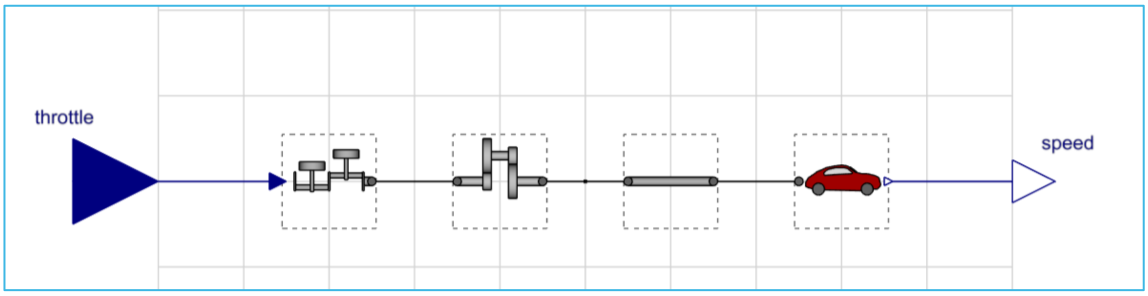
# WORKSHOP 2.1

Hierarchical modeling in Impact

## Contents

## Introduction

In this workshop, you will learn about the concepts of **templates** and **interfaces**.



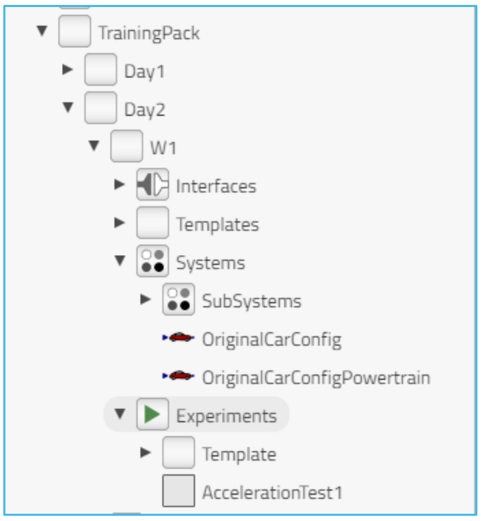
You will:

- Reconfigure an existing system using a new architecture.
- Reuse the **ElasticShaft** (from Workshop 1.3) to configure a compliant powertrain.
- Create a new template for an electric powertrain.
- Create tests and run analysis on the systems.

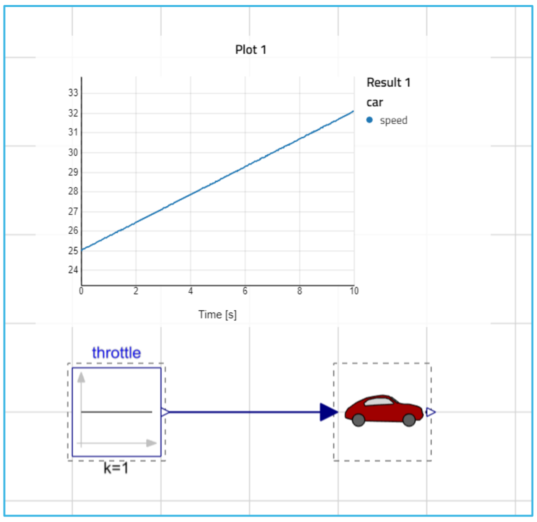
## Exploring the model package

A model package for workshop 3 has been provided to you.

1. Open the package **TrainingPack.Day2.W1 (**from here on referred to as **W1)** and inspect the library content. It will have a structure like that shown in the figure below.



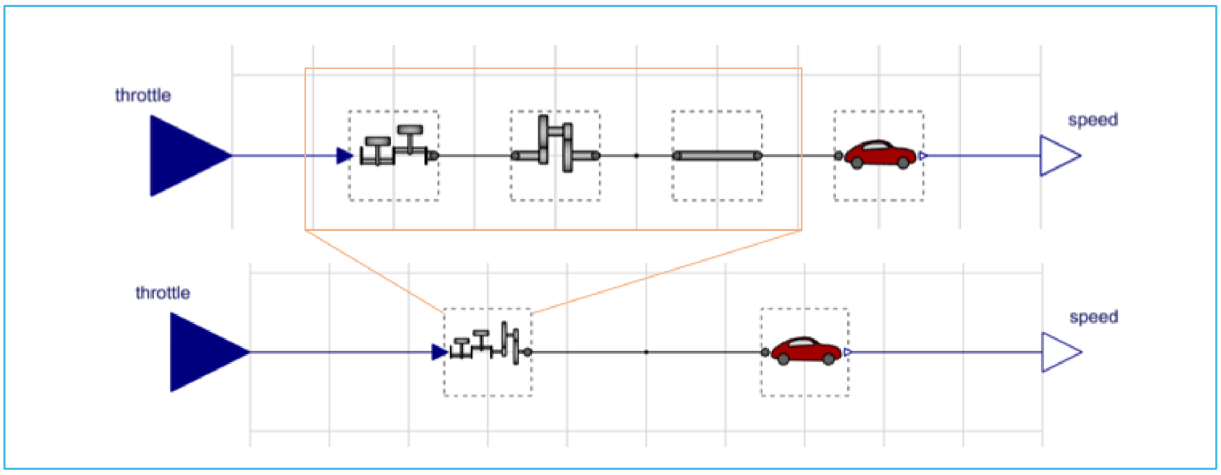
2. Go to **W1.Experiments.AccelerationTest1** and run the simulation. This experiment runs the car at full throttle. Set the simulation **Stop Time**=10s and plot the car speed (***car.speed***).



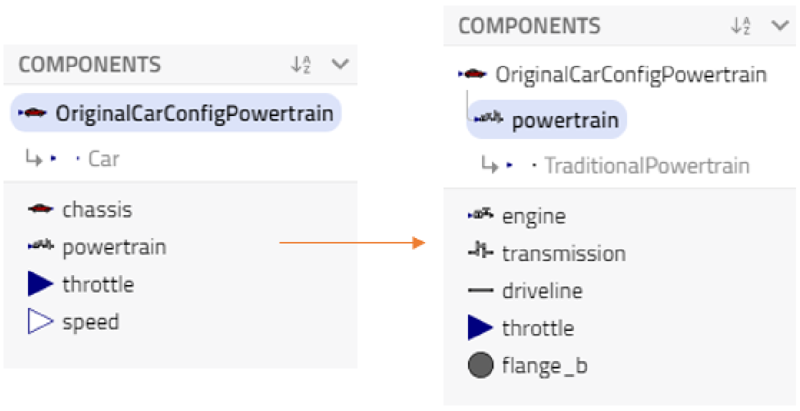
## Redesigned powertrain template

Now let us inspect a new powertrain model architecture. The idea is to have an easy way to change the whole powertrain in one go and be flexible enough to accommodate several different types. A conventional powertrain contains a completely different set of subsystems than an electric or hybrid system.

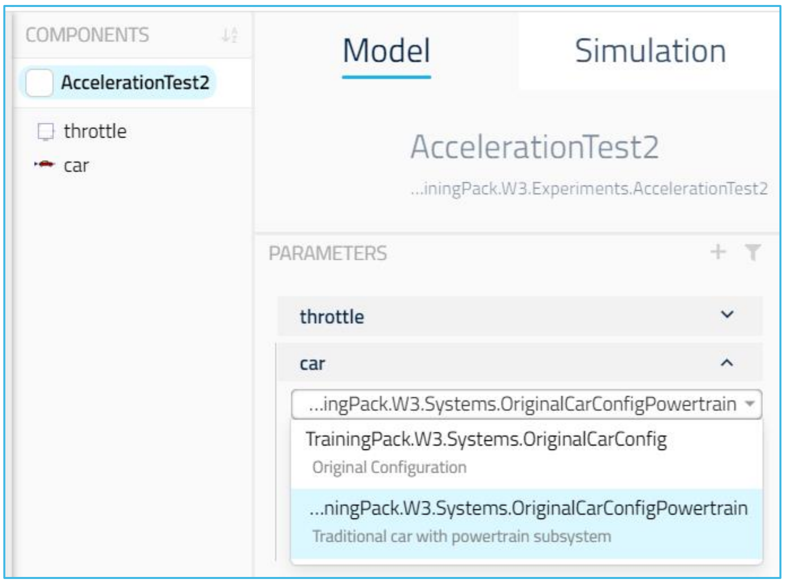
1. Open the **W1.Systems** package and compare models **OrigianlCarConfig** and **OriginalCarConfigPowertrain.** These system variants are based on different templates found in the **Templates** package.



2. Go to the model **W1.Systems.OriginalCarConfigPowertrain** and inspect the component browser.



3. Double click the **powertrain** instance, to go down one hierarchical layer in the instance tree. Here you can inspect and see that this model contains the same subsystems as the **OrigianlCarConfig** example. The new architecture is only a refactoring of the same interfaces used earlier. Now let us test the new architecture.
4. Open the **W1.Experiments** package, duplicate the model **AccelerationTest1** and rename it to **AccelerationTest2.**

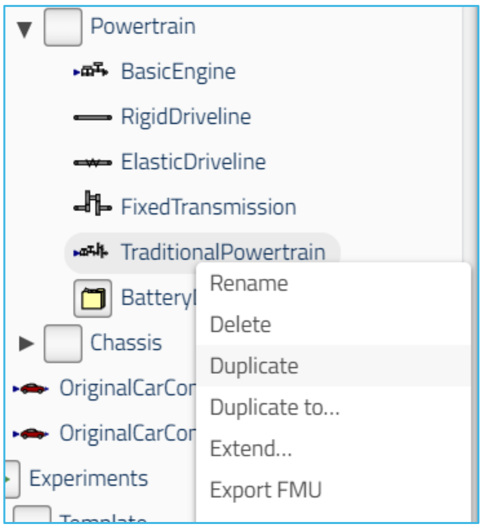

COMPONENTS
AccelerationTest2
throttle
car
Model
Simulation
AccelerationTest2
...iningPack.W3.Experiments.AccelerationTest2
PARAMETERS
throttle
car
...ingPack.W3.Systems.OriginalCarConfigPowertrain
TrainingPack.W3.Systems.OriginalCarConfig
Original Configuration
...ningPack.W3.Systems.OriginalCarConfigPowertrain
Traditional car with powertrain subsystem

5. Change the **car** component to use the new architecture.
6. Simulate the model and plot the car speed and verify that you get the same results.

## Compliant powertrain

In this part of the workshop, we will use the elastic shaft component developed in Workshop 2 and create a powertrain model with compliance.

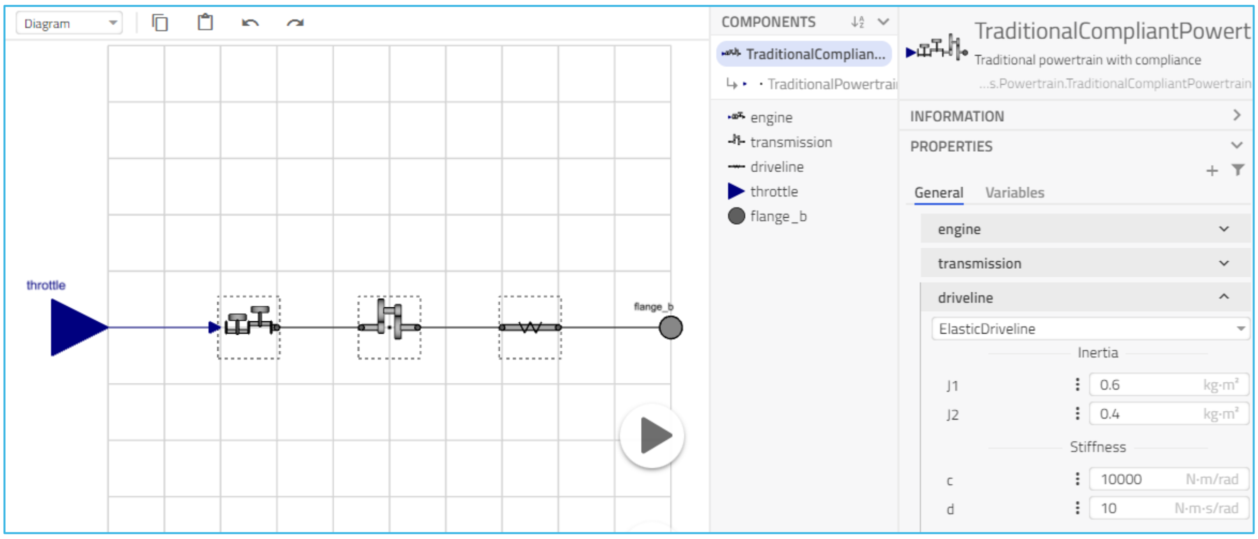
1. Open **W1.Systems.SubSystems.Powertrain** and look at the **ElasticDriveline** model. This is the same model as created before, but it also uses an interface class for driveline components.
2. To create a new variant, we have two optional methods to consider: either we extend the **W1.Systems.SubSystems.Powertrain** template class and populate the template, or we duplicate an existing variant and change the driveline component.
3. Since the **TraditionalPowertrain** has almost everything we need, lets duplicate that one.



Right click, and **Duplicate**. Rename it to **TraditionalCompliantPowertrain**.

4. Open the code editor and change the model description to "Compliant powertrain." Save the changes.

5. Go to the parameter dialog and change the ***driveline*** component to use the **ElasticDriveline.**

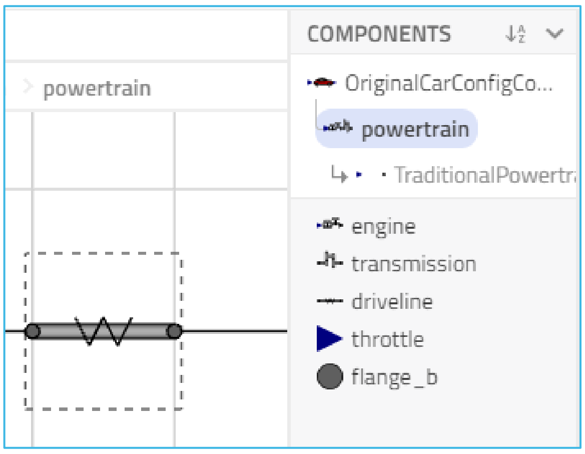


driveline: ElasticDriveline

Inertia
J1: 0.6 kg·m²
J2: 0.4 kg·m²

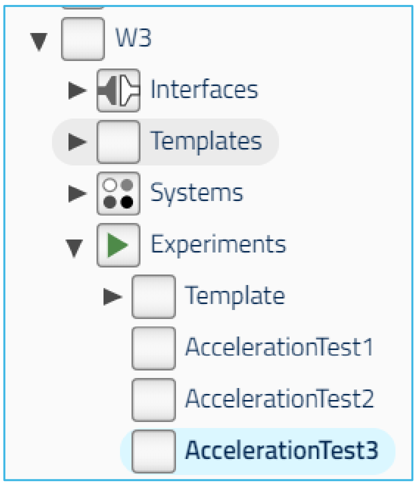
Stiffness
c: 10000 N·m/rad
d: 10 N·m·s/rad

6. Now that we have a compliant driveline, lets create a vehicle that uses it. Duplicate **W1.Systems.OriginalCarConfigPowertrain** and call it **OriginalCarConfigCompliantPowertrain.**
7. Configure it to use the **CompliantPowertrain** by redeclaring the ***powertrain*** subsystem. You can verify that its correctly configured, by browsing down the component browser. If it is correct, you will see the elastic shaft.

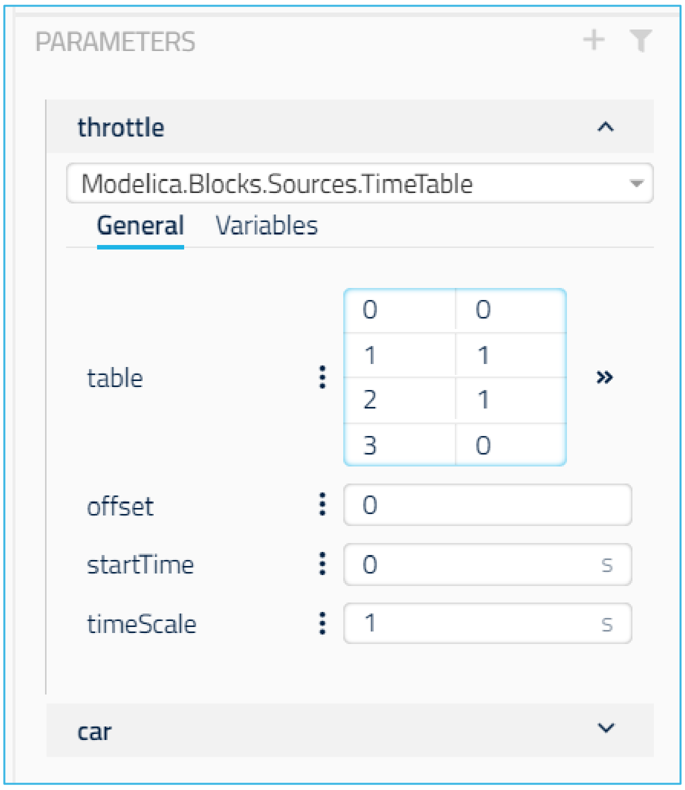


Let us take it for a spin! Now we are going to create an experiment that tests the influence of the compliance. We will give a series of acceleration inputs and study the vibrations in the shaft.

8. Go to the **Experiments** package and **Duplicate** one of the previous tests, and name it **AccelerationTest3.**



9. Change the car component to use the compliant variant.
10. To generate the desired input signal, instead of the constant input, we will use the **Modelica.Blocks.Sources.TimeTable.** You can copy paste the following table:
`{{0,0},{1,1},{2,1},{3,0}}`
Then you should get the following:



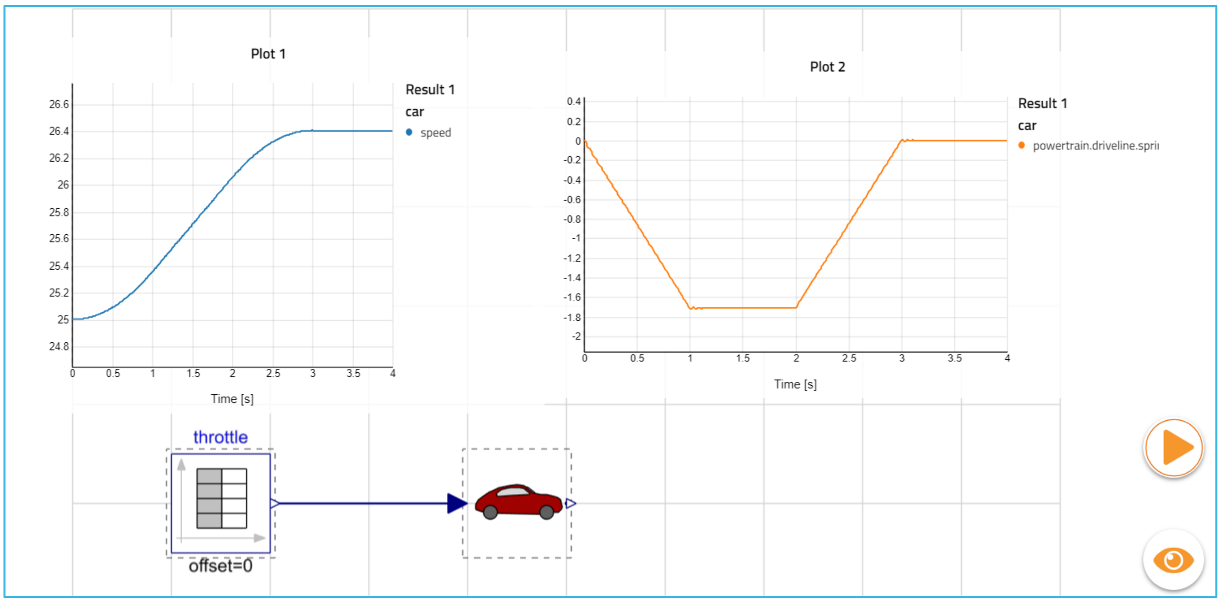
Modelica.Blocks.Sources.TimeTable

table:

| | |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 1 |
| 3 | 0 |

offset: 0
startTime: 0 s
timeScale: 1 s

11. The model is already initialized at v=25m/s. This input will test going from 25m/s cruise, ramping up to full throttle then releasing the gas again. Simulate the model for **4s** and plot ***car.speed*** and ***powertrain.driveline.springDamper.phi_rel***. You can zoom in and look at the vibrations.



## Electric powertrain

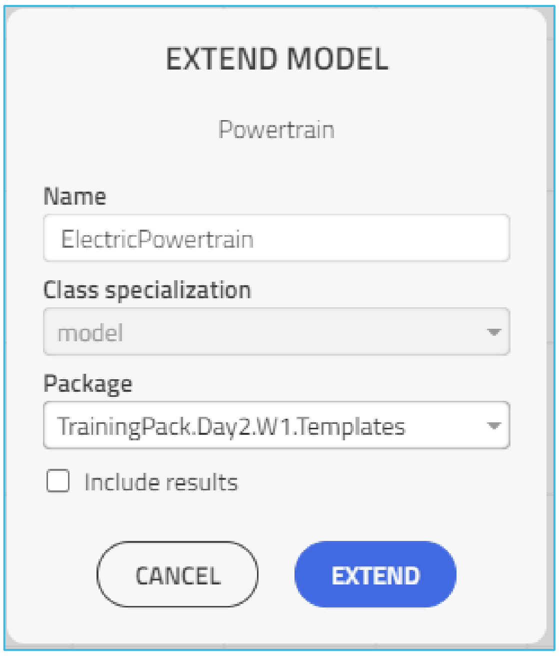
In the following part of the workshop, you will create a very simple electric powertrain with a single electric machine and a battery pack. To test the vehicle, you will create two experiments, one AccelerationTest, and one DriveCycle to analyze the range and battery performance.

You will create the following:

- Template for ElectricDriveline
- A data record for the battery
- A variant of the Electric Driveline
- Electric car variant
- DriveCycle experiment

To create the template, we start with the right template interface classes.

1. **Extend** the interface class **W1.Interfaces.Powertrain** and save it in the **Templates** package. Name it **ElectricPowertrain**.


EXTEND MODEL

Powertrain

Name: ElectricPowertrain

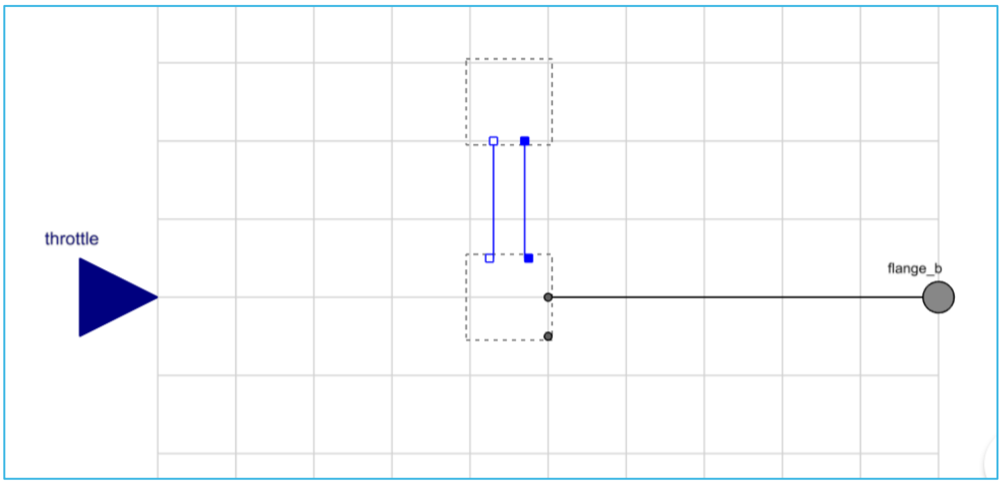
Class specialization: model

Package: TrainingPack.Day2.W1.Templates

☐ Include results

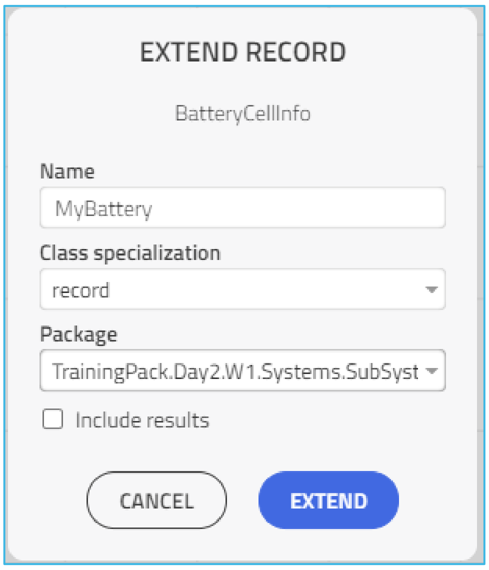
2. Add the following interface components for the electrical machine and the battery:
- **Modelon.Electrical.EnergyStorage.Interfaces.Base**
- **Modelon.Electrical.Machines.Interfaces.BaseDC**

3. Connect them like this:



Now we will create the **ElectricPowertrain** variant, but to parametrize the battery we need a data record, so we will create that first.

4. Extend the data record class **Modelon.Electrical.EnergyStorage.Components.CellInfo.BatteryCellInfo,** call it **MyBattery** and place it in the battery data folder **W1.Systems.SubSystems.Powertrain.BatteryData.** (See below)


EXTEND RECORD

BatteryCellInfo

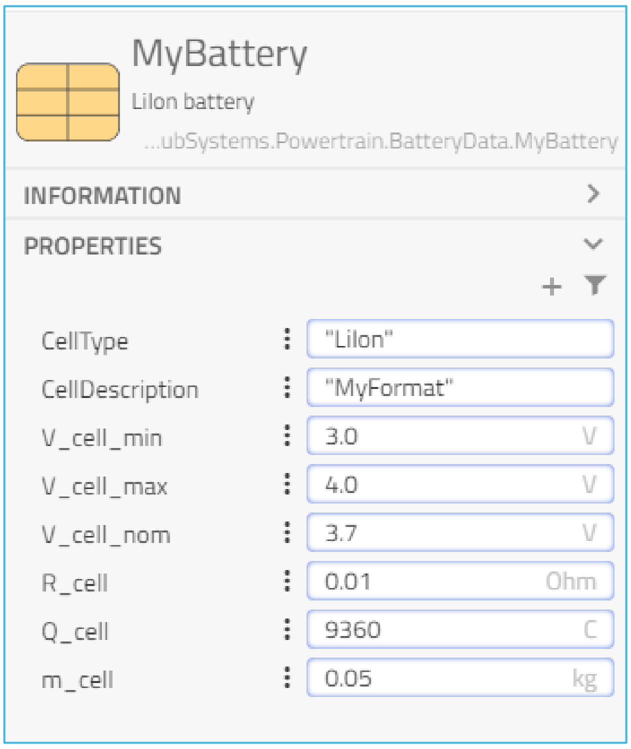
Name: MyBattery

Class specialization: record

Package: TrainingPack.Day2.W1.Systems.SubSyst

☐ Include results

5. Add the following data to the record.


MyBattery

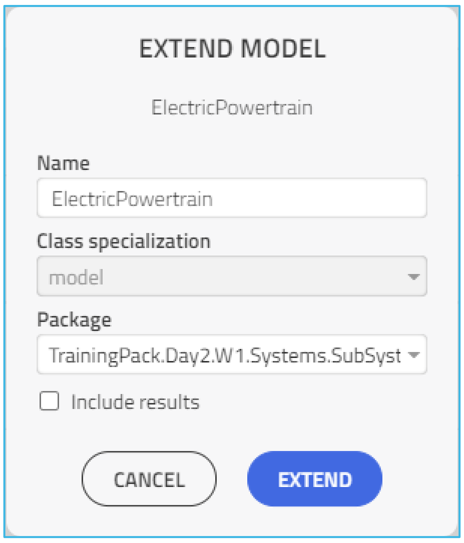
LiIon battery

...ubSystems.Powertrain.BatteryData.MyBattery

| | | |
|---|---|---|
| CellType | "LiIon" | |
| CellDescription | "MyFormat" | |
| V_cell_min | 3.0 | V |
| V_cell_max | 4.0 | V |
| V_cell_nom | 3.7 | V |
| R_cell | 0.01 | Ohm |
| Q_cell | 9360 | C |
| m_cell | 0.05 | kg |

Now that we have the data ready, lets create the **ElectricPowertrain** variant.

6. Start by extending the new **ElectricPowertrain** template and place it in the package **W1.Systems.SubSystems.Powertrain**. Name the subsystem **ElectricPowertrain**.


EXTEND MODEL

ElectricPowertrain

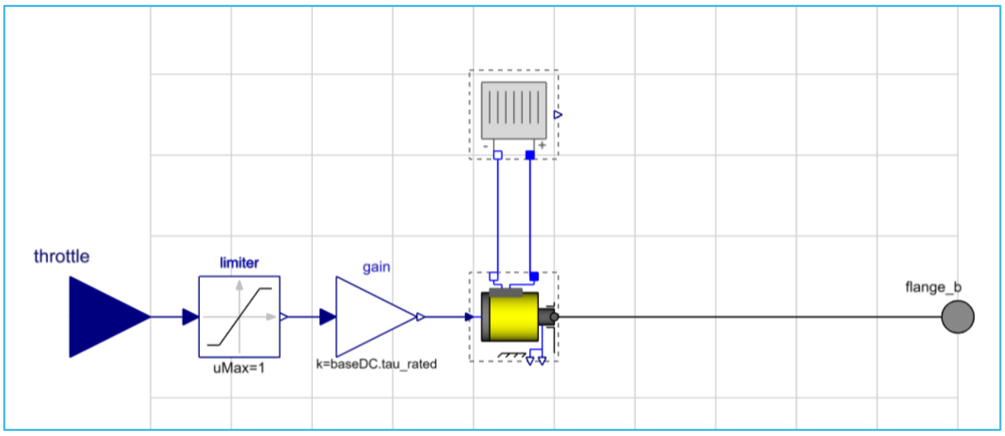
Name: ElectricPowertrain

Class specialization: model

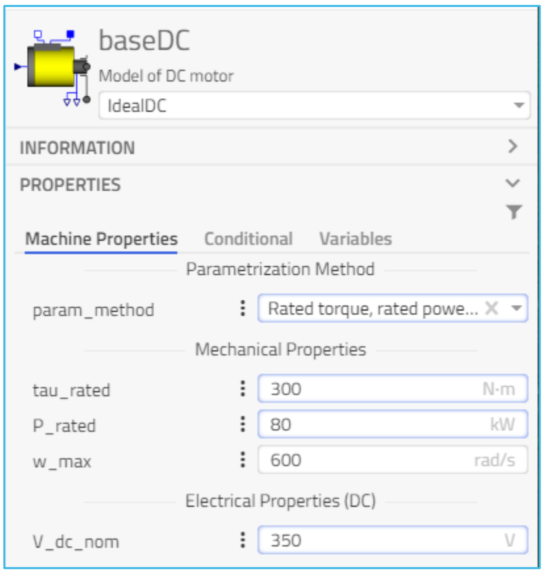
Package: TrainingPack.Day2.W1.Systems.SubSyst

7. To complete the variant, we need to do the following steps:
   - Choose an ElectricMachine and parametrize it.
   - Choose a Battery and parameterize it.
   - Limit and amplify the throttle input.

   The model assumes that the throttle input is between 0 and 1. To assert that the user does not make an error, we place a limit to the input. Likewise, the Electrical machine assumes torque input, which will be between 0 and max torque output of the machine. The result after the upcoming steps should look like this:



   a. Start with the electrical machine. Redeclare it and choose the **IdealDC** version and give it the following parameters:


baseDC

Model of DC motor

IdealDC

INFORMATION

PROPERTIES

Machine Properties | Conditional | Variables

Parametrization Method

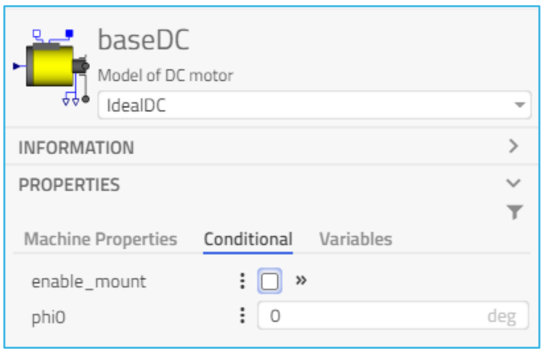
| | |
|---|---|
| param_method | Rated torque, rated powe... |

Mechanical Properties

| | |
|---|---|
| tau_rated | 300 N·m |
| P_rated | 80 kW |
| w_max | 600 rad/s |

Electrical Properties (DC)

| | |
|---|---|
| V_dc_nom | 350 V |


baseDC

Model of DC motor

IdealDC

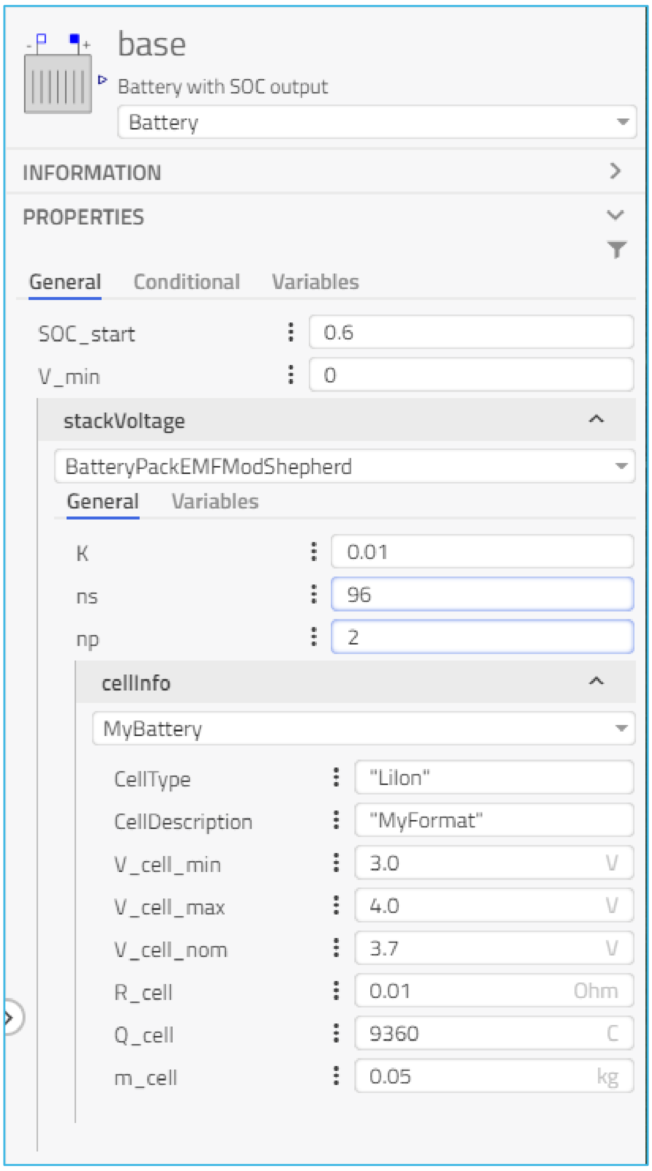
INFORMATION

PROPERTIES

Machine Properties | Conditional | Variables

| | |
|---|---|
| enable_mount | ☐ |
| phi0 | 0 deg |

b. Redeclare the battery and choose "**Battery with SOC output**". Set ***ns*** = 96 and ***np*** = 2 and change the ***cellInfo*** data input to **MyBattery**.



base

Battery with SOC output

Battery

INFORMATION

PROPERTIES

General | Conditional | Variables

SOC_start: 0.6

V_min: 0

stackVoltage

BatteryPackEMFModShepherd

General | Variables

K: 0.01

ns: 96

np: 2

cellInfo

MyBattery

CellType: "LiIon"

CellDescription: "MyFormat"

V_cell_min: 3.0 V

V_cell_max: 4.0 V

V_cell_nom: 3.7 V

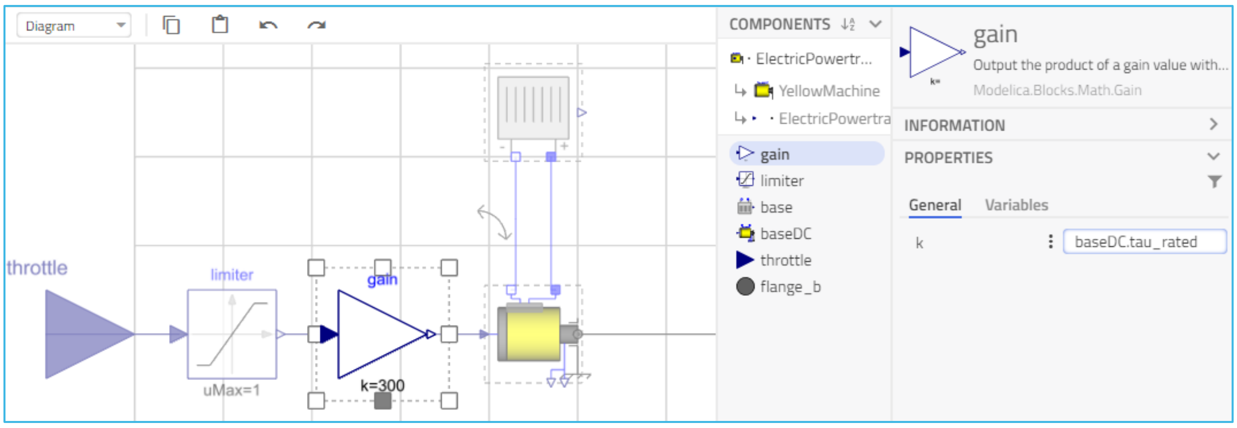
R_cell: 0.01 Ohm

Q_cell: 9360 C

m_cell: 0.05 kg

c. To add the limiter and gain, we can reuse the implementation used in the standard engine model. Go to the **W1.Systems.SubSystems.Powertrain.BasicEngine** and copy the two components, and paste them into the ElectricPowertrain, and connect them.

d. We need to change the parameter for the gain. The maximal torque the machine can give, is stored in the parameter **tau_rated** in the machine model. To set the gain, use component referencing. When you start to type, you can use autocompletion to browse the model hierarchy and find the parameter.

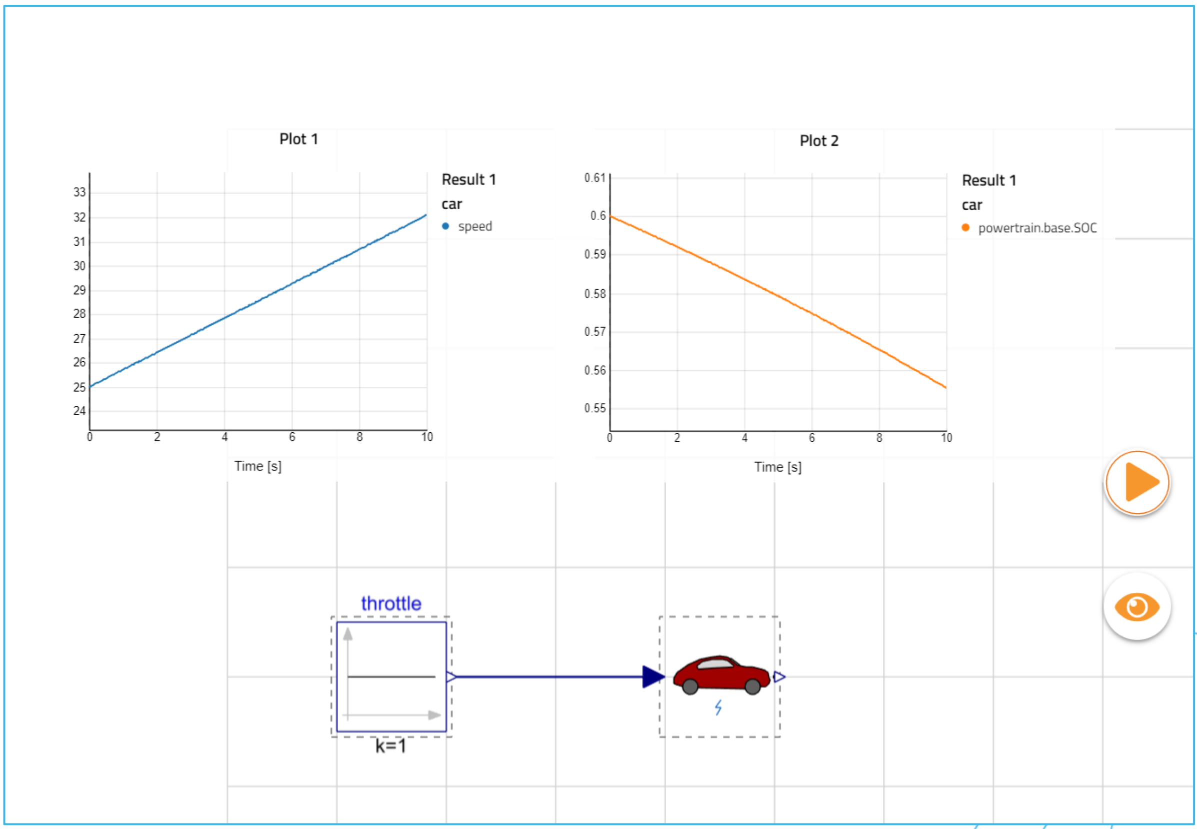


Now we have completed the creation of a new ElectricPowertrain. Let us create the electric vehicle.

8. Duplicate **W1.Systems.OriginalCarConfigPowertrain** and call it **ElectricCar.** Redeclare the Powertrain subsystem to use the **ElectricPowertrain.**

Let us take it for a spin!

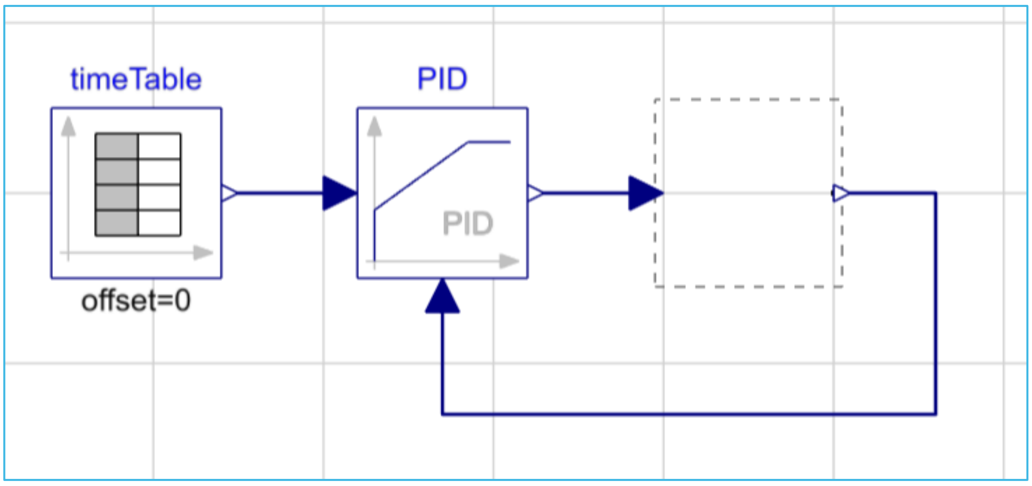
9. Duplicate the AccelerationTest2, call it AccelerationTest4 and replace the car with the electric car.
10. Run the simulation for **10s** and plot ***car.speed*** and ***powertrain.base.SOC***.



As we can see, the SOC is decreasing quite rapidly, so the current battery and electrical machine sizing is not well suited and needs to be redesigned.

Instead of giving direct throttle input, drive cycles can be described by speed profiles.

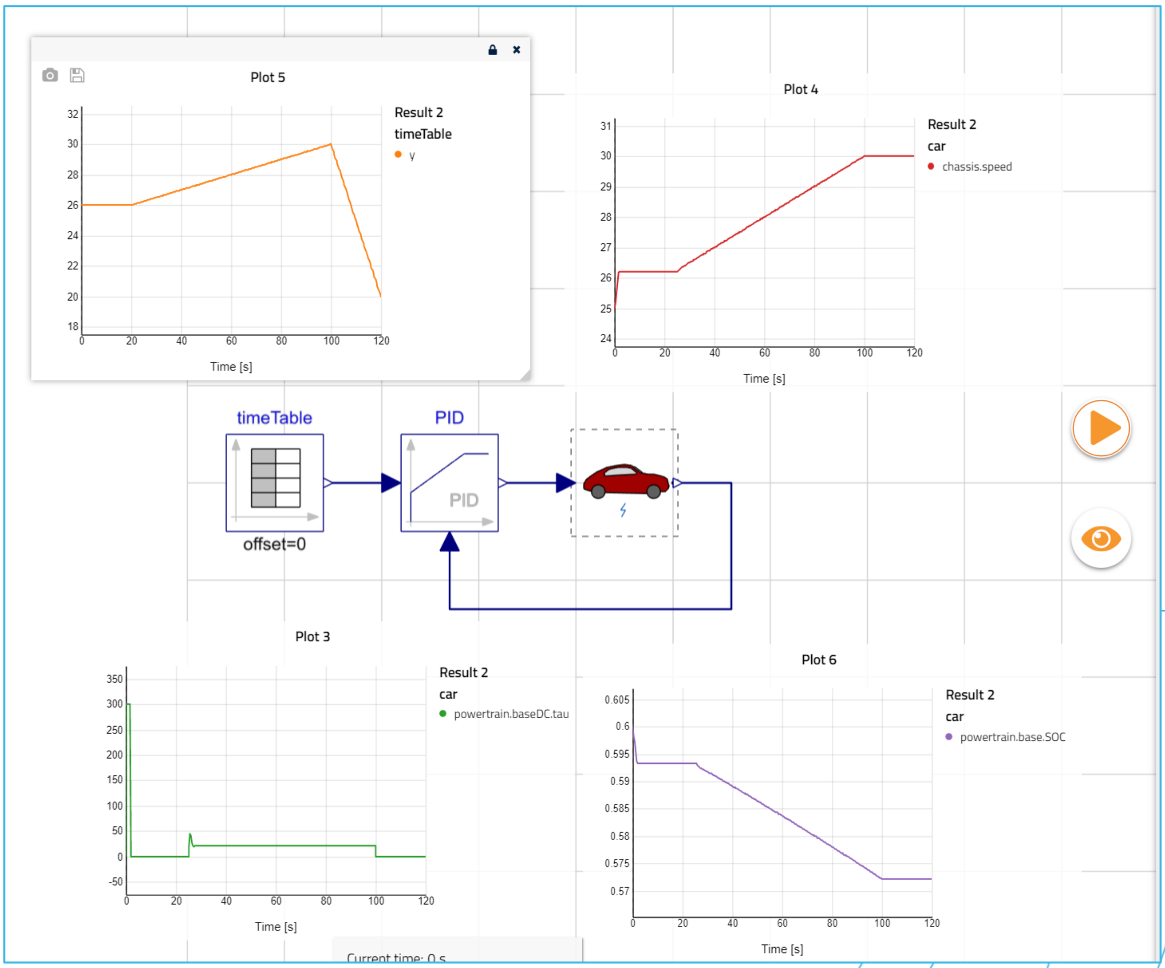
11. In the **W1.Experiments.Template** package, you can find a very simple **DriveCycleTest.**



12. Extend the template and insert the **ElectricCar**.

The controller represents the driver, given the instructions to follow a specified velocity profile given by the timetable input.

13. Run the simulation for **120s** and plot the timetable output, the car speed, the battery SOC, and the ElectricPowertrain torque output.



At *t=100s*, the driver is given the instruction to decelerate to 20m/s, but in the current model there is no possibility to brake. That would require additional modeling effort.

**This concludes workshop 2.1. Well done!**