How to dissolve complex dynamic systems for wanted unknowns with *Dymola / Modelica*

Jochen Köhler
ZF Friedrichshafen AG
Graf-von-Soden-Platz 1, D-88046 Friedrichshafen, Germany

Abstract

For developing optimized hybrid driveline strategies a way was found to make complex models of these systems available in an optimization process. The challenge here is to make models built up with *Modelica* available for an optimization process in other tools like *Matlab*. The simulation feature of *Dymola* does not help here because the optimization is done in one point in time.

After describing the optimization problem for these drivelines the special way of implementing it in Modelica is emphasized.

When the model is built it had to be exported into *Matlab* as a MEX-funtion to make it available for optimization algorithms.

Keywords: Dissolving equations; C-Interface; dsblock; Matlab Mex File

1 Introduction

One main issue in control theory is to drive a system in an optimal way. One task is to find an optimal trajectory to get to a final desired state that is known. If this final state is not known or if there is no "final" another way is to optimize just the present working point of the system. These optimal working points for different working conditions can be found by an optimization process. To be able to optimize it you have to describe the model mathematically. In principle, *Modelica* is a very good language to describe complex systems in a mathematical form and *Dymola* is a good and easy to use solver for it. With *Modelica* you can build up models very quickly and it's easy to modify these models.

But the normal way to use *Modelica* is to build models of complex systems to simulate them in time. This does not help here, because we need to dissolve such systems at one point in time for wanted unknowns under the assumption that certain values are known. In this case, it is no simulation issue anymore! This is not only useful for optimizing working points of dynamical systems (e.g. minimum consumption at a specified output power) but also for finding extreme working points of it (e.g. maximum possible Output torque of a driveline). It would be nice to benefit from *Modelica* also for this type of task.

In principle, *Dymola* does all the critical work to solve this problem: It manipulates the originally given equations to make the numeric system dissolvable and generates C-Code, but there is no standard way to make it useable for the described task.

The following approach to do this is elucidated on the basis of optimizing the operation of hybrid drivelines.

2 Optimizing the work point of a hybrid driveline

The prior task of a hybrid driveline is to save fuel and minimize emissions. Furthermore there are many other conditions that have to be taken into account

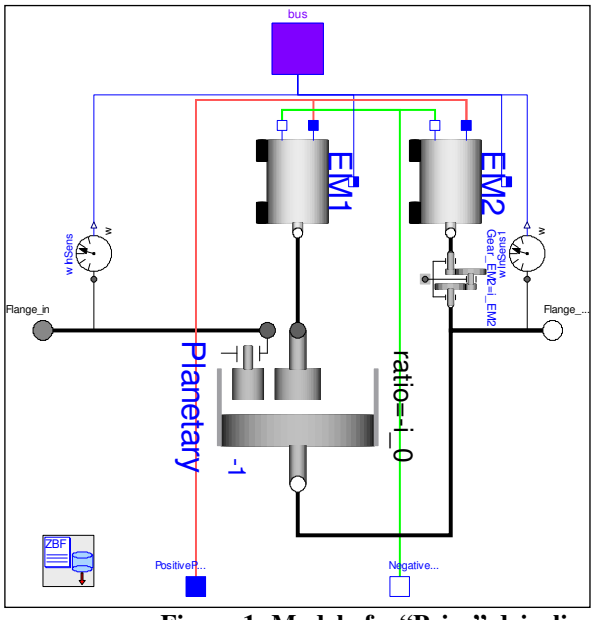

Figure 1: Model of a "Prius" driveline

like State of charge of the battery, performance demands and comfort issues. To get the "best compromise" of these goals you need a driving strategy that operates the hybrid system always on an appropriate working point.

2.1 Task of a hybrid driving strategy

A driving strategy must interpret the driver command given by the accelerator α and brake pedal β as a demanded torque $\hat{T}_{Out}$ at the actual driving speed ω_{Out}.

$$\hat{T}_{Out} = \hat{T}_{Out}(\alpha, \beta)\Big|_{\omega_{Out}}$$

Eq. 1

Dependant on the driveline topology there are many possibilities to perform this torque. Taking the "Toyota Prius system" (Figure 1) as an example of a power split transmission, this torque can be delivered as a combination of torques from the internal combustion engine (ICE), and both electric machines. You could define the work point with some heuristic rules in the driving strategy but then it's very difficult to get the over all optimum of the whole system. Another approach is to get this optimum using an optimizing process. Therefore you need all the torques and speeds of the ICE and all EM's.

When desired speed $\hat{\omega}_{ICE}$, acceleration $\dot{\hat{\omega}}_{ICE}$ and torque $\hat{T}_{ICE}$ of the ICE, furthermore the desired acceleration of the vehicle, represented as angular acceleration at the gearbox output $\dot{\hat{\omega}}_{Out}$, are selected as degrees of freedom, a well-defined system can be dissolved to this form.

$$(T_{EM1}, \omega_{EM1}, T_{EM2}, \omega_{EM2}) = \Phi\left(\hat{T}_{Out}, \omega_{Out}, \dot{\hat{\omega}}_{Out}, \hat{T}_{ICE}, \hat{\omega}_{ICE}, \dot{\hat{\omega}}_{ICE}\right)$$

Eq. 2

You don't have to describe the system in steady-state! However by setting the accelerations to zero, you get this special case.

This equation can be used to get the torques, speeds (and accelerations) of the EM's and in consequence the actual consumption of the ICE and the current through the battery.

2.2 Components that effect a performance index

To be able to optimize the hybrid system a performance index *G* (Eq. 3) has to be defined. In the end it depends on the actual consumption *C* of the ICE and the current *I* through the battery that in turn depend on the demanded torques and speeds given from the driving strategy.

$$G(C, I) = G\left(\hat{T}_{Out}, \omega_{Out}, \dot{\hat{\omega}}_{Out}, \hat{T}_{ICE}, \hat{\omega}_{ICE}, \dot{\hat{\omega}}_{ICE}\right)$$

Eq. 3

There are a lot of components that have effect on these values:

- Vehicle with its driving resistance
- ICE
- EM's
- Battery
- Gears in the driveline

The mathematical description of each component can be rather complex. Just as complex is the interaction between these components. For a *Modelica* user the obvious way to address this problem would be to build a model of the complete driveline using detailed models of the components. It's the task of the *Modelica* Interpreter / Solver to build a system of nonlinear equations that can be dissolved numerically.

3 Modeling the hybrid driveline for numeric solving

3.1 Defining knowns and unknowns

Having the mentioned components as model component in *Modelica* it is easy to build a model of the complete hybrid driveline. By using the block

`Modelica.Blocks.Math.TwoInputs`,

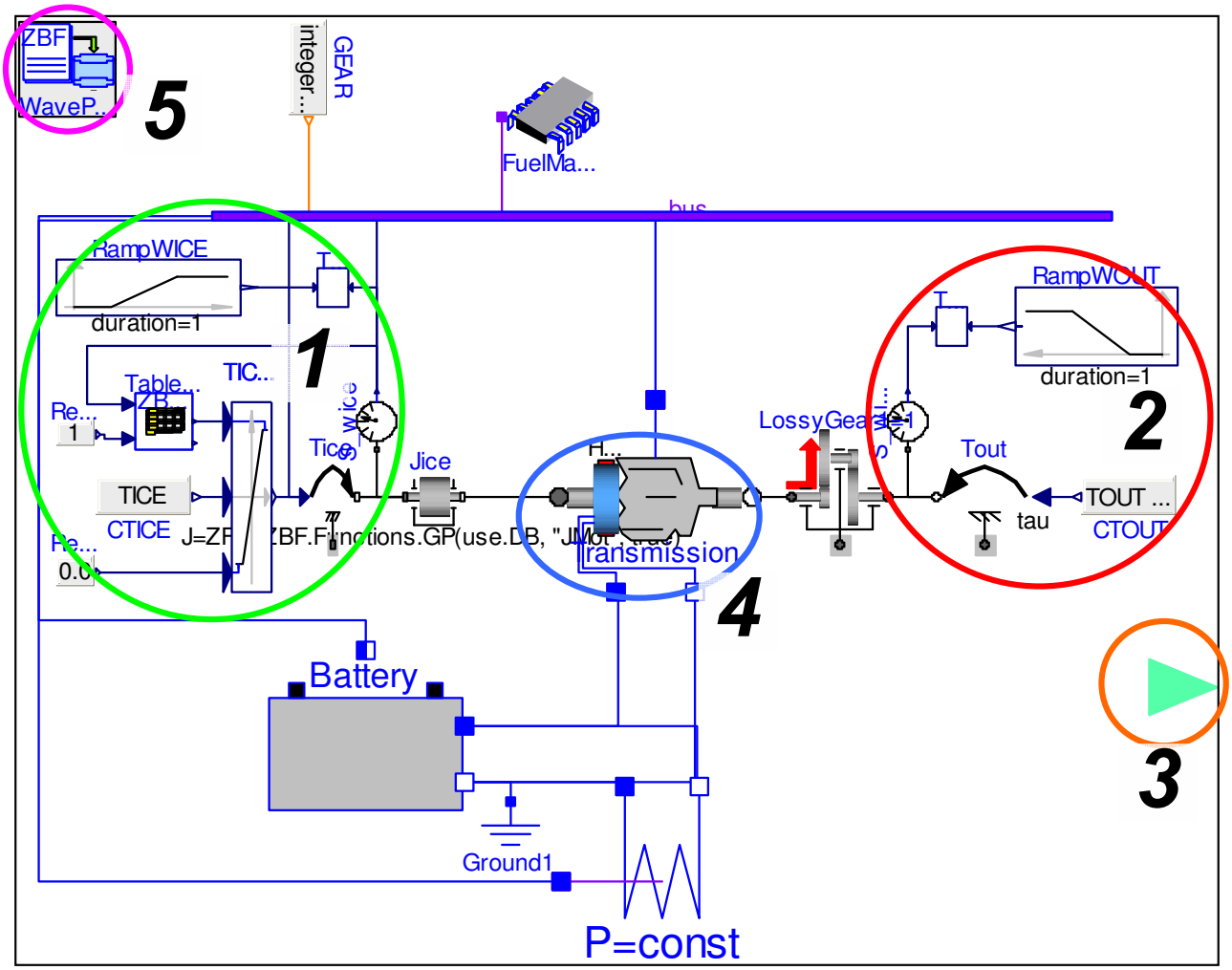

Figure 2: Modelica model of a hybrid driveline with given inputs and out put

The input values from function Φ (Eq. 2) can be defined as inputs to this model. It turned out to be the best way to declare the inputs as parameters in *Modelica*. The block

`Modelica.Blocks.Sources.Ramp`

is the best way to define the speeds and accelerations by putting the parameters in here.

In Figure 2 the Ramp for the ICE speed and acceleration can be found in circle No. 1 that is connected to the `TwoInputs` block. Accordingly the speed of the output shaft is defined in the circle No. 2 of Fiqure 2. The desired torque of ICE and output are defined in a Modelica expression block.

As output a connector (circle No. 3 in Figure 2) with the wanted unknowns is defined and connected to the appropriate components or bus signals.

The main driveline is represented by the transmission model in circle No. 4 of Figure 2. You can just replace this component by another transmission if you want to. The frame around it just keeps the same.

In this special case, to calculate the unknown values for a certain point in time the initialization feature of *Modelica* / *Dymola* is used.

3.2 Modifications of components for numeric solving

Depending on the possibly variable structure of the transmission, the number of degrees of freedom can change. This has to be handled in the `initial equation` block of the complete model. Another possible problem can be “inconsistent speeds” because of any ratios within the driveline or clutches.

The simpliest case is a clutch that is known to be stuck. Here it is useful to replace it by a fixed connection.

If there is a clutch used as launching unit in a parallel hybrid, it is a bit more complicated. We have “inconsistent speeds” if the given speeds of the ICE $\hat{\omega}_{ICE}$ and the output ω_{Out} does not match the condition

$$\hat{\omega}_{ICE} = i_{Gear} \cdot \omega_{Out}$$

Eq. 4

where i_{Gear} is the gearbox ratio. In this case the launching clutch has to be in a slipping state. Otherwise it has to be stuck. With the standard clutch model in the Modelica standard library, Dymola is not able to solve the problem. Introducing a "special clutch" just as a torque element, the system is solvable for Dymola (See Figure 3).

Doing it this way, Dymola calculates the needed torque `f_normalized` to satisfy the given inputs.

```
model Clutch_OutT extends
  Modelica.Mechanics.Rotational
    .Interfaces.Compliant;
public
Modelica.Blocks.Interfaces.RealInput
  f_normalized;
equation
  tau = f_normalized;
end Clutch_OutT;
```

Figure 3: Modelica Code of specialized clutch

Another point to be taken care of are possible delays in form of `FirstOrder` or `SecondOrder` blocks that are inserted in the model to make its behaviour smooth. In normal simulation mode they start normally at a zero initial state and get rather fast to their "steady state", but if you want to get this "steady" result just at initialization of the model you have to think of eliminating these blocks or to define some extra initial equations to avoid this problem.

4 Generate the numeric solver

After doing the steps described above, a *Modelica* model exists, that can be used to calculate the wanted unknowns with a defined input. You can perform this calculation within *Dymola* interactively by just starting the model for a short period of time and look for the results at initial time. But if you want to optimize working points you have to do this very often. In addition to this you should be able to make this calculation available for an optimizing algorithm as they are implemented for example in Matlab. Therefore the Modelica model built above has to be exported into an appropriate environment.

4.1 MEX-function generation in *Matlab*

Dymola provides an export to *Matlab Simulink* in form of a so called S-Function. To do this it embedds the C-Code generated during the translation process in the S-Function interface of *Simulink*. But this does not help her because the model shall not be simulated but be called for one point in time with certain inputs. To provide this functionality one reasonable way is to import the generated C-Code into *Matlab* as a so called MEX-function. Instead of the S-Function interface the MEX gateway function

```
void mexFunction(int nlhs, mxArray
                 *plhs[],
                  int nrhs, const
                 mxArray *prhs[])
```

has to be used. To dissolve the hybrid driveline, the code generated from *Dymola* (`dsmodel.c`) is called within this gateway function. In the generated code a function

```
dsblock(&idemand, …, inputs, para-
        meters, …, outputs)
```

is implemented that contains all system equations. The function dsblock can perform different tasks like initialization, calculating derivatives, handling events and so on. For this task, only the initialization functionality is needed. Here's the calling sequence:

1. Set `inputs` and `parameters` of the model from `prhs` of mex function.
2. Call initialization of `dsblock`
3. Put the wanted values from `outputs` of `dsblock` into `plhs` of mex function.

You can compile this MEX-gateway function with the MEX-compiler and get a DLL that can be called from the *Matlab* command line or in a *Matlab* M-Script as any other *Matlab* function. In this form you can use it for example within a fitting function for the *Matlab Optimization Toolbox*.

4.2 Parametrizing the function

For one kind of driveline the mex function should be built only once. As a consequence the parameters must not be defined within the model. Therefore all application dependable parameters are defined in ASCII files that can be read by the model itself during initialization. This is described in [1]. When including the generated code from *Dymola*, we have to insert a string variable to define the name of the ASCII file that shall be read by the model, because there is no possibility to define this as a variable within *Modelica*. This string variable is an input variable of the MEX-function and is propagated to the model.

5 Conclusions

The main goal to make the efforts described above is to get optimized driving strategies for many variants of hybrid drivelines in a short period of time.

Modeling complex systems in *Modelica* / *Dymola* is easy to do and fast. So this is the most efficient way to describe them and make them available for optimization issues. A lot of hybrid drivelines could be provided to optimization process this way. The final results when using the optimized working point in a simulation are very good.

The possibility to dissolve such systems not "only" for time simulation is a great enrichment in many other engineering tasks apart from optimizing hybrid driving strategies.

References

[1] Köhler J., Banerjee, A. Usage of Modelica in modeling transmissions in ZF, ZF Friedrichshafen AG, 2005.