Integrating the IEEE/CIGRE DLL Modeling Standard to Use "Real Code" Models for Power System Analysis in Modelica Tools

Hao Chang[1] Luigi Vanfretti[1]

[1]Electrical, Computer and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, NY, USA,
{changh7, vanfrl}@rpi.edu

Abstract

Vendors of power system simulation tools are investigating the incorporation of actual controller code into specialized simulation environments. To facilitate this, IEEE and CIGRE have collaboratively created the IEEE/CIGRE DLL Modeling Standard. However, adoption by simulation tool providers has been minimal. The limited adoption is because 'real code' models per the IEEE/CIGRE DLL Modeling Standard must be provided as DLLs by equipment vendors. Thus, to support the standard, tools need to support a standard-specific interface and provide additional functions to execute the models.

This paper presents a method for integrating 'real controller code' models (RCMs) built according to the IEEE/CIGRE DLL Modeling Standard into Modelica-based tools. This is achieved by linking precompiled C code to Modelica models and using components from the OpenIPSL library. The approach is demonstrated with an RCM of a simplified silicon-controlled rectifier excitation system (SCRX). The paper discusses the details of the implementation, challenges, and solutions. The findings show that this method allows RCMs to be used in Modelica tools for power system simulations, providing a valuable alternative to specialized simulation tools.

Keywords: IEEE/CIGRE DLL Modeling Standard, Generator excitation, Power Systems, Power System Simulation, External Object

1 Introduction

1.1 Motivation

In modern control systems engineering, the ability to test and validate control strategies under diverse and realistic conditions is paramount. Traditional controller testing methods often fail to replicate real-world scenarios, leading to discrepancies between the simulated and actual performance of the system under test. To bridge this gap, the integration of controller code into simulation environments has emerged as a crucial step, often referred to as "Software-In-the-Loop" (SIL) simulation (Schaub, Hellerer, and Bodenmüller 2012). By incorporating the controller code into SIL, the number of discrepancies between simulation results and field measurements can be reduced, improving the accuracy and reliability of simulation models (Ramasubramanian et al. 2024). However, in the field of power system simulation, this remains a challenging situation for multiple reasons. One of the difficulties faced is that of exchanging models between electromagnetic transient (EMT) simulation platforms and/or dynamic simulation tools (transient stability or phasor simulators). To a large extent, this is mainly due to the lack of a standardized equation-based modeling language for model exchange, leading to inconsistencies in simulation results between different tools. This inconsistency can result in speculation about the accuracy of the model or the adequacy of a simulation tool, highlighting the need for a more consistent model exchange mechanism (Rogersten, Vanfretti, and Li 2015).

Power system simulation tool vendors and users have started to explore the integration of 'real controller code' models (RCMs) into domain-specific simulation environments. They have established a joint effort within two professional organizations (CIGRE and IEEE [1]) to develop a domain-specific approach to perform such integration, known as the IEEE/CIGRE DLL Modeling Standard (ICDMS). Unfortunately, the proposed approach has only been adopted by a few power system simulation tool vendors, limiting the use of such RCMs to those tools. This adoption has been limited because the RCMs, according to the IEEE/CIGRE DLL Modeling Standard (ICDMS), are to be provided as DLLs (Dynamic Link Libraries) by equipment vendors. Hence, to support this standard within a simulation environment, a standard-specific interface needs to be called, and to run the models additional ancillary functions need to be developed.

To expand the potential use of such models beyond domain-specific power system tools and leverage the built-in features of the Modelica language for integrating external objects, this paper presents a novel method for incorporating precompiled C code to support the ICDMS

[1]According to https://www.electranix.com/ieee-pes-tass-realcodewg/ this is under the IEEE Task Force "Use of Real-Code in EMT Models for Power System Analysis" and according to https://tinyurl.com/ieee-cigre-dll-tor this is a Joint Task Force under CIGRE Study Committe B4, with Title: "Guidelines for Use of Real-Code in EMT Models for HVDC, FACTS and Inverter based generators in Power Systems Analysis".
CIGRE is the International Council on Large Electric Systems, which is a professional global non-profit in the field of high voltage.
The Institute of Electrical and Electronics Engineers (IEEE) is a professional association for electronics engineering, electrical engineering, and other related disciplines

domain-specific standard within the Modelica language and the OpenIPSL library.

1.2 Related Works

The modeling and simulation community has successfully developed interoperable standards, such as the Functional Mock-up Interface (FMI) (Junghanns et al. 2021) and the Functional Mock-up Interface for Embedded Systems (eFMI) (Lenord et al. 2021), which aim to streamline model exchange and integration in simulation environments. However, there are still significant challenges to achieve widespread adoption, especially in engineering areas where domain-specific approaches are the rule, which is the case for the electrical power industry (Vanfretti, Li, et al. 2013).

Meanwhile, within the power industry itself, previous efforts to standardize equipment models have not been successful due to their lack of adoption. One particular example is that of the generic software interface developed as part of the IEC 61400-27-1:2020 standard "Wind energy generation systems - Part 27-1: Electrical simulation models - Generic models" (see `https://webstore.iec.ch/publication/32564`), which intended to provide both generic models and an interface method for vendor-specific wind turbine models.

These grid standards have been unsuccessful as equipment manufacturers have been slow to adopt them and provide equipment models according to the standards, resulting in persistent difficulties in model exchange. Manufacturers are discouraged in adopting any of these standards due to the customers' preference for tool-specific implementations (e.g., PSCAD and PSS/E), which leads to tool lock-in. Although there have been efforts in Europe to develop the Common Grid Model Exchange Specification (see `https://tinyurl.com/cgmes2p5`); simulation tools built and used outside Europe have not yet adopted this standard. What this implies for user's that need functionalities not yet supported by domain-specific simulation tools, or that want to use Modelica-complaint simulation environments, is that the domain-specific approach has to be somehow supported within the Modelica ecosystem. This is what is attempted in this paper for the case of the IEEE/CIGRE DLL Modeling Standard (ICDMS).

In addition to implementing the ICDMS, means to simulate the reminder of the power grid in Modelica tools are required. Fortunately, an effort to port the behavioral model descriptions in Modelica replicating those of the PSS/E software (the simulation tool most used in the US and the Nordic countries) has been in place for almost a decade (T. Bogodorova et al. 2013; Vanfretti, Tetiana Bogodorova, and Baudette 2014; Zhang et al. 2015), which makes it possible to reproduce power system dynamic simulation results like those expected by industry practitioners. The OpenIPSL(de Castro et al. 2023) is a Modelica library that provides robust models and enhanced portability aimed at building an open-source software-based encyclopedia of dynamic power system models that can be exploited by multiple modeling tools that are compliant with the Modelica language specification. The OpenIPSL is used here to set up power grid simulation models in which the RCMs are included.

1.3 Contributions

This paper presents a method for integrating RCMs built according to the IEEE/CIGRE DLL Modeling Standard (ICDMS) into Modelica-based tools. This is achieved by linking precompiled C code to Modelica models and using components from the OpenIPSL library. The approach is demonstrated with an RCM of a simplified silicon-controlled rectifier excitation system (SCRX).

Our demonstration involves modifying and compiling the code of the SCRX RCM into Dynamic Link Libraries (DLLs), following the ICDMS. This standardization ensures compatibility with domain-specific standards and facilitates the seamless incorporation of controller code into Modelica simulations. The primary contribution of this paper is the detailed description of the process used to integrate the precompiled DLLs into the simulation environment, enabling extensive testing and validation of the controller code.

2 Background on the IEEE/CIGRE DLL Modeling Standard

To explain how the ICDMS functions, the simulation workflow shown in Figure 1 is used. It starts with "Allocate Memory", where memory for inputs, outputs, and parameters is allocated. Model Initialization then sets initial conditions and parameters. The Update Input step reads the current input values, followed by Run Calculation, where the model computes the output based on input values and parameters. Finally, Update Output writes the results to the output variables, completing one simulation cycle. This workflow repeats, allowing dynamic simulation of the controller's behavior.

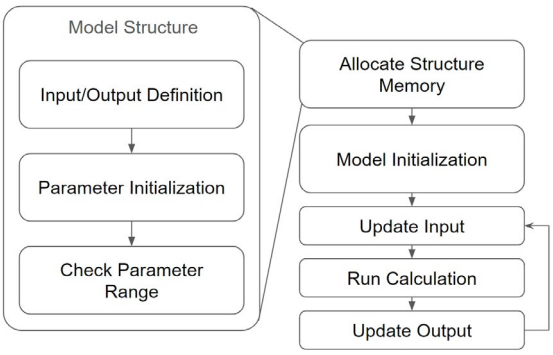

Figure 1. Model Structure and Simulation Workflow according to the IEEE/CIGRE DLL Modeling Standard

The implementation of the workflow in Figure 1 follows the ICDMS by defining a clear structure for input signals, output signals, and parameters using standardized data types and units. Standardized functions

such as *Model_GetInfo*, *Model_CheckParameters*, and *Model_Initialize* ensure proper initialization and parameter validation. The *Model_Outputs* function performs the main computational tasks, adhering to the fixed time step approach common in the real-world controller firmware. Each model to be developed using this approach needs to be compiled into a DLL, enabling its use across various simulation tools. However, this requires that the ICDMS be supported by the simulation tool.

3 Implementing the IEEE/CIGRE DLL Modeling Standard in Modelica

In the following section, the SCRX RCM is used as an example to illustrate the implementation of the interface between the C code and the Modelica language. The same methodology can be applied to other controller codes following the same ICDMS. It should be noted that the SCRX RCM is one of the examples used in the development of the ICDMS.

This example demonstrates the capability to interface a DDL of an RCM with a Modelica library and simulation tool. In most cases, controller manufacturers will not disclose their controller structure and may only provide parameter values, which would require a clean-room re-implementation similar to those in (Laera et al. 2022) to be used in a Modelica tool. However, if they follow the approach proposed in this paper, RCMs provided by manufacturers could be used to run simulations without the need of a complete re-implementation in Modelica.

3.1 External Object Integration

In ICDMS, structures store all the information about a controller including the simulation time step, the number of input/output, the parameter values and other information. To access a structure defined in `C`, we have to define a class in Modelica as shown in Listing 1. A `constructor` and `destructor` must be specified in a class to initialize and de-initialize an object from a class. This is essential for the computer to allocate and free the memory that stores the data of the structure.

Listing 1. SCRX Class Real-code Modelica Implementation.

```
class SCRX9_DLL
  extends ExternalObject;
  function constructor
    output SCRX9_DLL scrx9_dll;
    external "C" scrx9_dll = init_scrx_model()
        annotation (Library="SCRX9",
        LibraryDirectory="modelica://OpenIPSL/
        Resources/Library");
  end constructor;
  function destructor
    input SCRX9_DLL scrx9_dll;
    external "C" deinit_scrx_model(scrx9_dll)
        annotation (Library="SCRX9",
        LibraryDirectory="modelica://OpenIPSL/
        Resources/Library");
  end destructor;
end SCRX9_DLL;
```

The external C function `init_scrx_model` is called at line 5 of Listing 1 to allocate memory space. In `annotation`, the library name and directory have to be specified for the compiler to know where to look for the required functions.

Listing 2. `init_scrx_model` Function Implementation.

```
__declspec(dllexport) void* __cdecl
    init_scrx_model(void)
{
    IEEE_Cigre_DLLInterface_Instance* instance =
         (IEEE_Cigre_DLLInterface_Instance*)
        malloc(sizeof(
        IEEE_Cigre_DLLInterface_Instance));
    ...
    /*PARAMETER INITIALIZATION*/
    ...
    double * states = malloc(6 * sizeof (double)
        );
    instance->DoubleStates = states;
    Model_Initialize(instance);

    return (void *) instance;
}
```

The C functions shown in Listing 2 initialize all the parameters (Line 5) of the instance and allocate memory space (Line 7) to save key state values, when the constructor is called. Since most controller consists of integrators that require memory, line 7 allocates memory space to store the states of the integrators. Line 11 returns the address of the instance to access this initialized instance later in Modelica functions. From line 4 of Listing 1, the returned address is returned again by the constructor as an external object of class `SCRX9_DLL`.

Having initialized and allocated memory, the model needs to be accessed and integrated to a power system model. As an excitation control system, the example model features two primary inputs: ETERM, representing the generator's terminal voltage, and XADIFD, representing the field current, both initialized to steady-state values to avoid initialization problems. The EFD output is the generated field voltage. The object `scrx9_struct` wraps the states and parameters of the SCRX controller initialized in Line 3 of the Listing 2. The algorithm section updates the controller's state from the input port using the `update` function shown in Listing 4. The resulting field voltage is obtained through `model_output` function defined in Listing 5. In addition, the function `update_scrx_input` shown in Listing 4 reads the values from the input ports in Modelica and updates them in the defined C instance.

Listing 3. SCRX Controller Modelica Model.

```
model SCRX
  Modelica.Blocks.Interfaces.RealInput ETERM(
      start = 1);
  Modelica.Blocks.Interfaces.RealInput XADIFD(
      start = 1.325);
  Modelica.Blocks.Interfaces.RealOutput EFD;
```

 10.3384/ECP20772 DOI

```
  SCRX9_DLL scrx9_struct = SCRX9_DLL();
algorithm
  Functions.update(scrx9_struct,time,1,ETERM,0,
      XADIFD,ETERM,0,0);
  EFD:=Functions.model_output(scrx9_struct);
  when terminal() then
    Functions.save_ss_state(scrx9_struct);
  end when;
end SCRX;
```

Listing 4. `update` Function Implementation.

```
__declspec(dllexport) void __cdecl
    update_scrx_input(
    IEEE_Cigre_DLLInterface_Instance* instance,
    double sim_time_input, double vref,double ec
    ,
    double vs,double ifd,double vt,double vuel,
        double voel) {
    MyModelInputs* inputs = (MyModelInputs*)
        instance->ExternalInputs;
    inputs->IFD = ifd; // Field current
    inputs->VT = vt; // Terminal voltage
    ... // Other inputs
    sim_time = sim_time_input;
};
```

Listing 5. `model_output` Function Implementation.

```
__declspec(dllexport) double __cdecl
    model_calculate(
    IEEE_Cigre_DLLInterface_Instance* instance)
    {
    MyModelOutputs* outputs = (MyModelOutputs*)
        instance->ExternalOutputs;
    if (sim_time != pre_sim_time)
    {
        Model_Outputs(instance);
        pre_sim_time = sim_time;
    }
    return outputs->EFD;
};
```

At each time step of the simulation, the program will call `model_calculate` shown in Listing 5 to calculate the output with `Model_Outputs`. The calculation result will return to Modelica as a floating number or a list of floating numbers depend on the type of the controller (multiple input single output or multiple input multiple output).

3.2 Initialization of the External Object

Initializing the RCM requires us to ensure that the simulation starts from a valid equilibrium point. Consequently, this requires sending data to the external object and linking its output to the rest of the system model. In the case of the SCRX RCM, this means passing the measured voltage from the bus bar to the excitation system and returning the field voltage value at the equilibrium condition.

To this end, the C function shown in Listing 6 is called at the termination of each simulation (see Line 10 of Listing 3) to extract the current values of the controller's inputs, outputs, and state variables, storing them in an array for writing to a binary file. For the SCRX controller, the inputs include signals such as `VRef` (reference voltage), `Ec` (measured voltage), `VOEL` (over excitation limit), and others. The output, `EFD`, represents the generated field voltage. Furthermore, the state variables, stored in the `DoubleStates` array (see Line 16) within the instance, are also included. The function opens the file in binary write mode, populates the array with the extracted values, and writes the entire array to the file (see Line 18). This process ensures that all critical data required by the controller are preserved, enabling the analysis and potential reinitialization of the system at the desired state in future simulations.

Listing 6. `save_ss_state` Function Implementation.

```
__declspec(dllexport) void __cdecl
    save_states(
    IEEE_Cigre_DLLInterface_Instance*
    instance)
{
    MyModelOutputs* outputs = (MyModelOutputs
        *)instance->ExternalOutputs;
    MyModelInputs* inputs = (MyModelInputs*)
        instance->ExternalInputs;
    int listSize = 7+1+6; %input+output+
        states
    double list[listSize];
    FILE* file = fopen("list.dat", "wb");
    if (file != NULL)
    {
        list[0] = inputs->VRef;
        list[1] = inputs->Ec;
        .../*More Input states*/
        list[6] = inputs->VOEL;
        list[7] = outputs->EFD;
        for (int i = 0; i < 6; i++){
            list[8+i] = instance->
                DoubleStates[i];
        }
        fwrite(list, sizeof(double), listSize
            , file);
        fclose(file);
    }
}
```

3.3 Illustration with the SCRX Excitation Model

The SCRX excitation model is a simplified control system designed to regulate the field voltage of a synchronous generator, thereby maintaining the machine's AC voltage at a specified reference set-point. This section introduces the excitation controller and illustrates its block diagram and overall system structure shown in Fig.2.

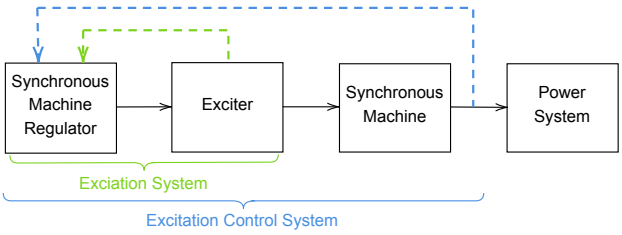

Figure 2. Synchronous Machine Control System(IEEE 2007).

The *Synchronous Machine Regulator* generates control

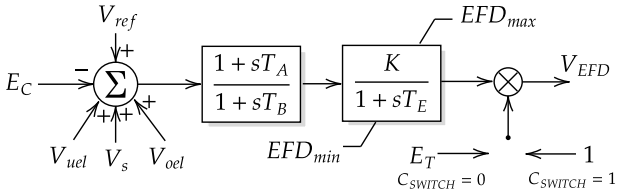

Figure 3. Excitation Control System ("IEEE Standard Definitions for Excitation Systems for Synchronous Machines" 2022).

signals based on the reference voltage (V_REF) and feedback signals from the synchronous machine. It takes these inputs to produce an appropriate control command for the exciter. The *Exciter* modulates the field voltage of the synchronous machine in response to the control commands from the regulator. It serves as an intermediary that translates the regulator's signals into field winding voltage adjustments. The *Synchronous Machine* is the plant of this system that converts rotational mechanical energy into electrical power. It responds to the field voltage adjustments made by the exciter and influences the voltage and stability of the broader power system. The *Power System* represents the electrical grid of which the synchronous machine is part. The primary goal of the excitation system is to maintain the desired voltage levels at the generator terminals.

The SCRX excitation model shown in Figure 3 presents a detailed block diagram and standardized modeling approach that is generally adopted to represent how the control of the field voltage of the synchronous generators is achieved ("IEEE Standard Definitions for Excitation Systems for Synchronous Machines" 2022). The integration of this model into power system simulations allows for extensive testing and validation, ensuring optimal performance under various operational conditions.

Table 1 lists the parameters and the default values required by the SCRX controller, including time constants (TAdTB, TB, TE), controller gain (K), and voltage limits (EMin, EMax), as well as the power source selection switch (CSwitch) and the field resistance ratio (RCdRFD). These parameters are essential for configuring the controller to operate within the desired specifications and to ensure compatibility with the ICDMS.

Table 2 lists the input signals such as the reference voltage (VRef), measured voltage (Ec), stabilizer signal (Vs), field current (IFD), terminal voltage (VT), and excitation limits (VUEL and VOEL), which are used to dynamically adjust the controller performance during simulation. Table 3 defines the output signal (EFD), representing the output machine field voltage.

These tables illustrate the format typically used to define the models of excitation control systems. Note that the ICDMS adopts this formating to specify the parameter, input, and output specifications of all RCMs. This would allow us to use the RCMs in any simulation environment adhering to the ICDMS.

Table 1. SCRX Parameters.

Parameters	*Description*	*Default*
TAdTB	Time Constant	0.1
TB	Time Constant	10
K	Controller Gain	100
TE	Time Constant	0.05
EMin	Min Field Voltage	-10
EMax	Max Field Voltage	10
CSwitch	Power Source Select	1
RCdRFD	Field resistance ratio	10

Table 2. SCRX Input Signals.

Vref	Reference voltage
Ec	Measured voltage
Vs	Stabilizer signal
IFD	Field Current
VT	Terminal Votlage
VUEL	Under Excitation Limit
VOEL	Over Excitation Limit

4 Results

4.1 Testing Power Network Model

The power network model that incorporates the SCRX model is constructed using the OpenIPSL and is shown in Fig.4. This power system model provides a platform for testing both RCM and standard OpenIPSL built-in SCRX9 example controllers.

The power network consists of a synchronous generator connected to an infinite bus through transmission lines, buses, and including a load. The generator is controlled by an SCRX excitation system, which regulates the field voltage (EFD) to maintain the desired power output. A short fault was applied between Bus2 and Bus3 starting at 2 seconds and stopping at 2.15 seconds. This network allows for comprehensive testing and validation of the SCRX controller integrated as an RCM in DLL form, following the ICDMS. By simulating a short fault (i.e., a large disturbance), the power network response can be

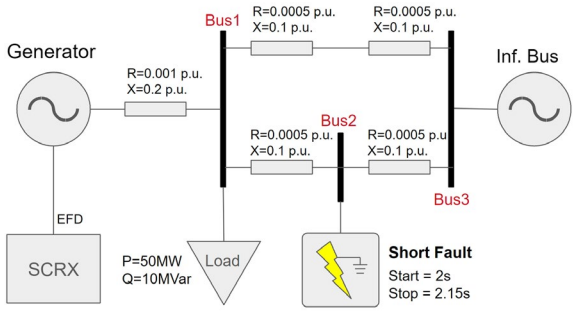

Figure 4. SCRX Controller within a Testing Power Network.

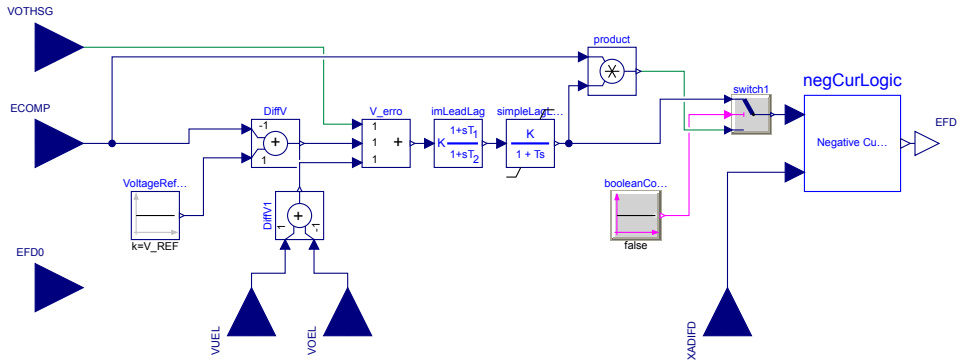

Figure 5. SCRX Controller in OpenIPSL.

Table 3. SCRX Output Signal.

EFD	Output Signal Voltage

used to evaluate the performance of the RCM in handling dynamic events.

4.2 Original SCRX Controller Simulation Result

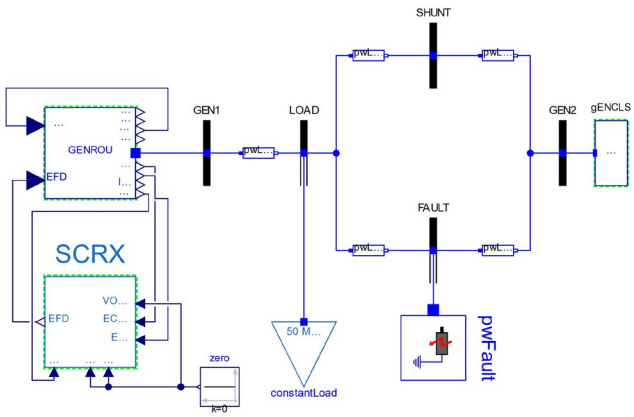

Figure 6. SCRX9 Simulation Example in OpenIPSL.

OpenIPSL contains a model of the SCRX excitation control system, which is shown in Figure 5. In this model, the SCRX is implemented with traditional lead-lag and phase-lag compensators, similar to what is specified in ("IEEE Standard Definitions for Excitation Systems for Synchronous Machines" 2022) and shown in Figure 2. The default parameters in Table 1 are used in this model to compare with the model implemented using the external DLL library. Meanwhile, the grid network is built with OpenIPSL and the generator is controlled by the SCRX RCM, as shown in Figure 6.

Simulating the model in Figure 6 yields the results shown in Figure 7, where two subplots: 1. Generator Voltage (p.u.); 2. SCRX Field Voltage (EFD, Volts). Before the fault occurs, the generator voltage is stable at the reference value of 1.0 p.u., and the field voltage (EFD) is maintained in steady state by the SCRX. When the fault

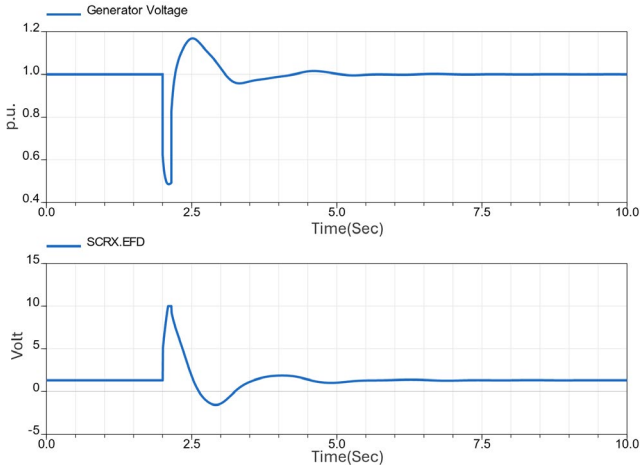

Figure 7. Simulation Result: Bus1 Voltage (Top); SCRX Output Voltage (Bottom).

occurs at 2.0 seconds, there is a significant drop in the generator voltage to approximately 0.4 p.u. The SCRX controller responds by sharply increasing the field voltage to counteract the voltage dip and stabilize the generator. The peak field voltage reaches around 10 Volts (EMAX) shortly after the fault initiation. Once the fault is cleared at 2.15 seconds, the generator voltage initially overshoots about 0.2 p.u. before settling back to the reference value. The SCRX controller adjusts the field voltage accordingly, first reducing it to correct the overshoot and then gradually stabilizing it around the required level to maintain the generator voltage at 1.0 p.u. The performance metrics observed in this simulation can be used as a reference for further testing and comparison with the RCM.

4.3 External Object SCRX Controller Simulation Result

Next, we compare the implementation of the ICDMS using the 'real code' implementation of the SCRX model.

Figure 8 shows the simulation diagram with an excitation controller of the generator replaced with the 'real-code' implementation. The original SCRX controller has 6 inputs. However, 4 of them remain zero during the simulation. Thus, for simplicity of the block, only two feedback ports are preserved, and the input voltage set point is

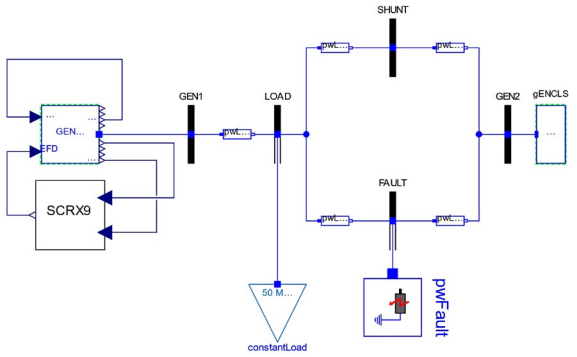

Figure 8. SCRX Controller With Real-Code Implementation.

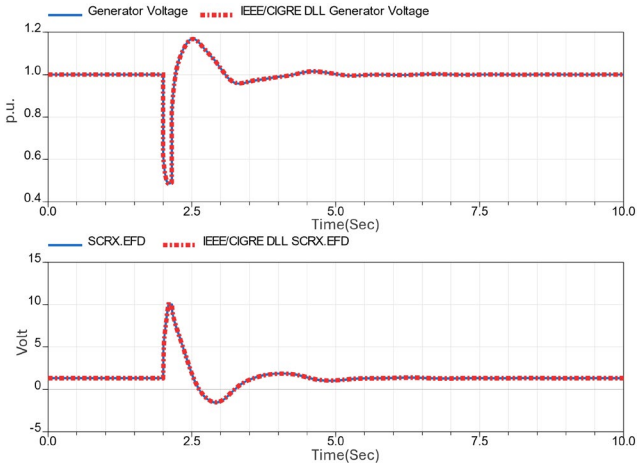

Figure 9. Simulation Result: Bus1 Voltage (Top); SCRX Output Voltage (Bottom).

always set to 1 p.u.

The comparison results shown in Figure 9 indicate that the real code SCRX controller, implemented as a DLL following the ICDMS , performs similarly to the original Modelica-based model from OpenIPSL. This successful integration and matching performance validate the RCM compatibility and robustness within the Modelica simulation environment. Although the comparison of simulation performance (e.g., time required to simulate) was part of our experimental analysis, we observed significant variability in simulation times between different runs. This variability led us to conclude that the operating system's task scheduling had a substantial impact on the simulation time. As a result, we were unable to provide a consistent and meaningful comparison of simulation times between the two approaches.

5 Conclusions and Future Work

By achieving consistent behavior across different implementations, this study confirms that the IEEE/CIGRE DLL Modeling Standard (IDMS) can be implemented in Modelica to support RCMs. These models can be seamlessly integrated with Modelica models of power system components, as shown using the OpenIPSL library, and tested in simulation scenarios. This offers the possibility of performing power system simulations without the need for domain-specific tools, which is valuable for practitioners and researchers who need to develop models that comply with the ICDMS. These results support the broader use of RCMs in power system simulations with Modelica, enhancing the flexibility and reliability of power system simulations and control systems for industrial applications.

The implementation has been tested using the Dymola software. Future work involves releasing the developed code to implement the ICDMS, integrating the examples in this paper into the OpenIPSL library, and conducting tests with OpenModelica.

Although the prototype implementation approach used herein requires one to create treat each RCM individually and, therefore, providing interfacing functions and a Modelica model for each RCM, this process can be automated by developing generic Modelica functions that extract and pass information to a generic DLL. This will be explored in future work.

In addition, future work includes the development of unit testing to assess the performance of the integrated DLLs and determine if additional error handling functions would be required to protect against unexpected DLL numerical errors or other unwanted simulation behavior.

Finally, the authors will explore the potential wrapping of RCMs with FMI and compare the benefits and drawbacks with the approach proposed herein.

Acknowledgements

This paper is in part, based upon work supported by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the Advanced Manufacturing Office, Award Number DE-EE0009139.

References

Bogodorova, T. et al. (2013). "A modelica power system library for phasor time-domain simulation". In: *IEEE PES ISGT Europe 2013*, pp. 1–5. DOI: 10.1109/ISGTEurope.2013.6695422.

de Castro, Marcelo et al. (2023). "Version [OpenIPSL 2.0.0] - [iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations]". In: *SoftwareX* 21, p. 101277. ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2022.101277. URL: https://www.sciencedirect.com/science/article/pii/S2352711022001959.

IEEE (2007). "IEEE Standard Definitions for Excitation Systems for Synchronous Machines". In: *IEEE Std 421.1-2007 (Revision of IEEE Std 421.1-1986)*, pp. 1–33. DOI: 10.1109/IEEESTD.2007.385319.

"IEEE Standard Definitions for Excitation Systems for Synchronous Machines" (2022). In: *IEEE Std 421.1-2021 (Revision of IEEE Std 421.1-2007)*, pp. 1–45. DOI: 10.1109/IEEESTD.2022.9737077.

Junghanns, Andreas et al. (2021-09-27). "The Functional Mockup Interface 3.0 - New Features Enabling New Applications". In: 14th Modelica Conference 2021. Linköping University

Electronic Press. DOI: 10.3384/ecp2118117. URL: http://dx.doi.org/10.3384/ecp2118117.

Laera, Giuseppe et al. (2022-10). "Guidelines and Use Cases for Power Systems Dynamic Modeling and Model Verification using Modelica and OpenIPSL". In: Proceedings of the American Modelica Conference 2022. Linköping University Electronic Press. DOI: 10.3384/ECP21186146.

Lenord, Oliver et al. (2021-09-27). "eFMI: An open standard for physical models in embedded software". In: 14th Modelica Conference 2021. Linköping University Electronic Press. DOI: 10.3384/ecp2118157. URL: http://dx.doi.org/10.3384/ecp2118157.

Ramasubramanian, Deepak et al. (2024). "Techniques and Methods for Validation of Inverter-Based Resource Unit and Plant Simulation Models Across Multiple Simulation Domains: An Engineering Judgment-Based Approach". In: *IEEE Power and Energy Magazine* 22.2, pp. 55–65. DOI: 10.1109/MPE.2023.3343679.

Rogersten, Robert, Luigi Vanfretti, and Wei Li (2015). "Towards consistent model exchange and simulation of VSC-HVdc controls for EMT studies". In: *2015 IEEE Power & Energy Society General Meeting*, pp. 1–5. DOI: 10.1109/PESGM.2015.7285986.

Schaub, Alexander, Matthias Hellerer, and Tim Bodenmüller (2012-09). "Simulation of Artificial Intelligence Agents using Modelica and the DLR Visualization Library". In: 9th International Modelica Conference. Linköping University Electronic Press. DOI: 10.3384/ecp12076339. URL: http://dx.doi.org/10.3384/ecp12076339.

Vanfretti, Luigi, Tetiana Bogodorova, and Maxime Baudette (2014-03-10). "A Modelica Power System Component Library for Model Validation and Parameter Identification". In: 10th International Modelica Conference. Linköping University Electronic Press. DOI: 10.3384/ecp140961195. URL: http://dx.doi.org/10.3384/ecp140961195.

Vanfretti, Luigi, Wei Li, et al. (2013-01). "Unambiguous power system dynamic modeling and simulation using modelica tools". In: pp. 1–5. DOI: 10.1109/PESMG.2013.6672476.

Zhang, Mengjia et al. (2015-10). "Modelica Implementation and Software-to-Software Validation of Power System Component Models Commonly used by Nordic TSOs for Dynamic Simulations". In: 56th Conference on Simulation and Modelling (SIMS 56). Linköping University Electronic Press. DOI: 10.3384/ecp15119105. URL: http://dx.doi.org/10.3384/ecp15119105.