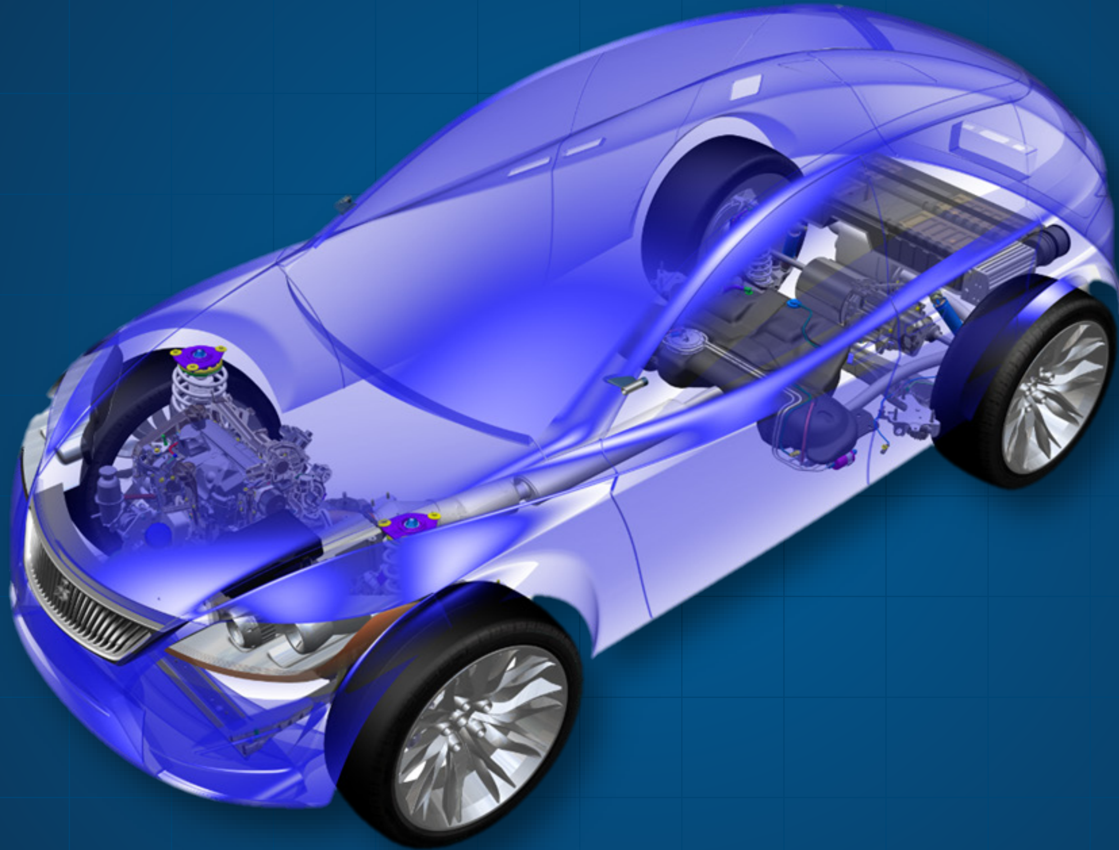
Dynamic Vehicle Behavior Modeling



DASSAULT SYSTEMES

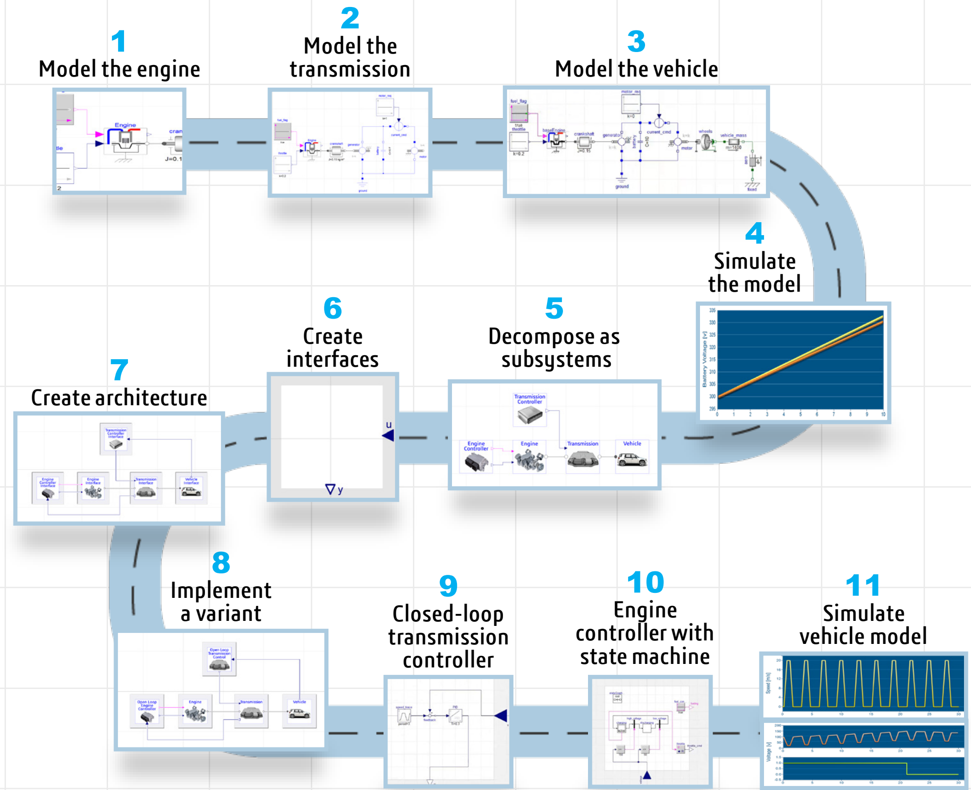
Deep Dive into Dynamic Vehicle Systems Modeling

Let's take a deep dive into dynamic vehicle systems modeling with a step-by-step example – modeling and simulating a series hybrid vehicle in Dymola®.

A series hybrid has an internal combustion engine that is only used to generate electricity. This means there is no direct connection from the engine to the wheels – instead electric motors are used to provide torque to the wheels.

This example not only demonstrates how components from different domains, like internal combustion engines and electric motors, can be combined to build a complete model of your vehicle, it also demonstrates how to model the control systems.

To watch a prerecorded video of this scenario please [Click here!]

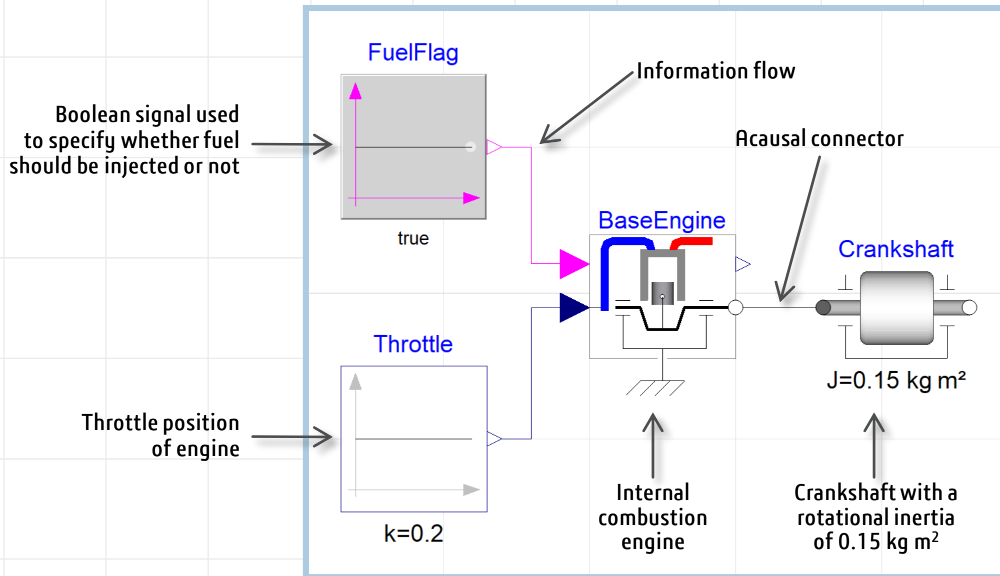


Dymola® is a registered trademark of Dassault Systèmes AB.

DASSAULT SYSTEMES

Step 1: Modeling the Engine and Crankshaft

We'll start by modeling the internal combustion engine model. Models are typically created by dragging and dropping component models into a schematic diagram. Notice that the engine model has two inputs. The first input is used to specify the normalized throttle position for the engine and the other, a Boolean signal, used to specify whether fuel should be injected or not. We'll revisit the topic of how to control this engine later. For now, we'll send constant signals into the engine as a starting point and switch to a closed-loop control strategy later. To finish building the engine portion of our model, let's add a rotational inertia of 0.15 kg m^2 to represent the crankshaft.



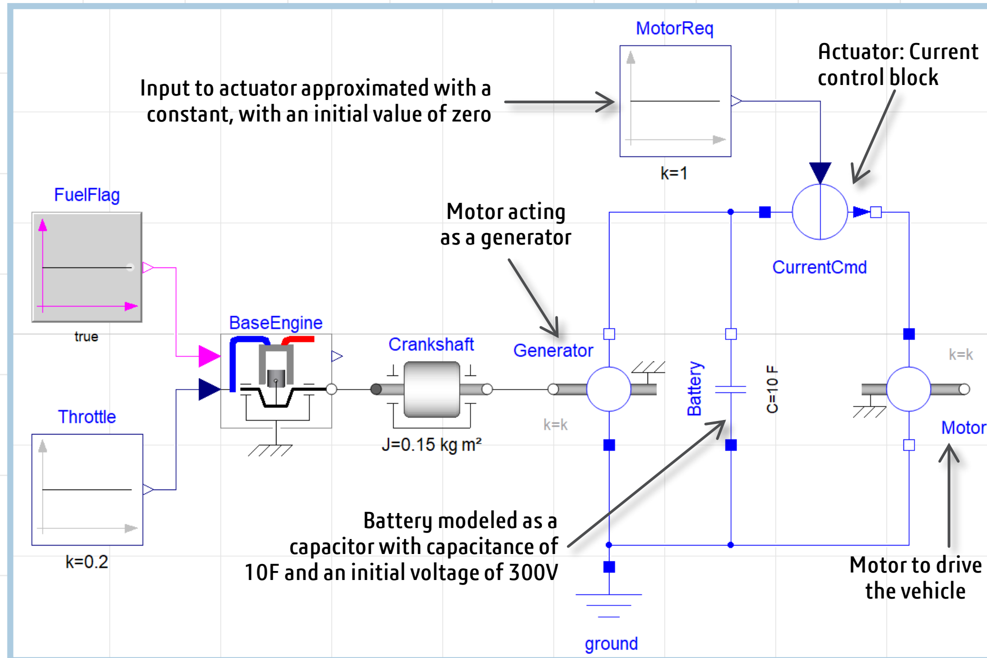
Note that the block diagrams used to model control functions are seamlessly combined with the mechanical components. More importantly, notice the difference in the connectors. The block diagram components have arrows on them, indicating information flows through the system. The mechanical connections on the other hand are directionless, acausal connectors. Acausal connectors allow us to build models that are flexible. In this case, we've connected the crankshaft to the engine model but we have the freedom to connect it to any other rotational component. We don't need to worry about whether that component will be a spring, an engine, or a clutch; whatever is needed, we just instantiate it from the library and connect it up.

BaseEngine is instantiated from the DLR Powertrain library. All other components are from the Modelica® standard library.

DASSAULT SYSTEMES

Step 2: Modeling the Transmission

With the engine out of the way, let's start looking at the transmission. Let's model a simple transmission with a pair of motors from the standard library. One motor is connected to the engine, acting as the generator and the other is connected to the wheels, driving the vehicle. To control the motor, let's insert a current control block. This component is essentially an actuator, controlled from outside the transmission. The input to this component is the requested motor current. The actual value for the requested current will have to be calculated based on the torque required by the vehicle. For now, let's simply add a constant input with an initial value zero (motor is not running).

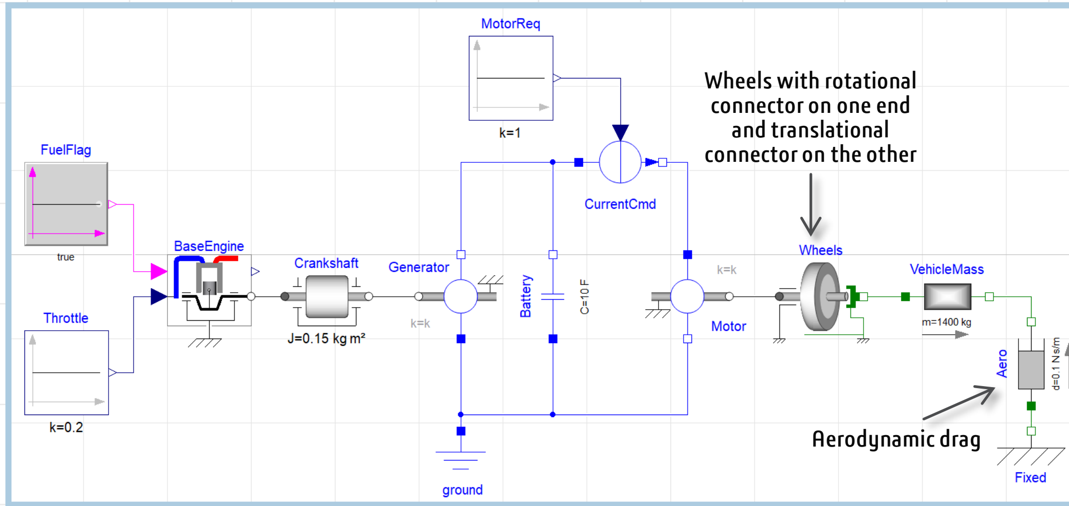


Next, we connect the generator and the motor, and add a ground to the circuit. Our model is still missing one important thing: batteries to store energy. There are many ways to model batteries. Just to keep things simple, let's use a large capacitor as the battery and add it in parallel with the motor. This means that electricity generated by the generator can flow either into the battery or into the motor. The motor current actuator determines how much flows one way and how much flows the other. To start the battery out charged, let's specify the initial voltage of the capacitor as 300V.

DASSAULT SYSTEMES

Step 3: Modeling the Vehicle

We'll start with a simple model for the vehicle. The main effects we need to capture are how torque is translated into a force on the vehicle, the drag force present on the vehicle and the overall vehicle inertia. For this model, we are only interested in longitudinal dynamics, that is, we are only interested in modeling the vehicle moving in a straight line. The first step in modeling the vehicle is to add wheels that transform the torque generated from the transmission into forces that move the vehicle forward in a straight line. Note how the wheel model has a rotational connector on one end indicated by a gray circle and a translational connector indicated by green square on the other. Let's also add the overall vehicle mass and a damper to represent losses that scale up with speed. In reality, aerodynamic drag scales differently, this is just an approximation. So far everything looks good!

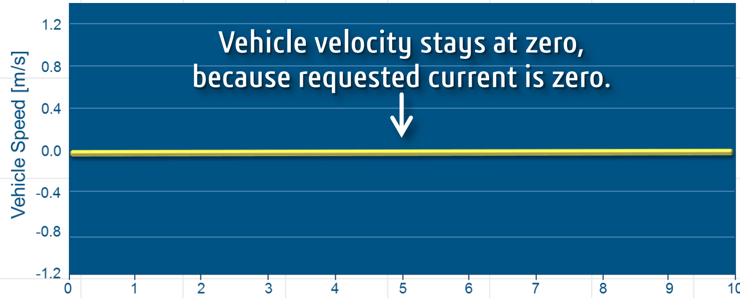


CHECKPOINT!

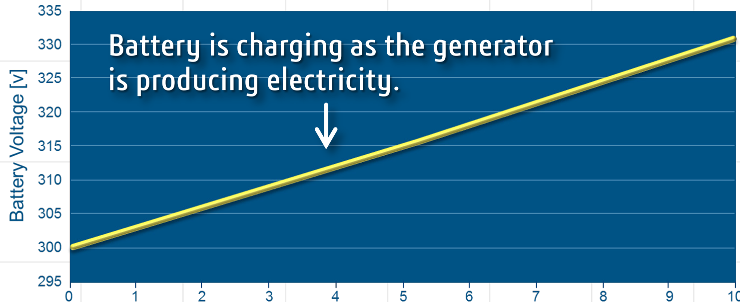
We started by adding the **engine model** and **transmission models**. The **engine control** has two inputs. One determines whether fuel is being injected and the other specifies the throttle position. Our **transmission control** model has one input which allows us to control how much current is input to the motor. The transmission contains a **battery** which allows us to store up electrical energy for use later. This is typically done to allow the engine to operate at its most efficient operating point. When large torque demand comes from the driver, the torque is delivered by draining the battery rather than forcing the engine to operate at an inefficient point. The battery is also important because it provides a place to store energy recovered from regenerative braking. Finally, we have added the **wheels** and attached the **overall vehicle mass**.

Step 4: Simulating the Behavior of the Vehicle

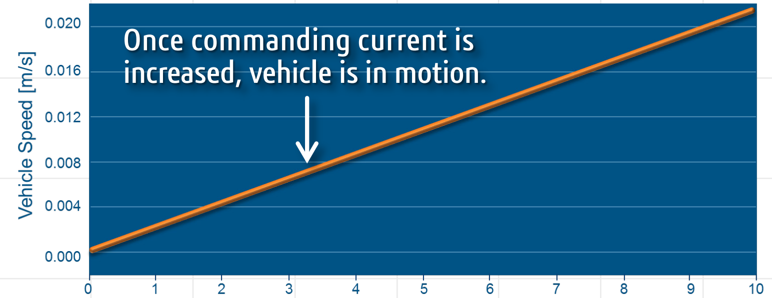
We can now simulate this model and study the behavior of the vehicle. Let's start by observing the vehicle velocity, we see that the vehicle isn't moving. This is expected since the transmission controller is requesting zero motor current.



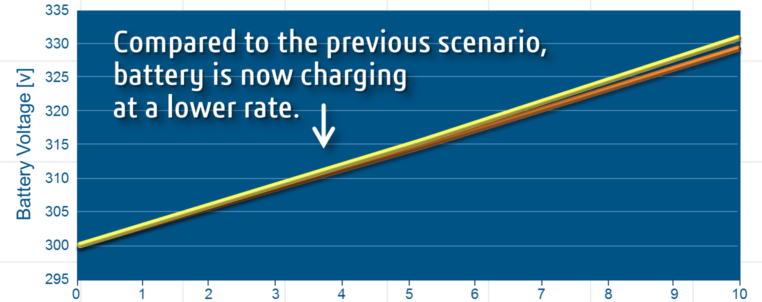
The generator is producing electricity and since current is not requested by the motor, it charges the battery instead, from its initial voltage of 300v.



Let's increase the value of the commanded current and rerun the simulation, making the vehicle accelerate.



Let's compare the battery voltage trajectory against the previous simulation. Sure enough, we see that the battery is now charging more slowly because power that was previously going into the battery is now powering the motor.

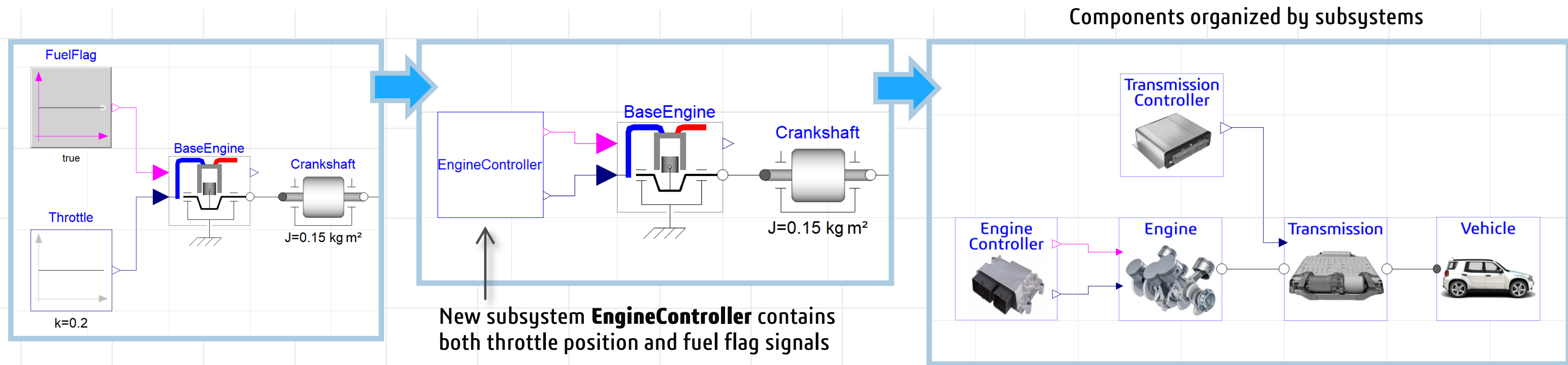


DASSAULT SYSTEMES

Step 5: Decomposing into Subsystems

To get closer to real-world conditions, we need to refactor this model and improve the control systems. We could start by changing and reconnecting components, but there are a couple things to watch out for. First, when reconnecting things, you run the risk of introducing errors. Second, we may want the original open-loop control version for testing. To address these concerns, we should follow standard configuration management guidelines.

Our first step is to organize the components by subsystems. To do this, we select the components that are part of the same subsystem and create a new subsystem model. Let's create a new subsystem called EngineController out of the engine control components – the throttle indicator and fuel flag, while preserving the original components as a new model called OpenLoopController. We perform the same actions for the engine, transmission, transmission-control and vehicle models. The system is now composed of subsystems. The next step is to standardize the interfaces for each subsystem.

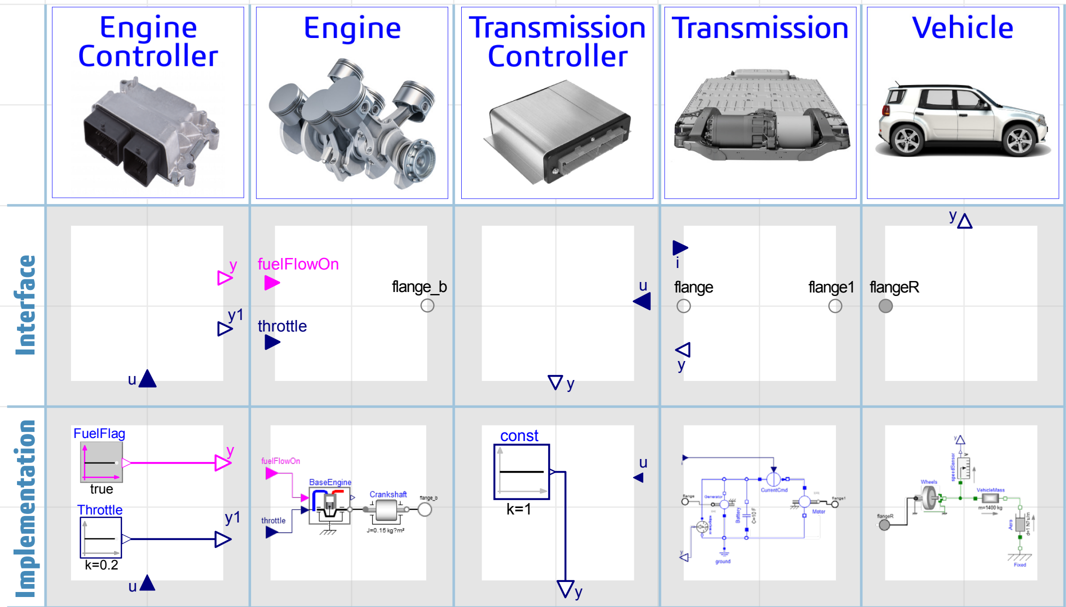


Step 6: Creating Interfaces

Let's define a standardized interface for the engine-controller. In our model, the engine control decides what the throttle position should be and whether to fuel the engine. Our current engine control model reflects this by including two output signals. One is a Boolean signal for the fueling command and the other is a continuous signal indicating normalized throttle position. The current engine control model defines both the interface, the above two signals that it needs to work with, as well as the implementation, that it uses open-loop commands. We will separate this model into an implementation and an interface.

We'll also add one additional input signal to the new interface to supply the engine controller with information about the state of the battery voltage.

We now have an interface which defines what is common across all potential engine control models and a specific implementation that just uses open loop commands.

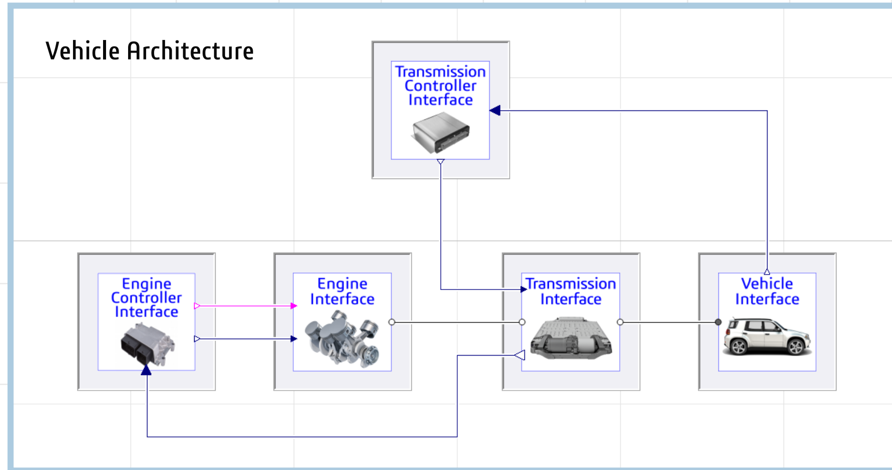


Let's follow the same procedure for the engine subsystem, splitting it into an interface and a specific implementation. This interface includes inputs from the engine controller and an output shaft for connection to the transmission. The implementation includes the internal combustion engine and the crankshaft. We will follow this procedure for the transmission, the transmission-controller and the vehicle model, adding additional sensors along the way.

Step 7: Creating Vehicle Architecture with Interfaces

To create the vehicle architecture, let's build a new version of our system model, but this time, using only the interfaces that we have developed. After connecting the interfaces together, we end up with a model that looks very similar to what we had before, except this time we haven't included any implementation details.

This architecture contains only the interfaces and no implementation has been specified. It captures the structure of our system, regardless of the specific implementations we choose to use. Once the architecture has been created we don't need to connect subsystem models anymore, all the interfaces have been connected to work across any implementation. Next step is to create a variant of the vehicle and decide which implementations of each subsystem will be included in the variant.



The vehicle architecture now captures the structure of the system, regardless of the specific implementation for each subsystem.

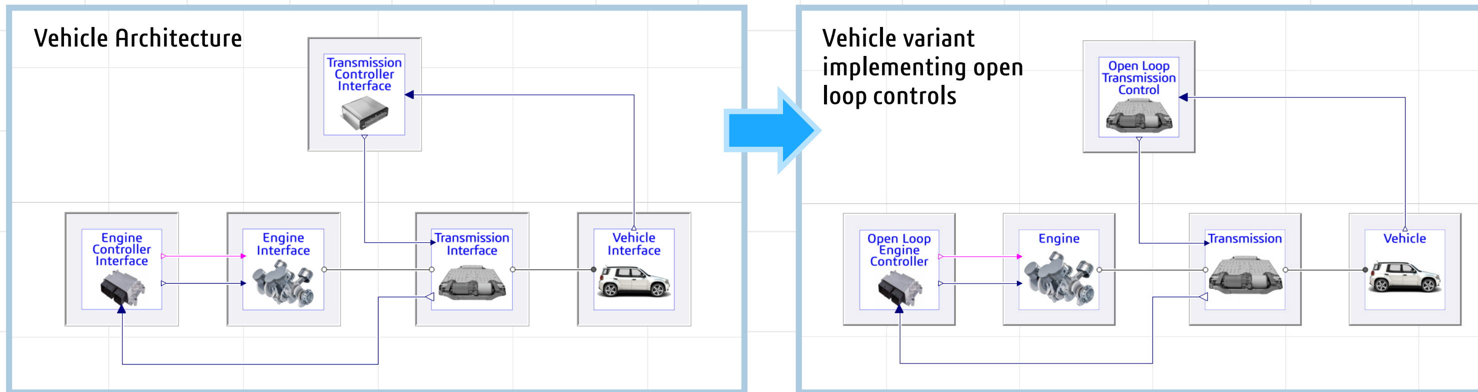
CHECKPOINT!

After simulating the initial vehicle, we created **subsystems** by aggregating multiple components with related functions. The **interfaces** for each subsystem was extracted, separating them from specific **implementations**. The vehicle **architecture** now captures the structure of the system, regardless of the specific implementation for each subsystem.

DASSAULT SYSTEMES

Step 8: Extending the Vehicle Architecture to Implement a Variant

For the base variant, let's recreate our original model with open-loop control. To do this we need to specify the implementations for each subsystem. At the moment, we only have one implementation for each subsystem. We could directly specify our implementation choices in the architecture model, but a better approach is to leave the architecture model as it is and create a variant from our architecture that captures our specific implementation choices. For this, we simply create a new model that extends from the architecture. When we extend, the new model starts from the old model. From there, we can make further changes, like specifying the implementation details for the different subsystems. This allows us to easily create many different variants of the same fundamental architecture. All these models can exist at the same time instead of constantly switching back and forth between different configurations. It is worth pointing out that there is no limit to how many times we can extend from a model. For example, we might extend from our architecture to create a baseline configuration of our vehicle where all the implementations are filled in. From there, the engine designers might extend from the baseline model but insert a more detailed engine model while keeping the transmission and vehicle subsystem models the same. Similarly, the transmission designers might do the same with the transmission while leaving the engine and the vehicle as is. These best practices for configuration management organize the models and support collaborative workflows.



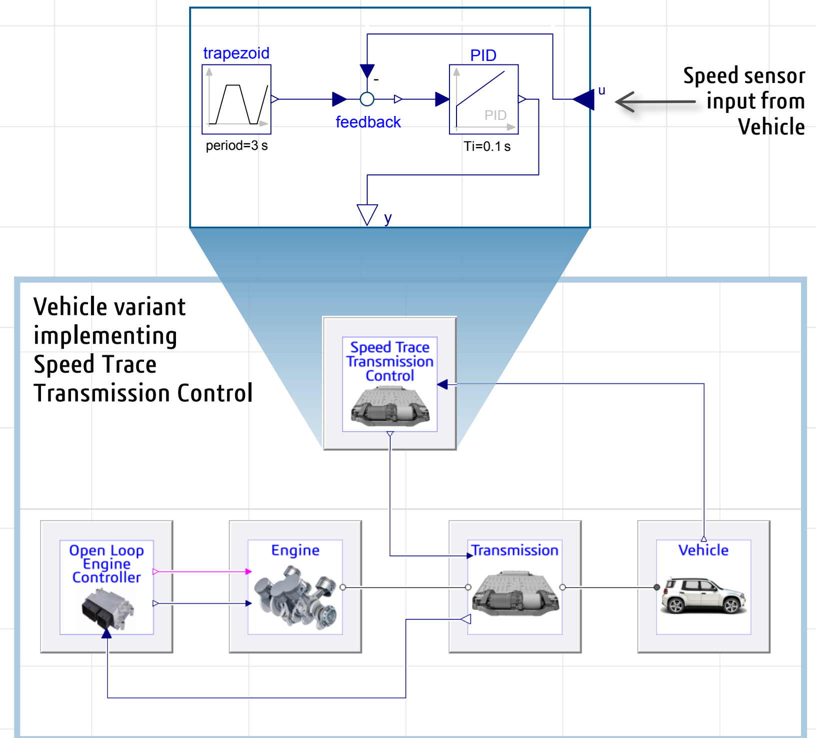
DASSAULT SYSTEMES

Step 9: Implementing a Closed-loop Transmission Controller

Once we've gone through and specified all of our initial implementations, we can again simulate the model. Of course, we'd still have the same uninteresting response because of the open loop elements, but now, we're in a position to quickly do something about that. For example, let's create a transmission controller that directs our vehicle to follow a specified speed profile. To do this, we'll create a new transmission controller implementation by extending from the interface. When we extend, we are not copying the contents of the interface into our implementation. This is important because copying and pasting creates redundancy. By extending, we avoid copying and pasting, making maintaining the models easier.

Once we create our new transmission controller model by extending from the appropriate interface, we just need to fill in the implementation details. Let's instantiate a PID controller for speed control with a trapezoidal wave pattern for the drive cycle.

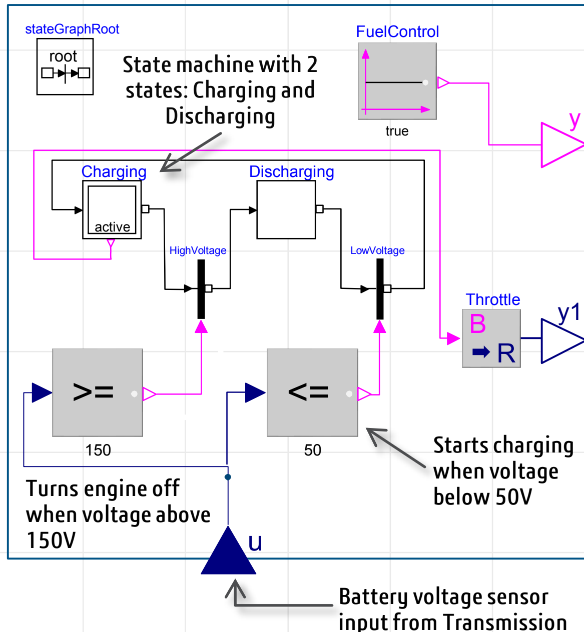
To incorporate the new controller, instead of creating a whole new vehicle model, we can extend from the original open loop vehicle model and simply change our selection of the transmission controller. Again, we select the subsystem we are interested in, the transmission controller in this case, and we select from a collection of existing controllers. The relationship between different variants of our model is concisely captured. In this case, our current vehicle model, extends from the open-loop vehicle model, but replaces the transmission controller with a different transmission controller.

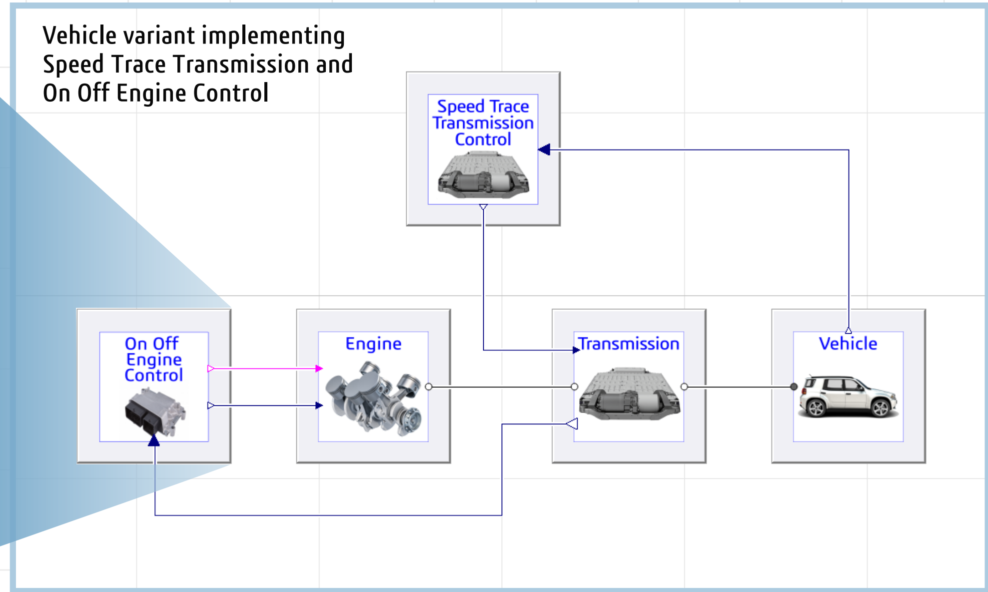


DASSAULT SYSTEMES

Step 10: Implementing a State Machine for the Engine Controller

Let's repeat the procedure for the engine controller, extending from the interface to create a new model and implement a state machine to turn the engine on and off. Taking in the battery voltage as input, the state machine turns on the engine to charge the battery when the voltage is too low, and turns off the engine to save fuel, when the battery voltage is too high. To select this variant of the engine controller, we select the engine controller subsystem and switch to this implementation. The variant choices in the architecture for each subsystem is automatically determined based on the interfaces they implement.



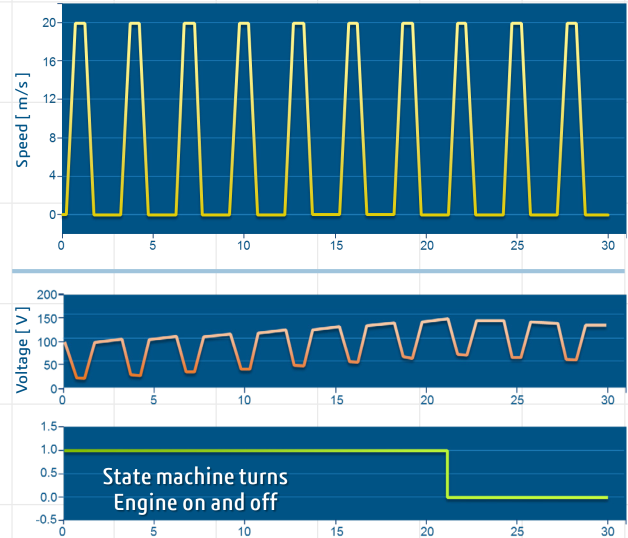


DASSAULT SYSTEMES

Step 11: Simulating Vehicle with the New Controllers

With these two new controller implementations, let's take the vehicle out for a spin. After we simulate the model, we plot the vehicle speed and compare it to the desired drive cycle profile. Here we see the transmission controller is doing a good job of following the drive cycle speed trace. Now let's look at what is going on with the engine and the battery. Notice how the engine comes on when the battery voltage gets too low and turns off when the battery voltage gets too high. Another important thing to know about the battery is that it is charging and discharging, even when the engine is off. The discharging comes when the vehicle accelerates because the motor takes energy from the batteries to increase the vehicle speed. But how is the battery recharging when the engine is off? The answer is regenerative braking. When did we implement regenerative braking? We did it in the transmission control. The PID controller in the transmission controller requests positive torque from the motor when the vehicle needs to accelerate, and requests negative torque when the vehicle needs to decelerate. Because we are using acausal models, all of our components include balance equations, for things like mass, momentum, charge and so on. In order for these balance equations to work out, the kinetic energy in the vehicle has to go somewhere. The motor turns it back into electricity when negative torque is requested and the resulting current flows into the battery, charging the battery in the process.

The important point here is that we don't need to implement regenerative braking. We implement a mathematical model of each component and then impose conservation equations across all the connections. As a result, we are always assured of accurate accounting. This is important because if model developers had to implement all the consequences of the different modes for each component it would be very easy to overlook something. With acausal modeling, all of this is taken care of.



CHECKPOINT!

Using this series hybrid example, we've shown how to build a complete multi-domain system model from scratch. We also organized the model into subsystems following configuration management best practices. We implemented some additional controllers to demonstrate typical capabilities seen in a series hybrid. Finally we simulated the vehicle with the new controllers and verified the controller behavior.

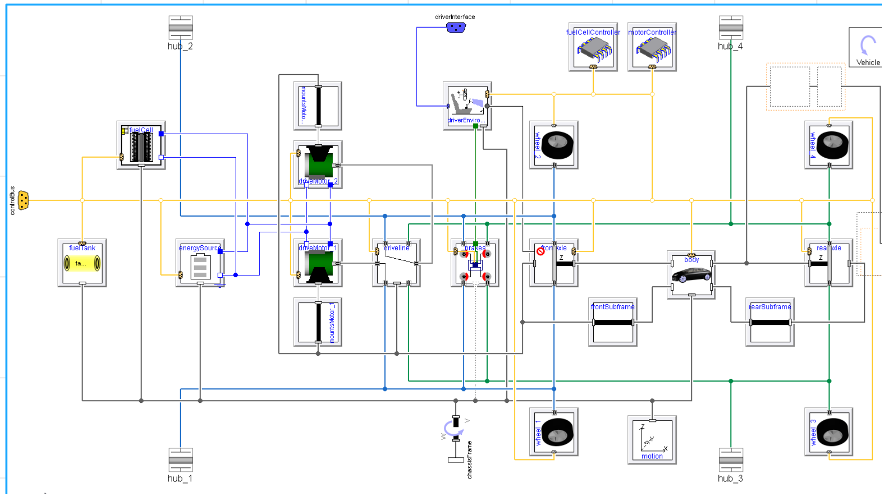
DASSAULT SYSTEMES

Next Step: Leverage the Vehicle Library to Model Your Vehicle

While it's good to know that all of this is possible with Dymola, it is also important to realize that you don't need to start from scratch. This architecture based approach is very common and many of the specialized libraries that come with Dymola include not only high quality component and subsystem models but also the interfaces and architectures that give you a head start in building models of your system. You can also import your legacy models into Dymola, either using direct interfaces, or by adopting the standard FMI® interface.

Vehicle Systems Modeling and Analysis (VeSyMA®) is a complete set of libraries for vehicle modeling and simulation. It includes engine, powertrain and suspensions libraries that work in conjunction with the Modelica® Standard Library. In addition, battery with electrified and hybrid powertrain libraries are available as well. Please watch the other videos in this series for more information on [Dymola.](#)

VeSyMA® Vehicle Library



CONCLUSION

The need for more intelligent and autonomous capabilities has resulted in increasing complexity of systems. Traditionally, building many prototypes and subjecting them to exhaustive tests was the only way to achieve convergence of multi-disciplinary designs. This reliance on prototypes however, has proven to be time consuming and expensive. Moreover, the sophistication of intelligent systems have made it more difficult to test all possible scenarios, thereby limiting what can be achieved with physical prototypes.

Building virtual models is the most effective way to identify and resolve integration issues upfront, reducing rework and improving quality.

VeSyMA® is registered trademark of Claytex Services Limited | Modelica® is a registered trademark of the Modelica Association
FMI® is a registered trademark of FMI Standard Org

DASSAULT SYSTEMES

Articles in the series:

Model-based Systems Engineering – A Primer

Dynamic Vehicle Behavior Simulation – A Deep Dive

System Architecture with SysML for Control Systems Engineers