WRIGHT STATE UNIVERSITY
College of Engineering and Computer Science
Department of
Mechanical and Materials Engineering

Getting Started with Modelica Modeling: A Short Course in OpenModelica and Dakota

Danielle Massé
Wright State University

Requirements for this course:

- Installation of OpenModelica v1.13.2 : https://build.openmodelica.org/omc/builds/windows/releases/1.13/2/64bit/
- Installation of Dakota v6.11 : https://dakota.sandia.gov/download.html
- Windows operating system, or access to Windows Powershell
  - Linux and MacOS users will have to navigate this installation/configuration independently
- Installation of python 3.x : https://www.python.org/downloads/

Agenda

- Fundamentals of the Modelica language
  - Object-oriented modeling concepts
  - Acausal modeling advantages and implications
- OpenModelica
  - OpenModelica source code processing and its implications
- The Modelica Standard Library (MSL)
  - Review of class libraries
  - Explanation of Fluid classes relevant to lab exercises
- Lab exercise prompt and preparation
  - Lab exercise prompt
  - Objectives of example system which will be developed
  - Assumptions made in all lab exercises
  - Basics of PID Control - single and cascade control structures
- Introduction to Dakota
  - Parameter optimization and lab exercise motivation

Fundamentals of the Modelica Language

What is Modelica?

- **Modelica** is a **modeling language** (not a tool) that:
  - Defines all parts of a model and structures the components
  - Has been in development since 1996, with quick application to thermal modeling
    - One of the first Modelica papers published focused on modeling heat exchangers with Modelica
  - Is non-proprietary and object-oriented
  - Has proven to be suitable for modeling complex multiphysical systems
    - E.g. systems with mechanical, electrical, electronic, hydraulic, thermal, control, electric power, or process-oriented subcomponents
  - Can make use of many libraries, both proprietary and open-source
    - The open-source Modelica Standard Library (MSL) contains ~1600 model components in many domains

Object-Oriented Modeling Constructs

- **In Modelica, everything is a class**
  - Classes can also be declared with other keywords, e.g. `model`, `record`, `block`, `connector`, `function`, etc…
  - Each of these keywords instantiates a class with specific restrictions
    - A `record` is a class that only contains data, with no equations
    - A `block` is a class with fixed input-output causality
    - A `model` is a class that cannot be used as a connector class
- As expected, **classes have the typical object-oriented properties – most importantly, inheritance**
  - Instantiated classes can extend existent classes, inheriting variable and parameter declarations, equations, etc…
  - The `extend` keyword is used to instantiate classes with inherited properties
  - Allows for the creation of partial models, which allow for the creation of basic models with common properties and need to be combined with a constitutive equation to be complete

Modeling vs. Programming Language Semantics

Compiled programming language

- e.g. C, C++, Fortran
- Source code is compiled i.e. machine code is generated from source code
- Based on assignment statements
- Input variables must be declared and specified, and can be used in assignment calculations of output variables
- Explicit causality is required before compilation

```
y = 2x + 3 - z
```

Modeling language

- e.g. Modelica, executable UML
- Source code is translated into executable objects that can be exercised by a simulation engine
- Based on equation statements
- Inputs and outputs are determined via equation manipulation by the simulation engine
- Acausal

```
y + z = 2x + 3
```

Advantages of Acausal Modeling

- **Acausal modeling increases the reusability of Modelica classes** because input-output causality is not fixed
  - Unlike some scripting and programming languages, the order of equations or variable declarations/assignments is not crucial
- Example: a resistor *equation*: V = I*R
  - In a language which requires explicit causality, two of the three variables would have to be defined, and the third calculated from them
    - E.g. I = 10, R = 0.5, then can find that V = 5
  - In Modelica, the given resistor equation can be left as-is, while changing the assigned variables
    - If I and R are assigned, then V will be calculated from V = I*R
    - If V and R are assigned, then I will be calculated from I = V/R
    - If V and I are assigned, then R with be calculated from R = V/I

→ The equation can be used in three ways, without having to be re-typed!

OpenModelica

Modelica Modeling and Simulation (M&S) Environments

- Popular M&S environments include text-based and diagram-based editors for defining components and their properties
- All changes made to either editor are also updated in the other
- This flexibility lowers the initial learning curve, and allows for intuitive model development via the diagram-based editor

- Many software distributors have developed proprietary Modelica simulation environments:
  - Dymola (Dassault Systemes)
  - Simplorer/Twin Builder (ANSYS)
  - MapleSim (Maplesoft)
  - Wolfram SystemModeler (Wolfram)
  - SimulationX (ITI-ESI Group)
  - Simcenter Amesim (Siemens)
  - Jmodelica.org (Modelon)

The leading open-source alternative is OpenModelica

OpenModelica

- The GUI consists of 3 main windows
  - Modeling – two tabs within this which allow for text-based or diagram-based modeling
  - Plotting
    - Once a simulation is run, the results will automatically populate within the plotting tab for visualization and/or exportation
  - Debugging
- The Libraries Browser is docked on the left-hand side
  - This allows for easy browsing of libraries, whether they be imported open-source libraries, part of the Modelica Standard Library, or user-defined

Steps Taken by OpenModelica

- Once a model has been developed that has the same number of variables as equations, **how does OpenModelica actually use the source code to eventually solve the equations?**
- It sends the source code through the following steps:
  - **Translator**: Source code is translated into a system of hybrid differential algebraic equations (DAEs)
  - **Analyzer**: The equations are sorted and optimized
  - **Code generator**: from the optimally sorted equations, C code is generated and compiled
  - **C compiler**: The generated C code is compiled
  - **Simulation engine**: Finally, the compiled C code can be exercised

Differential Algebraic Equations

- A system of equations of the form

$$F(t, x, \dot{x}) = 0$$

  is called a differential algebraic equation (DAE) if the Jacobian matrix $\frac{\partial F}{\partial \dot{x}}$ is singular
- Generally, if the Jacobian matrix is non-singular (i.e. invertible) then the system can be transformed into an ordinary differential equation (ODE) of the form $\dot{x} = f(t, x)$
- In engineering, DAEs arise from practical applications
  - Differential equations describing the dynamics of a process, plus
  - Algebraic equations describing
    - Conservation laws
    - Mass, molar, entropy balance equations
    - Desired constraints on the process dynamics

Implications of DAEs in OpenModelica

- Since OpenModelica translates the Modelica source code into DAEs, this imposes some constraints on the resultant matrix system
- For the system of DAEs to be solvable in OpenModelica, it must satisfy determined initialization
  - i.e. **the number of equations and number of variables must always be equal**
  - Under-determined systems cannot be handled by OpenModelica (some other Modelica tools have this capability, e.g. Wolfram SystemModeler)

The Modelica Standard Library (MSL)

Modelica Standard Library (MSL)

- The Modelica Standard Library contains components from various application areas, with ~1600 models
- Many components from this library will be used in the lab exercises and modified for our purposes
- MSL is hierarchical, with nested classes typified using dot notation as:

  Top_Class.Middle_Class.Innermost_Class
- For example, within the Fluid package there is a package named Pipes, and within it there is a StaticPipe model – this would be referred to as:

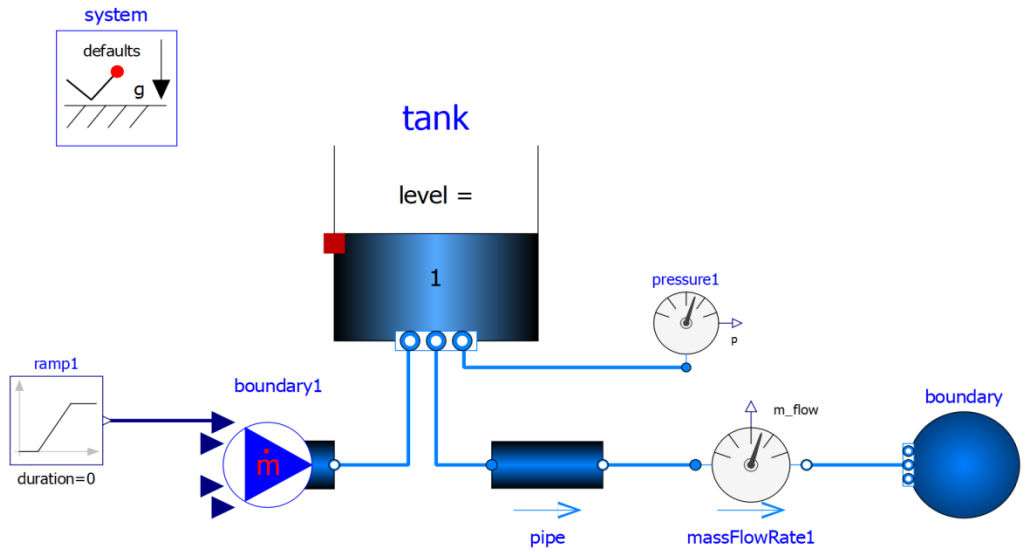
  Modelica.Fluid.Pipes.StaticPipe

MSL Continued

- The following sub-libraries are included in the MSL:
  - Blocks — Library for basic input/output control blocks
  - Constants — Mathematical constants and constants of nature
  - Electrical — Library for electrical models
  - Icons — Icon definition
  - Fluid — 1-dimensional flow in networks of vessels, pipes, valves, etc…
  - Math — Mathematical functions
  - Magnetic — Magnetic.Fluxtubes – for magnetic applications
  - Mechanics — Library for mechanical systems
  - Media — Media models for liquids and gases
  - SIunits — Type definitions based on SI units according to ISO 31-1992
  - Stategraph — Hierarchical state machines (analogous to Statecharts)
  - Thermal — Components for thermal systems
  - Utilities — Utility functions especially for scripting
- In addition to the MSL, there are many specialized libraries (free and proprietary)

Simple Tank Pipe Flow: Explanation of Fundamental Fluid Components

Diagram View



Text View

```
within ThermoFluidsLabLib.Examples;

model SimpleTankPipeFlow
  extends Modelica.Icons.Example;
  package water = Modelica.Media.Water.ConstantPropertyLiquidWater;
  Modelica.Fluid.Vessels.OpenTank tank(redeclare package Medium = water, crossArea = 1.5, height = 1.5, level_start = 1, nPorts = 3,
use_portsData = false)  annotation( ...);
  Modelica.Fluid.Pipes.StaticPipe pipe(redeclare package Medium = water, diameter = 0.1, length = 1) annotation( ...);
  Modelica.Fluid.Sources.FixedBoundary boundary(redeclare package Medium = water, nPorts = 1)  annotation( ...);
  Modelica.Fluid.Sensors.MassFlowRate massFlowRate1(redeclare package Medium = water) annotation( ...);
  inner Modelica.Fluid.System system(allowFlowReversal = false, m_flow_start = 0.06)  annotation( ...);
  Modelica.Fluid.Sources.MassFlowSource_T boundary1(redeclare package Medium = water, nPorts = 1, use_m_flow_in = true)  annotation( ...);
  Modelica.Blocks.Sources.Ramp ramp1(duration = 0, height = 0, offset = 1, startTime = 0)  annotation( ...);
  Modelica.Fluid.Sensors.Pressure pressure1(redeclare package Medium = water) annotation( ...);
equation
  connect(tank.ports[3], pressure1.port) annotation( ...);
  connect(ramp1.y, boundary1.m_flow_in) annotation( ...);
  connect(boundary1.ports[1], tank.ports[1]) annotation( ...);
  connect(massFlowRate1.port_b, boundary.ports[1]) annotation( ...);
  connect(pipe.port_b, massFlowRate1.port_a) annotation( ...);
  connect(tank.ports[2], pipe.port_a) annotation( ...);
  annotation( ...);
    end SimpleTankPipeFlow;
```

Lab Exercise Prompt and Preparation

Lab exercise prompt

- Consider the water loop of an industrial system. In this loop, water flows into a tank at a dynamic rate then out through a pipe and valve to some other components in the system
  - the rest of the system is not our concern – only modeling and evaluating different implementations of control within the water loop
- To maintain proper pressure in the larger system, it has been determined that a level of 0.6m needs to be maintained in the water tank.
- Even though the flow rate of water into the tank can change, the objective is to create the water loop and test different control systems.
  - This means that the control system ultimately used needs to be able to respond well to changes in the incoming water flow rate.
- The overall goal is to maintain the level in the water tank with ±0.1m tolerance.

Assumptions For All Models Developed in Lab Exercises

- All of these assumptions are made in the Modelica.Fluid.System component which dictates system-wide properties and assumptions
- Energy and mass both have a dynamic balance, with a guess for the initial values
- Momentum has a steady state balance, with a guess for the initial value
- No flow reversal is allowed – this restricts the flow from upstream to downstream ports throughout the models
- Ambient conditions are:
  - Atmospheric pressure (p = 1.01325 bar)
  - Temperature of 20°C
  - These conditions are used for to initialize the systems
- Fluid library components rely on a medium model
  - A medium is what will move through the fluid model
  - For all lab exercises, we will use constant property liquid water (i.e. incompressible)
- Default mass flow rate is 0.06 kg/s
  - This will be used as a start value for media source components if no other initialization condition is provided

Control

- Generally, control systems are implemented to achieve desired behavior in a dynamic process
- These systems use sensors and detectors to measure process variables of interest within the process being controlled.
- The measurements taken are then used to provide corrective feedback to achieve the desired process behavior
  - Essentially, the control system will drive the process variable toward a desired value or range of values
- There are many methods of control; far too many to review in this course
- For the scope of the lab exercises, we will explore PID control systems

PID Control

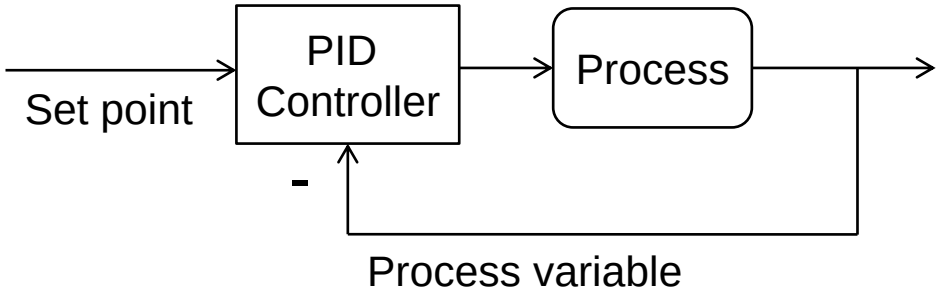
- A proportional-integral-derivative (PID) controller is a control loop mechanism used in industrial systems for continuously modulated process control
- These types of controllers require two signals:
  - A set point (SP) – the desired behavior
  - A process variable (PV) – the measured behavior
- PID controllers continuously calculate the error, $e(t) = SP - PV$, and apply a correction $u(t)$ as:

$$u(t) = k\left(e(t) + \frac{1}{T_i}\int_0^t e(t')dt' + T_d\frac{de(t)}{dt}\right)$$

where:

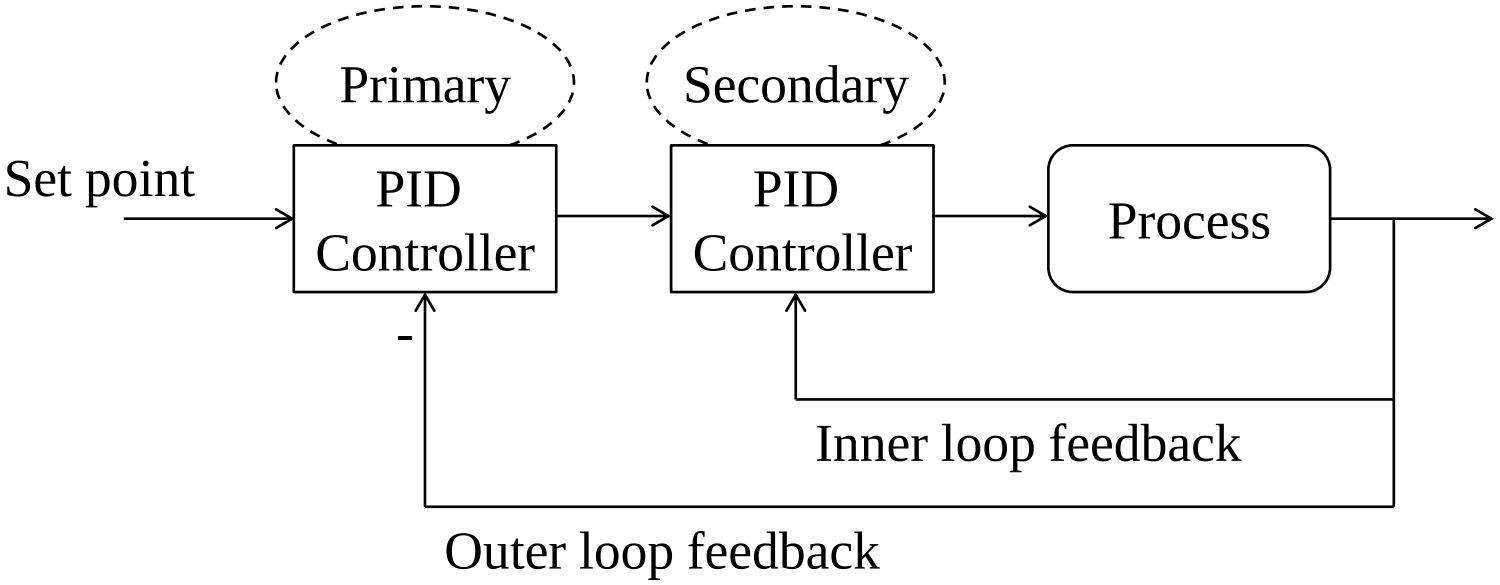
- k is the proportional coefficient
- T_i is the integral time coefficient
- T_d is the derivative time coefficient

These parameters are set in OpenModelica and changing them changes controller response



Cascade Control

- Cascade control is a control method which combines two feedback loops, using the output of one controller as the set point of a second controller
- This method is capable of improving response of the control system to disturbances
  - Generally, the **primary controller responds to measurements of a low frequency (slowly changing) process variable**
  - The **secondary controller responds to measurements of a high frequency (rapidly changing) process variable**



Single and Cascade PID Control for Maintaining Tank Level

- **Single loop control system**
  - the controller responds to the measured level in the tank
  - feedback controls an action (valve actuation) which dictates the rate at which fluid leaves the tank
    - depending on the cross-sectional area of the tank, changes in the tank level can happen quite slowly
    - there can be considerable delay before the level changes enough to initiate a correction
- **Cascade control system**
  - the primary controller still responds to the measured level in the tank, with the feedback now establishing the set point for flow rate of fluid leaving the tank
  - the secondary controller responds to measured flow rate, and its feedback now controls an action (valve actuation) which dictates the rate at which fluid leaves the tank
    - flow level loop now responds quickly to flow disturbances
    - reduces level fluctuations that can occur with single loop control

Introduction to DAKOTA

Parameter Optimization and Dakota

- How will we determine the (optimal) controller parameters?
  - In the proposed lab exercise, each PID controller has three coefficients which need to be determined for proper control of the tank level to be achieved
  - While iteratively changing the values of the coefficients and comparing results can lead to arguably good parameter values, many iterations will be needed to identify the best values
  - A better way to determine the coefficient values is by conducting a parameter search over an allowable range
  - To conduct such a parameter search, we will use another software called Dakota
- Dakota is an open-source software suite developed by Sandia National Laboratories which implements robust algorithms for optimization and uncertainty quantification
- The methods available through Dakota can be applied to any model which has an executable application format (.exe) and uses input/output files for parameters assignment and results

What can Dakota do?

- Dakota has implementations of many useful methods, including:
  - Parameter studies
    - Vector, list, centered, and multidimensional
  - Design of experiments
    - DDACE (design and analysis of computer experiments), sensitivity analysis, and many more developed by Sandia
  - Uncertainty quantification
    - Sampling, reliability, stochastic expansion, importance sampling, adaptive sampling, epistemic nondeterministic, and Bayesian calibration methods
  - Optimization
    - Gradient-based local, derivative-free local, multi-objective, and automatic scaling methods
  - Nonlinear least squares
    - Gauss-Newton, NLSSOL, NL2SOL

Lab exercises with Dakota

- For the purposes of the lab exercises, Dakota will be used to conduct a variety of parameter studies
- While the results are interesting, the real objective is to show how Dakota uses an input file, executable, output file, and driver to conduct a series of simulations
  - A driver in this context is a script file which interfaces between the model executable and its inputs/outputs
    - It creates modified input files for each simulation run, runs them, and collects the results from each run in a master summary results file
    - For the lab exercises, the driver is written in PowerShell (other scripting languages like python could also be used)
  - This can be done regardless of what software is used to create the executable
  - Even for a straightforward parameter study where all tested values are explicit and known, using Dakota saves time
  - The driver used to set up and execute the desired simulations can be adapted to calculate and report any desired metrics

Key References

- For more Modelica/OpenModelica information and exercises, refer to:
  - *Principles of Object Oriented Modeling and Simulation with Modelica 3.3* by Peter Fritzson, 2014 with accompanying short course material at:

    https://www.openmodelica.org/doc/ModelicaShortCourse/ModelicaTutorial-slides-PeterFritzson-160202-BT.pdf
  - *Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica* by Peter Fritzson, 2011
  - OpenModelica user documentation (includes tutorials):

    https://www.openmodelica.org/useresresources/userdocumentation

- For more information about Dakota, refer to:
  - Dakota user manual: https://dakota.sandia.gov/content/manuals