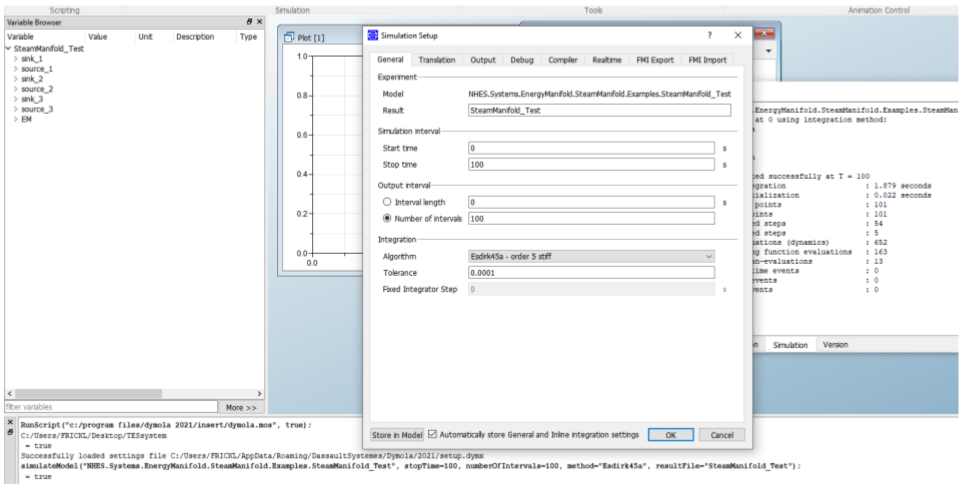


Figure 14: Simulation Setup

```
[Tests]
  [./]
   type = 'HYBRIDTester'
   input = 'Test_Example.mos'
   workingDir = '.'
   output = 'SteamManifold_Test.mat'
   dymola_mats = 'SteamManifold_Test.mat'
   rel_err = 0.001
  [../]
[]
```

where SteamManifold_Test.mat should be your result .mat file name, rel_error is the amount of error allowed between the gold file and the regression test output, and Test_Example.mos is the run script created.

3.4 Advanced Test File Options utilized for complex models

For complex models the initialization phase of a simulation can take the Modelica solvers a significant amount of time to find an initialization point. This occurs due to the highly nonlinear nature of the underlying physical equations. A way to avoid such situations is to provide a restart file to bypass the initialization phase of the simulation. A restart file is automatically created at the end of each simulation as the dsfin.txt file created in the folder where the simulation is run.

This file includes the final values of the previous simulation from which the new model can restart. Move this file to the gold folder for your new testing system. Once this file is created it can then be loaded automatically via the continue button in Modelica under the Simulation Tab. Select $Continue \rightarrow ImportInitial \rightarrow dsfin.txt$. See Figure 15.

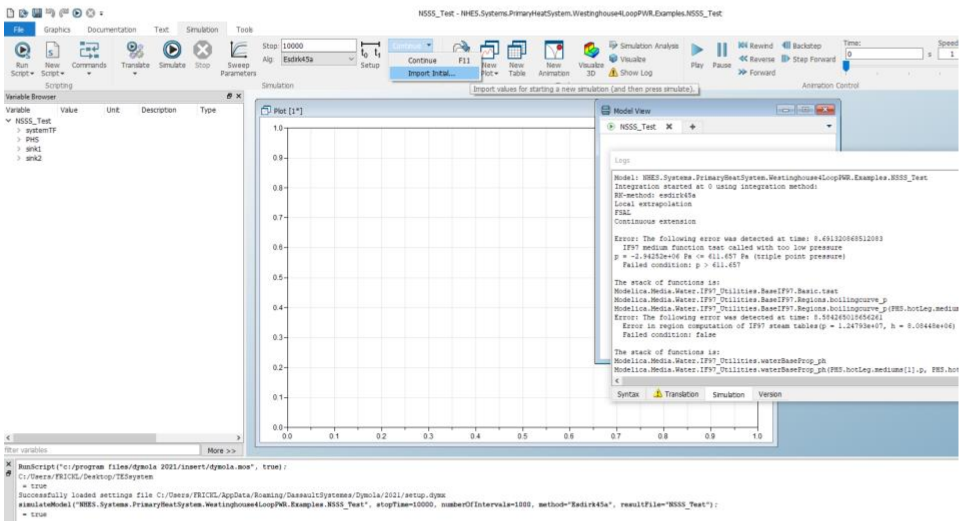


Figure 15: Import Initial conditions from a previously run simulation

Then once the dsfin.txt file is loaded go into the Setup tab and move the time back to start from zero and the end time to the desired simulation point for the test, shown in Figure 16. This is necessary since Dymola assumes the user wants to restart the simulation from where it ended in time as well. This is not the case for the test. Instead the goal is to skip the initialization phase of the simulation and provide a clean solution with which to compare.

Simulate this model and save the result file, in this example "NSSS_Test.mat" and place it into the gold folder of the testing system. Additionally, copy and paste the simulateModel command that is in the Dymola GUI as the last line of your .mos script file for the test. The first two lines should be translateModel to make sure the right model is loaded into the equation set, followed by the importInitial command that loads all the values into the translated Model. The final command should be the simulateModel command. The .mos file should look something like what is shown below.

```
translateModel("NHES.Systems.PrimaryHeatSystem.Westinghouse4LoopPWR
.Examples.NSSS_Test");

importInitial("./gold/dsfinal.txt");
```

23

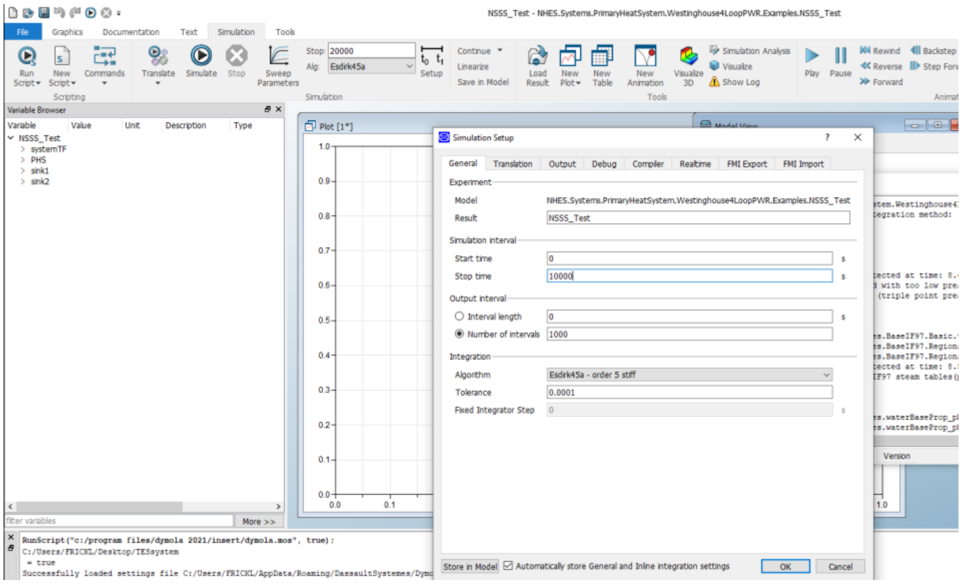


Figure 16: Realign the Simulation Time

```
simulateModel("NHES.Systems.PrimaryHeatSystem.Westinghouse4LoopPWR.
Examples.NSSS_Test", stopTime=10000, numberOfIntervals=250,
method="Esdirk45a", resultFile="NSSS_Test");
```

4 Model Description

It is the intent of this document to provide a level of understanding of each of the process models sufficient to allow users, with some background of Modelica, the ability to integrate, modify top level parameters, and run simulations of Integrated Energy Systems. Advanced users will be able to use the models as they see fit, but the descriptions provided here will not necessarily explain all facets of the models in detail.

4.1 Primary Heat System

In the Hybrid repository there are four potential primary heat sources: The Four-Loop PWR plant, the Generic Modular PWR, a natural circulation SMR power plant, and the Natural Gas Fired Turbine. Generally, we consider the Natural Gas Fired Turbine as a peaker unit and thus will save its' discussion and coverage for the secondary power source section of the Model Descriptions.

4.1.1 Four Loop Pressurized Water Reactor

The Four loop PWR system, Figure 17, is designed to be consistent with publicly available information for the Westinghouse plant design [7]. This is a Pressurized Water reactor with a nominal thermal power of 3400MWt and has control systems designed to output 1100MWe. All system parameters can be found in the SubSystem model under the "data" record. The steam generator is of U-tube design and operates at a nominal pressure 1000psia. Reactivity feedback can be found in the coreSubchannel module alongside an external source of activity that is designed to provide reactivity feedback from the control rods. Reactivity in the core is based on a point kinetics models, that includes feedback from fission products, boron, fuel temperature, and moderator temperature. System decay heat is calculated from the TRANSFORM package via an eleven-group decay heat correlation from the TRACE user manual.

4.1.2 Generic Modular PWR

The generic modular PWR unit, Figure 18, is sized to be 160 MWt with 50MWe output as is consistent with the NuScale power module. However, the generic modular PWR does not operate under natural circulation but instead operates under forced flow. Therefore, this unit provides more stability in the code since it does not rely on density differentials to drive flow. This makes the unit less useful than is the NuScale style reactor modeled below, but it does provide the user a power input consistent with NuScale style systems but without the need to tune system geometries, friction factors, etc.. to meet the proper flow dynamics. As with the Westinghouse plant the data file is included in the subsystem model and has reactivity controls within the core submodule. The

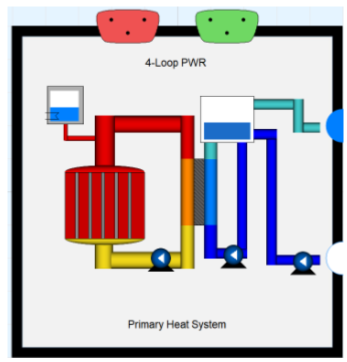


Figure 17: Top View of the Four-Loop PWR Plant

Generic Modular PWR relies heavily on the TRANSFORM library for its subcomponents. The steam generator is a once through design with geometrical orientation consistent with a helical coil steam generator.

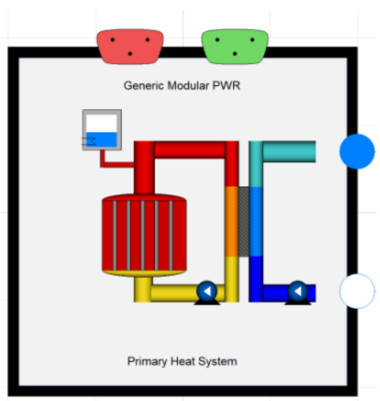


Figure 18: Top Level Depiction of the Generic Modular PWR in the NHES package.

4.1.3 Natural Circulation Small Modular Reactor

The natural circulation SMR power module,Figure 19, is an integral pressurized water reactor (IPWR) that operates with a nominal thermal power of 160 MWt capable of producing 50 MWe to the electric grid. Integral designs are fully self-contained, eliminating the need for large main steam lines that can potentially lead to large break loss of coolant accidents (LOCA). Instead the primary system has only an inlet of feed water into the bottom of the helical coil steam generator and an exit point for steam at the top of the steam generator. All sizes for components is held within the data record in the sub-system. These sizes are consistent with NRC design documentation that can be publicly viewed on the NuScale NRC design certification page.

The primary system does not include any pumps but instead operates under natural circulation. Natural circulation reactors rely on the height and density differentials between hot and cold water to drive circulation of water through the core. Through elimination of primary coolant pumps an entire class of accident scenarios is eliminated. Modeling efforts in this report focused on three main efforts: matching thermal and electric output, matching system geometry, and matching natural circulation efforts in the system via flow rates and temperature differentials. The primary side of the module has heights and cross-sectional areas in accordance with NRC design certification material. The primary and secondary sides were modeled in their entirety. The helical coil steam generator was modeled as a once through steam generator where the secondary side is on the inside of the tubes and the primary side fluid run along the outside of the tubes. The full report on this module is available on OSTI [4].

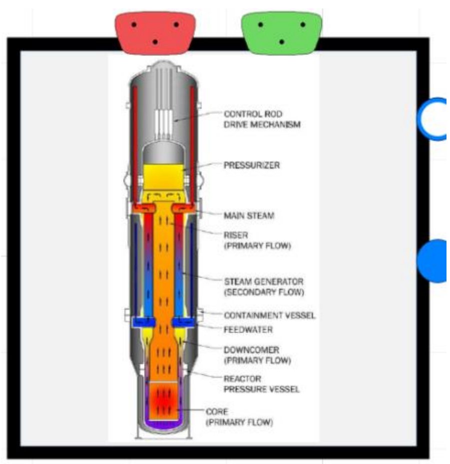


Figure 19: Top Level Depiction of the SMR System in the NHES package.

4.2 Energy Manifold

The energy manifolds intention is be a diversion module to as many different subunits as needed for fluid diversion. It consists of a series of pipes that can be extended to "n" submodules, see Figure 20. The unit has the capability of utilizing control schemes, however in many practical applications the control schemes are encapsulated within the subprocesses as opposed to within the energy manifold. There are currently four potential energy manifolds that can be used. For practical purposes only the model SteamManifold_L1_boundaries is used in integrated energy systems as it does not include control valves and supports "n" submodules going in and out. The other versions of the energy manifold exist for advanced users in the event the balance of plant or subprocess they are connecting to does not include sufficient valving and control to properly constrain the system. For example, simplified balance of plant systems that do not contain return flow would require the energy manifold to provide makeup water from the condenser, therefore for that scenario we would need to use the SteamManifold_L1_FWH_Cond model which contains a condenser and feedwater heater.

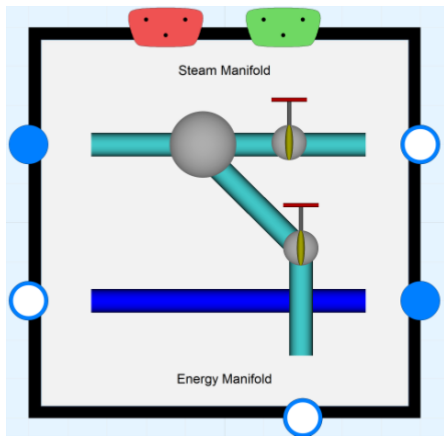


Figure 20: : Top Level Depiction of the Energy Manifold in the NHES package

4.3 Industrial Process

Integrated Energy Systems often include thermal and electrical energy users aside the electric grid. To accommodate thermal energy users and byproduct production the HYBRID repository includes a hydrogen production unit and a reverse osmosis unit.

4.3.1 Hydrogen Production

The hybrid repository includes hydrogen production via high temperature steam electrolysis (HTSE) as shown in Figure 21. HTSE utilizes a combination of thermal energy and electricity to split water into hydrogen (H2) and oxygen (O2) in Solid Oxide Electrolyzer Cells (SOECs), which can be seen in simple terms as the reverse operation of solid oxide fuel cells (SOFCs). The cathode-supported cell consists of a three-layer solid structure (composed of porous cathode, electrolyte, and porous anode) and an interconnect (separator) plate [8]. An oxygen-ion conducting electrolyte (e.g., yttria-stabilized zirconia [YSZ] or scandia-stabilized zirconia [ScSZ]) is generally used in SOECs [9]. For electrically conducting electrodes, a nickel cermet cathode, and a perovskite anode such as strontium-doped lanthanum manganite (LSM) are typically used. The interconnect plate separates the process gas streams; it must also be electrically conducting and is usually metallic, such as a ferritic stainless steel.

For the HTSE models there are four main models developed by INL, each relying on the same underlying physics of the system but with different control schemes. The HTSE units within the Modelica framework have been specifically designed for integration with light water reactor systems and have been sized with the necessary components to allow for steam side preheating under this assumption. It should be noted that in other HTSE designs there may be varying degrees of preheating equipment based on inlet conditions from the external process. For the HTSE process system parameters are finely tuned and highly non-linear when compared with other process models. Changes in heat exchanger design and sizing can be made directly within the subsystem model however due to the nonlinearity of the system convergence following any changes, a singular component cannot be guaranteed. To modify HTSE stack characteristics the user will need to go two levels down into the HTSE stack system the HTSE stack can be clicked on to open a parameter table where stack characteristics can be modified. Due to the high level of complexity required with HTSE stacks and the customization required depending on the inlet conditions of external system usage of the existing HTSE is preferred, with more details available in two reports published. [2], [10], [11]. Base classes for the HTSE system can found in the "Electrolysis" package within the NHES framework and can be utilized if one desires to create their own HTSE unit.

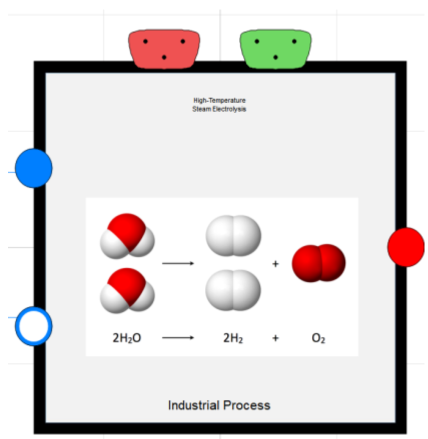


Figure 21: Top Level Depiction of the High Temperature Steam Electrolysis Unit in the NHES package

4.3.2 Desalination

The NHES repository includes a desalination industrial process based on reverse osmosis (RO), Figure 22, designed for brackish water desalination. RO desalination utilizes a semi-permeable membrane, which allows water to pass through but not salts, thus separating the fresh water from the saline feed water. A typical Brackish Water RO (BWRO) plant consists of four main components: feed water pretreatment, High-Pressure (HP) pumping, membrane separation, and permeate (fresh water) post-treatment. The concentrate water rejected by the first membrane module plays a role as the feed water for the second membrane module by the successive order, and so on. These pressure vessels are arranged in rows in each membrane stage, with two-stage membrane separation being typical in BWRO. Each stage has a recovery of 50–60 percent, achieving overall system recovery of 70–85 percent [12].

The Reverse Osmosis Subsystem unit provides the user the ability to modify the number of parallel reverse osmosis units being utilized within the plant alongside to specify how much power is being input into the RO system. Each one of these parallel systems is assumed to go through a two-step the desalination process. In addition, the unit provides the user the ability to alter the salinity of the brine coming into the plant alongside a specified pressure differential across the plant. If further alterations and control are desired from a user perspective reports detailing the full specifications of the plant designs are available in [3], [12]. Additionally, base components for the entire desalination plant can be found in the "Desalination" package within the NHES repository.

Plant.png

Figure 22: Top Level Depiction of the Brackish Water Desalination Process in the NHES package

4.4 Balance of Plant

There are two main balance of plant models in the hybrid repository. The standard ideal turbine (with and without a condenser and feedwater heater) and step down turbines that allow turbine tap offtake.

4.4.1 Simple Balance of Plant

The balance of plant system consists of an ideal steam turbine model, a condenser, feedwater system for reheat, and a couple of valves that allow steam flow either to the turbine or as bypass to the condenser, Figure 23. Additionally, piping exists to send condensate and rejected heat from ancillary processes directly to the condenser. The balance of plant model can handle supervisory control input for direct control of the turbine control valve and turbine bypass valve based on different sensor input. The main balance of plant system is designed to model Rankine systems.

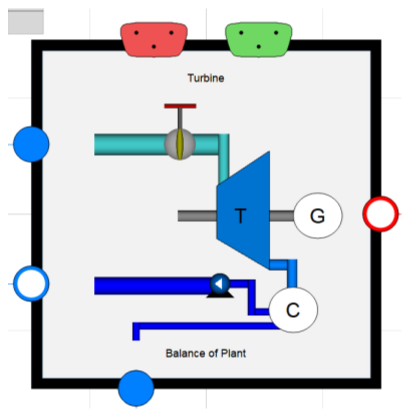


Figure 23: Top Level Depiction of the Balance of Plant in the NHES package

4.4.2 Step Down Turbines

The step-down turbines consist of a series of an ideal steam turbines connected via a singular rotational inertia shaft with bypass tap lines coming off the turbines, Figure 24. The purpose of this model is to allow turbine tap offtake in a dynamic system. The data record within the model includes a series of inputs that allows the user to specify the turbine tap offtake pressures. Additionally, each individual offtake fraction can be input from the data record. The outlet of the stepdown turbines does not include a condenser; therefore, a condenser model would need to be included in a separate system model if the fluid is to be re-introduced into an overall system model.

4.5 Energy Storage

Energy Storage is a large component of Integrated Energy Systems. Currently there are two models of Energy Storage in the repository. Electric Battery Storage, characterized largely as Li-ion

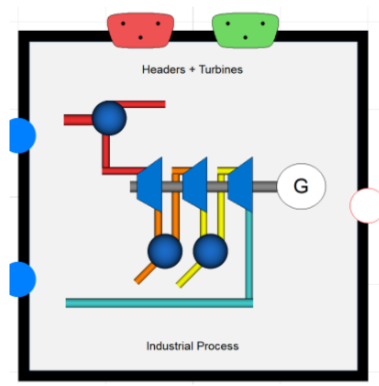


Figure 24: Top Level Depiction of the StepDown Turbines in the NHES package

battery technology, and two-tank sensible heat thermal energy storage that uses Therminol-66 as the working fluid.

4.5.1 Electric Battery Storage

Electric Battery Storage, shown in Figure 25, is largely characterized as fast and expensive. Due to the speed with which battery storage systems operate, on the order milliseconds, the battery within the hybrid repository has been modeled as a simple logical battery system. The battery can both charge and discharge based upon the direction of electricity flow through the port. It is assumed to be a "perfect" battery and due to the speed of the system, subcomponents have not been modeled simply because they would operate faster than would be useful for the types of analysis utilized with the system. The battery has user-based inputs that control how fast or slow the system can charge and discharge as well as how much energy can be stored within the battery before it is considered full.

4.5.2 Two-Tank Thermal Energy Storage

Sensible heat storage involves the heating of a solid or liquid without phase change and can be deconstructed into two operating modes: charging and discharging. A two-tank TES system, shown in Figure 26, is a common configuration for liquid sensible heat systems. In the charging mode cold fluid is pumped from a cold tank through an Intermediate Heat Exchanger (IHX), heated, and stored in a hot tank while the opposite occurs in the discharge mode. Such systems have been successfully demonstrated in the solar energy field as a load management strategy. The configuration of the TES system held within the repository involves an outer loop interfaces with the

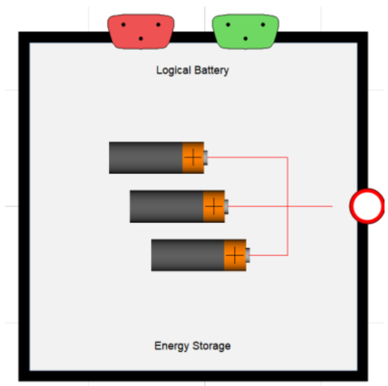


Figure 25: Top Level Depiction of the Logical Battery in the NHES package

energy manifold. Bypass steam is directed through an IHX and ultimately discharged to the main condenser of an Integrated system. An inner loop containing a TES fluid consists of two large storage tanks along with several pumps to transport the TES fluid between the tanks, the IHX and a steam generator. Flow Bypass Valves (FBVs) are included in the discharge lines of both the “hot” and “cold” tanks to prevent deadheading the pumps when the Flow Control Valves (FCVs) are closed. Therminol-66 is chosen as the TES fluid as it is readily available, can be pumped at low temperatures, and offers thermal stability over the range (-3°C–343°C) which covers the anticipated operating range of the light water reactor systems (203°C–260°C). Molten salts (e.g. 48 percent NaNO3 – 52 percent KNO3) were not considered, as the anticipated operating temperatures fall below their 222°C freezing temperature. The TES system is designed to allow the power plant to run continuously at 100 percent power over a wide range of operating conditions. During periods of excess capacity, bypass steam is directed to the TES unit through the auxiliary bypass valves where it condenses on the shell side of the IHX. TES fluid is pumped from the cold tank to the hot tank through the tube side of the IHX at a rate sufficient to raise the temperature of the TES fluid to some set point. The TES fluid is then stored in the hot tank at constant temperature. Condensate is collected in a hot well below the IHX and drains back to the main condenser or can be used for some other low pressure application such as chilled water production, desalination or feed-water heating. The system is discharged during periods of peak demand, or when process steam is desired, by pumping the TES fluid from the hot tank through a boiler (steam generator) to the cold tank. This process steam can then be reintroduced into the power conversion cycle for electricity production or directed to some other application through the PCV. A nitrogen cover gas dictates the tank pressures during charging and discharging operation. Full details of the model and its use within integrated energy systems can be found in report [3], [13].

The model itself is coded in a non-conventional manner compared to the rest of the modelica models. It is coded in an input, output sense rather than in a fluid-port, electric-port based modeling system. This is because the model was transferred over from a FORTRAN style code rather than initially coded in Modelica. To modify the two-tank thermal storage system the user will need to look at each individual model within the charging mode and the discharge mode. Base components within the models are fully commented within the code. Like the HTSE the thermal storage unit is finely tuned and thus use outside of its current state will take a bit of work. To help with this the thermal storage unit has been sized to be compatible for varying sizes of offtake from a power unit. One is sized to take 20 percent of nominal steam from a standard 3400MWt Westinghouse plant, and one is designed for 5 percent offtake. Both designed to provide energy back as a peaking unit. The peaking unit is held within the discharge side of the model and is assumed to have its own turbine or is sent back to the low-pressure turbine. Explicit modeling of the coupling back with the low-pressure turbine has not been done. Future updating of the two-tank thermal storage unit to be consistent with other models is planned.

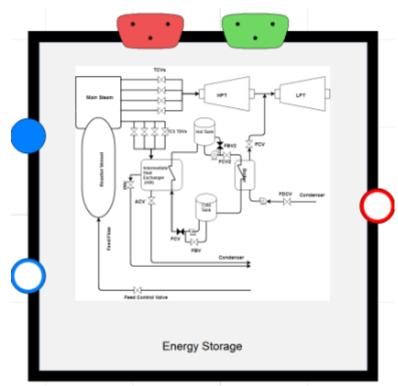


Figure 26: Top Level Depiction of the Two-Tank Sensible Heat Storage Unit in the NHES package

4.5.3 Thermocline Packed Bed Thermal Energy Storage

A thermocline storage system, shown in Figure 27, stores heat via hot and cold fluid separated by a thin thermocline region that arises due to density differential between the fluid. Assuming low mixing via internal flow characteristics and structural design, this thermocline region can be kept relatively small in comparison with the size of the tank. Additionally, large buoyancy changes and low internal thermal conductivity are also extremely useful in maintaining small relative thermocline thickness.

To increase the cost-effective nature of these designs, it is common to fill the tank with a low-cost filler material, such as concrete or quartzite. These filler materials are cheap, have high density, and high thermal conductivity. By using such material, a reduction in the amount of high cost thermal fluid can be achieved, thereby increasing the economic competitiveness of such designs.

The thermocline system was modeled from a modified set of Schumann equations that were originally introduced in 1927 [14]. The equation set governs energy conservation of fluid flow through porous media. His equation set has been widely adopted in the analysis of thermocline storage tanks. The modified equations adopted a new version of the convective heat-transfer coefficient to incorporate low and no-flow conditions from Gunn in 1978 [15]. Additionally, a conductive heat-transfer term was added for the heat conduction through the walls of the tank. Self-degradation of the thermocline in the axial direction is neglected due to low relative values when during standard operation, this is a known limit of the model during times of no flow.

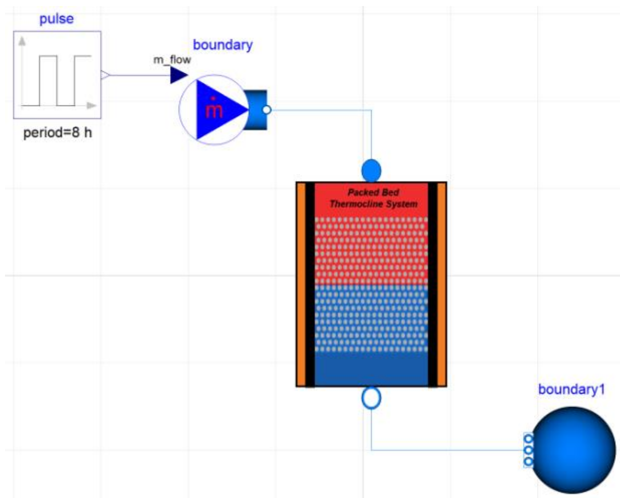


Figure 27: Top Level Depiction of the Single Tank Packed Bed Heat Storage Unit in the NHES package

4.6 Secondary Energy Source

Secondary Energy Sources, or commonly known as peaking units, are an essential part of the energy grid. These systems provide ”on-demand” energy during moments when the electrical demand is larger than what the rest of the grid can accomodate. A common feature of

4.6.1 Natural Gas Fired Turbine

Recently, natural gas-fired turbines have found widespread use because of their higher efficiencies, lower capital costs, shorter installation times, abundance of natural gas supplies, lower greenhouse

35

gas emissions compared to other energy sources; and fast start-up capability, which enables them to be used as peaking units that respond to peak demands [16], [17]. Due to their special characteristics, natural gas fired turbines are installed in numerous places around the world and have become an important source for power generation. This section is dedicated to detailed process and control designs of the GTPP, whose primary role is to cover rapid dynamics in grid demand that cannot be met by the remainder of the N-R HES. Simulation results involving several case studies are also provided. Full system details are available in OSTI [2].

The natural gas turbine, Figure 28, is designed with parameters embedded in each individual component. The top level variables can be edited directly within the GTTP_PowerCtrl system. This component is where things such as pressure ratios, flow rates, mechanical efficiencies, and shaft inertia can be modified. The natural gas turbine is designed with a nominal electrical power generation capacity of 35MWe but has a specially designed capacityScaler variable that allows the user to scale the system to between 17 and 70MWe for a singular load. If more are desired then the deployment of several natural gas fired turbines would be required.

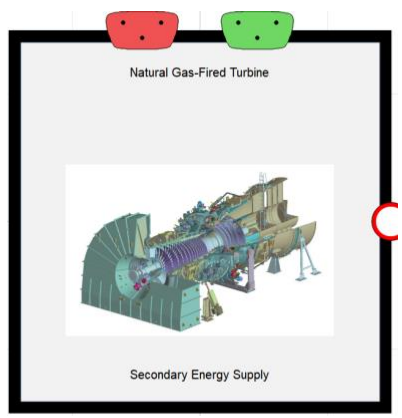


Figure 28: Top Level Depiction of the Natural Gas Fired Turbine in the NHES package

4.6.2 Hydrogen Turbine

With the increase in hydrogen production technologies comes on the other end hydrogen burning technologies. To accommodate a hydrogen burning technology the HYBRID repository has been outfitted with a retrofit natural gas burner that is capable of handling pure hydrogen.

The hydrogen turbine, Figure 29, is designed with parameters embedded in each individual component. The top level variables can be edited directly within the Hydrogen_PowerCtrl system. This component is where things such as pressure ratios, flow rates, mechanical efficiencies, and shaft inertia can be modified. The hydrogen turbine is designed with a nominal electrical power generation capacity of 35MWe but has a specially designed capacityScaler variable that allows the user to scale the system to between 17 and 70MWe for a singular load. If more are desired then the deployment of several hydrogen turbines would be required.

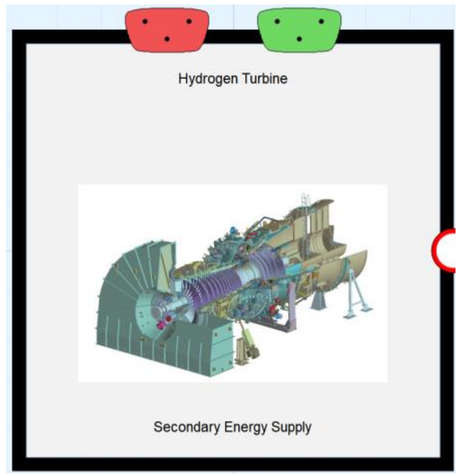


Figure 29: Top Level Depiction of the Hydrogen Turbine in the NHES package

Document Version Information

09a34b32ae7a18d79261047cb9da2e7e7d35523a alfoa Mon, 15 Mar 2021 14:09:50 -0600

References

[1] C. Rabiti, A. Epiney, P. Talbot, J. Kim, S. Bragg-Sitton, A. Yigitoglu, S. Greenwood, S. Cetiner, F. Ganda, and G. Maraonati, "Status report on modeling and simulation capabilities for nuclear-renewable hybrid energy systems," Tech. Rep. INL/EXT-17-43441, Idaho National Laboratory, 2017.

[2] J. Kim, M. Mckellar, S. Bragg-Sitton, and R. Boardman, "Status report on the component models developed in the modelica framework: High-temperature steam electrolysis and gas turbine power plant," Tech. Rep. INL/EXT-16-40305, Idaho National Laboratory, October 2016.

[3] J. Kim and K. Frick, "Status report on the component models developed in the modelica framework: Reverse osmosis desalination plant and thermal energy storage.," Tech. Rep. INL/EXT-18-45505, Idaho National Laboratory, May 2018.

[4] K. Frick, "Status report on the nuscale module development in the modelica framework," Tech. Rep. INL/EXT-19-55520, Idaho National Laboratory, August 2019.

[5] "Modelica standard library." https://github.com/modelica/Modelica.

[6] M. Greenwood, "Transform - transient simulation framework of reconfigurable models," no. doi.10.11578/dc.20171109.1, 2017.

[7] "Westinghouse pwr manual," 1984.

[8] J. Udagawa, P. Aguiar, and N. Brandon, "Improved goodness-of-fit tests," *Journal of Power Sources*, vol. 166, no. 1, pp. 127–136, 2007.

[9] J. O'Brien, "Thermodynamic considerations for thermal water splitting processes and high-temperature electrolysis," *IMECE2008*, 2008.

[10] J. Kim, S. Bragg-Sitton, and R. Boardman, "Status report on the high-temperature steam electrolysis plant model developed in the modelica framework (fy17)," Tech. Rep. doi:10.2172/1408745, Idaho National Laboratory, 2017.

[11] J. Suk Kim, R. Boardman, and S. Bragg-Sitton, "Dynamic performance analysis of a high temperature steam electrolysis plant integrated within nuclear-renewable hybrid energy systems," *Applied Energy*, vol. 228, pp. 2090–2110, 2018.

[12] J. Suk Kim, J. Chen, and H. Garcia, "Modeling, control, and dynamic performance analysis of a reverse osmosis desalination plant integrated within hybrid energy systems," *Energy*, vol. 112, pp. 52–66, 2016.

[13] K. Frick, J. Doster, and S. Bragg-Sitton, "Design and operation of a sensible heat peaking unit for small modular reactors," *Nuclear Technology*, vol. 205, pp. 415–441, 2018.

[14] J. Lew, P. Li, C. Chan, W. Karaki, and J. Stephens, "Analysis of heat storage and delivery of a thermocline tank having solid filler," *Journal of Solar Engineering*, vol. 133, 2011.

[15] T. Esence and et al, "Analysis of heat storage and delivery of a thermocline tank having solid filler," *Solar Energy*, vol. 153, pp. 628–654, 2017.

[16] S. Yee, J. Milanovic, and F. Hughes, "Overview and comparative analysis of gas turbine models for system stability studies," *IEEE Transactions of Power Systems*, vol. 23, pp. 108–118, 2008.

[17] J. Kim, K. Powell, and T. Edgar, "Nonlinear model predictive control for a heavy-duty gas turbine power plant," *American Control Conference*, vol. 23, pp. 2952–2957, 2013.

v1.28

APPENDIX B – SQA: SOFTWARE QUALITY ASSURANCE PLAN (SQAP)

Laboratory-wide	Template			

Document ID: PLN-6274
Revision ID: 0
Effective Date: 10/01/2020

Software Quality Assurance Plan (SQAP)

HYBRID Software Quality Assurance and Maintenance and Operations Plan

Andrea Alfonsi

Idaho National Laboratory

The INL is a U.S. Department of Energy National Laboratory operated by Battelle Energy Alliance.

$\boxed{\text{HYBRID}}$ Software Quality Assurance and Maintenance and Operations Plan

PLN-6274

Prepared by:

Andrea Alfonsi	09/01/2020
IT Project/M&O Manager	Date

Reviewed by:

Konor Frick	09/01/2020
Independent Reviewer	Date

Approved by:

Cristian Rabiti	09/01/2020
Asset Owner	Date

CONTENTS

1. PURPOSE

Software quality assurance (SQA) is a set of activities necessary to provide adequate confidence that a software item or product conforms to the set of functional and technical requirements specified for that item. This plan presents the required activities to enable consistent SQA implementation within the HYBRID Software. It provides a standardized method of capturing software requirements, how those requirements will be implemented, how the software will be tested, how changes to the software will be controlled, and how software deficiencies will be handled. This Software Quality Assurance Plan (SQAP) establishes the software Quality Assurance program for HYBRID. It covers the periods of software development, maintenance and operations (M&O), and retirement. It implements applicable requirements in conformance with PDD-13610, “Software Quality Assurance”. This plan is based on the RAVEN SQA process, documented in “PLN-5552, *RAVEN and RAVEN Plug- ins Software Quality Assurance and Maintenance and Operations Plan*”. The HYBRID software process follows the PLN-5552 and in this document, the deviations from such plan are documented.

1.1 HYBRID Description

One of the goals of the HYBRID software/product is to assess the economic viability of hybrid systems in a market that contains renewable energy sources (e.g. wind, solar, etc.). The hybrid system would be a nuclear reactor that not only generates electricity, but also provides heat to another plant that produces by-products, like hydrogen or desalinated water. The idea is that the possibility of selling heat to a heat user absorbs (at least part of) the volatility introduced by the renewable energy sources.

The HYBRID software/product is a container of systems/components models and analysis workflows for the deployment of a “plug and play” framework aimed to integrate *Modelica/Dymola* [see def.] with *RAVEN* in terms of both *FMI/FMU* [see def.] construction and repository structure that aims to ease the sharing and simulation of complex dynamic models.

HYBRID is operational within multiple projects. Ongoing support of HYBRID is required for the purpose of adding functionality, correcting model errors and improving the performance of the HYBRID models and analysis flows.

- HYBRID is maintained by a team of scientists/researchers, referred to herein as the HYBRID core team (see def.). HYBRID maintenance and operations, performed by the HYBRID core team, is an ongoing activity.

- This plan covers the maintenance of all existing and future components of HYBRID. This includes, but is not limited to, servers, server software, user workstations, HYBRID software, and control documents. Changes to this document will be completed through the Electronic Change Request (eCR) process.

1.2 Software Lifecycle

HYBRID is using an Agile life cycle methodology. The life cycle will be performed in an iterative manner and address the requirements, design, implementation, testing, installation and checkout, operations and maintenance, and retirement phases.

1.3 Assumption and Constraints

- The HYBRID core team will adhere to LWP-1303, "Management of Unclassified Cyber Security Information Systems" and LWP-1401, "Preparing and Releasing Scientific and Technical Information Products," where applicable.
- 29 USC 794d, Section 508 of the Workforce Investment Act of 1998 considerations will be made for the ability of disabled individuals to access the information or service provided by the software.
- INL will manage the software with support from vendors (for *acquired software* [see def.]) until the software is retired.
- Software vendor support agreements are maintained.
- For firmware, changes to acquired software including software updates and security patches will be implemented by the product vendor.
- The hardware that serves HYBRID is managed by the High-Performance Computing Group. The hardware is considered a configuration item (see def.) for the HYBRID asset, and changes impacting the HYBRID software must be reviewed by the HYBRID technical lead or designee; however, the management of the hardware is outside the scope of this plan.
-

1.4 Deviation Policy

All deviations from this plan require management approval. Whether planned or unplanned, if any deviation from this plan is necessary, the following components will be determined:

- Identification of task affected.

- Reasons for deviation defined.

- Effects on the quality of the project.

- Time and resource constraints affected.

A deviation report will be generated, and authorization will be required. Deviations that violate requirements must be documented within the relevant issue.

2. REFERENCES

The following source documents apply to this SQAP:

- 29 USC 794d, Section 508 Workforce Investment Act of 1998
- INL/EXT-18-44465, "RAVEN User Documentation"
- ISO/IEC/IEEE 24765:2010(E), "Systems and software engineering — Vocabulary"
- PDD-13610, "Software Quality Assurance Program."
- PDD-13000, "Quality Assurance Program Description"
- LWP-1201, "Document Management"
- LWP-1202, "Records Management"
- LWP-1305, "Acquisition of Computer Hardware/Software Resources"
- LWP-1306, "Management of IT Asset Minimum Security Configurations," Rev. 1, December 23, 2013.
- LWP-1401, "Preparing and Releasing Scientific & Technical Information Products"
- LWP-4001, "Material Acquisitions"
- LWP-4002, "Service Acquisitions"
- PLN-5552, "RAVEN and RAVEN Plug- ins Software Quality Assurance and Maintenance and Operations Plan"
- PLN-4653, "INL Records Management Plan"
- SDD-561, "HYBRID Software Design Description (SDD)"
- SPC-2990, "HYBRID Software Requirements Specification (SRS) and Traceability Matrix"

3. DEFINITIONS AND ACRONYMS

This section defines, or provides the definition of, all terms and acronyms required to properly understand this plan.

3.1 Definitions

Acquired software. Software generally supplied through basic procurements, two-party agreements, or other contractual arrangements. Acquired software includes commercial off-the-shelf software, support software such as operating systems, database management systems, compilers, software development tools, and commercial calculational software and spreadsheet tools (e.g. Microsoft's Excel). Downloadable software that is available at no cost to the user (referred to as freeware) is also considered acquired software. Firmware is acquired software. Firmware is usually provided by a hardware supplier through the procurement process and cannot be modified after receipt.

Agile development. Agile development is an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end user(s). It prescribes adaptive planning, continuous development, early delivery, and continual improvement, and it encourages rapid and flexible response to change.

Anomaly. Anything observed in the documentation or operation of software that deviates from expectations based on previously verified software products or reference documents.

Baseline. A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for use and further development, and that can be changed only by using an approved change control process. [ASME NQA-1-2008 with the NQA-1a-2009 addenda]

Change control. An element of configuration management, consisting of the evaluation, coordination, approval or disapproval, and implementation of changes to configuration items (CIs see def.) after formal establishment of their configuration identification. [ISO/IEC/IEEE 24765:2010(E)]

Change control board (CCB). The group by which a change is proposed, evaluated, approved or rejected, scheduled, and tracked. This board is also responsible for evaluating and approving or disapproving proposed changes to configuration items (CIs) and implementation of approved changes when required.

Change requests (CRs). CRs can be initiated by anyone, including off site users, and can be used for maintenance (fine-tuning and problem resolving), new development, and enhancements, or can be used to report program errors and problems.

Change request log. A log that provides a listing of all the change requests and the change request status used for application software, system software, and hardware configuration control.

Commercial off-the-shelf. (COTS) Usually refers to software purchased from a vendor "as-is" with minimal customization or configuration options that meets a requirement.

Configuration Control. An element of configuration management, consisting of the evaluation, coordination, approval or disapproval, and implementation of changes to configuration items after formal establishment of their configuration identification. [ISO/IEC/IEEE 24765:2010(E)]

Configuration identification. An element of configuration management, consisting of selecting the configuration items (see def.) for a system and recording their functional and physical characteristics in technical documentation.

Configuration item (CI). An item or aggregation of hardware or software (including documentation) or both that is designed to be managed as a single entity (ISO/IEC/IEEE 24765:2010(E) edited).

Configuration management. A discipline applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a configuration item (see def.), control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements (ISO/IEC/IEEE 24765:2010[E]).

Configuration Management (see def.) consists of activities to control and manage changes to items that have a *baseline* (see def.). It includes the process of identifying the *configuration items* (CIs) (see def.) in a system, controlling the release and change of these items, and recording and reporting the status of the CIs and their associated change requests.

Continuous Integration System (CIS). A system, linked to a central version control repository, such as *GitHub* and *GitLab* (see def.), aimed to automatically build and test a targeted software. Examples are CIVET, Jenkins, and GitLab Continuous Integration.

Custom-built IT assets. Information technology (IT) assets designed, developed, or modified internally or by a qualified subcontractor through the procurement process. Examples include custom-developed (see def.) or customized software, spreadsheet, and calculation and analysis applications (e.g., computer models), the implementation of a new network infrastructure or IT technology (e.g., Gmail, Internet Protocol Version 6, Internet Explorer 9). [Developed for internal laboratory use]

Custom-developed software. Software built specifically for a DOE application or to support the same function for a related government organization. It may be developed by DOE or one of its M&O contractors or contracted with a qualified software company through the procurement process. Examples of custom-developed software include material inventory and tracking database applications, accident consequence applications, control system applications, and embedded custom-developed software that controls a hardware device.

Defect. An error, fault or failure in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.

Doxygen. Standard tool for generating documentation from annotated C, C++, Fortran and Python sources.

Dymola. Dymola is a commercial modeling and simulation environment based on the open Modelica modeling language, Developed by the European company Dassault Systèmes.

Electronic Document Management System (EDMS). System approved for long-term storage, management, and maintenance of electronic and hardcopy records.

Enterprise Architecture (EA) Repository. An Oracle database that houses information about software applications and servers and is the source for the INL data dictionary. The applications are related to the management system business functions it supports or implements. EA is the repository for the technology (e.g., software/hardware) used to construct and implement software applications. EA contains links to the software documentation stored in *EDMS* (see def.) and includes a list of software owners.

FMI. The Functional Mock-up Interface (or FMI) defines a standardized interface to be used in computer simulations to develop complex cyber-physical systems.

FMU. Based on an FMI, the FMU is an executable called a Functional Mock-up Unit (FMU), which is "driven" by an FMI. A simulation environment can use the FMI to create an instance of the FMU and simulate it together with other FMUs or models native to the simulation environment.

GitHub. A web-based revision control hosting service for software development and code sharing. GitHub provides additional tools such as documentation generation, issue tracking, Wikis, nested task-lists within files, etc.

GitLab. A web-based revision control hosting service for software development and code sharing similar to GitHub. The CIS (see def.) connects to both the external and internal GitHub/GitLab to perform software builds.

Issue. Issues can be initiated by anyone, including off site users, and are used for maintenance (fine-tuning and problem resolving), new development, enhancements, or can be used to report program errors and problems.

Issue (GitHub). As defined for the GitHub environment, issues are suggested improvements, tasks, or questions related to the repository. Issues can be created by anyone (for public repositories) and are moderated by repository collaborators. Each issue contains its own discussion forum and can be labeled and assigned to a user/developer.

Major Change. A revision to software that, in the best judgment of authorizing personnel, has the potential to compromise the accuracy/validity of the output data, and as a result, could diminish the margin of safety to the public, worker, or environment.

Method. A reasonably complete set of rules and criteria that establish a precise and repeatable way of performing a task and arriving at a desired result. [The Configuration Management Manual Guideline for Improving the Software Process, Carnegie Mellon University Software Engineering Institute, 1995]

Minor Change. A revision to software that, in the best judgment of authorizing personnel, will not compromise the accuracy/validity of the output data and will not diminish the margin of safety to the public, worker, or environment.

Modelica. Object-oriented, declarative, multi-domain modeling language for component-oriented modeling of complex systems, e.g., systems containing mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents.

Open source. Denoting software for which the original source code is made freely available and may be redistributed and modified.

Pull requests. Pull requests can be initiated by anyone, including off-site users, and are used for maintenance (fine-tuning and problem resolving), new development, enhancements, or can be used to address program errors and problems. Pull requests allow informing others about changes pushed to a repository on a *version control system (see def.).* Once a pull request is sent, interested parties can review the set of changes, discuss potential modifications, and even push follow-up commits if necessary, as well as integrate changes into the maintained code.

Quality grade. The grade applied to the level of quality activities to be applied to the specific task or activity. Current quality grades are Nuclear Use QL and Commercial Use Quality Levels (QLs) High, Medium, and Low.

RAVEN core team. INL personnel who are in charge of the development of the RAVEN framework or software applications/extensions/plugins that are based on the RAVEN framework. A list of the current components of the RAVEN core team can be found at https://github.com/idaholab/raven/wiki/AboutUs#raven-core-team

HYBRID core team. INL personnel who are in charge of the development of the HYBRID software applications/extensions that are based on the HYBRID software. A list of the current components of the HYBRID core team can be found at https://github.com/idaholab/HYBRID/-/wikis/About-Us

RAVEN Software. Open source software that resides in a public repository (GitHub) that provides the capabilities needed to perform Uncertainty Quantification, Probabilistic Risk Assessment, Data Analysis, Validation and Parameter Optimization.

HYBRID Software. Collection of software/models/analysis workflows that resides in a public repository (GitHub) that provides the for the deployment of a "plug and play" framework aimed to integrate Modelica/Dymola with RAVEN in terms of both FMI/FMU construction and repository structure that aims to ease the sharing and simulation of complex dynamic models.

Regression testing. Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements.

Retirement. Permanent removal of an asset (e.g., system or component) and associated support from its operational environment.
[ISO/IEC/IEEE Std 24765-2010 edited]

Safety function. The performance of an item or service necessary to achieve safe, reliable, and effective utilization of nuclear energy and nuclear material processing. For INL, safety functions are identified and defined in a formal safety basis or commitment document as credited for achieving nuclear safety (e.g., safety structures, systems, and components; safety significant; safety class; safety related; or important to safety) (ASME NQA-1-2008 with the NQA-1a-2009 addenda edited).

Software. Computer programs and associated documentation and data pertaining to the operation of a computer system and includes application software and support software.

Software life cycle. The activities that comprise evolution of software from conception to retirement. The software life cycle typically includes the activities associated with requirements, design, implementation, test, installation, operation, maintenance, and retirement.

Software quality assurance. All actions that provide adequate confidence that software quality is achieved.

Software tool. A computer program used in development, testing, analysis, or maintenance of a program or its documentation. Examples include comparators, cross-reference generators, compilers, computer-aided software-engineering tools, configuration and code management software, flowcharters, monitor test case generators, and timing analyzers.

Support software. Software tools (see def.) and system software (see def.).

System software. Software designed to facilitate operation and maintenance of a computer system and its associated programs (e.g., operating systems and utilities).

System testing. Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

Task (GitHub). A suggested improvement or feature enhancement.

Test case. (1) A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. (2) Documentation specifying inputs, predicted results, and a set of execution conditions for a test item.

User documentation. Instructions for use describing the capabilities and intended use of the software within specified limits. May also include a theory manual, when relevant.

Validation. Confirmation, through the provision of objective evidence (e.g., acceptance test), that the requirements for a specific intended use or application have been fulfilled. [ISO/IEC/IEEE 24765:2010(E) edited].

Verification. (1) The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. (2) Formal proof of program correctness (e.g., requirements, design, implementation reviews, system tests). [ISO/IEC/IEEE 24765:2010(E) edited]

Version Control System. It is the system aimed to support the management of changes to files, in general, and computer programs, in particular. Changes are usually identified by a number, letter code or unique alphanumeric identifiers, termed the "revision number", "revision level", or simply "revision". Each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored, and with some types of files, merged. Examples of Version Control Systems are GitHub and GitLab (see def.)

3.2 Acronyms

ASME American Society of Mechanical Engineers

BEA Battelle Energy Alliance

CCB Change Control Board

CFR Code of Federal Regulations

CI Configuration Item

CIS Continuous Integration System

CM Configuration Management

CMP Configuration Management Plan

COTS Commercial off-the-shelf software

CR Change Request

CSV Comma Separated Value

DOE Department of Energy

EA Enterprise Architecture

EDMS	Electronic Document Management System
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
IAS	Integrated Assessment System
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
INL	Idaho National Laboratory
ISMS	Integrated Safety Management System
ISO	International Organization for Standardization
IT	Information Technology
LST	List
LWP	Lab-wide Procedure
M&O	Maintenance and Operations
NQA	Nuclear Quality Assurance
POSIX	Portable Operating System Interface
PRA	Probabilistic Risk Assessment
QA	Quality Assurance
QL	Quality Level
QLD	Quality Level Determination
RTM	Requirement Traceability Matrix
RAVEN	Risk Analysis and Virtual ENvironment
SRS	Software Requirements Specification
SSD	Safety Software Determination
SQA	Software Quality Assurance

SQAP Software Quality Assurance Plan

USGCB U.S. Government Configuration Baseline

V&V Verification and Validation

4. MANAGEMENT

The MANAGEMENT plan of the HYBRID Software fully adheres with the one spelled out in the *PLN-5552, "RAVEN and RAVEN Plug-ins Software Quality Assurance and Maintenance and Operations Plan"* (replacing the word RAVEN with HYBRID).

5. CONFIGURATION MANAGEMENT

The CONFIGURATION MANAGEMENT plan of the HYBRID Software fully adheres with the one spelled out in the *PLN-5552, "RAVEN and RAVEN Plug-ins Software Quality Assurance and Maintenance and Operations Plan"* (replacing the word RAVEN with HYBRID). The HYBRID configuration items' list can be found in LST-1296.

6. SUBCONTRACTOR.VENDOR

No subcontractors/vendors activities are envisioned for HYBRID Software. In case of a new strategy, involving subcontractors, is defined, this plan will be revised.

7. DOCUMENTATION

The purpose of this section is to define the minimum documentation required to properly implement the SQA requirements. At all times during the life cycle of HYBRID, the following documents will be maintained as part of the Asset Portfolio.

7.1 Minimum Documentation Requirements

As a minimum, the following documentation is required for the HYBRID software. These documents are managed as records in accordance with Section 15, "RECORDS COLLECTION, MAINTENANCE, AND RETENTION."

The following documentation is required as a minimum:

Document	Record Location	ID
Software Quality Assurance Plan	Electronic Document Management System (EDMS)	PLN-6274
Software Test Plan and Verification & Validation	GitHub	PLN-6274

Software Requirements Specification and Traceability Matrix	GitHub	SPC- 2990
Software Design Description	GitHub	SDD-561
User Documentation (see def.)	GitHub	INL/MIS-20-60624

7.2 Other Documentation

In addition to the above documents, the following are created during the procurement and baselining of the project. These may be used in support of Change Control Request implementation and M&O activities.

- SSD-000753, “HYBRID Safety Software Determination”
- QLD, “HYBRID Quality Level Determination”
- HYBRID CTM Entry: 3C9B336C-8262-4790-AEBD-582B1BD85CF5

All documents will be managed according to LWP-1201, “Document Management.”

All records generated as part of this plan will be processed and managed according to LWP-1202, “Records Management.”

8. STANDARDS, PRACTICES, CONVENTIONS, AND METRICS

8.1 Content

The standards for HYBRID are maintained/recorded in the HYBRID GitHub repository (Wiki section). Any developer of the HYBRID software need to be aware of the standards and to follow the development guidelines.

The HYBRID standards evolve around the following macro-areas:

- Software Coding Standards
- Commentary Standards
- Testing Standards and Practices

8.1.1 Software Coding Standards

The HYBRID software imposes a coding standard on all source code within the repository. This standard is publicly maintained on the HYBRID GitHub repository wiki website (https://github.com/idaholab/HYBRID/-/wikis/HYBRID-Code-Standards) and enforced through the continuous integration testing system.

8.1.2 Commentary Standards

The HYBRID software imposes a commentary standard on all source code within the repository. The standard is aimed to fully describe any module/method in the source code, guaranteeing the automatic generation of software documentation via doxygen (see def.). This standard is publicly maintained on the HYBRID GitHub repository wiki website (https://github.com/idaholab/HYBRID/-/wikis/Hybrid-Software-Commentary-Standard) and enforced through the continuous integration testing system.

8.1.3 Testing Standards and Practices

The HYBRID software imposes a testing standard and practices on all the capabilities/methods of the HYBRID software. This standard is publicly maintained on the HYBRID GitHub repository wiki website (https://github.com/idaholab/HYBRID/-/wikis/HYBRID-Testing-Standards-and-Practices) and enforced through the review process by a member of the CCB.

9. SOFTWARE REVIEWS

The SOFTWARE REVIEWS process of the HYBRID Software fully adheres with the one spelled out in the *PLN-5552, "RAVEN and RAVEN Plug-ins Software Quality Assurance and Maintenance and Operations Plan"* (replacing the word RAVEN with HYBRID).

10. TESTING

The goal of software *validation* (see def.) is to confirm that the requirements for a specific intended end use have been fulfilled. Software *verification* (see def.) evaluates a system or component to confirm that specified conditions have been satisfied and provides formal proof of correctness.

10.1 V&V Overview

10.1.1 Test & V&V Objectives

Test procedures or plans will specify the following as applicable:

- required tests and test sequence
- required ranges of input parameters
- identification of the stages at which testing is required
- criteria for establishing test cases
- requirements for testing logic branches
- requirements for hardware integration
- anticipated output values
- acceptance criteria
- reports, records, standard formatting, and conventions
- performance testing

Any developer, including externals, are responsible for ensuring the creation of a *test case* (see def.) that covers the new capability or code change. The *CCB* (any of its member not directly involved in the *CR*) is responsible, through the help of the Review Check Lists (see def.), for verifying that an appropriate test case is provided, and passes based on the supplied acceptance criteria. This verification is performed for any CR and failing to meet these requirements shall conclude in rejecting the *CR* by the *CCB* member/reviewer. The process for handling *CRs* that modify or add requirements is discussed in Section 5, Configuration Management Activities.

HYBRID is *open source* (see def.) software that is maintained and stored in *GitHub* (see def.), a public repository. In order to align the testing and V&V activities of the software with the nature of the *Agile development process* (see def.), the verification of the software has been designed in a multi-stage automated testing suite, using the *Continuous Integration System* (CIS) (see def.) in GitHub.

The main scope of the automated testing is to guarantee that any capability is properly tested and that new addition to the software do not impact the functionalities of the already-deployed capabilities.

Four types of testing, unit, integration, system, and deployment, are covered by the HYBRID software.

The project manager/technical leader oversees the testing and verification and validation (V&V) activities, including the analysis of test coverage and the determination of when new tests are necessary. The test coverage analysis is performed during the code review activities conducted by the *HYBRID core team* (see def.), and it is determined at that step in the process if one or more new tests needs to be created. V&V activities are distributed among the *HYBRID core team*.

Every time a new development or capability is performed by a software developer, the following shall be determined:

- Required test activities and method of documentation (e.g., test plans, procedures, checklists, etc.);
- Required *support software* (see def.) (e.g., automated test scripts, fault insertion tools, etc.);
- Type and extent of required testing; and
- Required reviews and approvals.

A component (or more) of the *change control board* (*CCB)* (see def.), not being part of the development, shall review the correct documentation of the tests and ensure that the documentation includes approved requirements (when necessary) that have valid acceptance criteria. This documentation may include:

- Documentation of the tests including acceptance criteria. The documentation procedure is defined in the HYBRID wiki page (https://github.com/idaholab/HYBRID/-/wikis/Developing-Regression-Tests)
- Software Requirements Specification or equivalent requirements document;
- Requirements Traceability Matrix;
- Software Design Description for guidance on testing methodologies and the operating environment (i.e., software, firmware, and hardware elements) to be used during testing;
- *User documentation* (see def.)

The CIS will verify that the provided documentation ensures that the software demonstrates adherence to the documented requirements and that the software produces correct results.

10.1.2 Master Schedule

The V&V tasks (as captured in the automated tests) are executed automatically for every change to HYBRID software (i.e. source code). At several steps during the change commit process, automated tests are executed.

10.1.3 Specific meaning of V&V activities for HYBRID software

The HYBRID software contains modelica models that will be, if available, compared with experimental results.

10.2 TYPES OF TESTS TO BE EXECUTED

Tests are defined using an input file syntax, which specifies what the test should do, the inputs, and the post conditions for determining test success or failure; and assuring that the software produces correct results. The guidelines for the creation of a new test are reported in the HYBRID wiki page (https://github.com/idaholab/HYBRID/-/wikis/Developing-Regression-Tests). Any test case that is connected with a requirement or modify/add a new requirement shall be tagged with the associated requirement ID.

Acceptance Criteria for each test is defined by the Test type (defined below).

The collection of Test types ensure that the software properly handles abnormal conditions and events as well as credible failures, does not perform adverse unintended functions, and does not degrade the system either by itself, or in combination with other functions or configuration items.

The Test types and acceptance criteria for each are as follows:

- CSVdiff: A test case that runs a simulation, terminates without error, and produces a previously defined comma separated value solution within a predefined tolerance (usually to at least single precision accuracy or better). The order of data in the CSV must exactly match the reference solution file.

- UnorderedCSVDiffer: A test case that runs a simulation, terminates without error, and produces a previously defined comma separated value solution within a predefined tolerance (usually to at least single precision accuracy or better). The order of data (rows) in the CSV can be different with respect the previously defined file. *Note:* This Test is generally used when multiple parallel executions of an underneath model are performed, and the collection of the data can be unsynchronized depending on the latency of the network/machine. ***This test is only allowed if a parallel test is created.***

- TextDiff: A test case that runs a simulation, terminates without error, and produces a previously defined text file that matches a reference solution file.

- XMLDiff: A test case that runs a simulation, terminates without error, and produces a previously defined Extensible Markup Language (XML) solution within a predefined tolerance (usually to at least single precision accuracy or better).

- RAVENImageDiff: A test case that runs a simulation, terminates without error, and produces a previously defined image or picture within a predefined tolerance (in terms of pixel difference).

- RavenErrors: A test case that runs and produces a specified console output or output pattern and terminates with an expected error code or message.

- DymolaMatDiff: A test case that runs a simulation, terminates without error, and produces a previously defined ". mat" solution file within a predefined tolerance (usually to at least single precision accuracy or better).

- HPCinteraction: A test case that runs a simulation in a High-Performance Computing System using its native Job Scheduler and Workload manager (e.g. Portable Batch System – PBS), terminates without error.

In addition to the above reported Test types, for any CR the following tests are performed:

- Documentation Test: The CIS tests that the User Documentation and SQA Documentation can correctly be generated.

- Code Standard Validation: The CIS tests that all the source code is compliant with the RAVEN software coding standards (e.g. source code syntax, formats, documentation, etc.).

- Code Coverage: The CIS tests that at least the 80% of the source code is tested by the test suite.

10.3 Test Automation

Testing is performed automatically as part of the CIS process when a user commits a change to the repository. The automated tests that are executed at subsequent steps in the process vary in scope and type and are described in Table 2. Tests of the framework across multiple platforms (operative systems and versions) are executed with each *pull request* (see def.).

In order to pass acceptance testing, all test cases are expected to pass under the environments identified in the configuration items for HYBRID software.

Use of the automated tests is integrated directly into GitHub, and as such does not require additional training other than general familiarity with performing a pull request in GitHub.

Results from each test execution are maintained in the CIS database, in an approved records repository along with results from the timing executions and code coverage.

10.4 APPROVAL REQUIREMENTS

The HYBRID software relies on a heavy automation of the verification and testing of any new or modified capability. This approach is required for the nature of the *Agile development* process. As mentioned in the previous section, any CR in the source code needs to be accompanied with a new (or modified) test to assess the correctness of the code and its functionality.

Depending of the type of test case that is added or modified, two different approval processes are followed:

10.5 Requirement tests

This category is about to test any functionality that is linked to any new or assessed requirements.

Table 3 - Requirement tests' responsibilities.

Test Case Reviewer(s):	Chair of the *CCB*, Technical Leader and Independent Reviewer (Member of the *CCB*)
Test Result Reviewer and Approver:	Chair of the *CCB* or Technical Leader and Independent Reviewer (Member of the *CCB*)
Acceptance Test Case Reviewer(s):	Chair of the *CCB*, Technical Leader and Independent Reviewer (Member of the *CCB*)
Acceptance Result Reviewer(s):	Automated CIS
Acceptance Result Approver:	Automated CIS

10.6 Other tests

This category is about to test any functionality that is not linked to any specific requirement (e.g. infrastructure tests, verification tests, etc.).

Table 4 - Other tests' responsibilities

Test Case Reviewer(s):	Independent Reviewer (Member of the *CCB*)
Test Result Reviewer and Approver:	Independent Reviewer (Member of the *CCB*)
Acceptance Test Case Reviewer(s):	Independent Reviewer (Member of the *CCB*)
Acceptance Result Reviewer(s):	Automated CIS
Acceptance Result Approver:	Automated CIS

10.7 TEST DEFINITION TASKS AND RESPONSIBILITIES

This section summarizes the tasks and associated roles in the definition of the test cases and their approval.

Table 5 - Tasks and responsibilities for tests creation.

Tasks	Responsibility
1. Complete programming and test creation	Developer of the proposed CR
2. Test data creation	Developer of the proposed CR
3. Set up test environment	Automated via CIS
4. Migrate services to test environment	Automated via CIS
5. Set up test database	Automated via CIS
6. Prepare test cases	Developer of the *CR*
7. Conduct test, record results, and communicate to the developers	Automated via CIS
8. Make corrections and updates to the processes	Developer of the *CR*
9. Review and approve final results of the test	Independent reviewer part of the *CCB* and Technical Leader (or Chair of *CCB*) in case of requirement test.

Note: *The above steps need to be conducted for every type of testing*

11. V&V PROCESSES

The V&V PROCESSES of the HYBRID Software fully adheres with the one spelled out in the *PLN-5552, "RAVEN and RAVEN Plug-ins Software Quality Assurance and Maintenance and Operations Plan"* (replacing the word RAVEN with HYBRID).

12. PROBLEM REPORTING AND CORRECTIVE ACTION

The PROBLEM REPORTING AND CORRECTIVE ACTION of the HYBRID Software fully adheres with the one spelled out in the *PLN-5552, "RAVEN and RAVEN Plug-ins Software Quality Assurance and Maintenance and Operations Plan"* (replacing the word RAVEN with HYBRID).

13. TOOLS, TECHNIQUES, AND METHODOLOGIES

The TOOLS, TECHNIQUES, AND METHODOLOGIES of the HYBRID Software fully adheres with the one spelled out in the *PLN-5552, "RAVEN and RAVEN Plug-ins Software Quality Assurance and Maintenance and Operations Plan"* (replacing the word RAVEN with HYBRID).

14. SUPPLIER CONTROL

No subcontractors/vendors activities are envisioned for HYBRID. In case of a new strategy, involving subcontractors, is defined, this plan will be revised.

15. RECORDS COLLECTION, MAINTENANCE, AND RETENTION

The RECORD COLLECTION, MAINTENANCE, AND RETENTION process of the HYBRID Software fully adheres with the one spelled out in the *PLN-5552, "RAVEN and RAVEN Plug-ins Software Quality Assurance and Maintenance and Operations Plan"* (replacing the word RAVEN with HYBRID).

16. TRAINING

The TRAINING process of the HYBRID Software fully adheres with the one spelled out in the *PLN-5552, "RAVEN and RAVEN Plug-ins Software Quality Assurance and Maintenance and Operations Plan"* (replacing the word RAVEN with HYBRID).

17. RISK MANAGEMENT

The risk analysis for each application is documented on the safety software determination (SSD) and quality level determination (QLD). The SSD and QLD are identified in the EA repository for each individual application. Risks associated with the HYBRID software are controlled via the rigor implemented in requirements identification, testing, verification and validation, and change control processes.

17.1 Safety Software Determination

The SSD documents the decision basis as to why a software application is or is not safety software. The record copy is maintained within the company approved

"Determining Quality Levels."

-

5552, "RAVEN and RAVEN Plug
Operations Plan"

Plans'
5552, "RAVEN and RAVEN Plug
and Maintenance and Operations Plan"

5552, "RAVEN and RAVEN Plug
Software Quality Assurance and Maintenance and Operations Plan"

5552, "RAVEN and RAVEN Plug
Assurance and Maintenance and Operations Plan"

APPENDIX C – SQA: SOFTWARE DESIGN DESCRIPTION (SDD)

MANUAL

SDD-561
Revision 0
Printed March 30, 2021

HYBRID Software Design Description

Konor Frick, Andrea Alfonsi

Prepared by
Idaho National Laboratory
Idaho Falls, Idaho 83415

The Idaho National Laboratory is a multiprogram laboratory operated by
Battelle Energy Alliance for the United States Department of Energy
under DOE Idaho Operations Office. Contract DE-AC07-05ID14517.

Approved for unlimited release.

Idaho National Laboratory

Issued by the Idaho National Laboratory, operated for the United States Department of Energy by Battelle Energy Alliance.

SDD-561
Revision 0
Printed March 30, 2021

HYBRID Software Design Description

Konor Frick, Andrea Alfonsi

Contents

6

1 Introduction

The HYBRID repository is a collection of models and workflows used to assess the technical and economic feasibility of different integrated energy systems. The repository includes a library of high fidelity Modelica models developed in Dymola that includes models of nuclear reactors, gas turbines, hydrogen production facilities, energy storage technology, and other integrated energy system technologies. Models have been developed since 2014 through a tri lab coordination between Idaho National Laboratory (INL), Oak Ridge National Laboratory (ORNL), and Argonne National Laboratory (ANL). With the models users can quickly create large scale integrated energy systems to test out the technical interoperability of systems and develop novel control strategies to best integrate systems together. In addition to the high fidelity modelica models the HYBRID repository provides workflows that allows direct integration with the Risk Analysis Virtual Environment (RAVEN) code (`https://github.com/idaholab/raven`) developed at INL and it's plugins the Heuristic Energy Resource Optimization Network (HERON - available at `https://github.com/idaholab/heron`) and the Tool for Economic AnaLysis (TEAL - available at `https://github.com/idaholab/teal`) packages.

1.1 User Characteristics

The users of the HYBRID software are expected to be part of any of the following categories:

- **Core developers (HYBRID core team)**: These are the developers of the HYBRID software. They will be responsible for following and enforcing the appropriate software development standards. They will be responsible for designing, implementing, and maintaining the software.
- **External developers**: A Scientist or Engineer that utilizes the HYBRID framework and wants to extend its capabilities (new modelica models, new workflow generation, etc).This user will typically have a background in modeling and simulation techniques and/or control systems but may only have a limited skill-set when it comes to repository structure, regression testing, and version control.
- **Analysts**: These are users that will run the code and perform various analysis on the simulations they perform. These users may interact with developers of the system requesting new features and reporting bugs found and will typically make heavy use of the input file format.

1.2 Other Design Documentation

Also available within the repository "Repository" is the HYBRID Repository User manual within the "docs" folder. This user manual gives a detailed explanation of the installation process, system dependencies alongside links upon which where to find them, and an explanation of the Modelica models within the repository.

1.3 Dependencies and Limitations

The software should be designed with the fewest possible constraints. Current constraints are:

1. Commercial Modelica platform Dymola – `https://www.3ds.com/products-services/catia/products/dymola/latest-release/`
2. Risk Analysis and Virtual ENviroment (RAVEN) – `https://raven.inl.gov/SitePages/Software%20Infrastructure.aspx`
3. Python 3 – `https://docs.conda.io/en/latest/miniconda.html`
4. Microsoft Visual Studio Community Edition. – `https://visualstudio.microsoft.com/downloads/`

2 References

- ASME NQA 1 2008 with the NQA-1a-2009 addenda, “Quality Assurance Requirements for Nuclear Facility Applications,” First Edition, August 31, 2009.
- ISO/IEC/IEEE 24765:2010(E), “Systems and software engineering Vocabulary,” First Edition, December 15, 2010.
- LWP 13620, “Managing Information Technology Assets”

3 Definitions and Acronyms

3.1 Definitions

- **Baseline.** A specification or product (e.g., project plan, maintenance and operations [M&O] plan, requirements, or design) that has been formally reviewed and agreed upon, that thereafter serves as the basis for use and further development, and that can be changed only by using an approved change control process. [ASME NQA-1-2008 with the NQA-1a-2009 addenda edited]
- **Validation.** Confirmation, through the provision of objective evidence (e.g., acceptance test), that the requirements for a specific intended use or application have been fulfilled. [ISO/IEC/IEEE 24765:2010(E) edited]
- **Verification.**
  - The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.
  - Formal proof of program correctness (e.g., requirements, design, implementation reviews, system tests). [ISO/IEC/IEEE 24765:2010(E) edited]

3.2 Acronyms

API Application Programming Interfaces

ANL Argonne National Laboratory

ARMA Auto-Regressive Moving Average

DOE Department of Energy

FMI Functional Muck-up Interface

FMU Functional Muck-up Unit

HERON Heuristic Energy Resource Optimization Network

IES Integrated Energy Systems

INL Idaho National Laboratory

NHES Nuclear-Renewable Hybrid Energy Systems

IT Information Technology

ORNL Oak Ridge National Laboratory

M&O Maintenance and Operations

NQA Nuclear Quality Assurance

POSIX Portable Operating System Interface

QA Quality Assurance

RAVEN Risk Analysis and Virtual ENviroment

SDD System Design Description

TEAL Tool for Economic Analysis

TRANSFORM Transient Simulation Framework of Reconfigurable Modules

XML eXtensible Markup Language

4 Design Stakeholders and Concerns

4.1 Design Stakeholders

- Integrated Energy Systems (IES) program
- Open-source community

4.2 Stakeholder Design Concerns

The Hybrid repository is to be deployed in accordance with the funding programs reported above. No specific concerns have been raised during the design and deployment of the Hybrid software.

5 Software Design

5.1 Introduction

The HYBRID repository is a collection of models and workflows used to assess the technical and economic feasibility of different integrated energy systems. The purpose of the software is to allow the user to design process models quickly and efficiently for use within a technoeconomic analysis. The models are to be designed with the goal of interoperability and "plug and play" design in mind. This design philosophy allows users to quickly create and test new integrated energy systems, control schemes, and energy offtake opportunities and pathways.

In addition to the high fidelity modelica models the HYBRID repository provides basic workflows that allows direct integration with the Risk Analysis Virtual Environment (RAVEN) code developed at INL and it's plugins the Heuristic Energy Resource Optimization Network (HERON) and the Tool for Economic AnaLysis (TEAL) packages. Through the integration of these different packages and repositories complete grid analysis and systemwide optimization can be achieved.

5.2 Hybrid Repository Structure

The HYBRID Repository structure is illustrated in Figure 1 where the components are:

- ***Models***: Folder containing the Modelica/Dymola models and future location of new RAVEN workflows generated.
- ***doc/user_manual***: Folder containing the user manual for the repository
- ***archive***: containing legacy workflows from previous externally released milestones.
- ***TRANSFORM library***: Submodule of the Oak Ridge National Laboratory (ORNL) based TRANSFORM library that is used as the base models for many of the integrated energy systems models
- ***raven***: submodule linking to the RAVEN ([1]) repository
- ***scripts***: containing the dymola_launcher and the tools for loading Modelica/Dymola outputs into a Python environment
- ***tests***: containing all the tests that are automatically executed by the Continuous Integration system and executable, locally, running the command "run_tests."

Within the ***Models*** folder there are two subfolders ***NHES*** and ***RAVEN_WORKFLOWS***. The RAVEN_WORKFLOWS folder is empty and is where future RAVEN_WORKFLOWS used in techno-economic analysis will be placed. The NHES folder contains all the Modelica models ranging from Gas Turbines to Nuclear power plant and all the associated subsystems. The doc/user_manual folder contains the user manual for the HYBRID repository. The archive folder contains the models executed for two 2017 milestones. The ***TRANSFORM*** submodule is a library of Modelica models created by ORNL in conjunction with the NHES library that is used as the base models for many of the integrated energy systems developed within the NHES library. The ***TRANSFORM*** submodule is updated on a six month basis and all regression tests are run to ensure none of the models are broken between updates. The raven submodule is a link to the RAVEN repository and is updated frequently to ensure all the latest optimization and processing capabilities are available. The ***scripts*** folder contains files to allow the automatic launching of Dymola for use within the regression system. Additionally, it holds a folder called testers that contains all the files used to enable the .mat differ used within the regression system ROOK. Then the final folder is tests which contains all the tests that are automatically executed by the Continuous Integration system. Within the ***tests*** folder there are ***dymola_tests*** and ***raven_tests/train***. The dymola_tests contains all of the dymola regression tests while the raven_tests/train contains all of the raven tests that may need to be added as additional raven workflows are added to the Hybrid repository. The repository structure is evolving, but the current topology of the folders will stay the same.

Name	Last commit	Last update
.gitlab	issue and MR templates added	2 years ago
Models	Addition of Initialization files and cleaning of example files in t...	6 days ago
archive	added description	2 months ago
developer_tools	added description	2 months ago
doc/user_manual	Latex File Tree Addition	1 week ago
scripts	fixed list	2 weeks ago
tests	Update of Readme for what the different tests are	1 week ago
TRANSFORM-Library @ 96d3493a	added transform submodule	2 months ago
raven @ 60137c37	updated raven	2 weeks ago
.gitignore	Changing the .gitignore file to not ignore .mat files.	1 week ago
.gitmodules	added transform submodule	2 months ago
00README.txt	Civet test dirtectory set-up	4 years ago
README.md	Update README.md	5 months ago
run_tests	Use proper pathsep for the os.	2 months ago

Figure 1: Structure of the HYBRID repository.

5.3 Regression Test System

HYBRID a repository that contains a series of Modelica models capable of producing potential integrated energy system configurations. To test these models the RAVEN based regression system ROOK has been utilized. This testing system has been linked with the Continuous Integration tool to automatically test the models when new modifications are added to the repository. To do this RAVEN has been sub-moduled within HYBRID.

5.3.1 Dymola Regression Tests

ROOK operates via a basic testing harness. The testing harness includes a “tests” file that contains the tolerance limits, a gold folder with a gold test file, a simulation file to run, a file with which to launch the simulation, and a directory of tests to run. Since the models to be run are Modelica models within the Dymola simulation platform a dymola_launcher file was created to automatically run Dymola via the command line interface. For Modelica models the gold file is checked via a .mat file differ that has been created within the ROOK testing harness. This .mat differ checks all variables created by the test and compares them with an earlier version of the system. If the new .mat file is within a specified “tolerance” then the testing harness comes back with a clean pass. A series of Modelica tests have been added to test the system-level interactions in the Nuclear Hybrid Energy Systems (NHES) Modelica repository and the collection of regression tests will continue to grow as the number of models and model uses grows.

5.3.2 Raven ”Hybrid Specific” Regression Tests

In addition to the Modelica testing conducted by ROOK additional RAVEN specific tests are run. These raven tests are of workflows that are specific to hybrid energy system workflow generation. HYBRID is designed to be able to provide a techno-economic assessment of different integrated energy systems. As part of this HYBRID includes the generation of RAVEN workflows capable of implementing stochastic time series of wind, solar, and electric price data created via the Auto Regressive Moving Averages (ARMAs) algorithms within RAVEN into a workflow that can then be integrated into the Modelica models. The creation of these ARMAs using data held within the HYBRID repository is maintained using the ROOK system.

6 Data Design and Control

The data transfer in the HYBRID framework is fully standardized:

- ***Modelica Models***: API deployed by Modelica Models;
- ***FMI/FMU***: API deployed by the FMI/FMU importer/exporter for RAVEN and Modelica models.

The documentation of these APIs is reported in the HYBRID user manual.

7 Human-Machine Interface Design

There are no human system integration requirements associated with this software.

8 System Interface Design

HYBRID framework contains Modelica models that can be interfaced directly by modelica packages (e.g. Dymola) or via the standardized FMI/FMU interface.

9 Security Structure

The software is accessible to the open-source community (Apache License, Version 2.0). No restrictions for downloading or redistributing is applicable.

10 REQUIREMENTS CROSS-REFERENCE

The requirements are detailed in SPC-2990, “HYBRID Software Requirements Specification (SRS) and Traceability Matrix”.

References

[1] C. Rabiti, A. Alfonsi, J. Cogliati, D. Mandelli, R. Kinoshita, S. Sen, C. Wang, P. W. Talbot, D. P. Maljovec, and J. Chen, "Raven user manual," tech. rep., Idaho National Laboratory, 2017.

Document Version Information

09a34b32ae7a18d79261047cb9da2e7e7d35523a alfoa Mon, 15 Mar 2021 14:09:50 -0600

v1.28

Idaho National Laboratory

APPENDIX D – SQA: HYBRID SOFTWARE REQUIREMENTS SPECIFICATION AND TRACEABILITY MATRIX (SPC)

MANUAL

SPC-2990
Revision 0
Printed March 30, 2021

HYBRID Software Requirements Specification and Traceability Matrix

Andrea Alfonsi, Konor Frick

Prepared by
Idaho National Laboratory
Idaho Falls, Idaho 83415

The Idaho National Laboratory is a multiprogram laboratory operated by Battelle Energy Alliance for the United States Department of Energy under DOE Idaho Operations Office. Contract DE-AC07-05ID14517.

Approved for unlimited release.

Idaho National Laboratory

Issued by the Idaho National Laboratory, operated for the United States Department of Energy by Battelle Energy Alliance.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

SPC-2990
Revision 0
Printed March 30, 2021

HYBRID Software Requirements Specification and Traceability Matrix

Andrea Alfonsi, Konor Frick

3

4

Contents

6

1 Introduction

The HYBRID repository is a collection of models and workflows used to assess the technical and economic feasibility of different integrated energy systems. The repository includes a library of high fidelity Modelica models developed in Dymola that includes models of nuclear reactors, gas turbines, hydrogen production facilities, energy storage technology, and other integrated energy system technologies. Models have been developed since 2014 through a tri lab coordination between Idaho National Laboratory (INL), Oak Ridge National Laboratory (ORNL), and Argonne National Laboratory (ANL). With the models users can quickly create large scale integrated energy systems to test out the technical interoperability of systems and develop novel control strategies to best integrate systems together. In addition to the high fidelity modelica models the HYBRID repository provides workflows that allows direct integration with the Risk Analysis Virtual Environment (RAVEN) code developed at INL and it's plugins the Heuristic Energy Resource Optimization Network (HERON) and the Tool for Economic AnaLysis (TEAL) packages.
This document is aimed to report and explain the HYBRID software requirements. In addition, it reports the traceability matrix between software requirements and requirement tests (tests that testify the software is compliant with respect its own requirements).

1.1 Other Design Documentation

Also available within the repository "Repository" is the HYBRID Repository User manual within the "docs" folder. This user manual gives a detailed explanation of the installation process, system dependencies alongside links upon which where to find them, and an explanation of the Modelica models within the repository.

1.2 Dependencies and Limitations

The software should be designed with the fewest possible constraints. Current constraints are:

1. Commercial Modelica platform Dymola – `https://www.3ds.com/products-services/catia/products/dymola/latest-release/`
2. Risk Analysis and Virtual ENviroment (RAVEN) – `https://raven.inl.gov/SitePages/Software%20Infrastructure.aspx`
3. Python 3 – `https://docs.conda.io/en/latest/miniconda.html`
4. Microsoft Visual Studio Community Edition. – `https://visualstudio.microsoft.com/downloads/`

2 References

- ASME NQA 1 2008 with the NQA-1a-2009 addenda, "Quality Assurance Requirements for Nuclear Facility Applications," First Edition, August 31, 2009.
- ISO/IEC/IEEE 24765:2010(E), "Systems and software engineering Vocabulary," First Edition, December 15, 2010.
- LWP 13620, "Managing Information Technology Assets"

3 Definitions and Acronyms

3.1 Definitions

- **Baseline.** A specification or product (e.g., project plan, maintenance and operations [M&O] plan, requirements, or design) that has been formally reviewed and agreed upon, that thereafter serves as the basis for use and further development, and that can be changed only by using an approved change control process. [ASME NQA-1-2008 with the NQA-1a-2009 addenda edited]
- **Validation.** Confirmation, through the provision of objective evidence (e.g., acceptance test), that the requirements for a specific intended use or application have been fulfilled. [ISO/IEC/IEEE 24765:2010(E) edited]
- **Verification.**
  - The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.
  - Formal proof of program correctness (e.g., requirements, design, implementation reviews, system tests). [ISO/IEC/IEEE 24765:2010(E) edited]

3.2 Acronyms

API Application Programming Interfaces

ANL Argonne National Laboratory

ARMA Auto-Regressive Moving Average

DOE Department of Energy

FMI Functional Muck-up Interface

FMU Functional Muck-up Unit

HERON Heuristic Energy Resource Optimization Network

IES Integrated Energy Systems

INL Idaho National Laboratory

NHES Nuclear-Renewable Hybrid Energy Systems

IT Information Technology

ORNL Oak Ridge National Laboratory

M&O Maintenance and Operations

NQA Nuclear Quality Assurance

POSIX Portable Operating System Interface

QA Quality Assurance

RAVEN Risk Analysis and Virtual ENviroment

SDD System Design Description

TEAL Tool for Economic Analysis

TRANSFORM Transient Simulation Framework of Reconfigurable Modules

XML eXtensible Markup Language

4 System Requirements: Hybrid

4.1 Minimum Requirements

4.1.1 Minimum Requirements

4.1.1.1 R-M-1

Dymola 2020x or higher

4.1.1.2 R-M-2

Visual Studio 2017 or higher with associated 64-bit Intel Compiler

4.1.1.3 R-M-3

Python 3 or higher to be able to execute RAVEN-based workflows

4.2 Functional Requirements

4.2.1 Modeling

4.2.1.1 R-F-1

HYBRID shall allow the user the leverage and use compoment models developed in Modelica language

4.2.1.2 R-F-2

HYBRID shall provide models to simulate component/system control in Modelica language

4.2.2 Framework, I/O, Execution Control

4.2.2.1 R-F-3

HYBRID shall allow the user the ability to interact with modelica models via the Dymola GUI.

4.3 Regression Requirements

4.3.1 Infrastructure Support

4.3.1.1 R-IS-1

HYBRID shall have regression tests to perform checks on modelica models.

4.4 System Interfaces

4.4.1 Interface with external applications

4.4.1.1 R-SI-1

HYBRID-based models shall be able to be coupled with external applications via input files

4.4.1.2 R-SI-2

HYBRID-based models shall be able to be coupled with standardized interface (FMI and FMU)

4.5 System Operations

4.5.1 Human System Integration Requirements

The command line interface shall support the ability to toggle any supported coloring schemes on or off pursuant to section 508 of the Rehabilitation Act of 1973.

4.5.2 Maintainability

- The latest working version (defined as the version that passes all tests in the current regression test suite) shall be publicly available at all times through the repository host provider.
- Flaws identified in the system shall be reported and tracked in a ticket or issue based system. The technical lead or any COB member will determine the severity and priority of all reported issues. The technical lead will assign resources at his or her discretion to resolve identified issues.
- The software maintainers will entertain all proposed changes to the system in a timely manner (within two business days).
- The HYBRID framework in its entirety is made publicly available under the Apache version 2.0 license.

4.5.3 Human System Integration Requirements

The regression test suite will cover at least 80% of all models at all times. The results of the regression tests will be stored in the Continuous Integration System.

4.6 Information Management

The HYBRID framework in its entirety is made publicly available on an appropriate repository hosting site (e.g. GitHub). Backups and security services will be provided by the hosting service.

5 Verification

The regression test suite shall employ several verification tests of the correct mechanical executions of the models and workflows reported in this repository.

6 HYBRID:SYSTEM REQUIREMENTS

6.1 Requirements Traceability Matrix

This section contains all of the requirements, requirements' description, and requirement test cases. The requirement tests are automatically tested for each CR (Change Request) by the CIS (Continuous Integration System).

6.1.1 Minimum Requirements

Requirment ID	**Requirment Description**	**Test(s)**
R-M-1	Dymola 2020x or higher	1)K. Frick, A. Alfonsi, C. Rabiti, "HYBRID User Manual", INL/MIS-20-60624
R-M-2	Visual Studio 2017 or higher with associated 64-bit Intel Compiler	1)K. Frick, A. Alfonsi, C. Rabiti, "HYBRID User Manual", INL/MIS-20-60624
R-M-3	Python 3 or higher to be able to execute RAVEN-based workflows	1)K. Frick, A. Alfonsi, C. Rabiti, "HYBRID User Manual", INL/MIS-20-60624

Minimum Requirements

6.1.2 Functional Requirements

Requirment ID	**Requirment Description**	**Test(s)**

R-F-1	HYBRID shall allow the user the leverage and use compoment models developed in Modelica language	1)/HYBRID/tests/dymola_tests/GTTP_Test/GTTP_Test.mos 2)/HYBRID/tests/dymola_tests/SMR_primary_test/SMR_Test.mos 3)/HYBRID/tests/dymola_tests/Desalination_2_pass/RO_2_pass.mos 4)/HYBRID/tests/dymola_tests/NSSS_test/NSSS_Test.mos 5)/HYBRID/tests/dymola_tests/Generic_Modular_PWR/Generic_Modular_Test.mo 6)/HYBRID/tests/dymola_tests/Desalination_ROmodule/RO_module.mos 7)/HYBRID/tests/dymola_tests/Bouncing_Ball/Bouncing_Ball.mos 8)/HYBRID/tests/dymola_tests/HTSE_Power_Test/HTSE_Power_Test.mos 9)/HYBRID/tests/dymola_tests/StepDownTurbines_complex/StepdownTurbinesco 10)/HYBRID/tests/dymola_tests/Desalination_NHES_basic/RO_Desal_Test.mos 11)/HYBRID/tests/dymola_tests/Hydrogen_Test/Hydrogen_Burn_Test.mos 12)/HYBRID/tests/dymola_tests/HTSE_Steam_Test/HTSE_Test.mos 13)/HYBRID/tests/dymola_tests/BOP_L1_Boundaries_a_Test/SteamTurbine_L1_bou 14)/HYBRID/tests/dymola_tests/BOP_L1_Boundaries_b_Test/SteamTurbine_L1_bou 15)/HYBRID/tests/dymola_tests/StepDownTurbines/StepdownTurbines_Test.mos 16)/HYBRID/tests/dymola_tests/SMR_Nominal_Test/SMR_Coupling_Test.mos 17)/HYBRID/tests/dymola_tests/Desalination_NHES_complex/RO_Desal_NHES_Te 18)/HYBRID/tests/dymola_tests/Desalination_1_pass/RO_1_pass.mos 19)/HYBRID/tests/dymola_tests/SMR_4Loop/SMR_4Loop.mos 20)/HYBRID/tests/dymola_tests/Desalination_2pass_mixing/RO_2pass_mixing.mos

R-F-2	HYBRID shall provide models to simulate component/system control in Modelica language	1)/HYBRID/tests/dy- mola_tests/Supervisory_Control_Test/InputSetpointData.mos

Modeling

Requirment ID	**Requirment Description**	**Test(s)**
R-F-3	HYBRID shall allow the user the ability to interact with modelica models via the Dymola GUI.	1)https://github.com/idaholab/hybrid/wiki

Framework, I/O, Execution Control

6.1.3 Regression Requirements

Requirment ID	**Requirment Description**	**Test(s)**
R-IS-1	HYBRID shall have regression tests to perform checks on modelica models.	1)/HYBRID/tests/dy- mola_tests/Simple_Breakers_Test/Simple_Breakers_Test.mos

Infrastructure Support

6.1.4 System Interfaces

Requirment ID	**Requirment Description**	**Test(s)**
R-SI-1	HYBRID-based models shall be able to be coupled with external applications via input files	1)/HYBRID/test- s/raven_tests/train/HYBRun_trainARMA_1day.xml

R-SI-2	HYBRID-based models shall be able to be coupled with standardized interface (FMI and FMU)	1)/HYBRID/tests/dymola_tests/FMI_Fluid_ME/FlowReversalME.mos 2)/HYBRID/tests/dymola_tests/FMI_Heat_ME/HeatME.mos 3)/HYBRID/tests/dymola_tests/FMI_heat_CS_capacity/HeatFlowCS.mos 4)/HYBRID/tests/dymola_tests/FMI_heat_CS_conduction/HeatFlowCS.mos 5)/HYBRID/tests/dymola_tests/FMI_Fluid_CS/FlowReversalCS.mos

Interface with external applications

v1.28

Idaho National Laboratory

APPENDIX E – SQA: HYBRID CONFIGURATION ITEM LIST

Document ID: LST-1296
Revision ID: 0
Effective Date: 10/01/2020

List

HYBRID Configuration Items List

Andrea Alfonsi

The INL is a U.S. Department of Energy National Laboratory operated by Battelle Energy Alliance.

Applicability:	Configuration Items List		eCR Number:

Manual:

REVISION LOG

Rev.	Date	Affected Pages	Revision Description
0	09/15/2020	All	Creation of the Configuration Items List

CONTENTS

1. PURPOSE

This document identifies all HYBRID Software *configuration items (CIs)* (see def.). This document also identifies the level designation needed to modify CIs that can potentially affect the ability of HYBRID Software to comply with NQA-1.

2. SCOPE

This list is intended to identify all CIs for HYBRID Software, to provide a document to submit into the CTM (https://ctm.inl.gov) repository, and an aid to identify how severe a change to HYBRID Software will be.

3. RESPONSIBILITIES

The Asset Owner is responsible for maintaining this list and, when necessary, updating the EA repository when the configuration items list changes.

The Asset Owner is also responsible for maintaining configuration management in accordance with PLN-6274, “HYBRID Software Quality Assurance and Maintenance and Operations Plan.”

4. LIST

4.01 Software, Hardware, and Documentation

Table 1. Software, Hardware, and Documentation

Configuration Item	Component	Description	Repository/Location
Application Source Code	HYBRID Software	Source Code and tools for HYBRID Software	GITHUB (https://github.com/idaholab/HYBRID)
	RAVEN	Source Code for the RAVEN code. The HYBRID Software requires RAVEN for some workflows to be functional.	GITHUB (https://github.com/idaholab/raven)
System Software	Modelica language	HYBRID Software modeling language ***(Current versions are maintained in the CTM repository)***	Capabilities & Technology Management (CTM) (3C9B336C-8262-4790-AEBD-582B1BD85CF5
	Python 3.x	HYBRID Software workflow language ***(Current versions are maintained in the CTM repository)***	Capabilities & Technology Management (CTM) (UUID: 3C9B336C-8262-4790-AEBD-582B1BD85CF5)
	Unix-compatible systems	Any compatible Unix system (or Unix-like)	N/A
Support Software	GITHUB CI	Continuous Integration, Verification, Enhancement, and Testing. This is the continuous integration system used by TEAL for automatic testing.	GITHUB Installed in all the Regression Automatic Test Machines (Test Servers)
Hardware	Test Servers	These servers are used to test the HYBRID Software functionality. It will use a "snapshot" of live data to perform the tests. If testing on the server fails, that version of HYBRID Software is sent back to the Development Server for further configuration. ***(Complete and up-to-date list of servers is maintained in CTM repository)***	General Purpose Enclave EROB

	Workstations Laptops Personal Computer	These consist of computer terminals that the end users use to access the software.	N/A
Documentation	SDD-000753	HYBRID Safety Software Determination	EDMS
	ALL-XXXX	HYBRID Quality Level Determination	EDMS
	UUID: CE17AF70-BAB9-46E6-9BB8-74484B7F1791	TEAL Capabilities & Technology Management (CTM)	EDMS
	PLN-6274	HYBRID Software Quality Assurance Plan	EDMS
	PLN-6274	HYBRID Configuration Management Plan	EDMS
	PLN-6274	HYBRID Software Test Plan and V&V	EDMS
	PLN-6274	HYBRID Asset Maintenance Plan	EDMS
	SPC-2990	HYBRID Software Requirements Specification and Traceability Matrix	EDMS
	SDD-561	HYBRID Software Design Description	EDMS
	SPC-2990	HYBRID Software Requirements Specification and Traceability Matrix	EDMS
	INL/MIS-20-60624	HYBRID User Manual	GITHUB