# Towards Modeling and Simulation of Dynamic Overconstrained Connectors in Modelica

John Tinnerholm[1] Francesco Casella[2] Adrian Pop[1]

[1]Department of Computer Science, Linköping University, Sweden, {john.tinnerholm, adrian.pop}@liu.se
[2]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy,
francesco.casella@polimi.it

## Abstract

Cyber-Physical Systems are ever-increasing in complexity and new methods and tools for developing them are needed. To support these highly dynamic systems, increasing the flexibility of the modeling languages is desirable. This paper proposes and examines a Modelica language extension to support dynamic overconstrained graphs with reconfiguration at runtime. Two applications of this new feature are also discussed: synchronous AC power systems and incompressible fluid networks. Reported findings suggest that supporting dynamic overconstrained graphs might yield performance benefits and provide the possibility of simulating systems that can not currently be simulated in existing Modelica tools.

*Keywords: dynamic overconstrained connection graph, runtime reconfiguration*

## 1 Introduction & Motivation

Overconstrained connector semantics was introduced in 2004 in version 2.1 of the Modelica Language Specification (The Modelica Association, 2004). It allows to add non-flow variables on connectors that are dependent on each other, which can lead to overconstrained equation systems when loops are formed in the connection graph. It also makes topological information about the connection graph available to the modeller, via the `Connections.*()` operators.

Overconstrained connectors have found at least two notable applications so far. The first is in the MultiBody package of the Modelica Standard Library (Martin Otter, 2003), were such a feature allowed to design of the library in a truly object-oriented way compared to previous versions. The second one is in the PowerSystems library (Rüdiger Franke, 2014), where overconstrained connector variables are used to carry around a reference phase signal for efficient numerical simulation.

As of the current Modelica Language Specification (The Modelica Association, 2021), it is only possible to define static connection graphs, which can be processed at compile time. This makes the implementation of overconstrained connectors in a Modelica compiler rather straightforward. However, it introduces a significant limitation when modelling AC power systems using phasors. In these models, the phase (or frequency) reference is generated by one component of the synchronous system (an infinite bus or a large synchronous generator for islanded systems) and then distributed throughout the entire connected synchronous system by the overconstrained connector variables. In this context, it is possible to have multiple independent synchronous systems in the same Modelica model that correspond to structurally disconnected connections sub-graphs, e.g., two national grids such as Germany and Denmark connected by an undersea DC link; each statically connected synchronous network gets its reference phase or frequency from the root node of its connection graph. However, in this case, their topology is fixed at compile time and cannot change at runtime.

When modeling AC transmission systems, particularly large ones, it is possible that, in case of severe perturbations, some key circuit breakers are switched open, effectively splitting a single synchronous system into multiple independent synchronous islands, which can permanently rotate at different frequency. For example, when modelling the European ENTSO-E synchronous system, the Spanish grid can become isolated by opening a few line breakers on the French border. Note that modelling this scenario requires no structural changes in the grid equations; it just needs some numerical admittance values to be set to zero.

When this happens, the two (or more) ensuing islands can settle down into new steady states with different steady-state frequencies. Hence, if a single, whole-system-wide reference is still used, the phase angle of currents and voltages of the islands that do not contain the root node of the static connection graph will end up rotating permanently, with a frequency that is the difference between the local island frequency and the root node frequency. As a consequence, when the steady-state is reached, the phasors of the new island will continue to change sinusoidally. This is very inconvenient from a performance point of view because it prevents variable step-size solvers from increasing the step size, once the system settles into the new steady state. It also triggers very frequent recomputations of the system Jacobian if implicit stiff solvers are used.

This problem could be avoided by allowing to dynamically add or remove the unbreakable branches correspond-

ing to `Connections.branch()` statements in the connection graph (of Section 9.4.1 in the Modelica Specification[1]), based on the status of the corresponding circuit breaker components, and to add or remove the equations that show up in the same if-equation branches where the `Connections.branch()` statements are declared. It would then be possible to break up the original synchronous connections established by transmission lines when their admittance is brought to zero, thus modelling the effects of circuit breakers on the synchronous subsystem topology.

As a consequence, two or more disjoint connection graphs would be formed at the time of the breaker openings, each corresponding to a new synchronous island. Therefore, the new graph topology should be analyzed at this point, picking a new root node for each newly formed island in the grid. Then, instead of having a single phase reference for the entire system (which is no longer adequate), one would now have two or more independent phase references, one for each island, which would ensure that the phasor variables of each island reach a steady state, thus avoiding the persistent sinusoidal oscillations found in the case of a statically determined connection topology.

In this paper, we investigate the effects of relaxing the constraints imposed on If-Equations (in Section 8.3.4 of the Modelica Specification[2]) by allowing a special If-equation construct where the `Connectors.branch` operator is allowed within these equations, thus making it possible to change the connection graph dynamically at runtime.

The applicability and usefulness of this concept are demonstrated in Section 2, using simple conceptual models in two different application domains: AC power systems and closed incompressible fluid networks. It is shown that in these two cases, the structural variability of the system of equations brought by the proposed extension is indeed very limited and can be handled by small extensions of existing Modelica compilers. A prototype implementation of this feature in the OpenModelica.jl Julia framework is presented in Section 3. Simulation results obtained with the prototype implementations are discussed in Section 4, while Section 5 concludes the paper with final remarks and suggestions for future work.

# 2 Dynamic Overconstrained Connectors in Modelica

A current limitation of Overconstrained Connectors in Modelica is that they cannot be used in If-Equations[3]. In this paper we relax this condition by allowing the `Connections.branch()` operator to occur within If-Equations, hence allowing conditional Connection operators.

To showcase this feature we developed an experimental package called *DynamicOverconstrainedConnectors*[4], containing three sub-packages. The first contains conceptual models of AC grids, using Complex types to represent phasors. The second contains the very same models, albeit with separate Real variables for the real and imaginary part - this is meant for experimental compiler frameworks that cannot handle operator records. Finally, the third contains conceptual models of incompressible fluid networks.

## 2.1 Use case: AC power systems

The main simplifying assumptions for this use case are:

- Purely inductive transmission lines.
- Idealized synchronous generators that impose a voltage at their port with fixed magnitude and a phase equal to the rotor angle.
- Droop-based primary frequency control of the generators.
- The reference frame for the phasors is rigidly connected to the rotor of the generator that is selected as the root node in the connection graph.

Listing 1 contains the definition of the overconstrained connectors; all quantity are in per-unit, to avoid any scaling issues.

Listing 2 shows two alternative implementations for the transmission line model with an embedded line breaker. The first is made possible by the current static connection graphs, whereby the unbreakable branches are always active, and the frequency reference is always the same at the two ports. The second uses the proposed extension, whereby the unbreakable branch is only active when the breakers are closed; so is the equality constraint between the phase reference on the two ports.

Listing 3 shows the conceptual synchronous generator model, which sets the overconstrained reference frequency variable to its own frequency if selected as the root node of the connection graph.

**Listing 1.** AC overconstrained connector.

```
type ReferenceAngularSpeed
  extends SI.PerUnit;
  function equalityConstraint
    input ReferenceAngularSpeed omega1;
    input ReferenceAngularSpeed omega2;
    output SI.PerUnit residue[0];
  end equalityConstraint;
end ReferenceAngularSpeed;
```

[1] https://specification.modelica.org/maint/3.5/connectors-and-connections.html#overconstrained-equation-operators-for-connection

[2] https://specification.modelica.org/maint/3.5/equations.html#if-equations

[3] https://specification.modelica.org/maint/3.5/connectors-and-connections.html#restrictions-of-connections-and-connectors

[4] https://github.com/looms-polimi/DynamicOverconstrainedConnectors


DOI
10.3384/ecp19335

```
connector ACPort
  SI.ComplexPerUnit v(re(start = 1));
  flow SI.ComplexPerUnit i;
  ReferenceAngularSpeed omegaRef;
end ACPort;
```

**Listing 2.** AC Transmission line models.

```
partial model TransmissionLineBase
  parameter SI.PerUnit B = -5.0;
  discrete SI.PerUnit B_act;
  Boolean closed;
  Boolean open;
  Boolean close;
  ACPort port_a;
  ACPort port_b;
equation
  port_a.i = Complex(0,B_act)*
             (port_a.v - port_b.v);
  when open then
    closed = false;
    B_act = 0;
  elsewhen close then
    closed = true;
    B_act = B;
  end when;
  ...
end TransmissionLineBase;

model TransmissionLine
  extends TransmissionLineBase;
equation
  port_a.omegaRef = port_b.omegaRef;
  Connections.branch(port_a.omegaRef,
                     port_b.omegaRef);
end TransmissionLine;

model TransmissionLineVariableBranch
  extends TransmissionLineBase;
equation
  if closed then
    port_a.omegaRef = port_b.omegaRef;
    Connections.branch(port_a.omegaRef,
                       port_b.omegaRef);
  end if;
end TransmissionLineVariableBranch;
```

**Listing 3.** AC generator model.

```
model Generator
  parameter SI.PerUnit V = 1;
  parameter SI.Time Ta = 10;
  parameter SI.PerUnit droop = 0.05;
  parameter Integer p = 0;
  ACPort port;
  SI.PerUnit Ps = 1, Pc, Pe;
  SI.Angle theta(start=0, fixed = true);
  SI.PerUnit omega(start=1, fixed = true);
equation
  der(theta) =
    (omega - port.omegaRef)*omega_n;
  Ta*omega*der(omega) = Ps + Pc - Pe;
  port.v = CM.fromPolar(V, theta);
  Pe = -CM.real(port.v*CM.conj(port.i));
  Pc = -(omega-1)/droop;
  Connections.potentialRoot(
    port.omegaRef, p);
  if Connections.isRoot(port.omegaRef) then
    port.omegaRef = omega;
  end if;
end Generator;
```

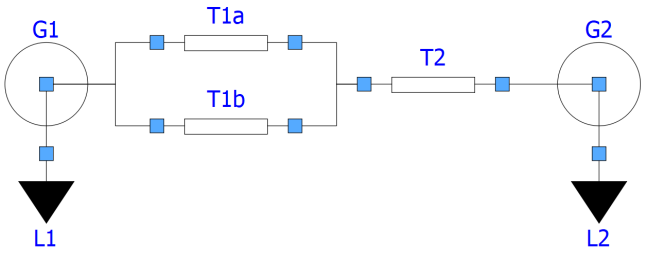


**Figure 1.** `System3` model diagram.

These components are used to build several test cases in the demonstration package; this paper focuses on the most relevant ones.

The base system model `System3` is built as shown in Fig. 1, using standard `TransmissionLine` components with static overconstrained connector semantics. Its connection graph is shown in Fig. 2, and it contains three unbreakable branches, seven connections, one broken connection, and one root node.

The system is initially fully connected, and undergoes an initial transient to get to its steady-state. At $t = 10$, the break of line `T2` is tripped open, so two synchronous island are formed, one containing `G1`, `L1`, `T1a`, `T1b`, and the left connector of `T2`, the other containing `G2`, `L2`, and the right connector of `T2`. The two islands settle to different steady-state frequency, but unfortunately there is only one reference frequency (set by `G1`), so the phasors of the right-hand-side island keep on rotating forever.

The next test model `System4` uses dynamic overconstrained `TransmissionLineVariableBranch` components instead. In this case, the connection diagram is initially the same as in Fig. 2, but after $t = 10$ it becomes as shown in Fig. 3: the deactivation of the unbreakable branch of `T2` splits the graph into two disconnected graphs, each with its own root node. From the point of view of the equation count, the additional equation brought in by the extra root node is balanced by the de-activated conditional equality equation of `T2`. In this way, also the phasors of the right-hand-side island eventually settle down to a constant value, because they are now referred to their proper reference, namely `G2.port.omegaRef`.

The test model `System6` is the same as `System4`, except that `T1a` is tripped open at $t = 10$, instead of `T2`. In this case the system remains fully connected, and a single root node can be used throughout the entire transient.

The model `System7`, shown in Fig. 4, demonstrates the use of root node priority. Two synchronous islands are formed when line `T2` is tripped open. The right-hand-side one contains two generators, `G2` and `G3`, which both contain potential root nodes. In this case, `G3` is selected as root node, since it has a higher priority than `G2`. This mecha-

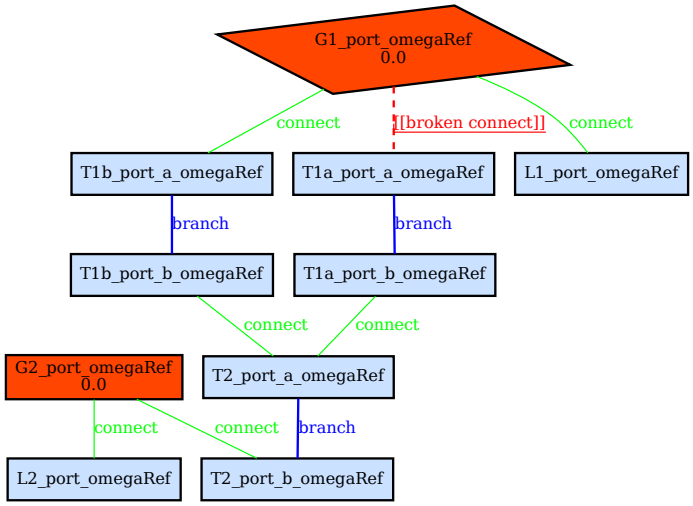


**Figure 2.** Static connection graph for `System3`.

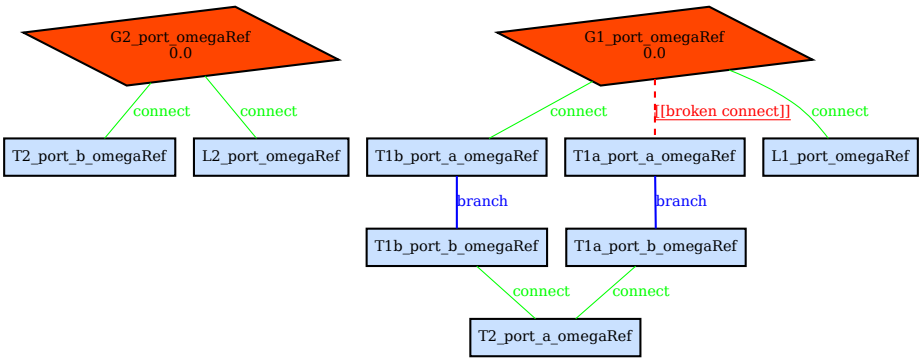


**Figure 3.** The connection graph for `System4` after $t = 10$.


DOI
10.3384/ecp19335

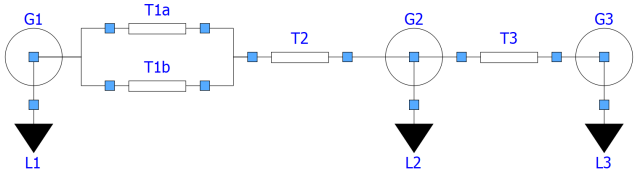


**Figure 4.** `System7` model diagram.

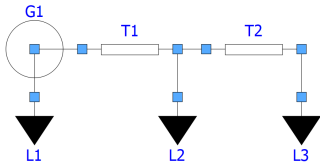


**Figure 5.** `System9` model diagram.

nism can be used to ensure that the larger generators are selected as reference nodes, by setting priorities correlated to the generator size.

Finally, another interesting modelling feature can be implemented with dynamic overconstrained connectors in synchronous AC system models. Consider the system shown in Fig. 5, with the three loads consuming 0.8, 0.1 and 0.1 p.u. power, respectively. Suppose that the total consumption threatens to cause a network breakdown, because the generator `G1` does not have enough power to supply it reliably. In such a case, the network operator can opt for some load shedding, i.e., it can trip open the line `T1`, preserving the service for the largest load `L1`, instead of risking a complete system blackout.

When this happens, `L2` and `L3` remain connected to an island without any power generation capability. Since load models normally prescribe a certain active and reactive consumption, the absence of any generation capacity in the island means that the system equations have no feasible solution, like in the equation $x^2 + 1 = 0$. As a consequence, the simulation aborts as soon as `T1` is tripped open, and it is not possible to continue the simulation of the left-hand-side island, even though it is perfectly functional and capable to carry on operating.

However, if the `TransmissionLineVariableBranch` model is employed for line `T1`, and the load model of Listing 4 is used, the simulation can be continued.

**Listing 4.** Extended load model.

```
model LoadVariableRoot
  extends LoadBase;
equation
  if port.omegaRef > 0 then
    port.v*CM.conj(port.i) = Complex(P,Q);
  elseif Connections.isRoot(port.omegaRef)
      then
    port.v = Complex(0);
  else
    port.i = Complex(0);
  end if;
  Connections.potentialRoot(port.omegaRef,
                            10000);
  if Connections.isRoot(port.omegaRef) then
    port.omegaRef = 0;
  end if;
end LoadVariableRoot;
```

As long as the load is connected to a connection graph that contains at least one generator, this will be selected as root node, because of the much higher priority, and it (greater than zero) frequency will show up as the overconstrained `omegaRef` variable; hence, the first branch of the If-equation will be active, setting the active and reactive power consumption to the given `P, Q` values.

However, if there are no generators in the dynamically formed island after the line tripping, then one load in the island will be selected as root node, and it will set both `omegaRef` and the port voltage to zero, thus conceptually connecting its port to ground. If the load finds itself in a generator-less island, which is characterized by a zero `omegaRef` value, but is not selected as root node, then its equations will set the absorbed current to zero. As a result, all voltages and all currents of the generator-less island will be computed to zero, describing a switched-off subnetwork, while allowing the simulation of the other island to continue undisturbed.

## 2.2 Use case: Closed incompressible fluid networks

Another use case for dynamic overconstrained connectors is incompressible fluid networks. This is demonstrated by the `IncompressibleFluid` sub-package. For the sake of brevity, only some short code fragments are reported in this paper; the reader is referred to the full Modelica code on GitHub for more details.

Any closed incompressible fluid systems, that is not connected to any pressure source of sink (e.g. the atmosphere, or a fixed pressure representing the supply point of a water supply system), needs to be connected to a component known as expansion tank or vessel. In real life the purpose of this component is to set the pressure level of the circuit, which would otherwise be floating freely, and also accommodate for the thermal expansion of the fluid without blowing the circuit up.

When modelling incompressible fluid systems, the thermal expansion effect is normally not explicitly included, because it is very well compensated by the presence of such expansion tanks and has a negligible effect on the actual flow rates, so the density of the fluid is assumed to be a constant. A very simple expansion tank model can then just set the pressure at its port to a fixed value; the entering flow will eventually turn out to be zero, due to the overall mass conservation of the closed circuit. This is demonstrated by the `System1` model in the `IncompressibleFluid` sub-package.

Consider now the `System2` fluid model, shown in Fig. 6. The system is closed and circulates a fluid in the left and right meshes, as well as through the two valves in case the pressure distribution is not fully symmetric.

As long as at least one of the two valves is open, the

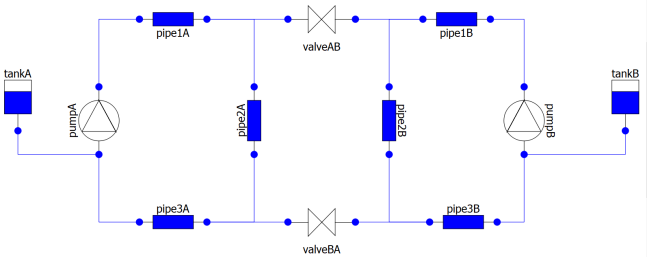


**Figure 6.** `System2` fluid system model diagram.

closed system is fully connected, so a single expansion vessel (e.g. `Tank1`) is required to make sure that all the pressures around the circuit are well-defined. However, when both valves are closed, i.e., their flow coefficients set to zero, the system is effectively split into two independent closed system. At that point, a second expansion tank is needed in the right-hand-side half of the circuit, to make sure that all the pressures there remain well defined.

Without that provision, the equations corresponding to that part of the system will have a unique solution concerning the mass flow rates, but infinitely many when looking at the pressures. Although some tools allow to manage such a situation by picking one of them and continuing the simulation, it remains a fact that the model in those conditions is not well-posed.

Note that, as in the case of AC networks, the splitting into independent sub-systems only depends on numerical values of some coefficients (line admittances there, flow coefficients here), not on structural changes of the system of equations such as, e.g. disabling some connections or some models in the system.

Dynamic overconstrained connectors allow solving this problem. The overdetermined connector can be defined as shown in Listing 5

**Listing 5.** A overconstrained fluid connector.

```
type CircuitIdentifier
  extends SI.PerUnit;
  function equalityConstraint
    input CircuitIdentifier id1;
    input CircuitIdentifier id2;
    output SI.PerUnit residue[0];
  end equalityConstraint;
end CircuitIdentifier;

connector FluidPort
  SI.Pressure p;
  flow SI.MassFlowRate w;
  CircuitIdentifier id;
end FluidPort;
```

In this case, there is no need to carry around any information throughout connected components, as in the previous case; the only thing that is needed for the modelling is the information about the dynamic connection graph topology. However, since the connection graph is always referred to some overconstrained connector variable, one possible choice is to define it as an Integer *connected circuit identifier*.

The `ValveDynamicBranch` model is analogous to the `TransmissionLine` model; in particular, it has a conditional activated `Connections.branch()` statement and a conditionally activated equation stating the equality of the ID on both connectors; both are only active when the valve is open, see Listing 6.

**Listing 6.** Valve model.

```
model ValveDynamicBranch
  extends BaseValve;
equation
  if closed then
    Connections.branch(inlet.id,outlet.id);
    inlet.id = outlet.id;
  end if;
end ValveDynamicBranch;
```

The `ExpansionTank` model is shown in Listing 7. If the tank is selected as root node, then it means the tank is the only component having such a property in the effectively connected circuit; in this case, it sets the port pressure to a fixed parameter value, and the overconstrained `id` connector variable to an ID parameter.

If instead, it is not selected as the root node, then it just acts as a plug, i.e., it sets the port flow rate to zero. This avoids getting inconsistent systems of equations, which would arise if two or more expansion tank components tried to set their port pressure in a connected circuit.

**Listing 7.** Valve model.

```
model ExpansionTank
  parameter SI.Pressure p0;
  parameter Integer id = 0;
  parameter Integer priority = 0;

  FluidPort inlet;
equation
  Connections.potentialRoot(inlet.id,
      priority);
  if Connections.isRoot(inlet.id) then
    inlet.p = p0;
    inlet.id = id;
  else
    inlet.w = 0;
  end if;
end ExpansionTank;
```

When this dynamic overconstrained component is used for the tanks, the model is always well-posed. Initially, when the valves are open, only one tank is selected as a root node and sets the pressure at its port, while the other tank behaves as a plug. As soon as both valves are closed, the connection graph is split into two disconnected graphs, each having its own root node tank. Therefore, each newly formed sub-circuit ends up with a tank setting its pressure level.

## 2.3 Outlook

The two presented modelling scenarios have two important factors in common. One is the need to identify effectively connected connection graphs, when some compo-


DOI
10.3384/ecp19335

nents that normally establish a branch in the graph actually do not do so in some cases, when parameters such as admittance or flow coefficients are brought to zero. The other is the need to set some value in the root node of the effectively connected sub-graph and then propagate it through the actually connected sub-network. Many other models in different domains could have the same requirement and, therefore, a similar structure.

As will be discussed in the next section, this modelling pattern, tackled with dynamic overconstrained connectors, leads to a very restricted structural variability of the corresponding equations, where the overconstrained connector variables are set by conditional equations that are activated when the variable is selected as a root node, and then propagated throughout the actually connected subnetwork. However, this can be handled in terms of code generation and runtime code, without requiring general-purpose handling of structural variability, which is still an open problem for Modelica tools.

# 3 Implementation

A typical Modelica Compiler first translates a textual representation of a Modelica model into executable simulation code through a series of phases, and the semantics for overconstrained connectors in the Modelica language is handled statically during the preprocessing of a model before the generation of simulation code, see Figure 7.

We choose to implement our extension within *OpenModelica.jl* (Tinnerholm et al., 2022). OpenModelica.jl is an experimental Modelica implementation implemented in the Julia language and compiles Modelica code to ModelingToolkit (MTK) (Ma et al., 2021) and is capable of runtime reconfiguration of models.

To handle the proposed extension for dynamically overconstrained connectors, we extended the flat Modelica representation to also contain a self-reference before connections are resolved. Furthermore, we reused the dynamical capabilities of *OpenModelica.jl* described in (Tinnerholm et al., 2022).

When the condition for a *DOCC-If-Equation* such as the one for the dynamic transmission line in Listing 2 is fulfilled at the time of the change $t_\Delta$ the following steps are taken:

- The simulation halts, and a *Connection* operator is either inserted or removed, and the virtual connection graph is updated.
- A new equation system is derived from the resulting connections and the changed overconstrained connection graph.
- The system is recompiled with the new equation system
- The simulation restarts using the previous values before the event at $t_\Delta$.

It should be noted that this could allow for a more general treatment of other types of structural variability, e.g., conditional `connect()` statements, since the OpenModelica.jl framework allows for dynamic reconfiguration of Modelica models.

However, as discussed previously, in some cases, recompilation is not necessary. The examples included in the DynamicOverconstrainedConnectors package all fall in this category; they are static[5], while the virtual overconstrained connection graph, for `System4` as depicted in Figure 2 and Figure 3, changes its structure during the simulation.

In the applications showcased by the exemplary library, the overconstrained variables carry around a scalar value that is relevant to the behavior of the connected subsystems. For power systems, it is the AC phase or frequency. For incompressible fluid networks, it is the network ID. As noted in Section 2.3, this means that the selected root node sets the value of the overconstrained connector variable, which is then propagated through connection equations and the conditional equality equations when the corresponding `Connections.branch()` statement is either activated or deactivated.

For `System4` this variable is `G1_port_omegaref` before the changes in the virtual connection graph. After this change, this value is provided by `G1_port_omegaref` and `G2_port_omegaref` as depicted in Figure 3. This means that some of the general steps of handling this solution described earlier can be omitted. Instead of recompiling the system at time $t_\Delta$, the set of variables that are part of the virtual connection graph can instead be reinitialized at that time $t_\Delta$ using the root value. Hence, recompilation of the system is not needed.

In `System4` these roots are `G1_port_omegaRef` and `G2_port_omegaRef` after $t = 10.0$ as depicted in Figure 3. Instead of recompiling, the second and third steps are as follows:

- The required modification is derived from the resulting connections and the changed overconstrained connection graph.
- The causality is changed for the equations involving the new roots

That is, the difference between the reinitialization approach and the recompilation approach is that instead of recompiling the system and regenerating the equations, we change the reference values of the roots based on the overconstrained connection graph in the simulation runtime. In this example this is done by identifying the new roots and the corresponding root sources. For `System4` this is `G2_port_omegaRef` with the source being `G2_omega`. In the case of `System4` only one equation is modified, that of `G2_port_omegaRef`.

[5]The causality changes, but the number of equations and variables remains the same

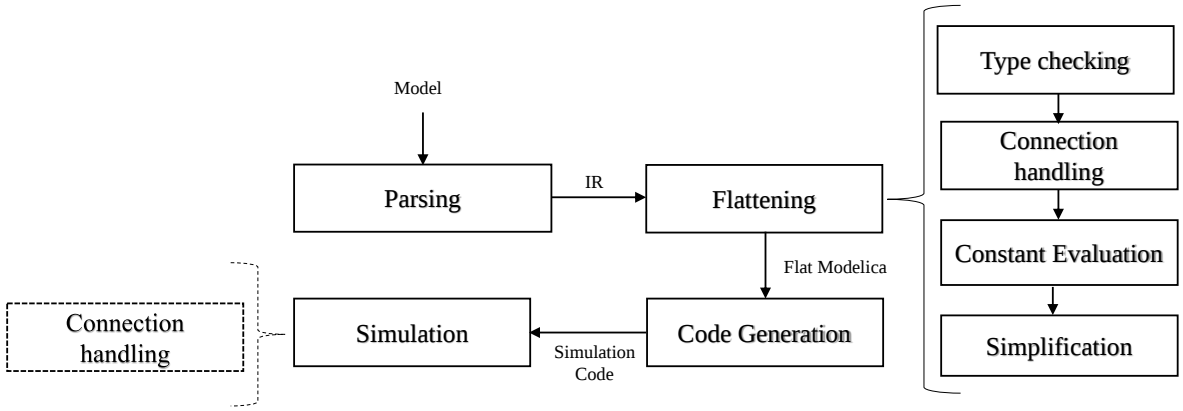


**Figure 7.** The translation process of a Modelica Compiler. The model is first translated to an internal intermediate representation (IR) where typing and type checking is performed and where the declared connections are handled and expanded before the simulation code is generated. The dashed box to the left shows where the new extension is handled in the compilation process.

# 4 Simulation Results

Handling overconstrained connector variables dynamically at runtime allows for the successful simulation of models that existing Modelica tools cannot currently handle because of model singularities. It also allows stiff solvers to increase the step size in some situations, which leads to improved simulation performance.

In this section, we demonstrate these benefits on a selection of the models presented in Section 2.1.

The systems were simulated using the RODAS5 solver available from DifferentialEquations.jl (Rackauckas and Nie, 2017).

## 4.1 Synchronous Power Grid models

The `System3` and `System4` models are equivalent with respect to the generator frequency and power variables `G1.omega`, `G2.omega`, `G1.Pe`, `G1.Pc`, `G2.Pe`, and `G2.Pc`. Indeed, the transients of those variables turn out to be identical in the simulations of the two models.

As explained in Section 2.1, this is not the case for the voltage and current phasors. Figures 8 and 9 show the real part of the voltage phasor of `G2`. During the first 10 seconds (left-hand-side plots), the grid is fully connected in one synchronous system using the frequency of `G1.omega` as reference, so both phasors settle down to a steady state after about 8 s.

However, when the breaker `T2` is tripped open, the right-hand-side island, to which `G2` belongs, settles down to a slightly different frequency than the left-hand-side one. As anticipated, the voltage phasor of `G2` continues to oscillate forever in the model with static overconstrained connectors, while it remains practically constant in the model with dynamic overconstrained connectors, thanks to the correct choice of reference frequency after the splitting into two islands.

This allows a stiff solver such as *RODAS5* to take much longer steps, completing the simulation with less steps and less Jacobian calculations, as shown in Table 1.

The situation is similar when comparing the simulations of `System7`, which has a static connection graph, and `System8`, which has a dynamic connection graph. When the breaker of line `T2` is opened, two synchronous islands are formed, one including `G1`, and one including `G2` and `G3`.

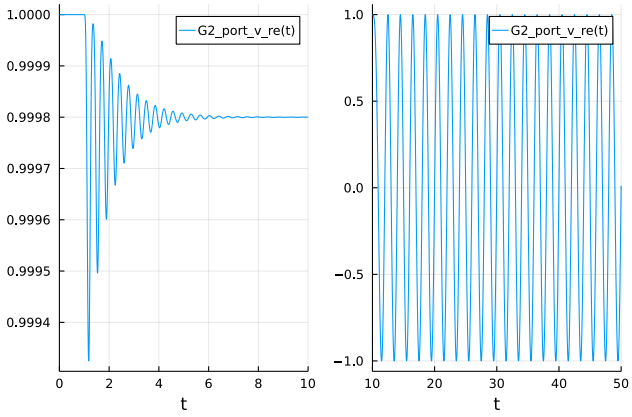


**Figure 8.** Plots of the `G2.port.v_re` variable in `System3` before and after the susceptance of line `T2` is brought to zero at $t = 10$. The phasor oscillates forever because the system only has one root node also after the network splitting.

Figures 10 and 11 show again the real part of the voltage phasor of `G2` for the two models, before and after the splitting. In this case, once the island containing `G2` and `G3` is formed, these two generators oscillate against each other for a while, but eventually end up rotating at the same speed. When using dynamic overconstrained connectors, the frequency of `G3` is used as a reference, so the voltage phasor of `G2` eventually becomes constant, while it does not in the static overconstrained connector case, for which the frequency of `G1` is still used as a reference.

As in the previous case, using dynamic overconstrained connectors has beneficial effects in terms of less integration steps and less Jacobian computations, see Table 1.


DOI
10.3384/ecp19335

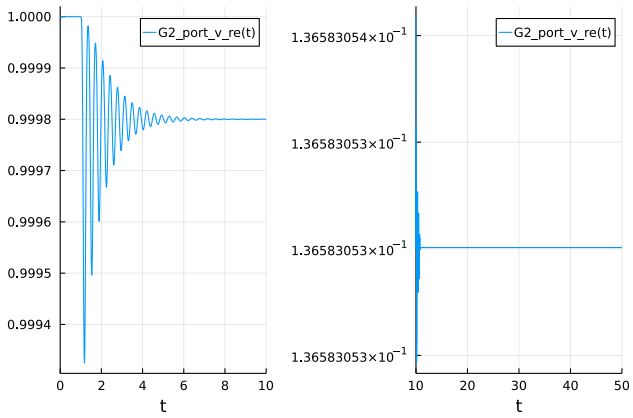


**Figure 9.** Plots of the `G2.port.v_re` variable in `System4` before and after the susceptance of line `T2` is brought to zero at $t = 10$. The phasor oscillates remains practically constant after the splitting thanks to the correct choice of reference after the splitting.

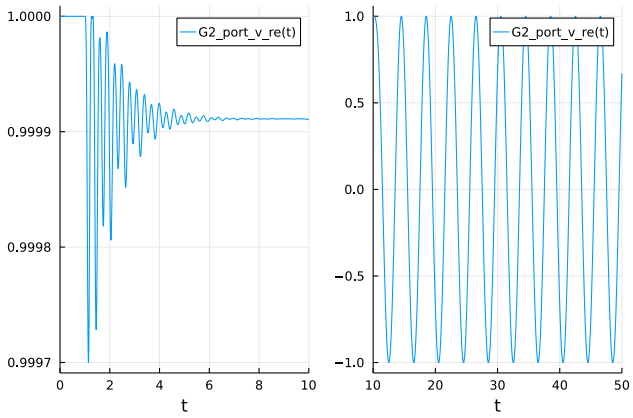


**Figure 10.** Plots of the `G2.port.v_re` variable in `System7` before and after the susceptance of line `T2` is brought to zero at $t = 10$. The phasor oscillates forever because the system only has one root node also after the network splitting.

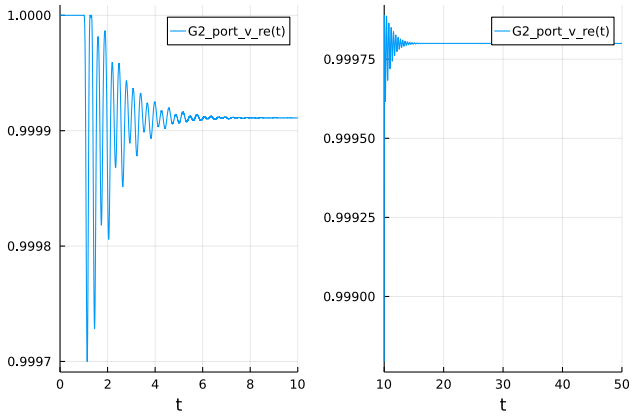


**Figure 11.** Plots of the `G2.port.v_re` variable in `System8` before and after the susceptance of line `T2` is brought to zero at $t = 10$. The phasor oscillates for a while but then settles to a constant value after the splitting, thanks to the correct choice of reference after the splitting.

**Table 1.** Number of accepted steps and the total number of Jacobians created for Systems 3, 4, 7 and 8. Systems 3 and 4 are identical except for the use of dynamically overconstrained connectors in System 4. Systems 7 and 8 have the same relationship.

| System | Accepted Steps | Jacobians Created |
|---|---|---|
| System 3 | 565 | 605 |
| System 4 | 125 | 132 |
| System 7 | 374 | 389 |
| System 8 | 169 | 175 |

### 4.2 Incompressible Fluid networks

When simulating `System3`, as described in Section 2.2, the first valve is closed at $t = 2$, and then the second is closed at $t = 4$. From that point in time, the algebraic system of equations determining the circuit pressures and flows is reported as singular when simulating the system using the OpenModelica tool (Fritzson et al., 2020).

As anticipated, the reason is that the pressures in the right-hand-side part of the system have infinitely many solutions[6]. Model `System4` instead uses the proposed extension and, as expected, the system shows no singularity for $t \geq 4$.

## 5 Conclusion and Future Work

In this paper, we have illustrated and discussed the benefits if some of the current constraints of the Modelica language are lifted, allowing for dynamic overconstrained connection graph, with application in synchronous AC system models and closed incompressible fluid system models. With reference to phasor-based models of synchronous AC systems, one benefit is that solvers can take much larger steps and subsequently need to create fewer Jacobians if the system is split into multiple, independent synchronous sub-systems by opening breakers on strategic transmission lines, as was shown in Section 4. Another benefit is handling the formation of islands without generation capacity, avoiding the termination of the simulation because of unsolvable equations. With reference to the models of closed hydraulic systems with an incompressible fluid, the benefit is that it is possible to handle the formation of independent closed sub-systems by closing valves that separate two or more parts of the circuit without leading to singular systems of equations.

Furthermore, the reconfiguration approach described in Section 3 could be improved. Currently, runtime reconfiguration in OpenModelica.jl requires the system to keep the equation structure before and during simulation; hence we have to omit important optimisation phases such as removing trivial equality constraints. Consequently, there is currently a trade-off between optimisation and the speed of system reconfiguration. Still, if we consider the cost of

[6]It should be noted that existing Modelica tools can handle this scenario by selecting one of the several solutions.

recompilation, this approach should be more efficient for small systems even though it currently only works without some optimisation phases.

In general, it seems that the extension proposed in this paper can be implemented with a reasonable effort in mainstream Modelica compilers, as long as the structure of the system using it is similar to that of the presented use cases, which will indeed be the case for a significant number of real-life use cases.

Although the current study is based on a small set of examples from a conceptual library, the findings suggest that dynamically overconstrained connectors could be employed to simulate real-life systems that is not possible to simulate in existing Modelica tools and provide possible performance benefits. Therefore, a direction for future work would involve implementing support for dynamically overconstrained connectors in the OpenModelica Compiler to investigate the general applicability of this construct on larger systems, for example using the PowerGrids library (Bartolini et al., 2019).

The final goal is to get this extension into a future version of the Modelica Language Specification, so that it gets eventually supported by a growing number of Modelica tools, allowing library developers to use it without concerns about limited support.

## Acknowledgement

This work has been supported by the Swedish Government in the ELLIIT project and has been partially funded by Vinnova in the context of the ITEA project EMBRACE. Furthermore, the authors gratefully acknowledge the financial support of the French Transmission System Operator RTE to the Open Source Modelica Consortium, which made this initial development possible.

## References

Andrea Bartolini, Francesco Casella, and Adrien Guironnet. Towards pan-european power grid modelling in Modelica: Design principles and a prototype for a reference power system library. In Anton Haumer, editor, *Proc. 13th International Modelica Conference*, pages 627–636, Regensburg, Germany, Mar 4–6 2019. doi:10.3384/ecp19157627.

Peter Fritzson, Adrian Pop, Karim Abdelhak, Adeel Ashgar, Bernhard Bachmann, Willi Braun, Daniel Bouskela, Robert Braun, Lena Buffoni, Francesco Casella, Rodrigo Castro, Rüdiger Franke, Dag Fritzson, Mahder Gebremedhin, Andreas Heuermann, Bernt Lie, Alachew Mengist, Lars Mikelsons, Kannan Moudgalya, Lennart Ochel, Arunkumar Palanisamy, Vitalij Ruge, Wladimir Schamai, Martin Sjölund, Bernhard Thiele, John Tinnerholm, and Per Östlund. The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development. *Modeling, Identification and Control*, 41(4):241–295, 2020. doi:10.4173/mic.2020.4.1.

Yingbo Ma, Shashi Gowda, Ranjan Anantharaman, Chris Laughman, Viral Shah, and Chris Rackauckas. Modelingtoolkit: A composable graph transformation system for equation-based modeling, 2021.

Sven Erik Mattsson Martin Otter, Hilding Elmqvist. The new Modelica MultiBody library. In *Proceedings 3rd International Modelica Conference*, pages 311–330, Linköping, Sweden, Nov 3–4 2003.

Christopher Rackauckas and Qing Nie. DifferentialEquations.jl–a performant and feature-rich ecosystem for solving differential equations in Julia. *Journal of Open Research Software*, 5(1), 2017.

Hans-Jürg Wiesmann Rüdiger Franke. Flexible modeling of electrical power systems – the Modelica PowerSystems library. In *Proceedings 10th International Modelica Conference*, pages 515–522, Lund, Sweden, Mar 10–12 2014. The Modelica Association. doi:10.3384/ecp14096515.

The Modelica Association. Modelica - A unified object-oriented language for physical systems modeling - Language specification version 2.1. Online, Jan. 30 2004. URL `https://modelica.org/documents/ModelicaSpec21.pdf`.

The Modelica Association. Modelica - A unified object-oriented language for physical systems modeling - Language specification version 3.5. Online, Feb. 19 2021. URL `https://modelica.org/documents/ModelicaSpec35.pdf`.

John Tinnerholm, Adrian Pop, and Martin Sjölund. A Modular, Extensible, and Modelica-Standard-Compliant OpenModelica Compiler Framework in Julia Supporting Structural Variability. *Electronics*, 11(11), 2022. ISSN 2079-9292. doi:10.3390/electronics11111772.