IDAES Advanced Model Development

Chinedu Okoli[a], Miguel Zamarripa[a], Andrew Lee[a], Paul Akula[b], Anca Ostace[b], Jaffer Ghouse[a], Debangsu Bhattacharyya[b], David Miller[a]

[a] National Energy Technology Laboratory, Pittsburgh, PA; [b] West Virginia University, Morgantown, WV

Advanced Model Development Tools

Unit Model Library

IDAES Unit **Model Library** provides basic unit models (i.e. mixers, splitters, flash drums) that can be used as building **blocks** for more **complex models**.

Property Blocks

- Implemented as modular blocks
- Inlet/Outlet ports allowing unit models to be applied to advanced processes
- Flexible formulation of equations, **for example**:

$V = f(T, P, x)$ $\quad h = f(T, x)$ $\quad P_{sat} = f(T)$

$Cp = f(T)$ $\quad y_i = K_i x_i$ $\quad r = f(T, P, x)$

Ports

- Defined via the property package.
- Can connect to any other Port with the same members automatically
- Translator blocks to connect Ports with different state variables

Control Volumes

- Three types of Control Volumes available for different applications
  - 0-D (Inlet-Outlet type)
  - 1-D (Pipes, PFRs)
  - Static (Dead zones)
- Control Volumes can be connected together to form advanced unit models
- Prebuilt **methods** for common forms of **balance equations**:

$$\frac{\partial M_j}{\partial t} = F_{in,j} - F_{out,j} + N_{rxn,j} + N_{eq,j} + N_{phase,j}$$

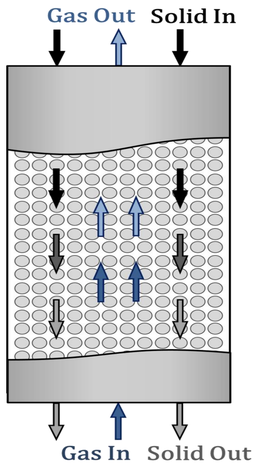
$$\frac{\partial E}{\partial t} = H_{in} - H_{out} + Q + W$$

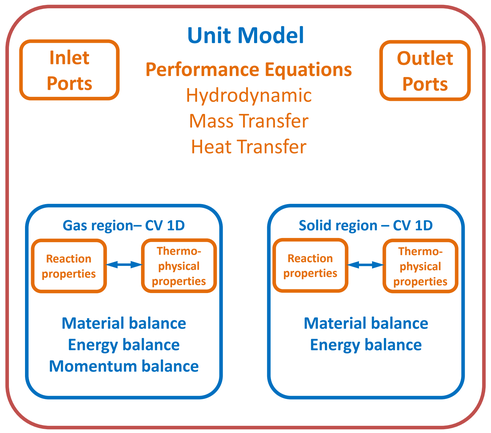
$$0 = P_{in} - P_{out} + \Delta P$$

Initialization

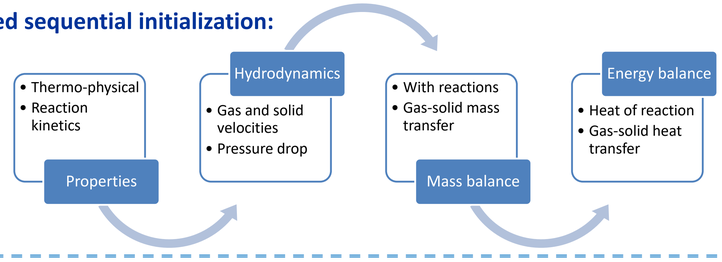
- Custom initialization routines
- IDAES is developing for solving complex models
  - Decomposition solvers
  - Reliability and convergence tools

Advanced Reactor Model: Moving Bed

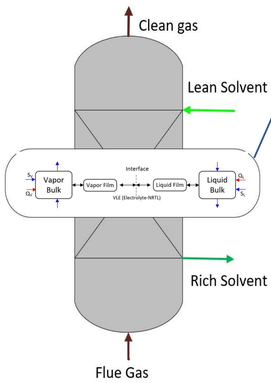


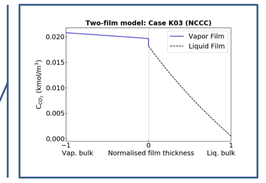


Automated sequential initialization:



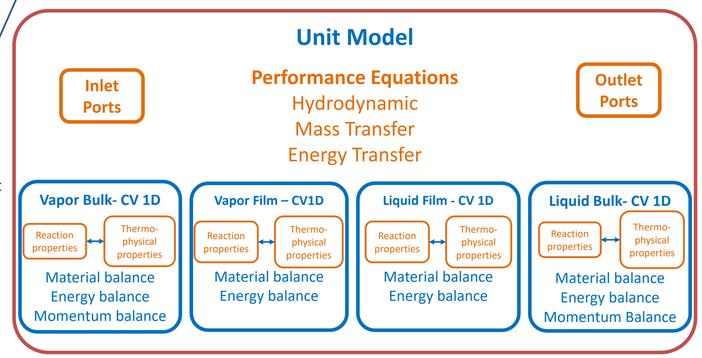
Advanced Column Models: Rate-based Column



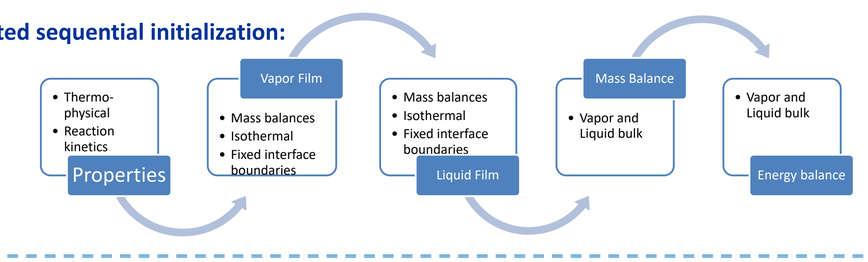


Model Features:

- Electrolyte NRTL model for vapor-liquid equilibrium
- Rigorous two film model for electrolytes
- Multi-component mass transfer across films involving molecular and ionic species



Automated sequential initialization:



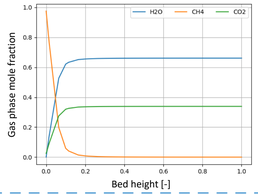
Case Study – Simulation of a Moving Bed chemical looping combustion fuel reactor

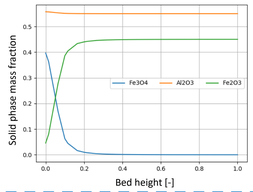
```
#Create flowsheet and instantiate Moving Bed model
m = ConcreteModel()
m.fs = FlowsheetBlock(default={"dynamic": False})
m.fs.MB = MovingBed(default={"gas_side":
                                {"property_package": m.fs.gas_properties}
                             "solid_side":
                                {"property_package":m.fs.solid_properties,
                                 "reaction_package": m.fs.hetero_reactions}})

#Initialize model
m.fs.MB.initialize()

#  Solve model
solver = SolverFactory('ipopt')
results = solver.solve(m.fs, tee=True)

#  Plot results
Plot  results(m.fs)
```





Moving Bed Unit model construction

```
def build(self):
    # Build Control Volumes for gas and solid phases
    self.gas_phase = ControlVolume1DBlock(default={ gas_phase_args})
    self.solid_phase = ControlVolume1DBlock(default={ solid_phase_args})

    # Populate gas control volume
    # Method to create a spatial domain and volume Variable in ControlVolume
    self.gas_phase.add_geometry(flow_direction= FlowDirection.forward, length_domain_set= [0.0, 1.0])
    # This method constructs the state blocks for the control volume
    self.gas_phase.add_state_blocks(information_flow= FlowDirection.forward, has_phase_equilibrium=False)
    # This method constructs a set of 1D material balances indexed by time length and component
    self.gas_phase.add_material_balances(balance_type= MaterialBalanceType.componentTotal,
                                         has_phase_equilibrium=False,  has_mass_transfer=True,
                                         has_rate_reactions=False)
    # This method constructs a set of 1D enthalpy balances indexed by time and phase
    self.gas_phase.add_energy_balances(balance_type= EnergyBalanceType.enthalpyTotal,
                                       has_heat_transfer=True)
    # Method to apply DAE transformation to the Control Volume length domain
    self.gas_phase.apply_transformation()

    # Add performance equations
    # Gas phase - gas to solid heat transfer
    @self.Constraint(self.time_ref, self.gas_phase.length_domain,
        doc="Gas phase - gas to solid heat transfer")
     def gas_phase_heat_transfer(b, t, x):
         return (b.gas_phase.heat[t, x]*b.solid_phase.properties[t, x]._params.particle_dia  == -6*b.gas_solid_htc[t,x]*
                 (b.gas_phase.properties[t, x].temperature - b.solid_phase.properties[t, x].temperature)*
                 b.solid_phase.area[t, x])
```

Remarks:

IDAES modeling framework provides the foundation for more advanced modeling capabilities:

- Multiple IDAES modeling components can be combined to build advanced models
- Optimization ready models - advanced NLP solvers
- Custom Initialization routines
- Compatible with other IDAES/Pyomo tools:
  - Parameter Estimation (Parmest/RIPE/ALAMO)
  - Conceptual Design (PyoSyn)
  - Dynamics and Control (Cappresse)

Contact:

Chinedu Okoli, Chinedu.Okoli@netl.doe.gov, (412)-386-7103

KeyLogic Systems, Inc.'s contributions to this work were funded by the National Energy Technology Laboratory under the Mission Execution and Strategic Analysis contract (DE-FE0025912) for support services.