Techniques for Modeling And Simulation of Dynamic Overconstrained Connectors

John Tinnerholm¹, Francesco Casella², Adrian Pop¹

1 Department of Computer and Information Science, Linköping University, Sweden
2 Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy
Introduction and Motivation

• Increase flexibility for better modeling and simulation of Cyber Physical Systems

• Overconstrained connector semantics was introduced (2004)
  • MultiBody package of the Modelica Standard Library\(^1\)
  • PowerSystems library\(^2\)

• Current Modelica language specification only allows static connection graphs

• Limitation when modelling AC power systems using phasors

---


Introduction and Motivation

• AC transmission systems
  ➢ Possible that, in case of severe perturbations, some key circuit breakers are switched open splitting a single synchronous system into multiple independent synchronous islands

• DOCC
  ➢ Performance Benefits
  ➢ Avoiding Singularities

• Increasing the flexibility of Modelica Models
  • Applicable to other modeling domains as well
    • AC power systems
    • Closed incompressible fluid networks
OpenModelica.jl

- A Modelica Compiler implemented in Julia
- Use the frontend of the OMC, translated into Julia
- Backend generating Julia code
- For this work
  - Implemented Discrete event handling
  - Backend Handling of OCC
  - Expanded the intermediate representations

➢ Feature
  ➢ Blurring the border between compilation and simulation
  ➢ Used for experimental features
Dynamic Overconstrained Connectors

- Currently, Overconstrained Connectors in Modelica can not be used in If-Equations\(^3\)
  - Relaxing constraints
- Allowing a special If-Equation construct where the `Connectors.branch` operator is allowed
  - Allowing changing the connection graph dynamically at runtime
- Implementation in OM.jl

```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;
```

\(^3\)https://specification.modelica.org/maint/3.5/connectors-and-connections.html#restrictions-of-connections-and-connectors
OpenModelica.jl

• A Modelica Compiler implemented in Julia
• Use the frontend of the OMC, translated into Julia
• Backend generating Julia code
• Does some things better than omc…

➢ Feature
  ➢ Vague border between compilation and simulation runtime
  ➢ Used for experimental features
Example Library

- Example Library to illustrate this construct (Dynamic Overconstrained Connectors)\(^1\)
  - AC power systems
  - Closed incompressible fluid networks
- Simplifying assumptions\(^2\)
  - 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

\[^1\] More information available in the paper
\[^2\] Techniques for Modeling And Simulation of Dynamic Overconstrained Connectors
Implementation

- Implemented in OpenModelica.jl
- Runtime supports two alternatives
  - Reconfiguration/Recompilation of the System
  - Reinitialization without recompilation
- Constructs for handling Overconstrained Connectors (OCCs) are moved to simulation runtime
- Supports Dynamic Overconstrained Connectors (DOCC)
- Size of implementation
  - ~1000 LOC
  - Frontend procedures reused
Example System 4

• Same as System 3
  • Difference is the transmission line model

• The line breaker implements the proposed extension
  • Dynamically removes the unbreakable branch between its to connectors when the susceptance B is brought to zero

• The OCC graph is split into two at time $t = 10$

```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;```
Example System 4

- Once the new frequency steady state is reached, phasors in both islands will remain constant
- Stiff solvers may take longer steps
- Phasor representation of currents and voltages in the connectors will be different
- Physically meaningful variables, specifically the generator frequencies $G_1.\omega$ and $G_2.\omega$, $G_1.P_e$, $G_1.P_c$, $G_2.P_e$, and $G_2.P_c$ will be the same
Runtime Recompilation

• Recompilation
  • At the time of the structural change recompile the system
  • System is initialized with the values of the variables before the structural change
  • Simulation is restarted
• Simple implementation for compilers/environments that support JIT
  • Recompilation should not be not needed
Runtime Reconfiguration

• Instead of Runtime Compilation
  • Runtime Configuration
• Extensions to the simulation runtime
• Pause the simulation at the time of the DOCC event
• Reinitialize

• Advantage
  • Reinitialization itself is less costly
• Disadvantage
  • More complicated implementation
  • DOCC chains need to be preserved, partially hinders optimizations
Runtime Reconfiguration Continued

- Detecting Structural Change
- Find the final roots in the OCC forest
- Propagate the values of the root variables
- Change System Causality
- Reinitialize the System
- Continue Simulation
Runtime Reconfiguration, datastructures

Additions to the simulation runtime

- The New Frontend **NFOCConnectionGraph** module as a part of the Simulation Code IR
- In Julia no change is needed since the frontend can be called directly
- The simulation runtime need to keep an instance of the current and previous virtual connection graph

Tracking OCC Variables

- Avoiding optimizations that breaks OCC chains

Structural Event Handler

- Supervise the simulation
- Update the Virtual Connection Graph
Runtime Reconfiguration
Runtime Reconfiguration Continued

- The first two steps are the same as in the Recompilation scheme
- Change System Causality
  - Here one equation need to change
    - $G^2_{\omega} = G^2_{\text{port}_\omega \text{Ref}}$
- In OM.jl a single new equation is created and inserted
  - Swapping pointers in OMC
Runtime Reconfiguration

The first two steps are the same as in the Recompilation scheme

Change System Causality

• For System 4 only one equation need to change
  
  I. \( T_2 \text{port\_omegaRef} = G_2 \text{port\_omegaRef} \)
  
  II. \( G_2 \text{omega} = G_2 \text{port\_omegaRef} \)

• Causality changes but the number of equation and variables in the system remains the same

• Change in equations may be achieved by swapping pointers in a Language such as C
  
  • No recompilation needed

• In Julia we insert new equations symbolically

• Overhead since optimizations are disabled for the equality chains involved in the Dynamic OCC Graph

➢ Assignments for G1_port_omegaRef:
  1. T1b_port_a_omegaRef := G1_port_omegaRef
  2. T1b_port_b_omegaRef := G1_port_omegaRef
  3. T2_port_a_omegaRef := G1_port_omegaRef
  4. T1a_port_b_omegaRef := G1_port_omegaRef
  5. T1a_port_a_omegaRef := G1_port_omegaRef
  6. L1_port_omegaRef := G1_port_omegaRef

➢ Assignments for G2_port_omegaRef:
  1. T2_port_b_omegaRef := G2_port_omegaRef
  2. L2_port_omegaRef := G2_port_omegaRef

Assignments for G1_port_omegaRef:
Applications
Applications

• DOCC allows stiff solvers to increase the step size in some situations, which leads to improved simulation performance.
  ➢ See the plot of System 3 and the equivalent System 4 using DOCC to the right.
• DOCC allows for the successful simulation of models that existing Modelica tools cannot currently handle because of model singularities.

Plots of the G2.port.v_re variable in System 3 before and after the susceptance of line T2 is brought to zero at $t = 10$. The phasor remains practically constant after the splitting thanks to the correct choice of reference after the splitting.

Plots of the G2.port.v_re variable in System 4 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.
Applications

• Cost/Benefits
  • Recompilation is expensive
  • Reinitialization is not as expensive but not free
• Drastically reduce the number of Jacobians needed to be created
• Likely Outcome
  • Might not be a good modeling technique for smaller systems in terms of performance

<table>
<thead>
<tr>
<th>System</th>
<th>Accepted Steps</th>
<th>Jacobians Created</th>
</tr>
</thead>
<tbody>
<tr>
<td>System 3</td>
<td>565</td>
<td>605</td>
</tr>
<tr>
<td>System 4</td>
<td>125</td>
<td>132</td>
</tr>
<tr>
<td>System 7</td>
<td>374</td>
<td>389</td>
</tr>
<tr>
<td>System 8</td>
<td>169</td>
<td>175</td>
</tr>
</tbody>
</table>
Applications

- DOCC allows for the successful simulation of models that existing Modelica tools cannot currently handle because of model singularities.
- DOCC allows this type of systems to be simulated.

https://github.com/looms-polimi/DynamicOverconstrainedConnectors
Applications
Future Work

- More experimentation
- Formalize backend methods
- Implementation in OMC
  - Mockup C program
  - Experiment with extending the Simulation runtime
Thank you for your attention
Questions?