Real-Code Models for Electromagnetic Transient Simulations: INTEROPERABILITY WITH MODELICA

Luigi Vanfretti and Hao Chang
ALSETLab, Dept. of Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY, USA

American Modelica & FMI Conference 2026 — Industrial Presentations 2
Georgia Institute of Technology, Aerospace Systems Design Laboratory, Atlanta, GA, USA
October 14, 2026

LUIGI VANFRETTI
Professor, ECSE Dept., RPI
vanfrl@rpi.edu
https://alsetlab.github.io/

Context: phasor vs. EMT simulation, and the status quo of EMT tools

Phasor (RMS, positive-sequence) simulation — OpenIPSL

- **Fundamental-frequency phasors:**
  → algebraic network equations,
  → time steps of milliseconds;
  → transient (rotor-angle, voltage, frequency) stability studies

- **Our other three OpenIPSL presentations at this conference** are in this domain (the PSS/E-like phasor models of OpenIPSL)

Electromagnetic transient (EMT) simulation — this presentation

- **Instantaneous three-phase waveforms:**
  → differential network equations, time steps of microseconds;
  → fast, control-dependent converter dynamics
- **Converter-dominated grids** (Inverter-Based Resources (IBRs), Distributed Energy Resources (DERs) → need EMT fidelity to assess interactions, controller performance and system-level coordination

Status quo of EMT tools: closed-source, proprietary

- **PSCAD** holds almost a monopoly in the US
- **EMTP** is trusted in Quebec, France, other places in Europe and Latin America
- Both are proprietary *domain-specific* monolithic simulators (recall our previous presentation on PowerFactory & FMI integration of OpenIPSL/Modelica models)
- **Vendor controller models** are tied to these tools: → tool lock-in, no validated open-source alternative

What this work tries to do!

- **A step towards a validated, open-source Modelica alternative** to closed-source EMT software
- **"Piggy-back" on one of the *TIMELIEST* topics in the power grid EMT community:**
  using the so-called (incorrectly) "real-code" models to represent power electronic control loops
- **Demonstrate the way forward using open-access, interoperable standard** (Modelica, FMI) vs. closed-source proprietary solutions

Our proposed alternative

Open-access interoperable standards (Modelica, FMI, eFMI) around vendor "real-code" controllers, instead of closed-source proprietary EMT tools

Motivation: why portable "real-code" EMT models, and why in Modelica?

Status quo: one DLL per tool, non-standard interfaces

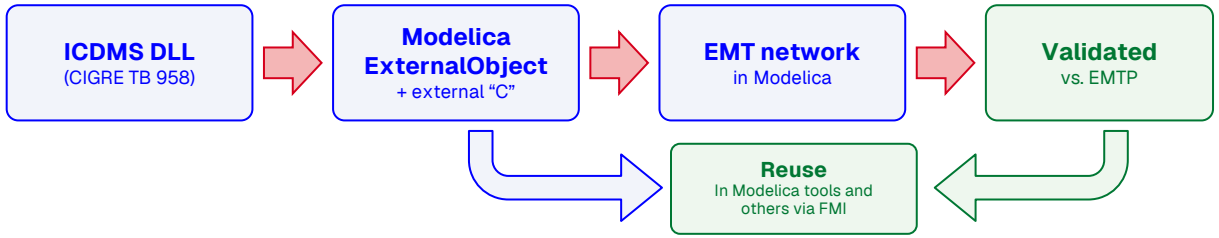
- **Converter-dominated grids need EMT fidelity:**
  - IBRs, DERs and VPPs bring fast, control-dependent dynamics
- **Vendors ship controllers as compiled DLLs:**
  - Vendor perspective: "protects" IP and preserves fidelity —
  - Assumption (incorrect or partially-correct only) of "real code" models → contain the same algorithms as the field controller
- **But each DLL targets a specific tool** → "DLL Hell":
  - high integration effort and limited portability,
  - limited model exchange across utilities, OEMs, consultants and researchers
- **Recent effort → IEEE/CIGRE TB 958 (ICDMS)** defines a common interface — but so far only a few commercial EMT tools implement it, all of them proprietary closed-source. Still requires to re-compile source at each tool…



✗ NO REUSE ACROSS TOOLS — **each tool needs its own DLL and interface**

This work: one ICDMS DLL, executed natively in Modelica

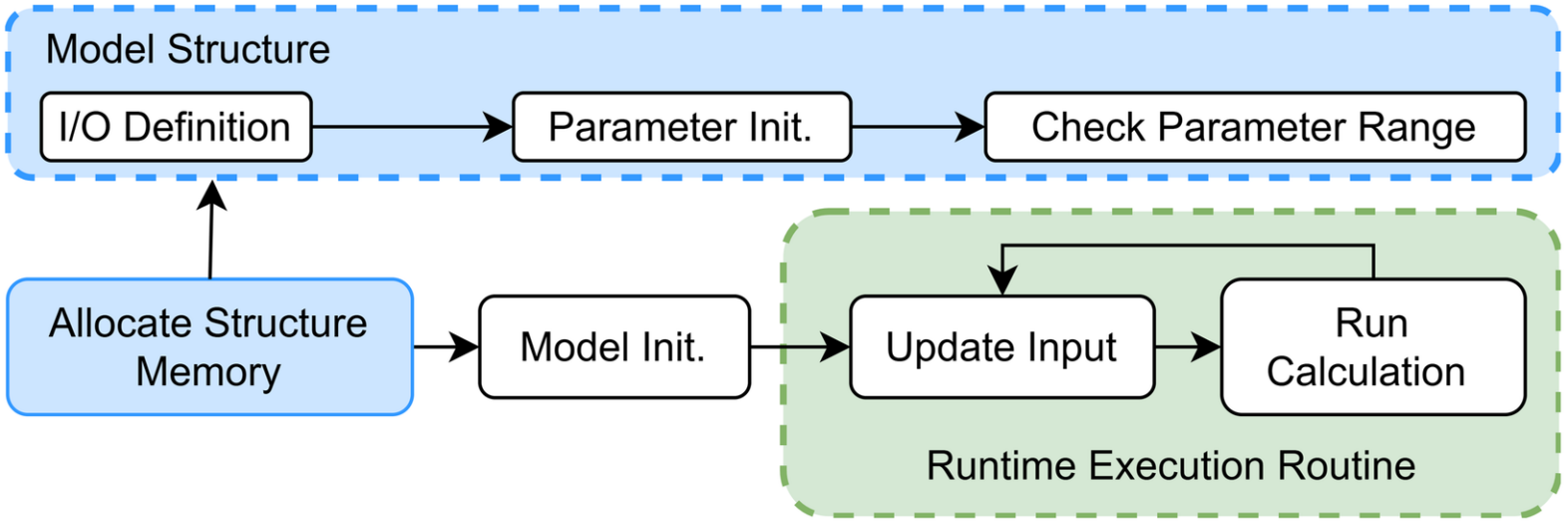
- **Modelica ExternalObject** + external "C" interface wrap the DLL: no controller code is re-implemented
- **EMT network model in Modelica** (Modelica Standard Library components) around the DLL controller
- **Validated** against the same DLL running in EMTP
- **Standards-based way out of the lock-in:** any Modelica-compliant tool can instantiate the object without modifying the DLL



Question:

- **Can ICDMS-compliant DLLs run natively in the open-access, standardized Modelica language for EMT studies of converter-rich systems?**

The IEEE/CIGRE DLL interface (ICDMS, CIGRE TB 958): what Modelica users should know



ICDMS model structure (I/O definition, parameter init., range check) and runtime execution routine (allocate memory, model init., update input, run calculation loop) — paper Fig. 1

What is the ICDMS?

- **A common set of C-style function calls** — initialization, step execution, memory de-allocation — so "real code" controllers compile into DLLs that are portable across EMT tools
- **Shared memory buffers** for inputs, outputs and states; the DLL behaves as a black-box unit; in principle the same code that runs on the field controller can be tested across multiple EMT tools

Why it matters — and its limits

- **OEMs would share some form of their control algorithms without exposing source code:** IP protection with model fidelity; not an official standard yet, but adopted by some EMT tools and OEMs
- **Use has stayed within a few commercial EMT tools** → limited accessibility for broader modeling environments such as Modelica

Our contribution: promoting the use of Modelica Association Standards for "real-code" with proof-of-concept demonstrations

ALSETlab

AMC 2024 — phasor domain

- **Positive-sequence** excitation system DLL built from TB 958 Appendix E reference C code
- **Compiled into a DLL following the ICDMS,**
  → wrapped as an `ExternalObject` in an OpenIPSL network
  → performs similarly to the original OpenIPSL `SCRX9` model (next slide)

CIGRE Paris 2026 — EMT

- **SC C4, paper 10724:** grid-following VSC with its ICDMS controller DLL in a Modelica EMT model
- **Cross-tool validation** against EMTP: ramp, fault, voltage step, frequency step
- **Core wrappers** contributed to OpenIPSL (PR #362)

PMAPS 2026 — why it matters

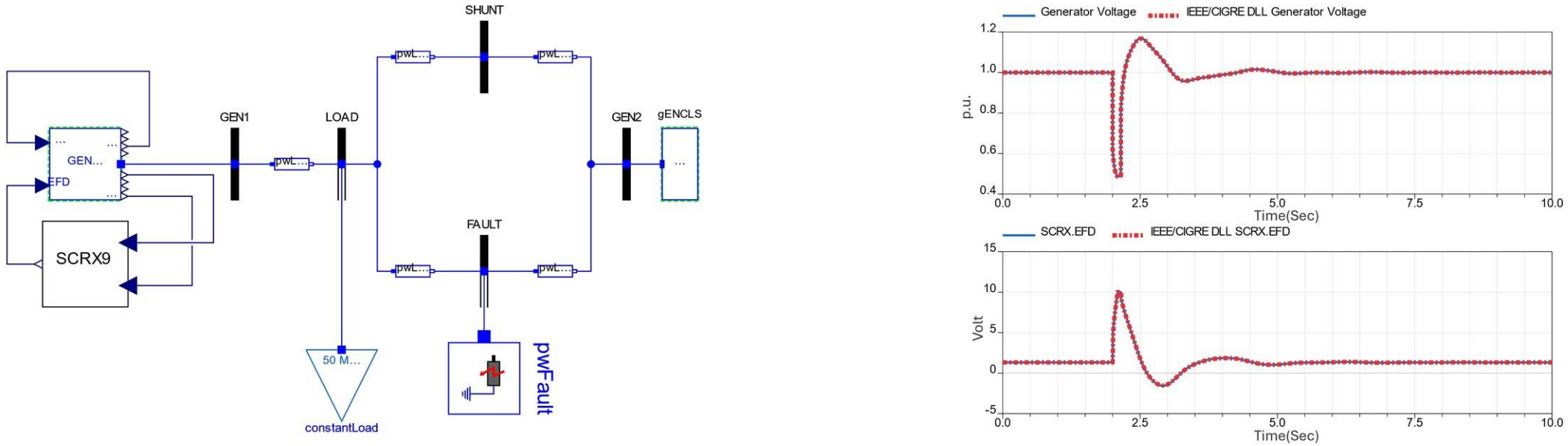
- **Portable, vendor-validated controllers** are the practical bottleneck for many-scenario / Monte-Carlo EMT campaigns
- **Stability, risk and reliability** assessment of converter-dominated systems depends on such models

AMFC 2026 Today

What this presentation covers

- **The `ExternalObject` bridge:** how an ICDMS DLL becomes a Modelica object — constructor, destructor, I/O every step
- **The Modelica implementation:** the EMT model of the grid-following VSC and the grid components (Δ–Y transformer, three-phase fault) built from MSL
- **The EMT difficulty:** the ICDMS DLL expects fixed-step execution; what that costs, and what the results tell us for future work
- **Cross-tool results** and where the residual differences come from
- **What practitioners gain:** the Modelica toolchain — FMI export into commercial phasor tools, eFMI for CHiL — around vendor-validated controllers
- *Luigi's Goal:* open access interoperable standards from the Modelica Association used instead of "oligopolistic"-pseudo-standards of the IEEE, IEC and CIGRE → bring competition and support innovation in simulation in the power grid industry.

AMC 2024 recap: SCRX excitation “real-code” DLL in an OpenIPSL phasor simulation



Left: SCRX controller with real-code implementation in the OpenIPSL test network (AMC 2024, Fig. 8). Right: Bus1 voltage and SCRX output voltage, OpenIPSL SCRX9 vs. IEEE/CIGRE DLL (Fig. 9)

Integrating the IEEE/CIGRE DLL Modeling Standard to use “real code” models for power system analysis in Modelica tools (Chang & Vanfretti, AMC 2024)

- **SCRX real-code model (RCM)** — one of the examples used in the development of the ICDMS — compiled into a DLL following the ICDMS and linked to Modelica as an ExternalObject (`SCRX9_DLL`, annotation `Library="SCRX9"`)
- **Test network in OpenIPSL:** synchronous generator connected to an infinite bus through transmission lines, buses and a load; short fault between Bus2 and Bus3 from 2 s to 2.15 s
- **Result:** the real-code SCRX controller, implemented as a DLL following the ICDMS, performs similarly to the original Modelica-based model from OpenIPSL; implementation tested using Dymola

The ExternalObject bridge: the ICDMS DLL as a Modelica class

The pattern reduces to two functions (Listing 1)

- **Lines 1–2 — define the ExternalObject class:** `GFM_DLL` extends `ExternalObject` and wraps the ICDMS C data structures; a constructor and a destructor manage its lifetime
- **Lines 3–14 — construct = initialize the DLL:** the `external "C"` call on line 13 runs `init_gfm_model()` (next slide) and returns the pointer held in `gfm_dll` (line 12) for the whole simulation
- **Lines 4–11 — physical parameters are constructor inputs:** `c_filt, r_filt, l_filt, r_l, Vbase, Sbase, VDC_base, w_type` stay visible and tunable from the Modelica environment — the controller logic stays inside the vendor DLL
- **Lines 16–19 — destruct = free memory:** `deinit_gfm_model()` on line 18 releases the DLL instance when the simulation ends
- **Every time step:** network measurements go in, controller commands come out — at the fixed step the ICDMS DLL expects

Any Modelica-compliant tool can instantiate and simulate the object without modifying the DLL!

→ No need to recompile it for each individual tool.

```
1  class GFM_DLL "TB 958 real-code controller DLL"
2    extends ExternalObject;
3    function constructor
4      input Real c_filt;
5      input Real r_filt;
6      input Real l_filt;
7      input Real r_l;
8      input Real Vbase;
9      input Real Sbase;
10     input Real VDC_base;
11     input Real w_type;
12     output GFM_DLL gfm_dll;
13     external "C" gfm_dll = init_gfm_model(...)
14   end constructor;
15
16   function destructor
17     input GFM_DLL gfm_dll;
18     external "C" deinit_gfm_model(...)
19   end destructor;
20 end GFM_DLL;
```

Listing 1 – GFM_DLL ExternalObject class (Modelica); the name is inherited from the DLL, which supports grid-forming and grid-following modes

What the constructor calls: init_gfm_model() in C

Initialization handled on the C side (Listing 2)

- **Invoked during construction** of `GFM_DLL`: when a Modelica simulation starts, the constructor (Listing 1, line 13) automatically calls `init_gfm_model` (line 1)
- **Lines 3–6 — allocate dynamic memory** for the ICDMS instance and three key structures: `Inputs`, `Outputs`, `Param` — the I/O interface and tuneable settings of the controller
- **Line 7 — parameter initialization** (values set in the instance; abridged in the listing)
- **Lines 8–10 — link the structures into the ICDMS instance; line 11** calls `Model_Initialize` to complete the internal setup
- **Line 12 — return the pointer** to the initialized instance, which Modelica maintains throughout the simulation via the `ExternalObject`

Result

- **The DLL is fully prepared before the first time step**
  - no controller code is re-implemented in Modelica
- **Integration is at the standardized interface level**
  - agnostic to the controller internals and portable across Modelica tools

```
 1  void* init_gfm_model(...)
 2  {
 3    ICDMS*   instance = (ICDMS*)
malloc(sizeof(ICDMS));
 4    Inputs*  inputs   = (Inputs*)
malloc(sizeof(Inputs));
 5    Outputs* outputs  =
(Outputs*)malloc(sizeof(Outputs));
 6    Param*   param    = (Param*)
malloc(sizeof(Params));
 7    //***PARAMETER INITIALIZATION****//
 8    instance->Inputs  = inputs;
 9    instance->Outputs = outputs;
10    instance->Param   = param;
11    Model_Initialize(instance);
12    return (void*)instance;
13  }
```

Listing 2 – Initialization function called by the constructor (C)

Test system: grid-following VSC EMT model connected to the grid

System under study

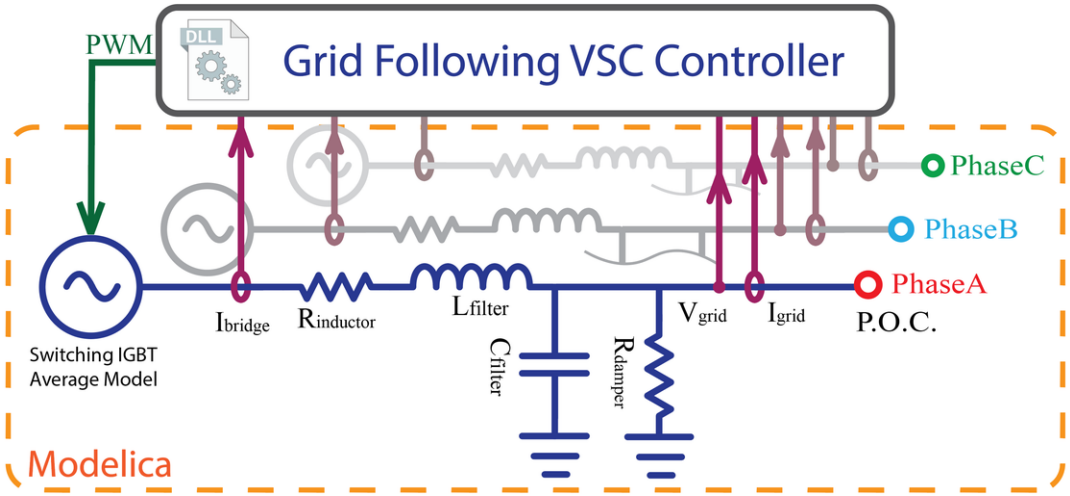
- **Grid-following VSC** connected to:
  - an infinite bus through an LC filter and
  - a Δ–Y transformer;
  - DC link = modeled as a constant DC voltage source
- **Averaged-value power stage:** the bridge is a controlled voltage source following the controller's modulation reference (no device-level IGBT switching or PWM carrier harmonics)
- **Controller:** compiled ICDMS DLL — a black box for the network

Exchanged with the DLL every time step

- **Voltages and currents** at the converter output, across the filter and at the point of connection (POC)

Network model

- **Built from MSL components →** a second-order low-pass filter at the transformer secondary improves numerical conditioning
- **Why an averaged-value model?** Sufficient and appropriate to validate the integrated controller against the reference tool; switching-level modeling with MSL switches is feasible but out of scope



Grid-following VSC EMT model: averaged bridge, LC filter with damper, POC measurements feeding the DLL controller (paper Fig. 3)

Model and parameters online

- H. Chang and L. Vanfretti, "EMT CIGRE-IEEE DLL Modelling Interface Testing Environment Configuration," 2025, DOI: 10.13140/RG.2.2.21778.77764
- The grid-following VSC controller DLL used in this work was developed by Deepak Ramasubramanian, EPRI

VSC controller executed from the real-code DLL: what Modelica sees

Controller DLL

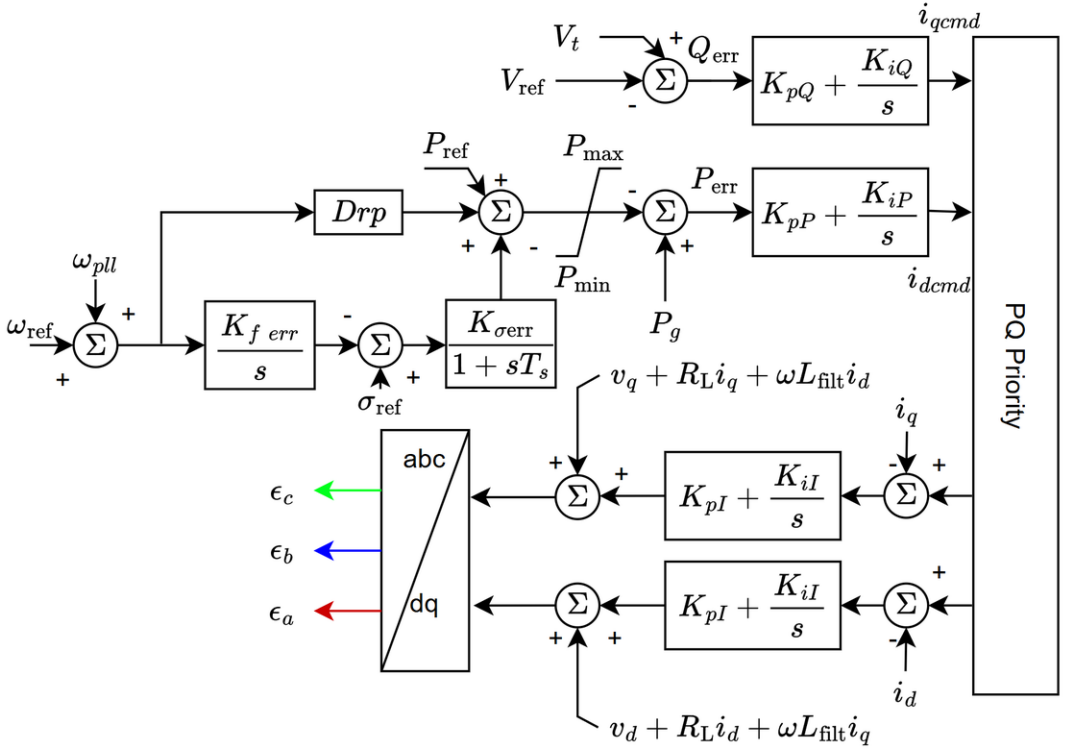
- **Developed by Deepak Ramasubramanian (EPRI):** a flexible inverter controller with grid-forming (PLL-based) or grid-following modes — here the grid-following droop scheme
- **Decentralized primary control:** frequency and voltage respond to P and Q mismatches, like a governor and an AVR do in a conventional synchronous generator

Cascaded loop structure

- **Outer droop loops** → frequency and voltage-magnitude references
- **Intermediate voltage / power PI loops** → internal inverter voltage and angle
- **Fast inner PI current loop** → tracks d- and q-axis current references; PQ-priority current limiting

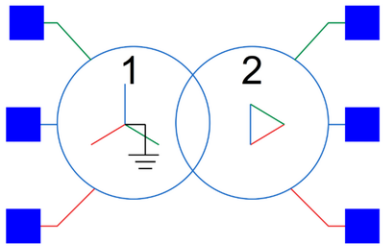
What the Modelica sees

- **Only the ICDMS inputs and outputs** — the control logic stays inside the vendor DLL

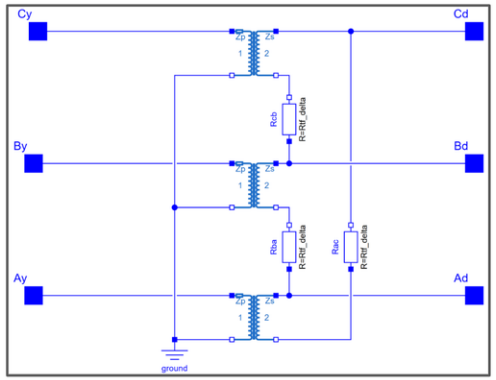


VSC controller block diagram: droop, voltage/power PI loops, PQ-priority limiter, inner dq current loops (paper Fig. 4)

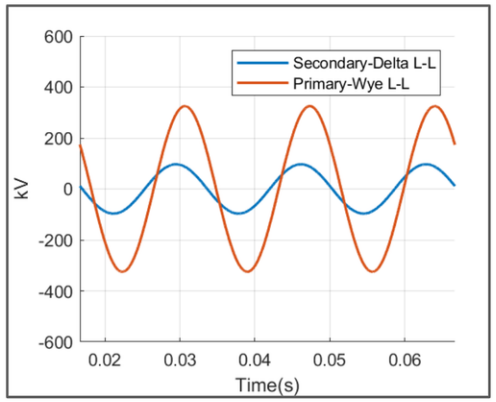
Modelica implementation (1/2): Δ–Y transformer from MSL



(a): Icon



(b): Subcircuit



(c): Simple Test

(a) icon, (b) wye–delta subcircuit, (c) simple test: primary wye and secondary delta line-to-line voltages (CIGRE paper Fig. 5)

All grid components from the Modelica Standard Library (MSL)

- AC voltage source, Δ–Y transformer and fault models constructed using standardized electrical models available in the MSL

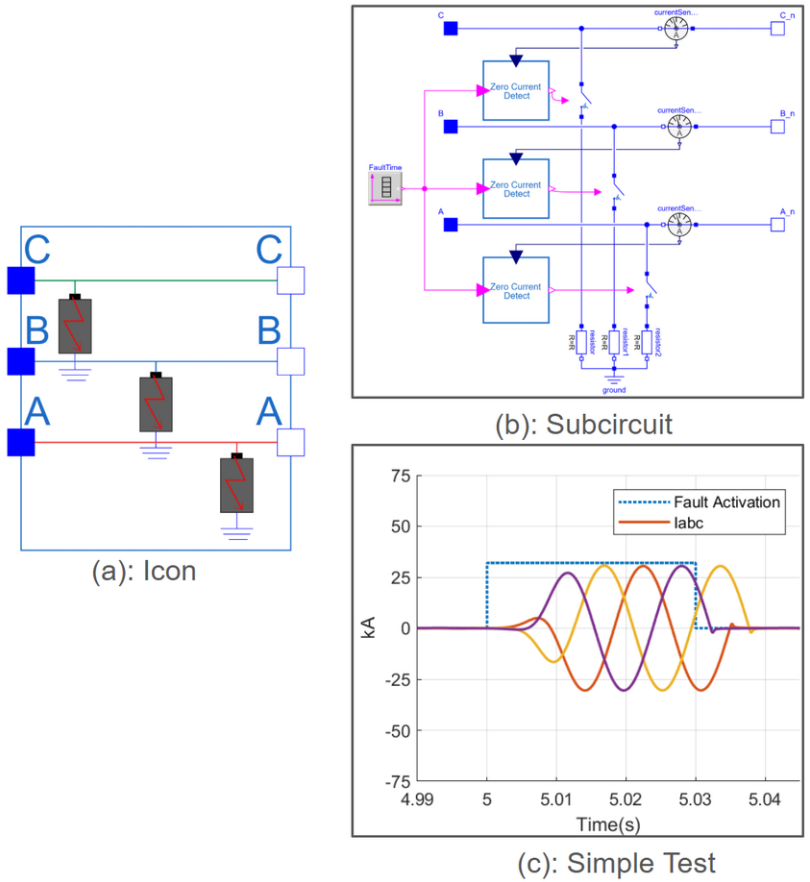
Delta-Wye transformer (CIGRE paper §3.4.1)

- **Role:** interface between the medium-voltage inverter output and the high-voltage transmission network;
- **Each phase** is a two-winding transformer model with specified leakage impedance and winding resistance; primary-side wye (Y), secondary-side delta (Δ); each phase winding represented explicitly
- **No magnetic saturation** (linear MSL inductor elements): the focus is leakage impedance, winding resistance and zero-sequence behavior
- **Series resistances on the secondary side**
  - damp zero-sequence currents that would otherwise circulate within the delta loop during unbalanced or faulted conditions, and
  - enhance numerical stability by preventing ideal short-circuit behavior at the delta connection
- **Validation test (c):** the delta-connected secondary voltages exhibit a 30° phase shift relative to the primary, and the magnitude transformation confirms correct scaling according to the specified ratio

Where it sits

- Between the VSC LC filter and the infinite bus

Modelica implementation (2/2): 3Φ fault w/zero-current detection



(a) icon, (b) switch-based subcircuit with zero-current detection, (c) simple test: fault currents and activation signal (CIGRE paper Fig. 6)

Purpose

- Evaluate the dynamic response of the VSC controller under severe grid disturbances — fault ride-through of the grid-following inverter

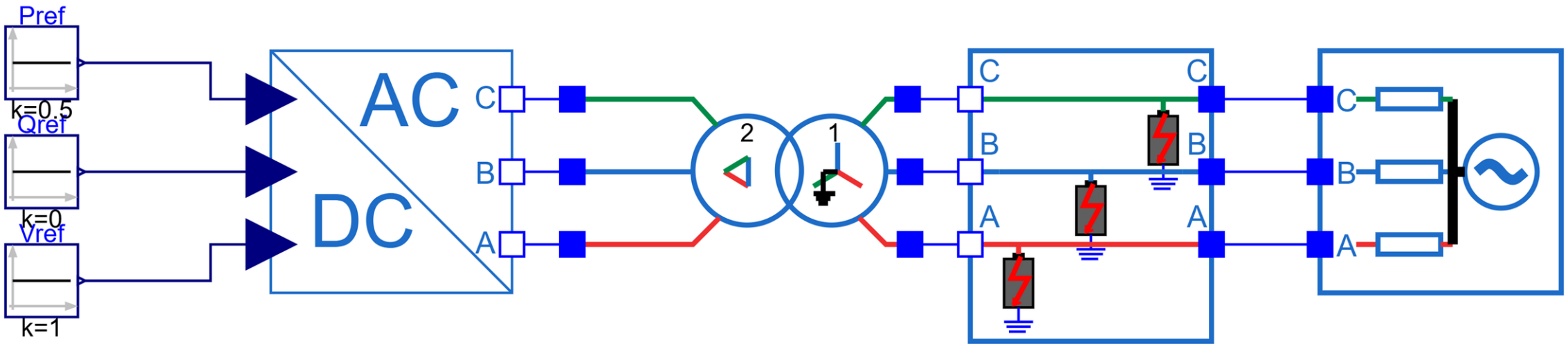
Three-phase symmetrical fault (CIGRE paper §3.4.2)

- **Built from MSL components:** simultaneously shorts all three phases (A, B, C) to ground through low-resistance paths
- **Three independent fault branches,** one per phase, equipped with controlled switches triggered simultaneously by a fault activation signal
- **Fault logic (b):** a user-defined fault initiation time is programmed, after which the switches close — a solid three-phase-to-ground fault; the fault is cleared automatically after a brief duration
- **Zero-current detection** synchronizes clearing with the zero crossing of each phase current → smooth fault clearing, enhanced numerical stability, no sudden discontinuities in the simulation
- **Validation test (c):** expected high current magnitude immediately after fault initiation, followed by a smooth recovery of the system once the fault is cleared

Where it is used

- Applied at the POC in the grid-fault test (Results 2/4)

Simulation setup in Dymola and the four cross-tool test scenarios



Overall system simulation diagram in the Modelica simulation environment : reference blocks, DC/AC converter with the DLL, transformer, three-phase fault block, grid source

Solver and execution

- **Fixed-step Euler, T_{step} = 100 µs** — consistent with the fixed-step execution required by the ICDMS DLL
- **Reference results** from EMTP, identical operating conditions
- **Same approach** implemented and tested in two different Modelica tools
- **Model and simulation parameters** available online (DOI 10.13140/RG.2.2.21778.77764)

1. Active power ramp

- P setpoint 0 → 0.4 p.u. at 0.4 p.u./s,
  Q setpoint = 0 → tracking and steady-state agreement

2. Three-phase fault at the POC

- Symmetrical fault → stability and recovery under a severe voltage disturbance

3. POC voltage step –0.05 p.u.

- Small-signal dynamic performance and Q–V droop response

4. POC frequency step –0.5 Hz

- P–f droop response of the grid-following VSC at t = 2 s

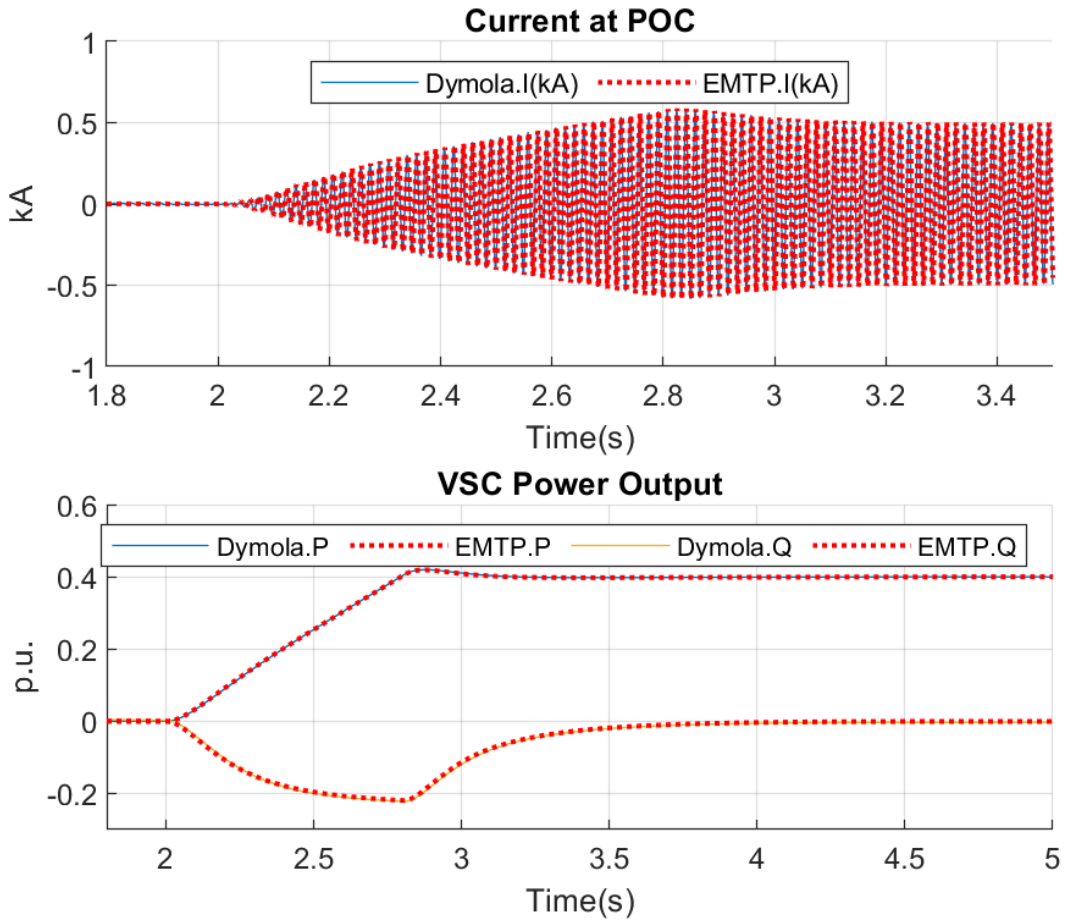
Results 1/4: active power ramping test — Modelica vs. EMTP

Scenario

- **Active-power setpoint** ramped from 0 to 0.4 p.u. at 0.4 p.u./s; reactive-power setpoint held at zero

Observations

- **Three-phase current magnitude at the POC** rises gradually as P increases
- **Current, active- and reactive-power traces** from Modelica and EMTP show very close agreement in amplitude and waveform shape throughout the event
- **Q dips transiently** to about –0.2 p.u. during the ramp and returns to zero — reproduced by both tools



Plot legend: "Dymola" = Modelica implementation (solid); dotted = EMTP. Same operating conditions in both tools

Results 2/4: three-phase grid fault at the POC — where the small deviations come from

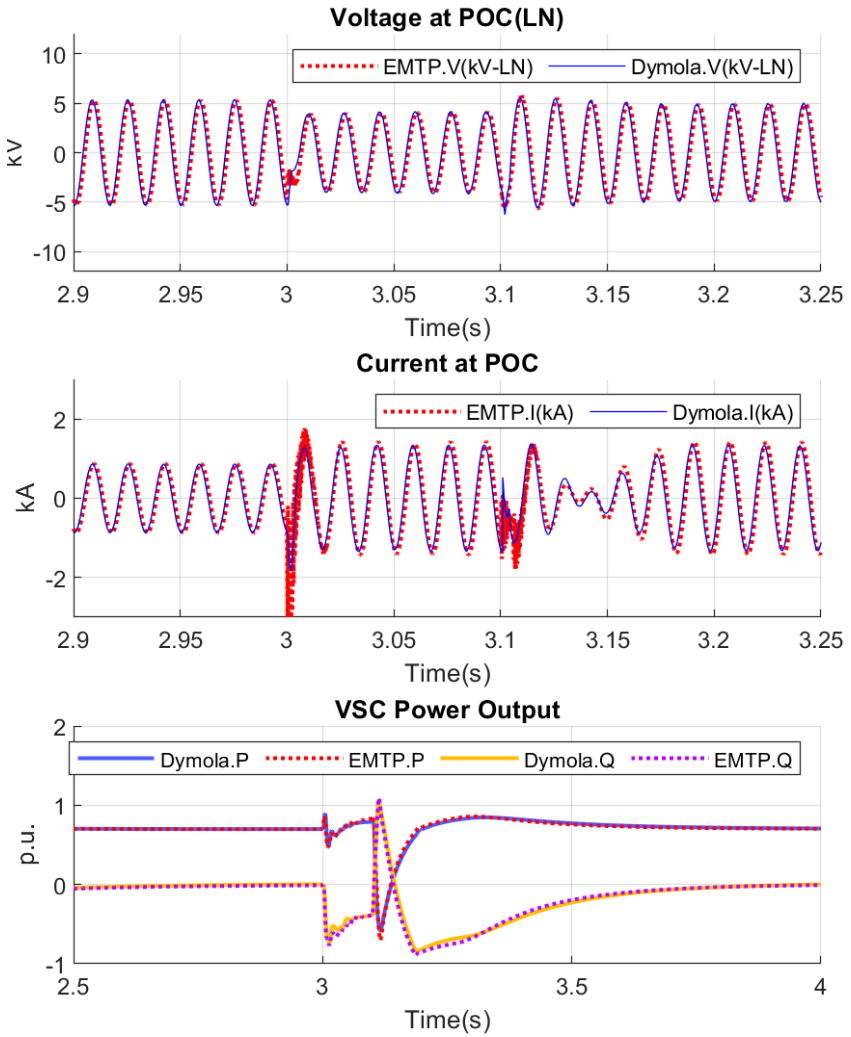
Scenario

- **Three-phase symmetrical fault at the POC** → controller stability and recovery under a severe voltage disturbance

Observations

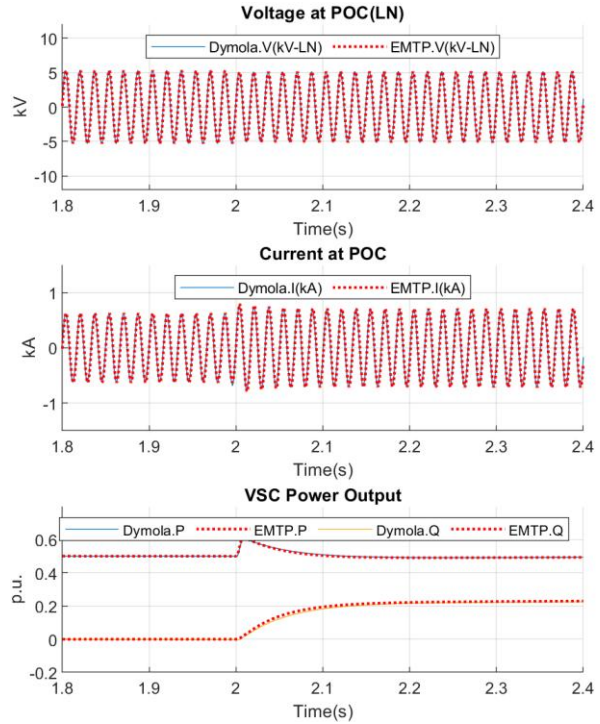
- **Main dynamic response** — voltage dip and recovery, current surge, P/Q swing — is captured with minor deviations
- **Some high-frequency transients are attenuated in Modelica:** a second-order low-pass filter at the transformer secondary suppresses voltage impulses from circuit inductances and improves solver convergence at each step
- **Despite this simplification,** Modelica closely reproduces the critical transient and steady-state VSC behavior during the fault

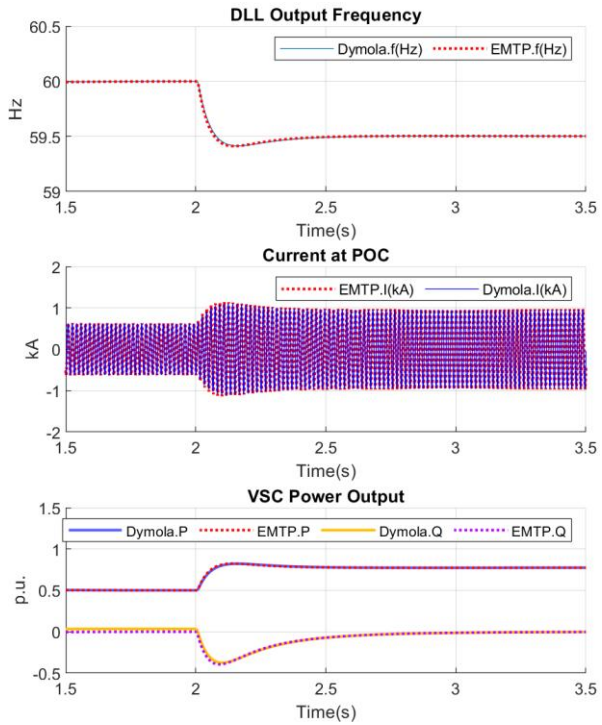
The residual differences trace back to the surrounding Modelica EMT model, built from scratch and not yet fully validated — not to the DLL integration.



Plot legend: "Dymola" = Modelica implementation (solid); dotted = EMTP. Same operating conditions in both tools

Results 3/4 and 4/4: voltage step (–0.05 p.u.) and frequency step (–0.5 Hz)





Left: POC voltage step –0.05 p.u. (paper Fig. 10). Right: POC grid frequency step –0.5 Hz at t = 2 s (CIGRE paper Fig. 11). "Dymola" = Modelica (solid); dotted = EMTP

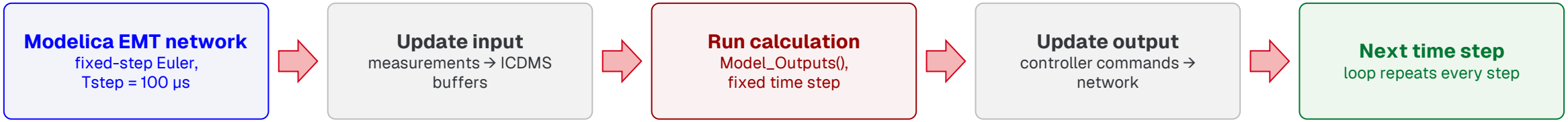
Voltage step: Q–V droop raises reactive power to support the voltage

- **Reactive power rises after the step:** the Q–V droop in the VSC voltage-regulation scheme injects reactive power; active power shows a brief transient and returns to its setpoint; Modelica and EMTP traces overlap for voltage, current and power

Frequency step: P–f droop increases active power

- **At 2.0 s the grid frequency is decreased by 0.5 Hz:** the VSC increases its active power output as expected from the droop characteristic; active power curves from both tools show identical transient peaks and settling behavior, with minor differences at initialization

What the results tell us: fixed-step execution and the Modelica EMT model — motivation for future work



100 µs
fixed-step Euler for the entire model, both papers

2 tools
same approach implemented and tested in two Modelica tools

Residual differences trace back to the surrounding Modelica EMT model (transformer low-pass filter, built from scratch) and to running the entire simulation at the DLL's fixed step — not to the DLL integration.

What limited the match and the efficiency

- **Fault case:** certain high-frequency transients are attenuated in Modelica tools due to 2nd-order LP filter at the transformer secondary, added for numerical stability — effective, but it may introduce artificial dynamics
- **Fixed-step execution:** `Model_Outputs` adheres to the fixed time step of real-world controller firmware, so the entire Modelica simulation runs under a fixed-step solver (Euler, T_{step} = 100 µs) — inefficient for the remainder of the network, where adaptive time-step solvers could give faster simulation and better numerical efficiency

Future work this motivates

- **Clocked partitions (Modelica 3.3 synchronous language elements):** the DLL subsystem in a clocked partition at its fixed sample rate, the EMT network solved by a continuous-time variable-step DAE solver, `sample` / `hold` at the boundaries; open: algebraic loops across partitions, sample-hold accuracy during fast transients
- **Numerical stabilization without the filter:** stiff solvers, impulse elimination, staggered solvers; advanced solver settings and regularization
- **Validation with commercial OEM-supplied ICDMS DLLs** as CIGRE TB 958 adoption grows; many-scenario analysis (Monte Carlo)

Conclusions and two take-aways for the Modelica community

Conclusions

ICDMS-compliant "real-code" DLL models can be linked natively into Modelica through the standardized `ExternalObject` / external "C" interface — no re-implementation of controller code

- **A grid-following droop-controlled VSC** was simulated with its actual controller code; close agreement with EMTP in ramp, fault, voltage-step and frequency-step tests (minor high-frequency deviations in the fault case)
- **The approach worked in two different Modelica tools** →
  - an open-access, standards-based path to interoperable EMT modeling and simulation
  - initial validation for one controller type — needs broader validation across controller types and EMT platforms

Take-away 1 — the DLL integration is validated; the open work is the Modelica EMT model

- **The integration operates at the standardized interface level** → agnostic to the controller internals, portable across Modelica tools
- **What is not yet validated is ours:** the surrounding EMT model built from scratch from MSL components — validated EMT component models (cf. MSEMT) and numerical stabilization without the transformer filter are concrete work for the Modelica power systems community

Take-away 2 — power system practitioners keep vendor-validated controllers and gain the Modelica standards

- **FMI model exchange** → deployment in commercial phasor simulators such as DIgSILENT PowerFactory (Chang & Vanfretti, Paper Session 2B);
- **eFMI** → traceability from Modelica source to embedded C code for CHiL testing (Vanfretti et al., same session)
- **Open source:** core wrappers and interface logic in OpenIPSL, Pull Request #362; model and parameters: DOI 10.13140/RG.2.2.21778.77764 — no proprietary EMT licence needed to run the vendor controller

Thank you — Questions?



Luigi Vanfretti · Hao Chang
ALSETLab, Dept. of Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute
vanfrl@rpi.edu · changh7@rpi.edu · https://alsetlab.github.io/
Model and parameters: DOI 10.13140/RG.2.2.21778.77764 · Wrappers: OpenIPSL PR #362



Rensselaer

Backup slides

Why Modelica for EMT · take-away details · references

References

[1] Modelica Association, “Modelica — A Unified Object-Oriented Language for Systems Modeling, Language Specification Version 3.6,” 2023, Sec. 12.9.7 “External Objects”; Ch. 16 “Synchronous Language Elements.”

[2] CIGRE JWG B4.82 / IEEE, “Technical Brochure 958: Guidelines for Use of Real-Code in EMT Models for HVDC, FACTs and Inverter Based Generators in Power Systems Analysis,” CIGRE, 2025.

[3] H. Chang and L. Vanfretti, “EMT modelling and simulation of grid following converters in Modelica integrating the IEEE/CIGRE ‘real code’/DLL modelling guidelines,” CIGRE Paris Session 2026, SC C4, paper 10724.

[4] H. Chang and L. Vanfretti, “EMT Modelling and Simulation of Grid Following Converters in Modelica Integrating the IEEE/CIGRE ‘Real Code’ DLL Modeling Guidelines,” PMAPS 2026, Salt Lake City, UT, Sep. 2026.

[5] H. Chang and L. Vanfretti, “Integrating the IEEE/CIGRE DLL modeling standard to use ‘real code’ models for power system analysis in Modelica tools,” Proc. American Modelica Conference 2024, Storrs, CT, doi: 10.3384/ecp207.

[6] H. Chang and L. Vanfretti, “Deploying OpenIPSL models into DIgSILENT PowerFactory via FMI model exchange,” American Modelica & FMI Conference 2026, Atlanta. Preprint: doi: 10.13140/RG.2.2.19334.54089.

[7] L. Vanfretti et al., “Real-time simulation and CHiL testing of power system stabilizers on microcontrollers with Modelica and eFMI,” American Modelica & FMI Conference 2026, Atlanta, doi: 10.13140/RG.2.2.32454.84808.

[8] H. Chang and L. Vanfretti, “EMT CIGRE-IEEE DLL Modelling Interface Testing Environment Configuration,” May 2025, doi: 10.13140/RG.2.2.21778.77764.

[9] D. Ramasubramanian, “Differentiating between plant level and inverter level voltage control to bring about operation of 100% inverter based resource grids,” Electric Power Systems Research, vol. 205, 107739, 2022.

[10] M. de Castro et al., “OpenIPSL 2.0.0 — iTesla Power Systems Library: A Modelica library for phasor time-domain simulations,” SoftwareX, vol. 21, 101277, 2023.

[11] A. Masoom et al., “MSEMT: An Advanced Modelica Library for Power System Electromagnetic Transient Studies,” IEEE Trans. Power Delivery, vol. 37, no. 4, pp. 2453–2463, 2022.

[12] K. De Brabandere et al., “A Voltage and Frequency Droop Control Method for Parallel Inverters,” IEEE Trans. Power Electronics, vol. 22, no. 4, pp. 1107–1115, 2007.