Peter Junglas

Mathematical Problems due to Oversimplification

Abstract. An important step in modeling is simplification: A model should contain only those aspects of a system that are relevant for the modeling purpose. But sometimes the exclusion of seemingly unimportant physical details leads to models that are mathematically ill defined and cannot be simulated. In such cases it can be helpful to include a rudimentary version of the omitted physics. Using four examples from different application areas it will be shown, how this can be done efficiently and without the introduction of lots of additional parameters, for which proper values would have to be supplied.

Introduction

The modeling of a technical system always begins with the reduction of its complexity. This serves several purposes: First of all, it reduces the size of the model and the simulation time. More important is the decrease in the number of parameters one has to supply. However, the fundamental point is the simplicity of the obtained model, which allows to identify the core mechanism of the problem under study.

But in some cases this crucial step leads to models which are difficult to simulate. A typical error message in Simulink would be

`Cannot solve algebraic loop`,

whereas in a Modelica-based environment one encounters the notorious message

`Failed to reduce the DAE index.`

While a novice modeller might suspect a bug in the simulation program, more experienced modellers look for a bug in their own model. But the real problem may lie in the mathematical description of the model and is often hard to find. The cause of the problem might be a case of oversimplification: Some physical details have been erased, because one is not interested in them, but the resulting mathematical representation of the model becomes incomplete or inconsistent.

A simple remedy seems to be to fix the mathematical problem, but this can be difficult to implement in a given simulation environment or may lead to unphysical behaviour. Often, a better approach is to fix the physics of the model by adding additional elements. This can make the mathematical problems disappear, but leads to new challenges: The expanded model requires (often lots of) new parameters that are unknown and in which one is not interested. Therefore, one needs a simple model of the missing physics that is sufficient to cure the mathematical description without introducing too many new quantities.

In the following, we will describe four examples of oversimplification from very different applications and show, how they can be fixed without introducing a lot of additional details.

Example 1: RS Flip-flop

The first example is the RS Flip-flop, a basic digital storage element with two inputs (S, R) and two outputs (Q, $\bar{Q}$) (Figure 1). Its behaviour is

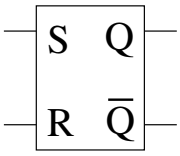

Figure 1: RS Flip-flop.

given by the following table:

S	R	Q	$\bar{Q}$	Remark
1	0	1	0	store is set (i. e. 1 is stored)
0	1	0	1	store is reset (i. e. 0 is stored)
0	0	hold	hold	last value is hold (stored)
1	1	0	0	forbidden!

Using standard electronic elements, it can be built with OR and NOT gates in an astonishingly simple way (Figure 2).

To model an RS Flip-flop in Simulink, one can utilize the OR and NOT components from the Simulink standard library to implement it directly along the lines of Figure 2. But after providing some additional input and output elements, running the model leads to the error message

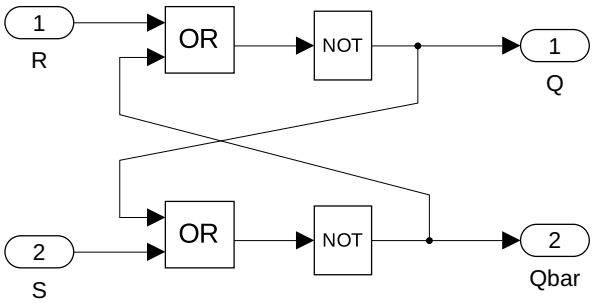

Figure 2: Simple implementation of an RS Flip-flop.

```
Error:  Cannot solve algebraic loop involving
'rsflipflopA/RS flip-flop/Or2' because it consists of
blocks that cannot be assigned algebraic variables ...

Consider breaking the algebraic loop.  For example,
add a delay or a memory block to the loop ...
```

The mathematical problem leading to this error is the simple fact that for input values $S = R = 0$ the model can have two possible output values for $(Q, \bar{Q})$ – namely $(0, 1)$ or $(1, 0)$ according to the previously stored value.

The real electronic system works well, but shows an interesting initialization behaviour: If one switches it on with input values $S = R = 0$, the output value is random between different circuits, but generally constant for a given one. The reason for this behaviour is the obvious fact that the transport of physical signals needs time. Depending on complex details like inhomogenities in the semiconductor crystals that make up the components, the internal delays vary between different circuits, which makes one or the other of the two outputs "win".

These considerations – and the error message above – suggest that one can solve the simulation problem simply by adding a `Memory` component to the model (Figure 3), which introduces a small delay. The improved model works as requested, even without defining an additional parameter, and the initial output value can be set inside the `Memory` block.

A possible problem may be that the actual delay is defined by the ODE solver and might be larger than expected. A slightly more elaborate model that is available in the Simulink standard library [1] cures this behaviour.

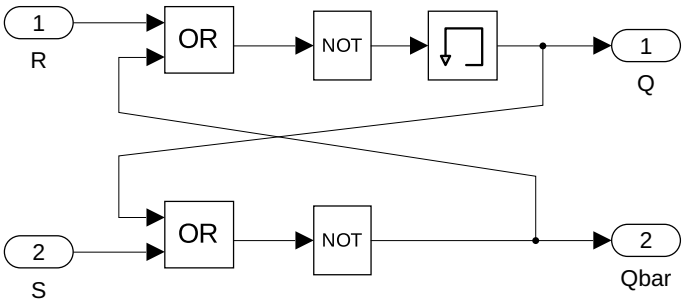

Figure 3: Working implementation of an RS Flip-flop.

Example 2: Gear Shift

In the field of automotive engineering, an important task is the modeling and simulation of a drive train, which consists of all parts that transport the power from the engine to the wheels. An important component is the manual transmission, which allows to change the ratio of rotational speeds from motor and drive shaft. It usually provides a few fixed ratios ("gears") and has a complex internal structure (Figure 4).

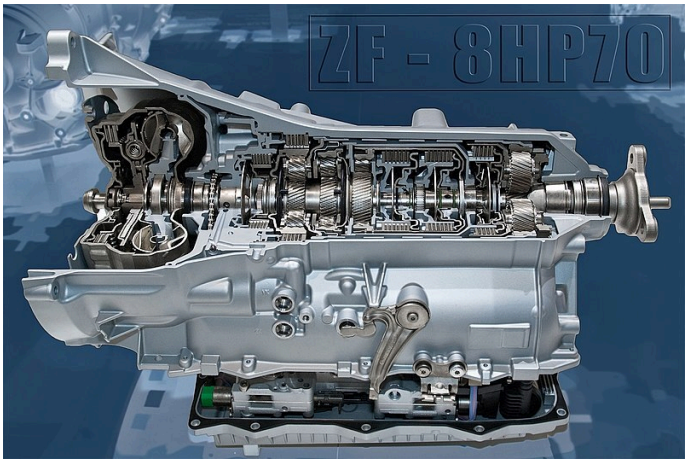

Figure 4: Eight speed transmission [2].

A corresponding drive train model using Modelica has been studied in the text book [3]. For simplicity the model has been reduced here and ends after the drive shaft and a final inertia, which represents all further parts such as a differential gear and wheels (Figure 5). The basic component of the gear shift model is a simple variant of the `IdealGear` from the Modelica Standard Library. It has two connectors describing torque τ and angle φ on each side and an additional input that specifies

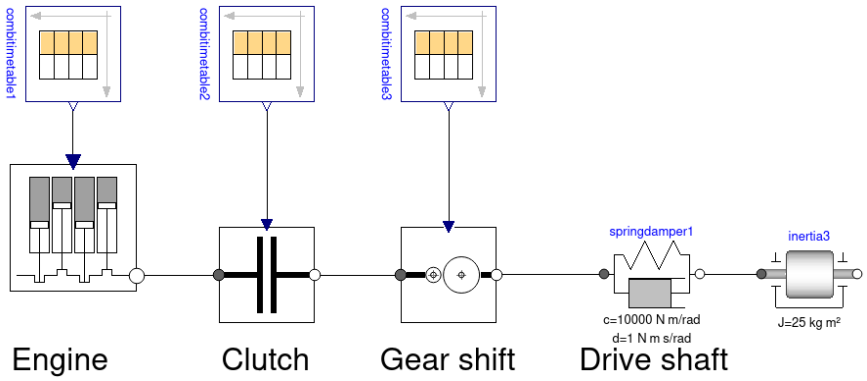

Figure 5: Simplified drive train model.

the transmission ratio r, and it defines the usual equations

$$\frac{\tau_i}{\tau_o} = -\frac{1}{r}\,, \qquad \frac{\varphi_i}{\varphi_o} = r\,.$$

The complete gear shift model adds inertias on both sides and a table that computes the ratio from the gear (Figure 6). Running the model in

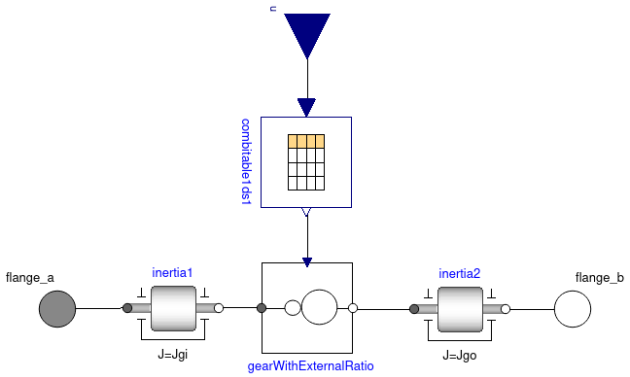

Figure 6: Simple gear shift model.

Dymola works perfectly well, as long as the gear is constant. But when the model describes a gear change, the simulation doesn't start, but the error message `Failed to reduce the DAE index` is displayed.

Again, the origin of the error is a mathematical problem: When the gear changes from r_1 to r_2, the rotational speeds on the input and output side of the transmission have to change from $\omega_i/\omega_o = r_1$ to $\omega_i/\omega_o = r_2$. But their start values after the event are not defined anywhere in the model. A simple solution would be to fix one of the two ω values and compute the other one, but this often leads to an unrealistic behaviour of the model.

That the speeds at the two sides of the transmission have to be adapted during a gear shift, is a real-world problem. In older cars, the driver had to change one speed manually, a technique called "double-clutching". A modern transmission uses synchronizer rings to adapt the rotational speeds using friction – basically, they work like tiny clutches. As a result, the speeds adapt dynamically, mainly according to the connected inertias on both sides.

The improved gear shift model implements this idea in a very simple way: The gears are given by fixed `IdealGear` components and are all connected to the engine via clutches (Figure 7). The clutch of the current

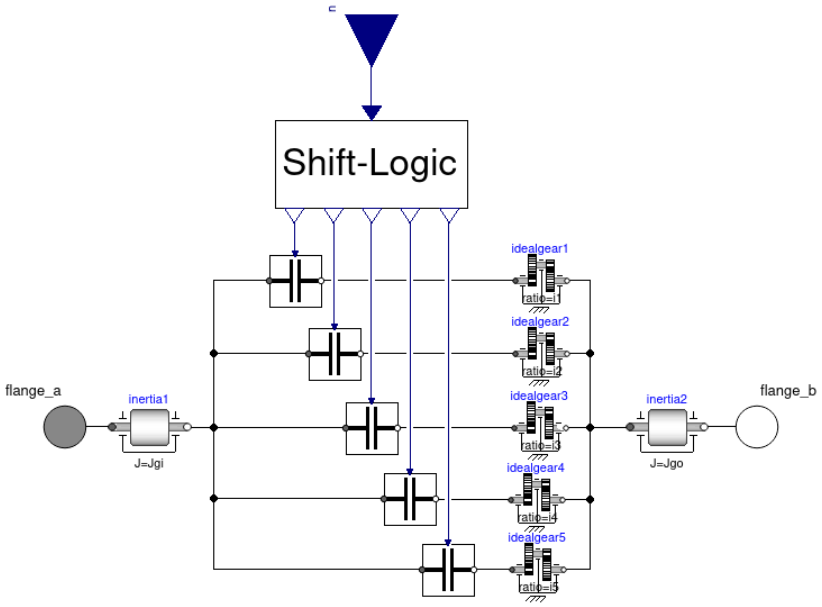

Figure 7: Improved gear shift model.

gear is closed, all other clutches are completely disengaged. A shift-logic component detects a gear change and gradually opens or closes the corresponding clutch using a given switching-time constant. This is the only additional parameter that is needed for the extended model. Its value can either be measured in real transmission processes or chosen to minimize the vibrations that appear in the model during the gear switch (Figure 8).

Example 3: Queueing system

In many application areas, such as logistics, manufacturing or computer networks, a standard situation appears over and over again: Entities (cus-

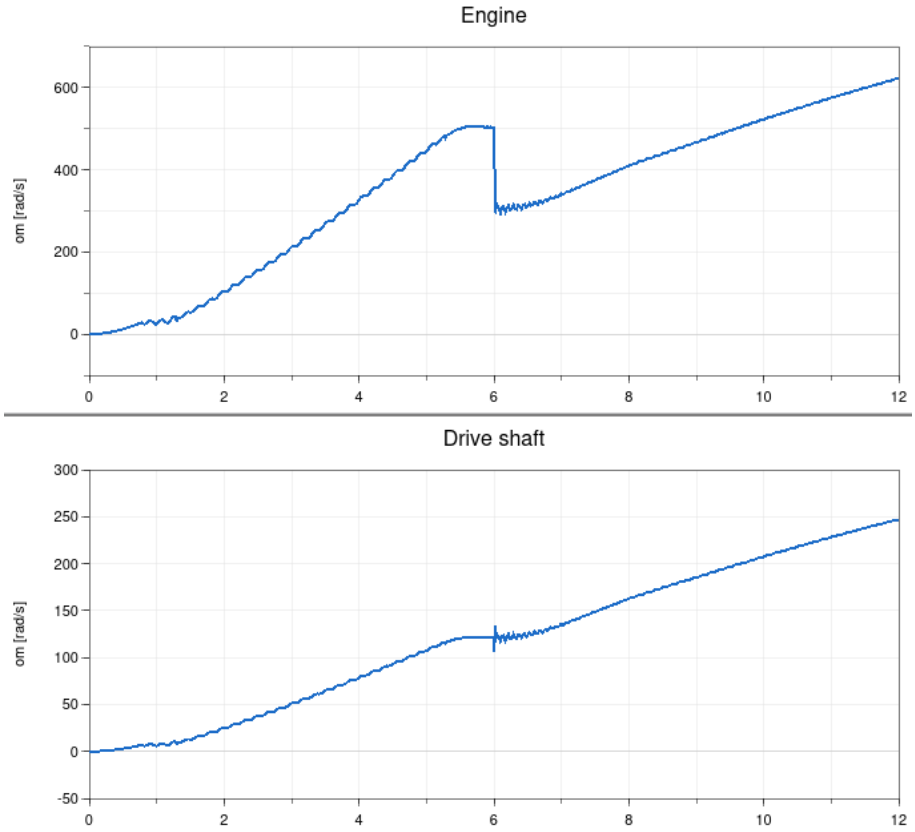

Figure 8: Engine and drive shaft speeds during a gear shift.

tomers, parts or data packets) appear in regular or random time intervals and wait in a queue until they are served (by a checkout, a machine or a router). Figure 9 shows the basic parts of such a queueing system together with an important detail: The server sends a signal *bl* to the queue, whenever its state (*free* or *working*) changes, which in turn changes the state of the queue output between *open* and *blocked*.

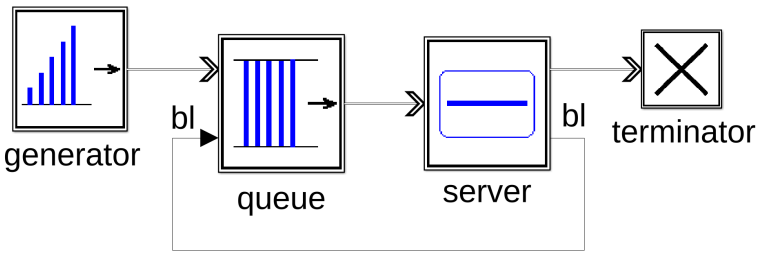

Figure 9: Basic queueing system.

A widely used modeling technique to describe such systems is the discrete event-based modeling, where components change their states for only one of two reasons: Either an external event arrives or an internal

state change is due that has been scheduled before. The DEVS formalism [4] has been introduced to define such models, together with their behaviour in a simulator, with mathematical precision. In order to simplify the construction of large models consisting of many basic components the variant RPDEVS has been proposed in [5], which especially allows for Mealy components, where output events can depend directly on the input events. To make this possible the abstract model of the simulator had to be adapted [6], it now contains special internal iterations ("λ iterations") to follow a chain of events through the system. A concrete implementation of the RPDEVS simulator, called PowerRPDEVS, and a set of basic components, are available freely from [7].

The basic components of the queueing example can be easily built using C++ and a set of basic functions that are provided with the simulator. The construction of the complete example is done in a graphical environment. Simple test models can be used to check the correctness of the components, but running the complete model still leads to the error message

```
Error at t = 1.
maximum number of lambda steps has been reached.
illegitimate model due to a non-resolvable algebraic
loop.
```

Tracking the outputs through the complete model, the solver iterates, until a steady state is reached. During the loop, outputs can be withdrawn, if the states of the components change. This leads to the following sequence of events in the example:

- generator sends entity to queue,
- queue sends entity to server,
- server sends $bl = true$ ("blocked") to queue,
- queue changes output state to "blocked",
- queue doesn't send entity (i. e. previously sent entity is retrieved)
- server is not blocked,
- queue sends entity to server ...

The sequence repeats, until it is stopped with an error, when the maximal number or λ-steps is reached.

The mathematical problem behind the scenes is that the states and output values of the queue and server components are undefined at $t = 1$. The solver tries hard to find proper values, but does not succeed. Actually, both components go through several states and outputs in immediate succession (an "event cascade"), which is a very reasonable behaviour, but can not be resolved by the simulator.

Going back a step and having a look at some of the concrete systems, one wants to model here, one finds how the problem is solved in reality: The transport of a signal and the change of a state always need a (usually small) time. Therefore the steps in an event cascade happen at increasing times. But introducing these time delays in the modeling leads to a large amount of additional parameters with unknown values, which clutter up the model and have to be supplied somehow.

A way how to include delays without adding an abundance of new parameters has been proposed in [8]: by using infinitesimal delays for all signal transports and "immediate" state changes. This can be done in a mathematically sound way by resorting to the field of hyperreal numbers ${}^*\mathbb{R}$ [9], which extends $\mathbb{R}$ by a value $\varepsilon > 0$ that is smaller than any positive real number. Being a field, ${}^*\mathbb{R}$ contains numbers such as ε^2, but for the implementation in a simulation program, time values of the form $a + b\varepsilon$ with $a, b \in \mathbb{R}$ are sufficient. To explicitely order concurrent events, one can use different values of b, but very often, a simple default delay ε is sufficient almost everywhere. This has been confirmed by a systematical study of a set of standard examples in [10]. The only exception that has been found are components that – like the queue – can output a "concurrent" sequence of several entities. Here the internal delay has to be set to 2ε, which can be predefined in a standard component library. In this way, the number of delay parameters a user has to consider, can be reduced drastically to an easily manageable number. With a corresponding simulator [11] the queueing example runs without problems and reproduces the expected results.

Example 4: Distribution of compressed air

Pneumatic systems are used frequently to distribute power in factory buildings. They often consist of large distribution networks with several pressure sources and a lot of time-varying consumers. A very simple example with one source, a tee branch, a pipe and two different consumers is shown in Figure 10.

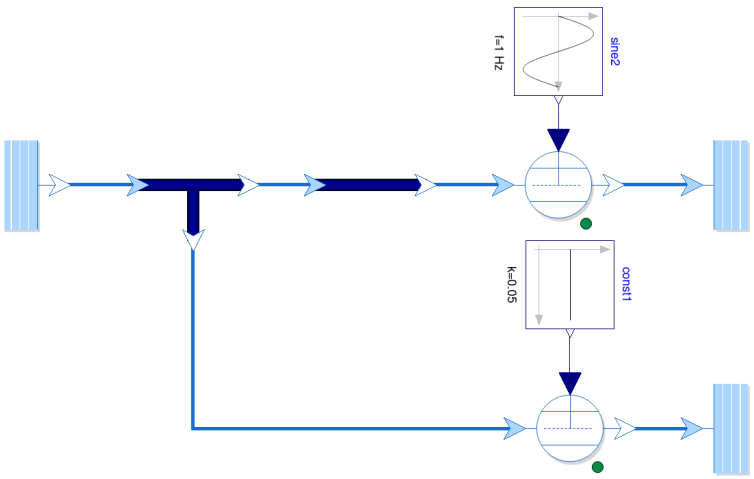

Figure 10: Simple pneumatics system.

To model this system in Modelica, one can use the Fluid library that is part of the Modelica Standard library (MSL). It basically describes quasistatic processes and contains components for one-dimensional thermofluid flow in pipes, vessels or machines [12]. For the special needs of pneumatics modeling, the additional PneuBib library has been introduced in [13] that is based on the MSL Fluid library. Using these tools in Dymola, the example can be implemented easily, but running the simulation, one gets the error message:

```
Warning:  Failed to solve nonlinear system using Newton
solver...
At time T = 5.130275e-04 ...  the corrector could not
converge because there were model evaluation failures
and the stepsize cannot be reduced further.
Integration will be terminated..
```

The origin of this error does not lie in the mathematical formulation of the model, but in the numerical method. The model consists of 170 equations, many of which are coupled and highly non-linear. To solve them with a standard Newton solver, one needs proper initial values, but

these are unknown here, and the values guessed by the solver do not work – the Newton iteration does not converge. This is a well-known problem, when using the MSL Fluid library, therefore mathematical methods based on homotopy have been introduced in Modelica to solve the initialization problem [14]. But at least the authors of [13] were not able to find a working solution along these lines.

The model starts with given initial values for the pressure and the mass flow of the consumers. But in reality a system usually starts with an ambient pressure and zero mass flow everywhere. After switching on the compressors, the global pressure difference rises. Accordingly, the mass flows increase with a small delay due to the inertia of the accelerated fluid masses. This in turn leads to the build-up of the local pressure differences, which finally reach the values that the quasistatic model tried to find immediately.

Following these lines, Zimmer incorporated the described real-world behaviour as simply as possible to construct a thermo-fluid library that is not plagued by initialization problems [15]. For this purpose, he extended the model equations by introducing the inertial pressure r, which – according to the Euler equation – is given by

$$r = -L\frac{d\dot{m}}{dt}.$$

The inertance L is defined by the geometry of a component and is usually set to a model-wide default value. The actual pressure p can now be split into a quasi-static part $\hat{p}$ and the inertial part,

$$p = \hat{p} + r,$$

where one is interested only in the behaviour of $\hat{p}$.

Still, the coupling of several components leads to large systems of non-linear equations. This problem is solved in [15] by using the quasi-static pressure $\hat{p}$ instead of the actual pressure p in the junction equations. The transient behaviour at the initialization phase is now only computed approximately, but one is not interested in it anyhow. As a consequence, the huge coupled system is decomposed into independent small non-linear parts, while the total coupling is done by the relaxation using a simple linear differential equation.

These ideas have been incorporated into the DLR ThermoFluidStream Library (TFS) [16], which provides components for vessels, pipes and machines and is freely available. Using it as the basis for the PneuBib library, the example model runs without problems. Its simulation results are shown in Figure 11 and clearly show the progression from the approximate relaxation at $t < 1$ to the quasistatic behaviour for $t > 1$.

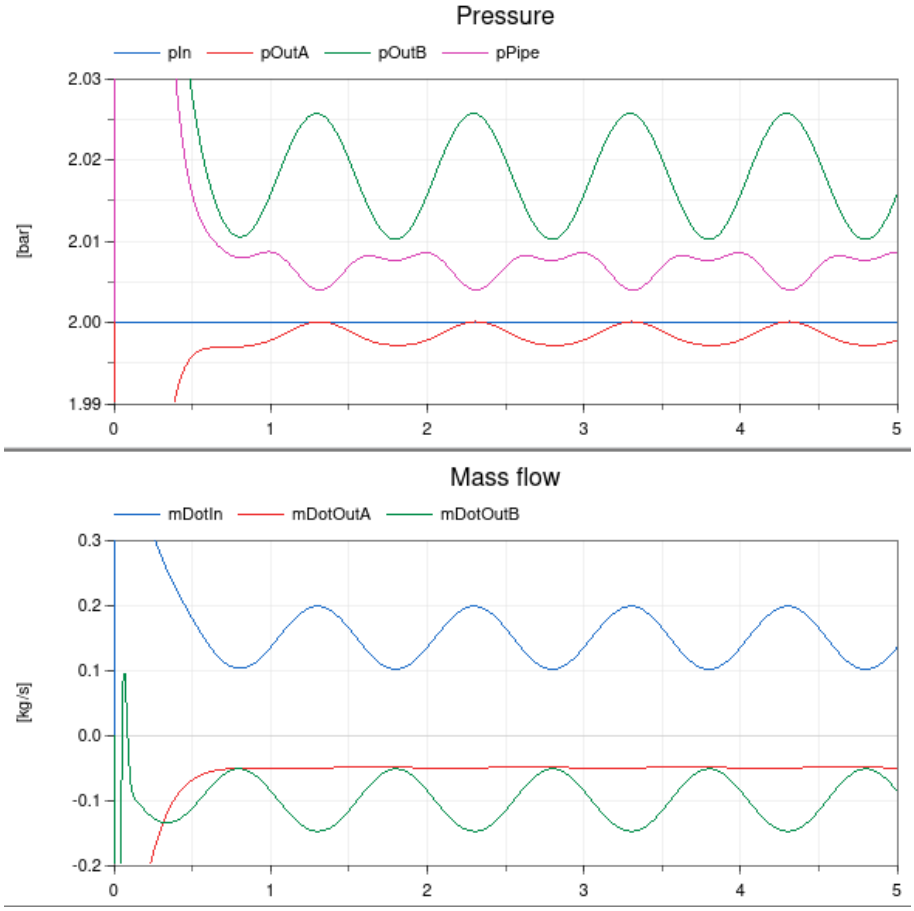

Figure 11: Pressures and mass flows in the TFS based pneumatics model.

Conclusions

As we have seen, oversimplification can lead to very different kinds of mathematical problems. In the examples we have found an equation with several solutions (ex. 1), undefined restart values (ex. 2), insufficient start values for a Newton solver (ex. 4) and "functions" with several values at the same time (ex. 3).

The reason, why these problems appeared, was the neglect of certain physical details. Though the examples and the mathematical problems

were quite different, the physical reason was basically always the same: the assumption that physical processes, which we are not interested in, can be modeled as if they occured in zero time. This is obvious in ex. 1 and 3, where the propagation time of signals or the switching time of states have been discarded. But it is true even for ex. 2 and 4, where the mass inertia has been omitted, which leads to instanteneous changes instead of gradual relaxation or adaption processes.

While the inclusion of the missing physics in the model can resolve the mathematical problems, it leads to new difficulties, namely, the appearance of – often many – new parameters that need meaningful values. The examples have shown different ways how to cope with them: An easy way is to aggregate them all into one place (ex. 1). If this is not possible, because the effect has to be included in many parts of the model, one can set the parameters to a default value (ex. 2, 4). This has the advantage that one can adapt individual values, where necessary. A different strategy has been used in ex. 3: The unknown physical values have been replaced by abstract entities, which subsume the basic properties ("very small, but not zero") without needing a concrete numerical value.

One could boil down the lessons learned from the four examples into a short recipe, what to do, when the simulator strikes (and it is definitely not a bug):

1. Isolate the mathematical problem.

2. Find the physical cause.

3. Model the missing physics as simply as possible.

4. Avoid the introduction of many new parameters.

Unfortunately, these steps are often hard to implement in real-world problems. They need insight into the physical properties of the system at hand and creativity especially for steps 3 and 4. Nevertheless, it is often worth the effort, because instead of a model containing obscure hacks to "fix the mathematics" one gets a model that is easier to understand and generally more robust.

Bibliography

[1] **The MathWorks, Inc.**: *Simulink: Simulation and Model-Based Design.* https://www.mathworks.com/products/simulink/ .

[2] **Stefan Krause**: *BMW's ZF 8HP 8 speed transmission.* https://commons.wikimedia.org/wiki/File:ZF_Stufenautomatgetriebe_8HP70.jpg, licensed under Free Art License (https://en.wikipedia.org/wiki/en:Free_Art_License).

[3] **Junglas, P.**: *Praxis der Simulationstechnik.* Verlag Europa-Lehrmittel Haan-Gruiten (2014).

[4] **Zeigler, B. P., Muzy, A.; Kofman, E.**: *Theory of Modeling and Simulation.* Academic Press San Diego, 3rd ed. (2019).

[5] **Preyser, F. J.; Heinzl, B.; Kastner, W.**: *RPDEVS: Revising the Parallel Discrete Event System Specification.* In: Proc. 9th Vienna Int. Conf. Mathematical Modelling, Wien, Austria (2018).

[6] **Preyser, F. J.; Heinzl, B.; Kastner, W.**: *RPDEVS Abstract Simulator.* SNE Simulation Notes Europe, 2, **29**, 79–84 (2019).

[7] **Preyser, F. J.**: *PowerRPDEVS on sourceforge.* https://sourceforge.net/projects/powerrpdevs/ .

[8] **Junglas, P.**: *NSA-DEVS: Combining Mealy Behaviour and Causality.* SNE Simulation Notes Europe, 2, **31**, 73–80 (2021).

[9] **Goldblatt, R.**: *Lectures on the Hyperreals.* Springer New York (1998).

[10] **Jammer, D.; Junglas, P.; Pawletta, T.; Pawletta, S.**: *Implementing Standard Examples with NSA-DEVS.* SNE Simulation Notes Europe, 4, **32**, 195-202 (2022).

[11] **Jammer, D.; Junglas, P.; Pawletta, T.; Pawletta, S.**: *A Simulator for NSA-DEVS in Matlab.* In: Proc. of ASIM 2022 – 26. Symposium Simulationstechnik, Wien, Austria (2022).

[12] **Franke, R.; Casella, F.; Sielemann, M.; Proelss, K.; Otter, M.**: *Standardization of thermo-fluid modeling in Modelica.Fluid.* In: Proc. 7th Int. Modelica Conference, Como, Italy (2009).

[13] **Drente, P.; Junglas, P.**: *Simulating a simple pneumatics network using the Modelica Fluid library.* SNE Simulation Notes Europe, 2, **25**, 85–92 (2015).

[14] **Casella, F.; Sielemann, M.; Savoldelli, L.**: *Steady-state initialization of object-oriented thermo-fluid models by homotopy methods.* In: Proc. 8th Int. Modelica Conference, Dresden, Germany (2011).

[15] **Zimmer, D.**: *Robust object-oriented formulation of directed thermofluid stream networks.* Mathematical and Computer Modelling of Dynamical Systems, 3, **26**, 204-233 (2020).

[16] **Zimmer, D.; Weber, N.; Meißner, M.**: *The DLR ThermoFluidStream Library.* In: Proc. 14th Int. Modelica Conference, Linköping, Sweden (2021).

Author

Prof. Dr. rer. nat. Peter Junglas
Private Hochschule für Wirtschaft und Technik Vechta/Diepholz
Am Campus 2
D-49356 Diepholz
E-Mail: peter@peter-junglas.de