HIERARCHICAL SYSTEM MODELING

Lecture 2.1

Modelon

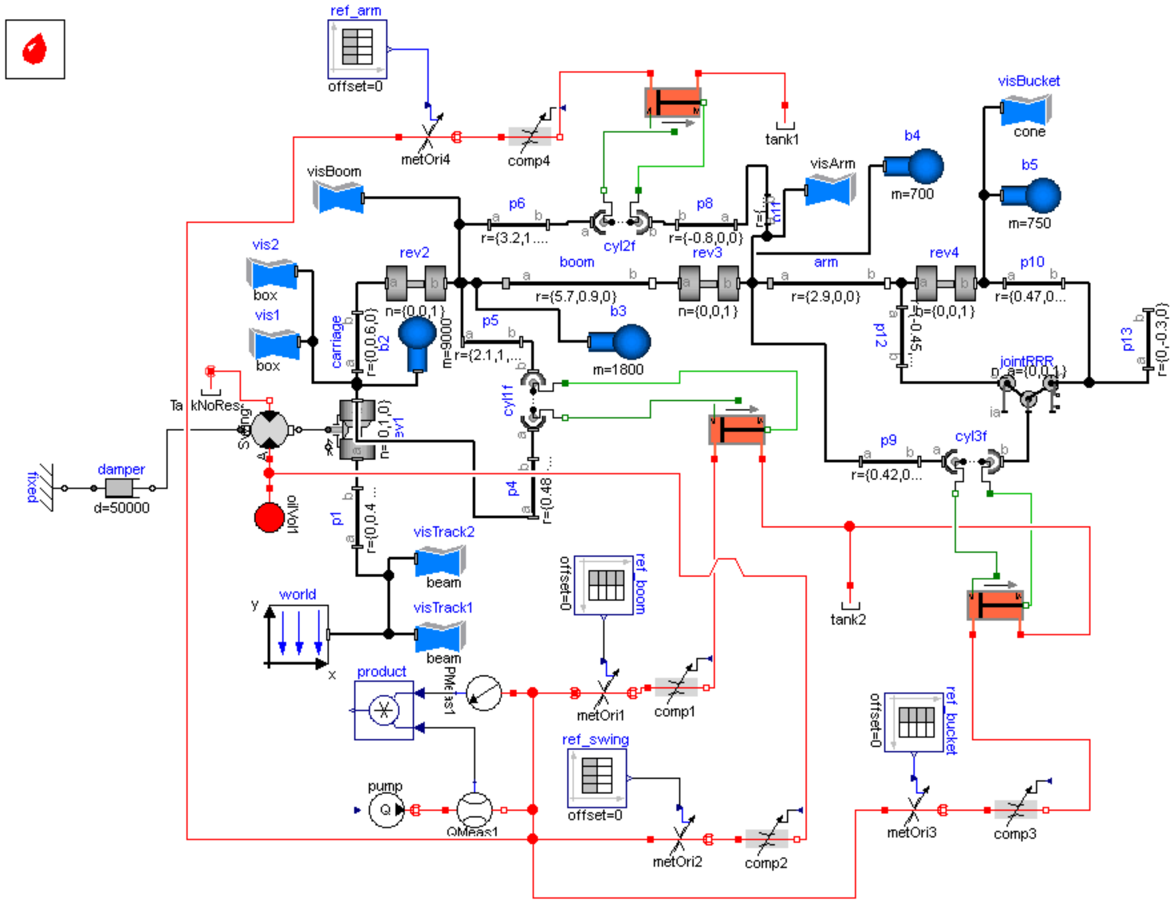
OVERVIEW

- Benefits with hierarchical models
- Library Architecture and Model Structure
- Browse model hierarchy
- Parameter propagation and modifiers
- Reconfigurable models
- System stickies and views

BENEFITS WITH HIERARCHICAL MODELS

FLAT VS. STRUCTURED MODEL

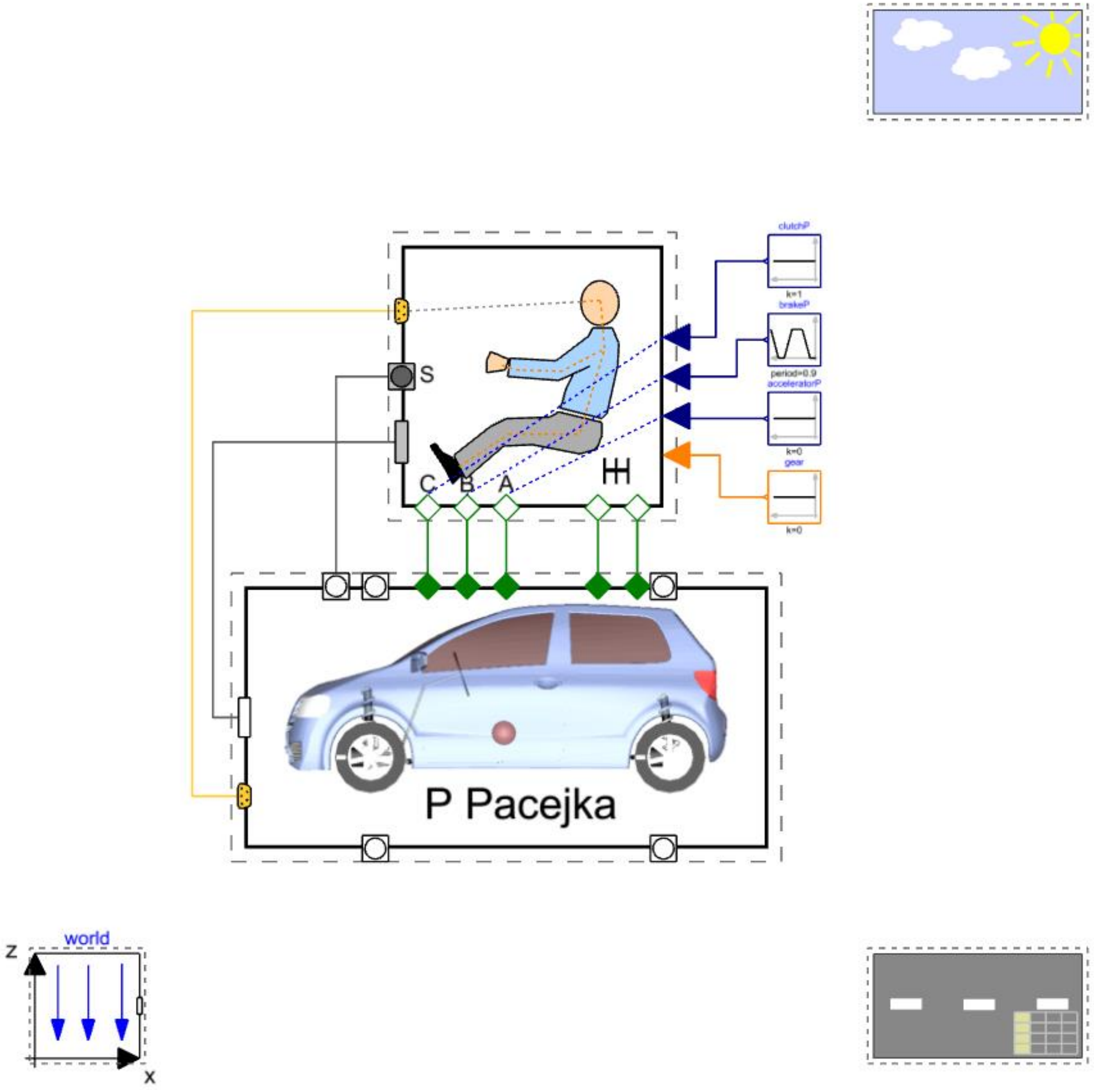
- Example of a hydraulic excavator
  - flat structure
  - one level for all components
  - no structural distinction between different domains as well as experiment boundary conditions
  - difficult to quickly understand the basic idea of the model for someone who has not created it



FLAT VS. STRUCTURED MODEL

Example of a driver - passenger car experiment

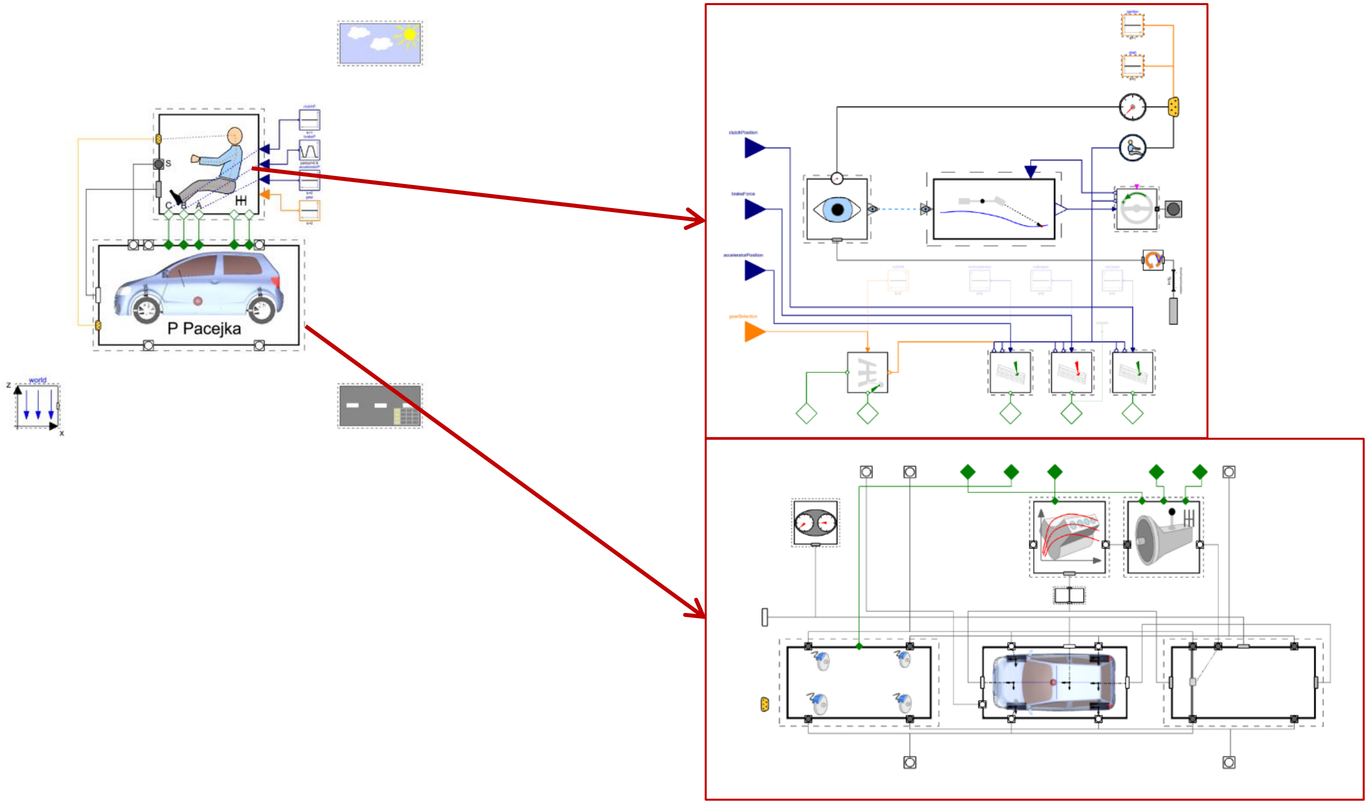
- Modular system decomposition makes reuse and exchange of system parts easier.



FLAT VS. STRUCTURED MODEL

Example of a driver - passenger car experiment

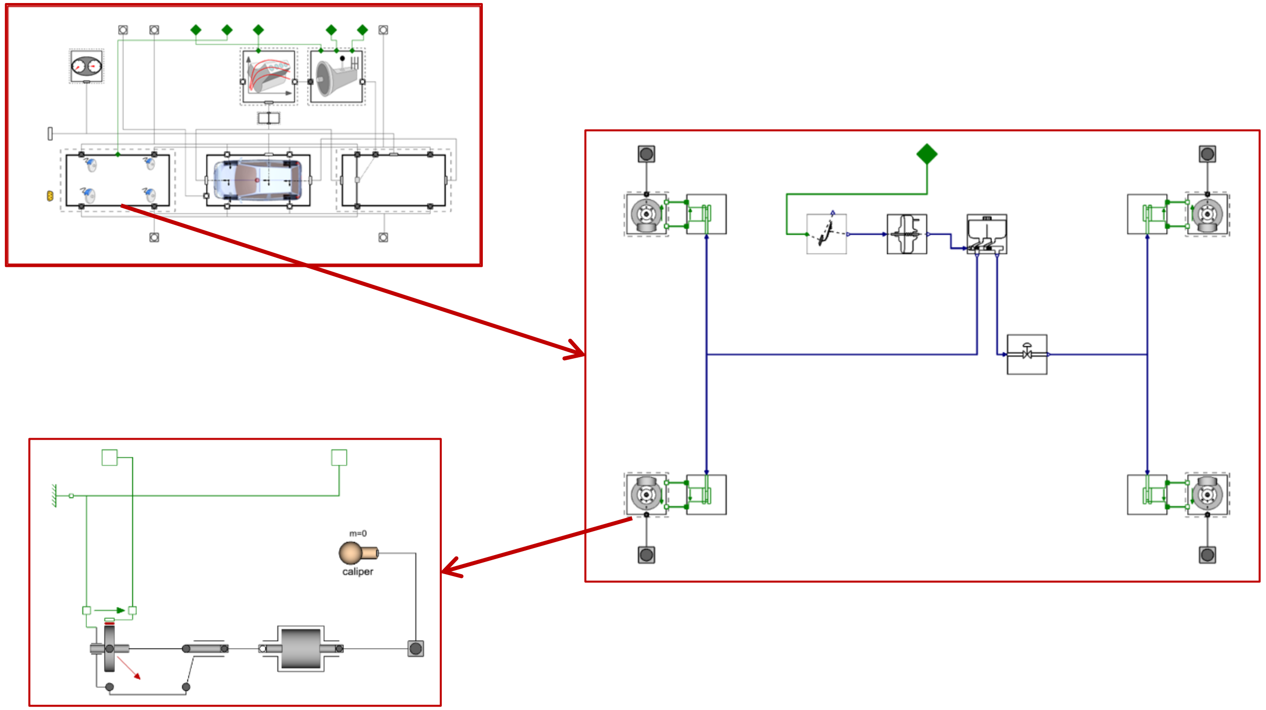
- Modular system decomposition makes reuse and exchange of system parts easier.
- Individual components are grouped into subsystems.



FLAT VS. STRUCTURED MODEL

Example of a driver - passenger car experiment

- Modular system decomposition makes reuse and exchange of system parts easier.
- Individual components are grouped into subsystems.
- Physical interaction comprehensible from the graphical view.



Modelon

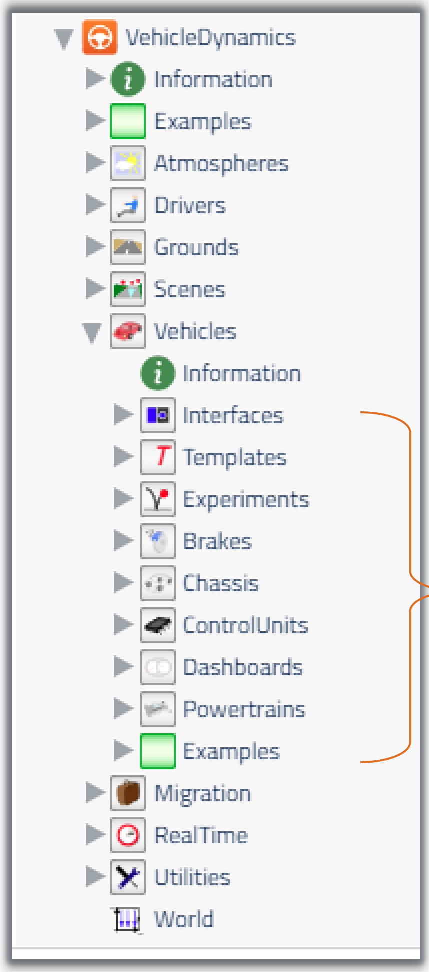
LIBRARY ARCHITECTURE AND MODEL STRUCTURE

FEATURES OF A GOOD MODEL LIBRARY

- If a large complex system model can be used and understood by someone other than the person who modeled it
  - The structure of the real-world system is reflected
  - Physical component interactions can be recognized easily in the graphical view
  - The system model can be graphically reconfigured and adapted to different boundary conditions

- Model classes are easy to find
  - Well organized package structure with descriptive names

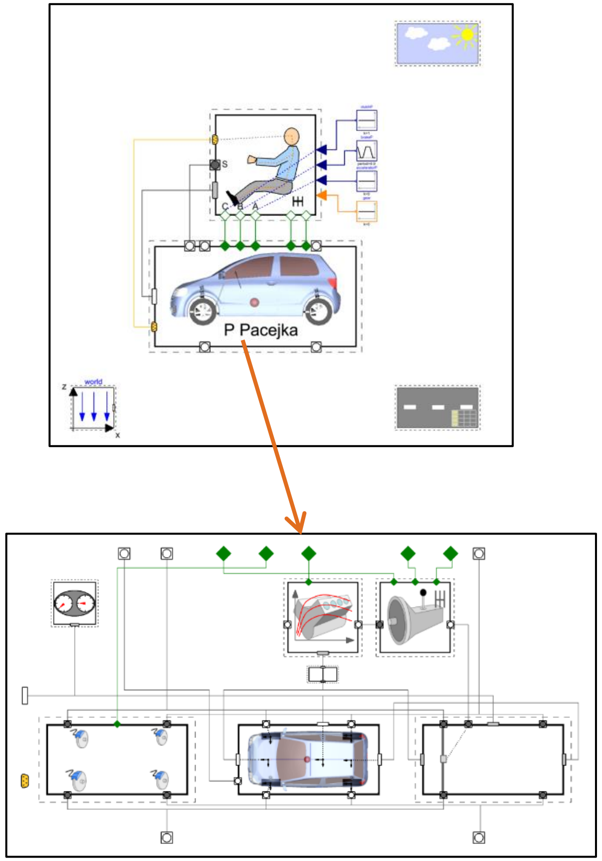
PACKAGE STRUCTURE



Good modeling practice:

- Package structure resembles system structure to a certain extent
- Within reasonable depths, i.e., only upper levels

Example: Sub-systems within *Vehicle* package correspond to components in diagram view of a vehicle



BROWSE MODEL HIERARCHY

CLASS VS. COMPONENT

- Class
  - *Modelica* keyword
  - model, connector, package, type...
  - defines behavior
  - convention: name starts with upper case
- Component
  - not a Modelica word
  - instance of a class
  - convention: name starts with lower case

Modelica text view:

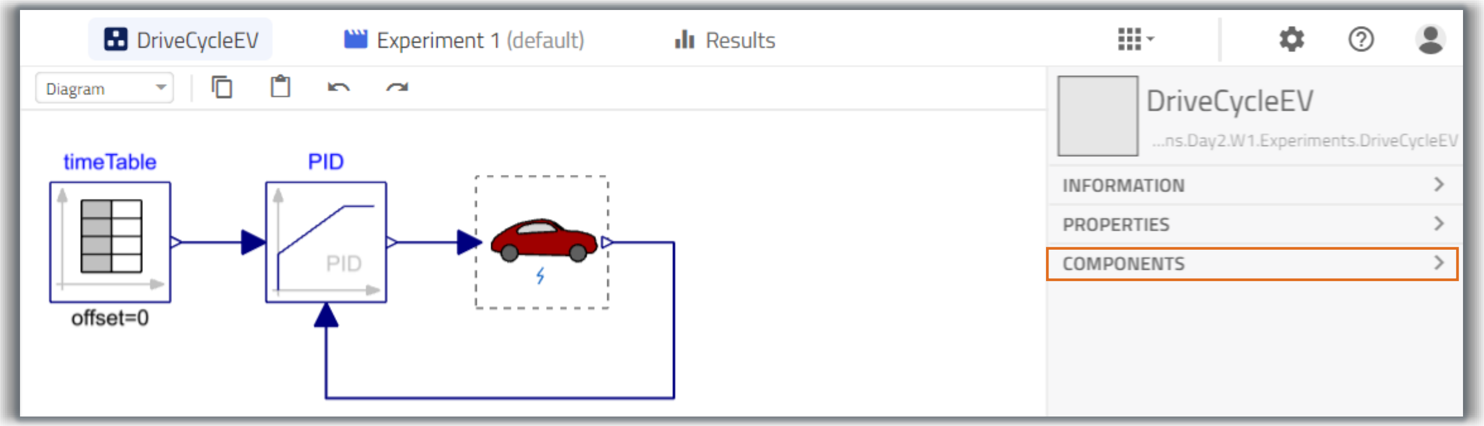
```
model Unnamed
  .Modelica.Mechanics.Rotational.Components.Inertia inertia;
end Unnamed;
```

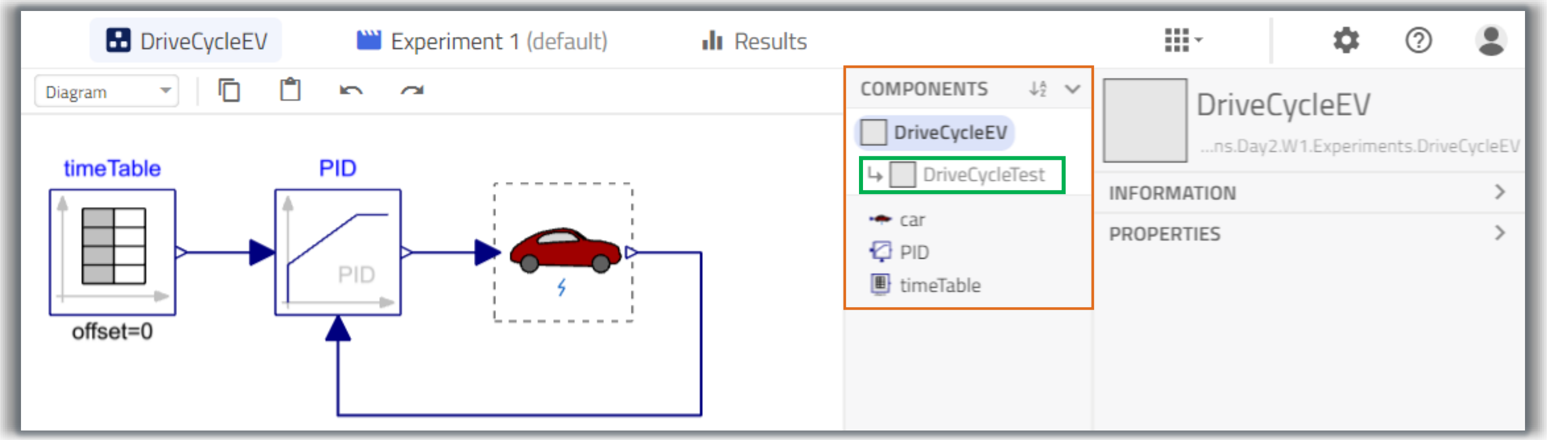
component instance name

class name

COMPONENT BROWSER

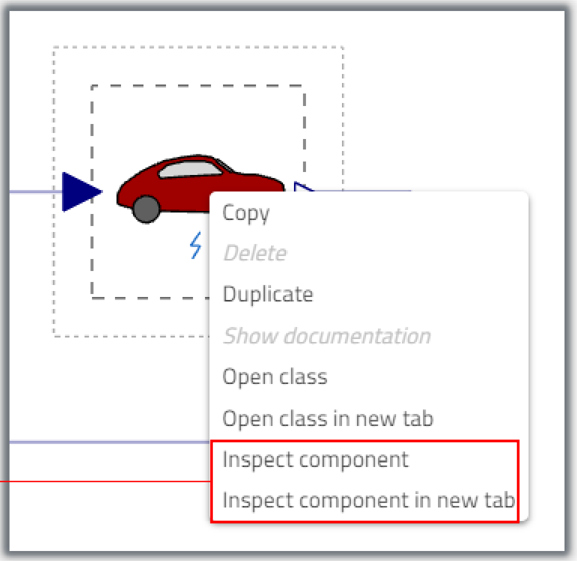
- We saw that Modelica enables hierarchical modeling – you can inspect the contents/hierarchy of a model using the ***Components Browser***
- It is available for all three modes
- Useful to get an overall idea of the components present in the model
- It also shows any classes the current model extends



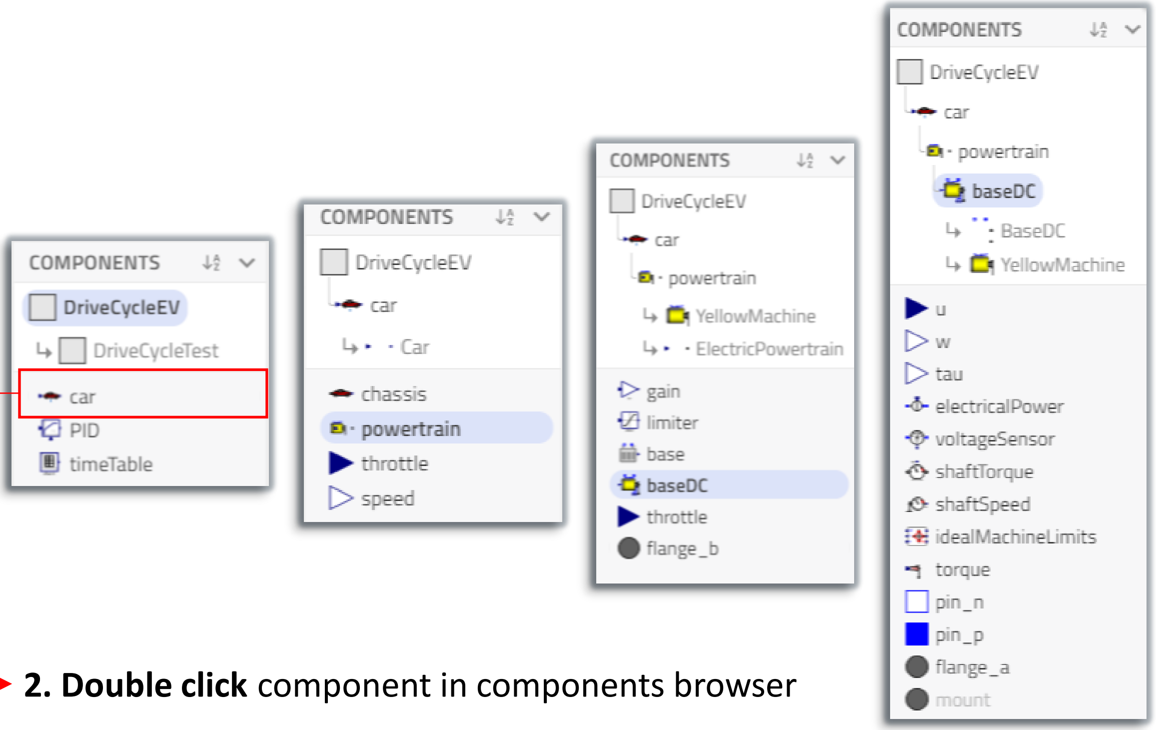


INSPECT COMPONENT

- Inspecting a component helps understand what components are present inside it
- Modelon Impact allows you to go down in the hierarchy in two ways:



1. Right-click the component on canvas then **Inspect component** or **Inspect component in new tab**



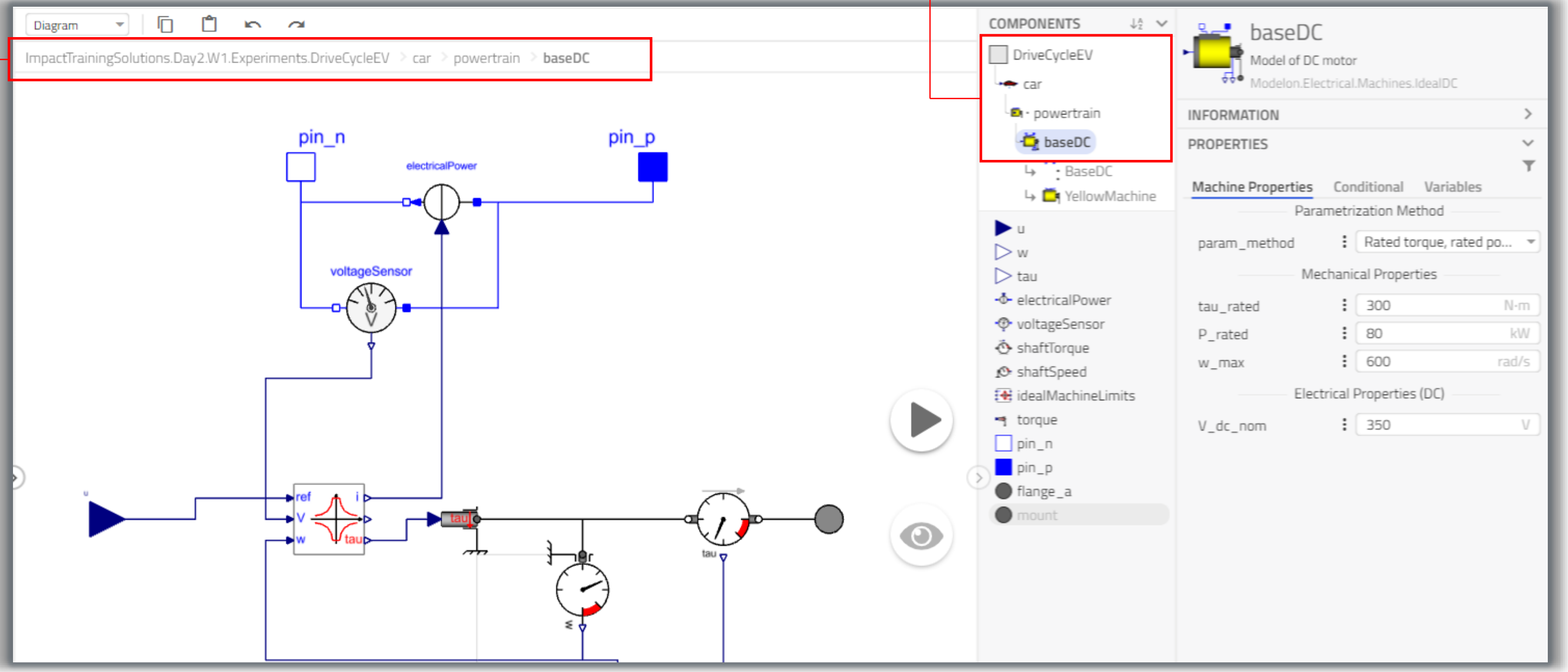
2. Double click component in components browser

BROWSE INSTANCE TREE

- To go back up, click on any of the **Bread-crumbs**

Bread-crumbs

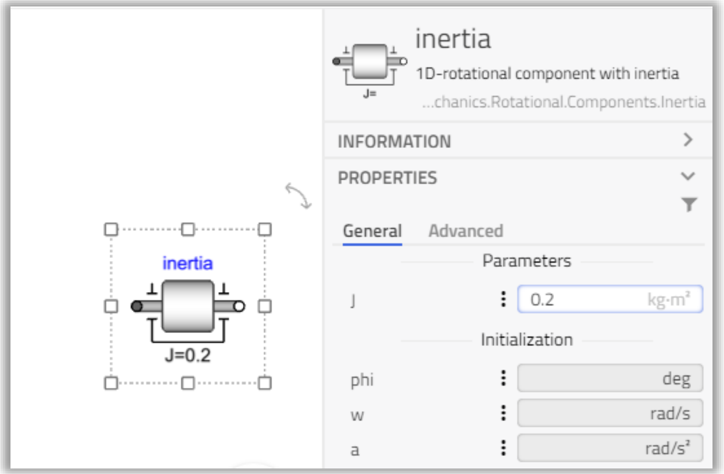
Hierarchy



PARAMETER PROPAGATION AND MODIFIERS

WHAT ARE MODIFIERS?

- "*A modifier is a Modelica language construct that allows to modify an existing class by setting a variable, parameter or attribute value, by binding a parameter or variable to an expression containing other parameters or to change a class reference to another class.*“
- Examples:
  - Simple parameter modifier: *inertia(J = 0.2)*

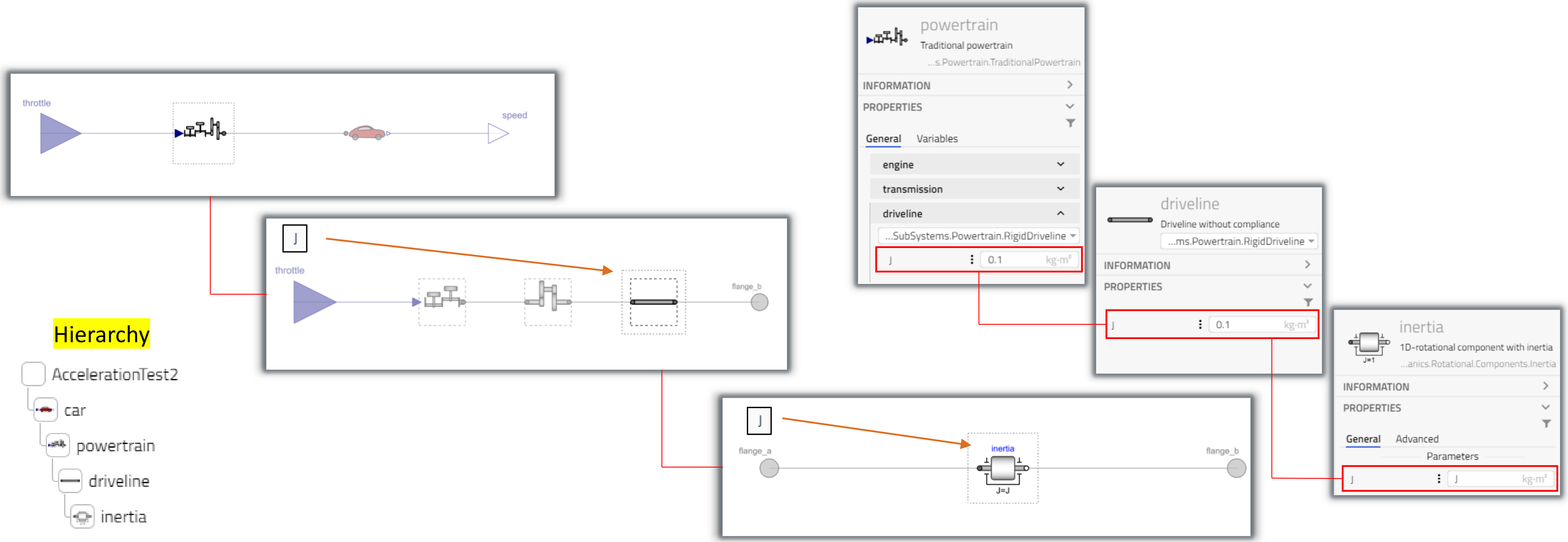


  - Class modifier: *extends Template.FullCarAccelerationTest(redeclare Systems.OriginalCarConfig car, throttle(k=1))*
  - Binding equation: *volume(v = tank.level*tank.crossArea)*

Modelon

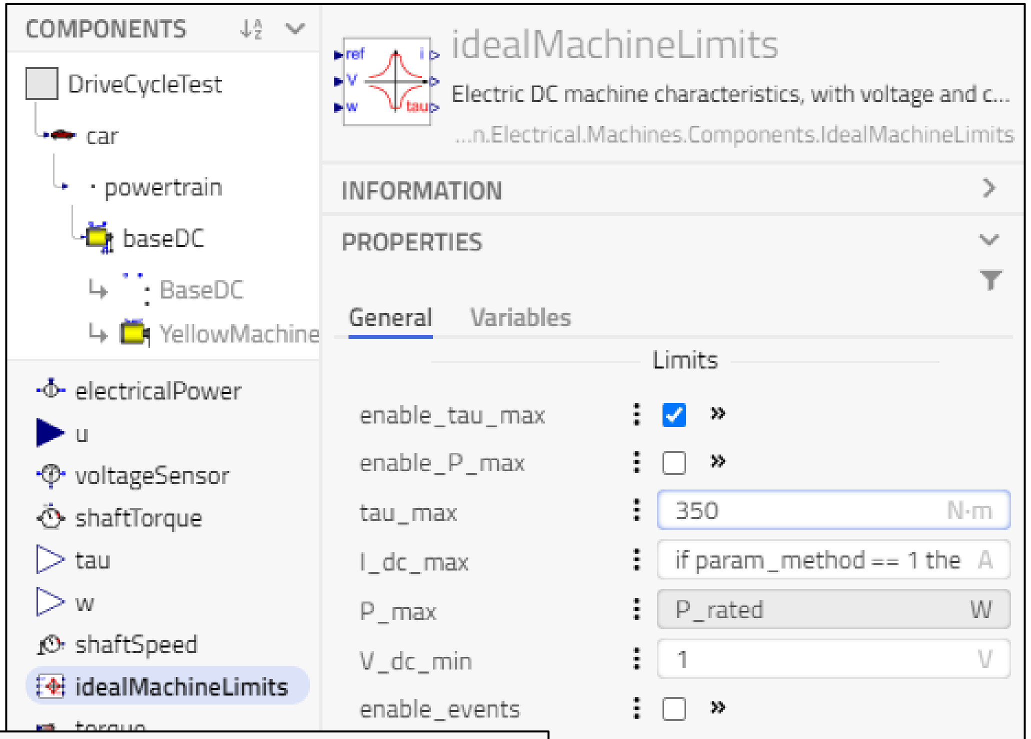
MODIFIERS

- The preferred workflow to create a proper data flow through the model tree is parameter propagation
- This was introduced in lecture Reusable Components



HIERACHICAL MODIFIERS IN SYSTEM MODELS

- A ”system model” here is a fully-paramatrized model ready to be simulated
- You can modify any part of the final model
  - Modifier only resides within the experiment container
    - In the example, the modified value of ***350*** for the parameter ***tau_max*** is only applicable to the particular experiment – ***DriveCycleTest***



TrainingPack.W3.Experiments.DriveCycleTest

(read-only)

```
model DriveCycleTest
    extends .TrainingPack.W3.Experiments.Template.DriveCycleTest(redeclare replaceable
    .TrainingPack.W3.Systems.ElectricCar car(powertrain(baseDC(idealMachineLimits(tau_max=350)))));
end DriveCycleTest;
```

HIERARCHICAL MODIFIERS IN SYSTEM MODELS

- If modifiers are applied in the modeling mode, its stored in the modelica code

DriveCycleTest Experimentation Results

TrainingPack.W3.Experiments.DriveCycleTest

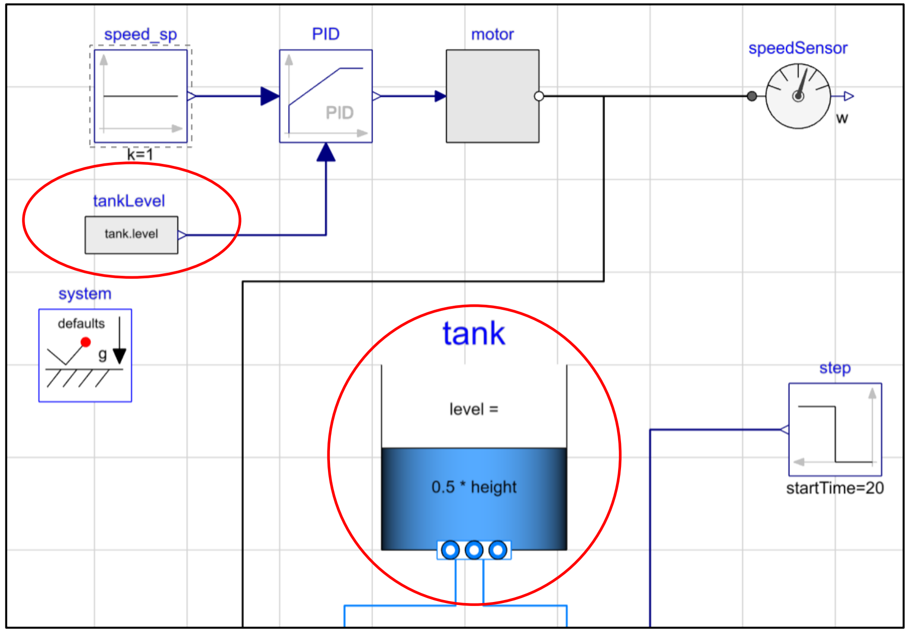
(read-only)

```
model DriveCycleTest
    extends .TrainingPack.W3.Experiments.Template.DriveCycleTest(redeclare replaceable
    .TrainingPack.W3.Systems.ElectricCar car(powertrain(baseDC(idealMachineLimits(tau_max=350)))));
end DriveCycleTest;
```

- If modification is done in Experimentation mode, it will be stored in that specific Experiment definition.

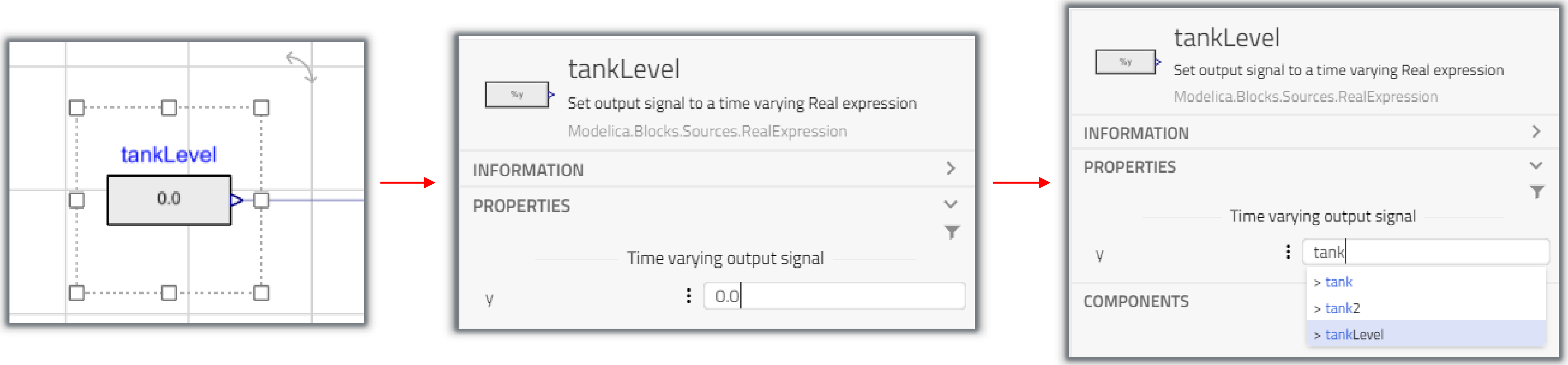
MODIFIERS BY COMPONENT REFERENCE

- Component reference is an option when you want to define variables by referencing variables or parameters from other components
- Data for the referenced variable is automatically retrieved (from the model instance tree)
  - In the example, value of ***tank.level*** (level of fluid in the tank) has been used in a **realExpression** block



MODIFIERS BY COMPONENT REFERENCE

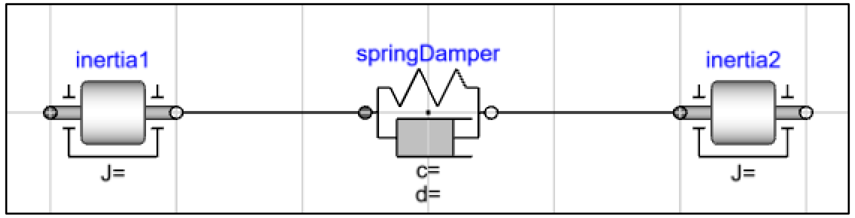
- Modification by component reference can be done by browsing or autocompletion
  1. Select the component where a parameter needs to be modified
  2. Go to **Details Panel-> Properties**
  3. Start typing the parameter by dot notation
     - Start typing to enable instance browser
     - Use arrow keys + (enter or tab)

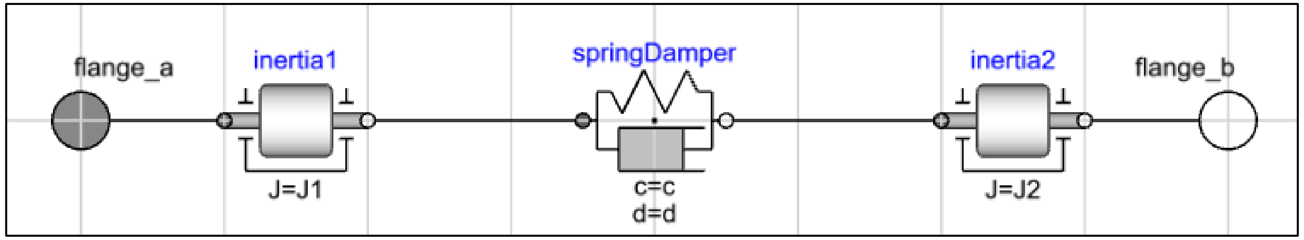


RECONFIGURABLE MODELS

INTERFACING A COMPONENT

- When creating a new component or subsystem you need to design its interface
  - Connector interface (how it interacts physically)
  - Parameter interface (what data is needed)



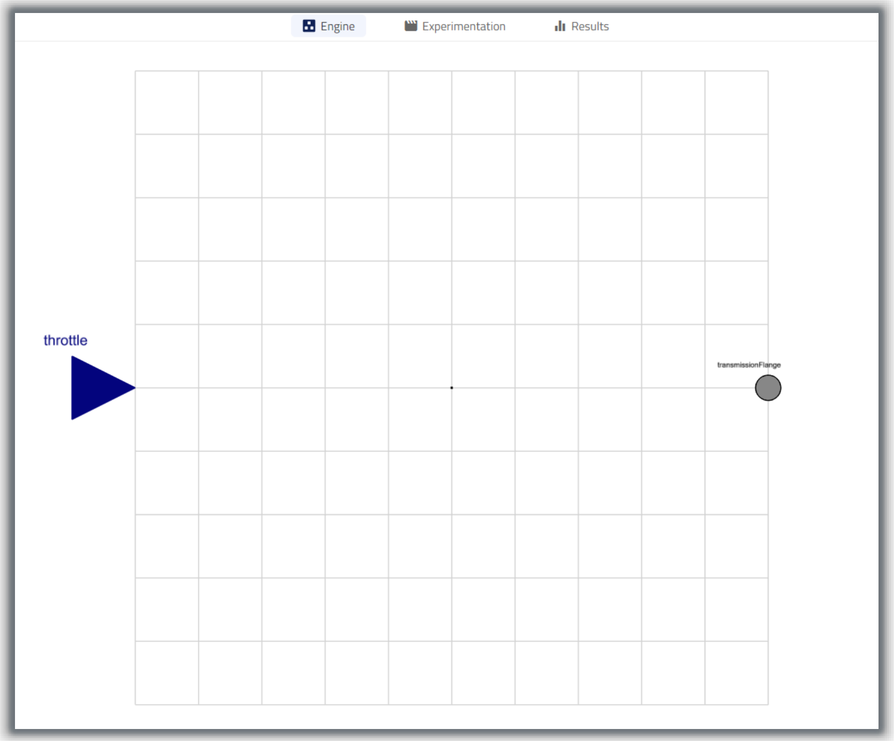


INTERFACE CLASSES

- An interface contains
  - Connectors
  - Common parameters
- Well-defined interfaces ensure plug-compatibility: all models that share an interface will also fit in the template
- Interfaces are used with inheritance, in Modelica the keyword is 'extends'

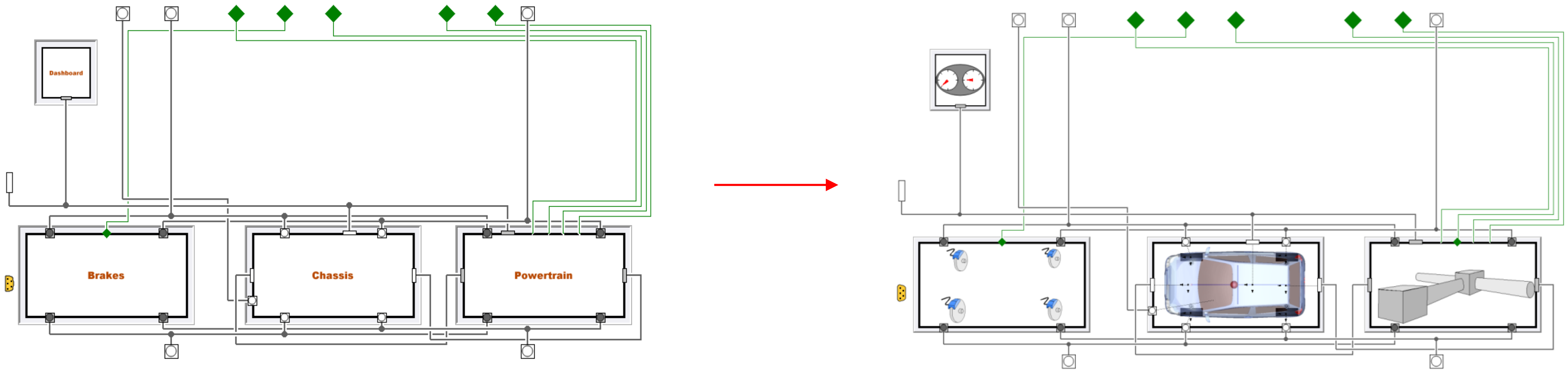
```
partial model EngineInterface
    Modelica.Mechanics.Rotational.Interfaces.Flange_a shaft
    annotation (...);
    Modelica.Blocks.Interfaces.RealInput throttle
    annotation (...);
end EngineInterface;
model Engine
   extends EngineInterface;
    ...
end Engine;
```

- ***Engine*** has all the properties of ***EngineInterface***



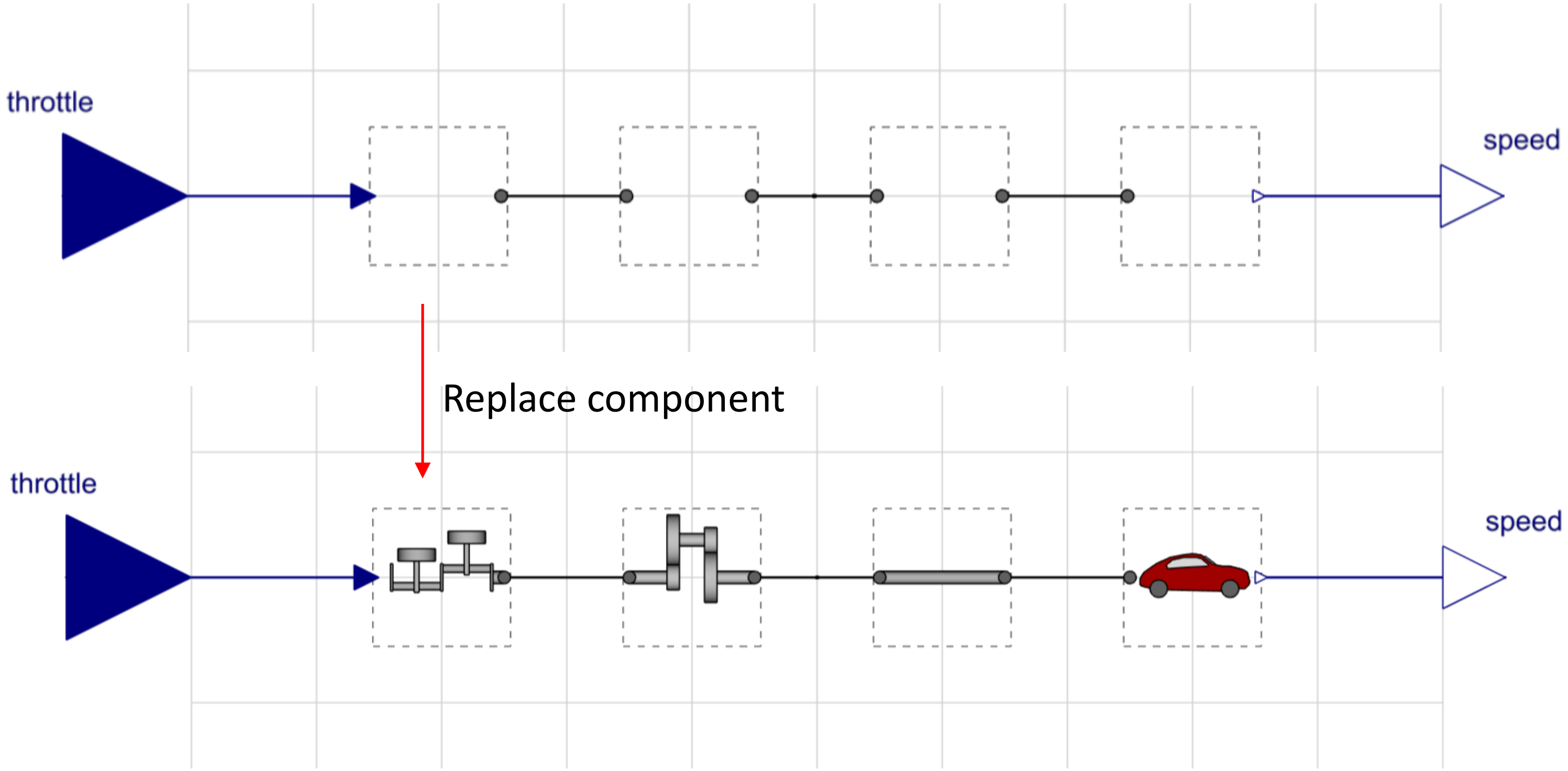
TEMPLATE CLASSES

- A template is a topology definition
- A template consist of
  - interfaces that act as placeholders
  - connections between the placeholders
- Using a template, configurations can be created by just specifying the components / subsystems



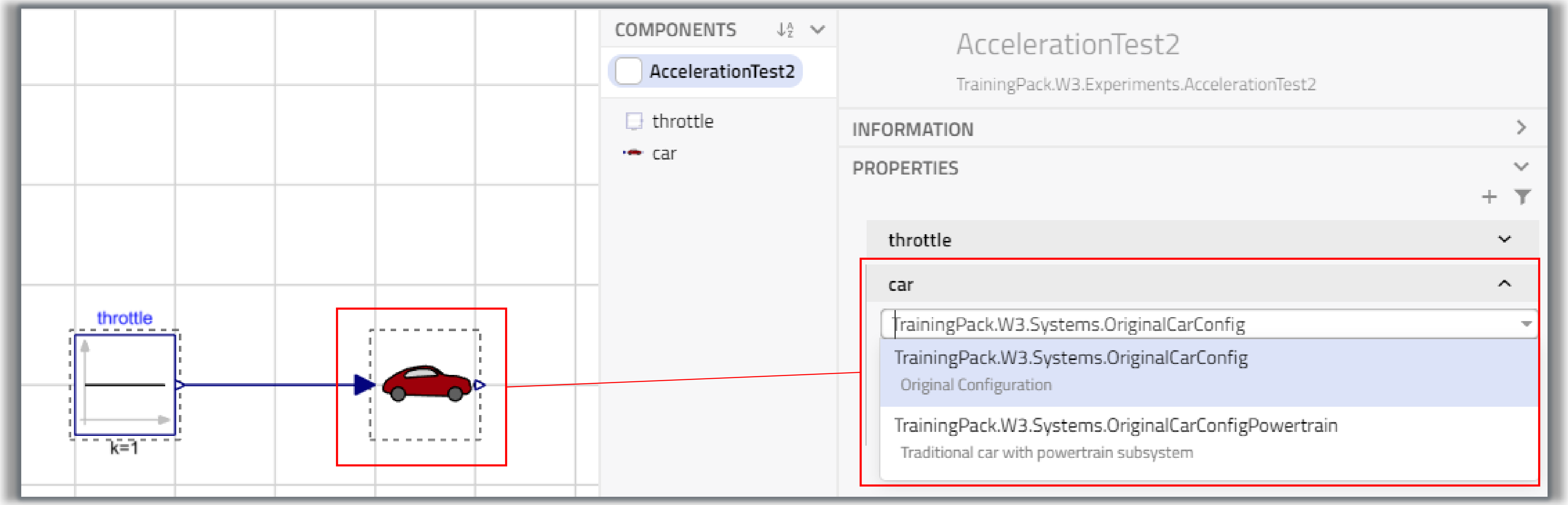
SYSTEM VARIANT

- If we have a template, with well defined interfaces, we can extend that and create a specific system variant:



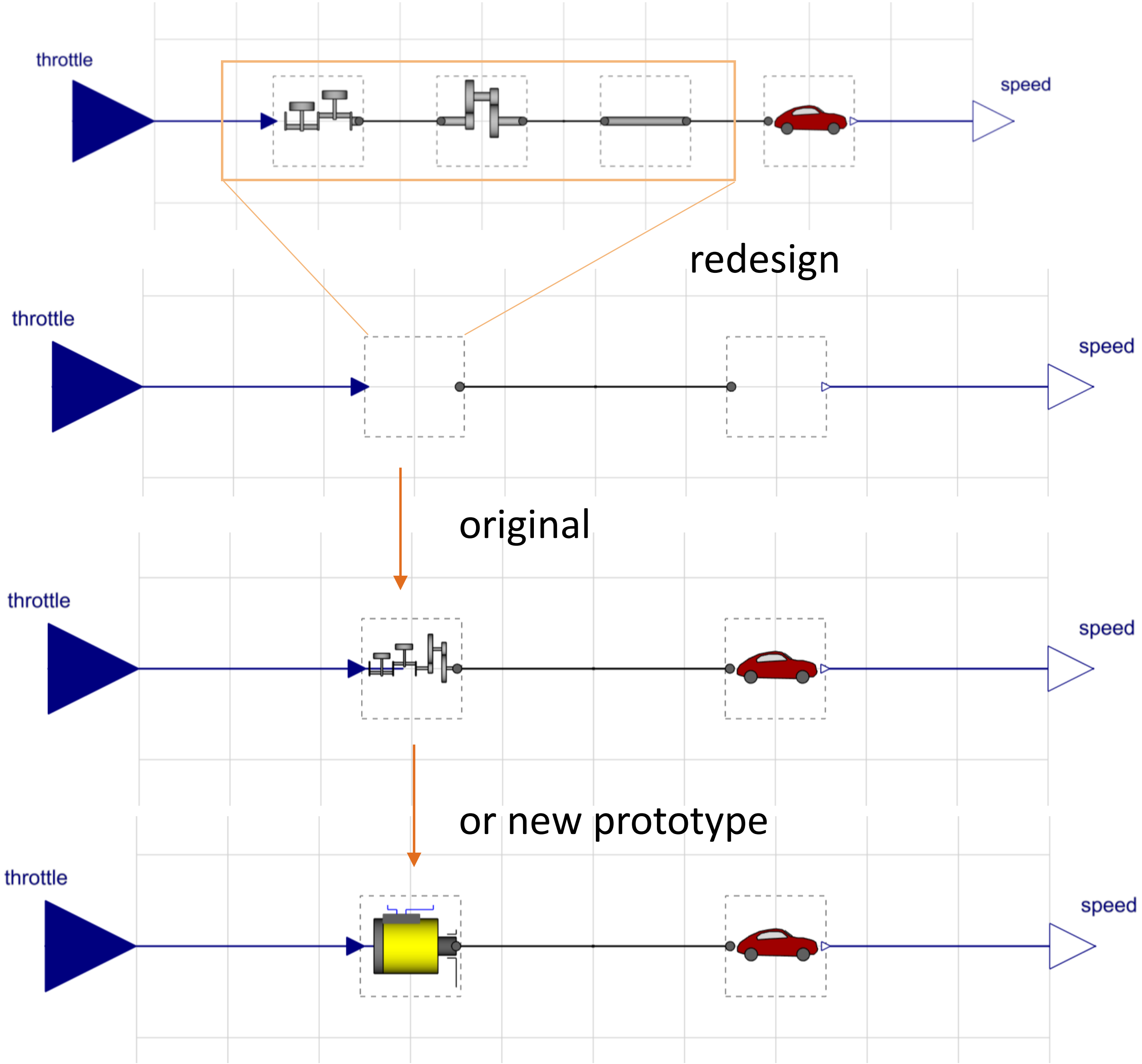
CHANGE CLASS THROUGH DROP-DOWN

- The parameter tab of the replaceable model will show a drop-down list
- The drop-down list will contain all matching choices
  - Matching choices here include any *car* model extending the ***car*** interface



TEMPLATE VARIATIONS

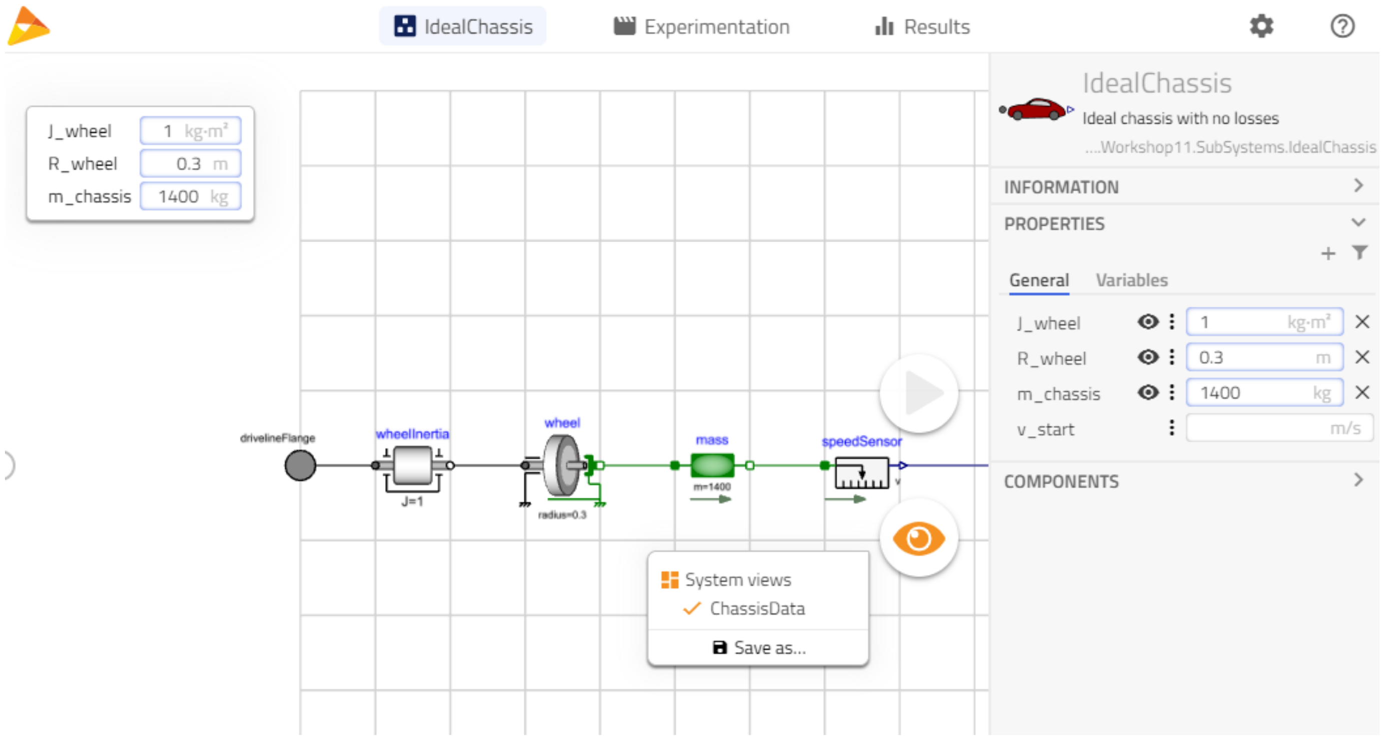
- Templates are easy to create
- Serve different architectures
- Reuses all subsystems



SYSTEM STICKIES AND VIEWS

STICKIES AND VIEWS

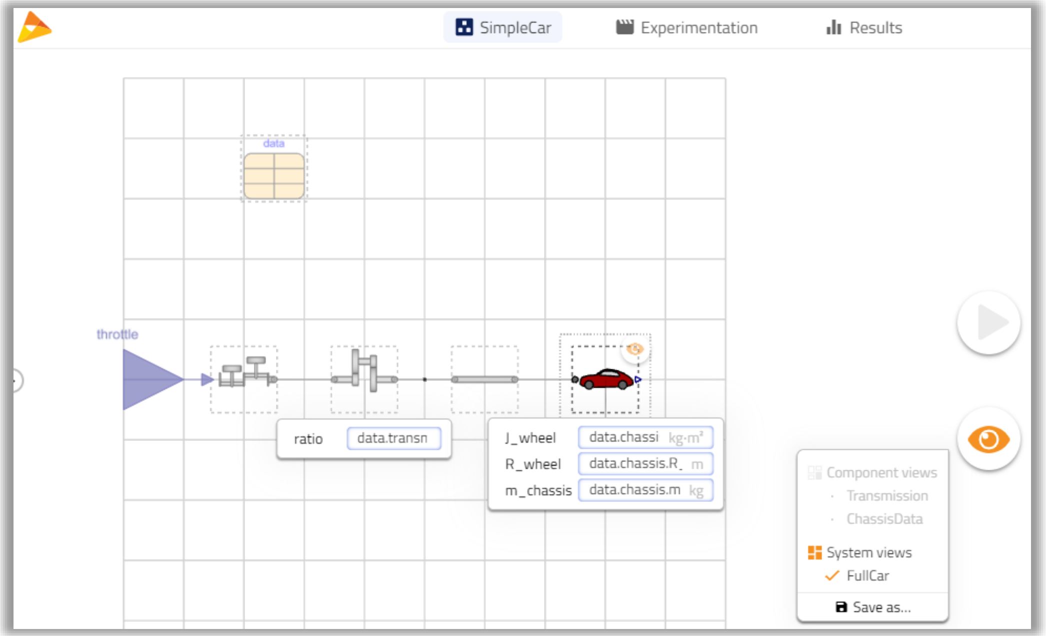
- Can be created for each subsystem

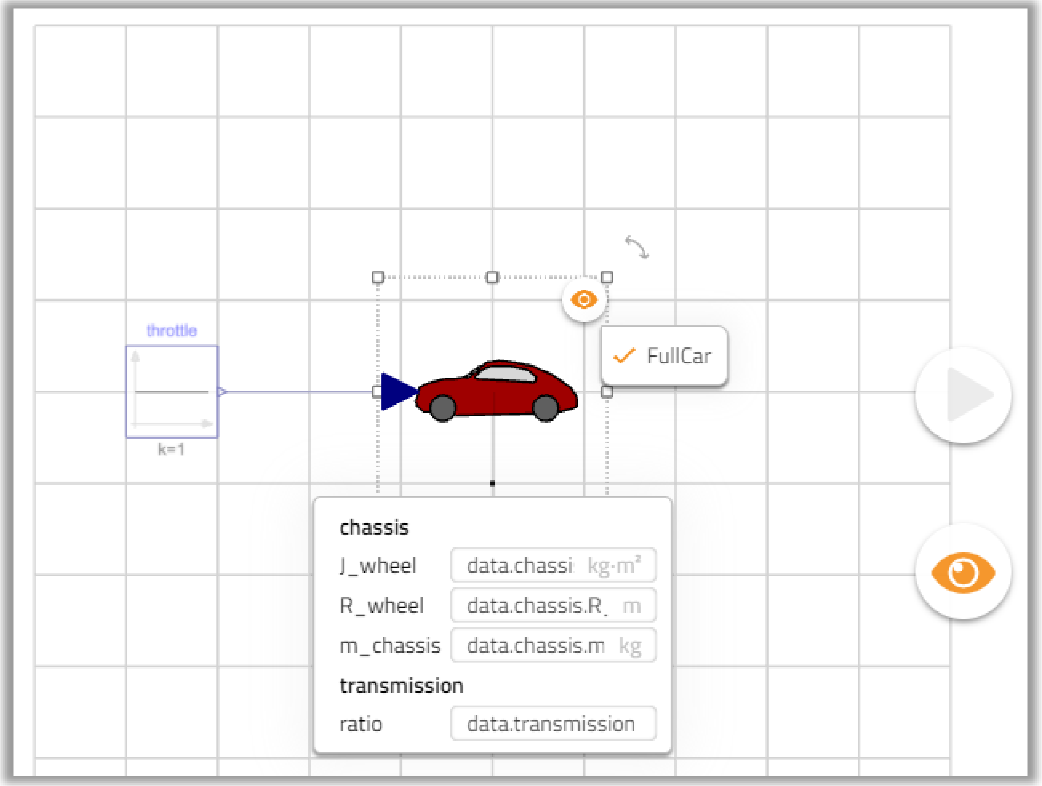


Modelon

STICKIES AND VIEWS AGGREGATION

- Multiple component views can be aggregated to a new system view





WORKSHOP 2.1

In this workshop you will:

- Browse a model library
- Inspect a hierachical model
- Redesign an architecture
- Create new configurations