Input Smoothing for Faster Co-Simulation using FMI

Erik Henningsson[1] Christian Schulze[2] Manuel Gräber[3] Elmir Nahodovic[1] Dag Brück[1]
Julius Aka[4] Oliver Lenord[5]

[1]Dassault Systèmes AB, Sweden, {Erik.Henningsson,Elmir.Nahodovic,Dag.Brueck}@3ds.com
[2]TLK-Thermo GmbH, Germany, c.schulze@tlk-thermo.com
[3]TLK Energy GmbH, Germany, manuel.graeber@tlk-energy.de
[4]University of Augsburg, Germany, julius.aka@uni-a.de
[5]Robert Bosch GmbH, Germany, oliver.lenord@de.bosch.com

Abstract

We present two methods for speeding up co-simulations under the FMI standards. By smoothing the input signals inside each FMU, the internal integrator may avoid re-initialization. This can significantly reduce the number of model and Jacobian evaluations. To further help the integrator we also propose a predictor compensation technique tailored to the input smoother. The main benefit of our methods is the ease-of-use, requiring no model manipulations, nor any special co-simulation master algorithms. The methods are implemented in Dymola 2025x and validated with both an academic mechanical model as well as industrial thermo-fluid examples where we often observe over 10 times faster co-simulation and, in the best cases, up to 400 times faster. One of these thermo-fluid examples is used in the *OpenSCALING* research project to generate training data for constructing surrogate models, for which the input smoothing is especially important to speed up the dataset creation. *Keywords: FMI, Efficient co-simulation, Input smoothing, Thermo-fluid*

1 Introduction

To facilitate simulation of complex and large systems spanning over multiple technical domains, co-simulation techniques are commonly employed. This may, for example, include the integration of several domain-specific tools into one single simulation.

In this article we present two methods for speeding up co-simulation under the FMI standards, versions 2 and 3 (Modelica Association 2022; Modelica Association 2023). Co-simulation of multiple FMUs face several numerical challenges. Among others, these involve stability, accuracy, efficiency, and step-size control. A wide range of methods for handling these challenges can be found in the literature. For stabilization techniques see for example Benedikt and Drenth (2019) for non-iterative co-simulation and Arnold (2010) for iterative co-simulation. For adaptive co-simulation with step-size control we refer to Schierz, Arnold, and Clauß (2012).

Here we consider the efficiency challenge by proposing two methods to speed up the numerical integration inside each co-simulation FMU. More precisely, our techniques are useful for FMUs using variable-step solvers. Firstly, we apply *input smoothing* to create continuous inputs to the FMUs. Thus, we may avoid restarts of the integrators inside the FMUs. To further help the integrator we also present a *predictor compensation* method.

Input smoothing methods have previously been discussed in the literature, see e.g. Andersson (2016) and Busch (2016) where the smoothing is achieved by the master. These references also include stability and consistency analyses of related smoothing techniques.

However, the main benefit of the input smoothing and predictor compensation presented in this article is the ease-of-use. No special co-simulation master is required. Nor do we require any changes to the models that are exported to FMUs.

Indeed, in our implementation in Dymola 2025x you only need to set the corresponding options before exporting your model to an FMU and the input smoother, with or without predictor compensation, will then be included in the generated code. The resulting FMU can then be used with most co-simulation masters, in particular versions of the classical non-iterative, parallel master.[1] Further, FMUs using input smoothing may be co-simulated with FMUs not using it.

Avoiding integrator re-initialization at communication points may improve performance for any variable step-size solver. But this is especially true for multistep methods as they have to restart at order one, with a tiny internal step size. While ramping up both the order and the step size, several costly model and Jacobian evaluations are required. In this article we consider this class of integrators. More precisely the implementations CVODE and IDA from SUNDIALS (Gardner et al. 2022) and DASSL from Brenan, Campbell, and Petzold (1996).[2] In the context of models exported from Dymola these are arguably the most

[1]The exceptions involve masters that set the input derivatives. Further, since the input to an FMU using input smoothing is delayed, such FMUs cannot be used in co-simulation masters which rely on setting an input and then immediately reading the new output without first performing a `doStep`.

[2]The input smoothing does not depend on which solver is used, but the predictor compensation requires modifications tailored to the solver implementation.

widely used variable-step solvers in co-simulation FMUs. Note that fixed-step solvers, e.g. the commonly used Dymola inline methods, do not benefit from input smoothing.

However, even with smoothed inputs, continuing the integration over communication points comes with additional challenges and problems. We identify and address these issues. To alleviate the main problem caused by avoiding integrators restarts, we arrive at the aforementioned predictor compensation. In Andersson (2016), this technique was discussed in a similar context for integrators with a Nordsieck history representation (e.g. CVODE). In this article we extend the idea to the DAE solvers DASSL and IDA and tailor it to work optimally in combination with input smoothing.

To validate these methods we study their application in both academic models and thermo-fluid examples. One important example is how input smoothing can benefit the creation of datasets for training surrogate models. For this purpose we examine the effect of smoothing inputs on a demonstrator of the *OpenSCALING* research project, delivered by the industry partner *Bosch*.

Finally, we note that FMI 3 comes with the powerful capability *Intermediate Update Mode*, where inputs can be communicated between FMUs also during the communication intervals. This allows for more advanced co-simulation masters and can also be used to reduce the number of integrator restarts. As an example we mention the *Transmission Line Method* (R. Braun and Krus 2013; Robert Braun and Fritzson 2022) where physically motivated time-delays are introduced. While our methods are not motivated by physical considerations, their benefit is that they do not require any special modeling methodology or corresponding co-simulation master.

2 Co-simulation with input smoothing

Co-simulation of multiple FMUs can be performed using a variety of methods. For example iterative methods, higher-order input extrapolation, or non-iterative stabilization techniques. However, we consider this to be out-of-scope for the current presentation. Rather, the purpose of this article is to investigate techniques for speeding up the simulation inside each FMU. Therefore, we here only consider the classical parallel co-simulation algorithm (also known as non-iterative or weak-coupling co-simulation).

2.1 Classical co-simulation

During classical co-simulation the FMUs communicate with each other at *communication points*. This means that the input to each FMU may be computed from outputs of other FMUs. The interval between communication points (the *communication interval*) may vary in size. To advance the time from one communication point to the next, a `doStep` call is made in each FMU. The `doStep` computations may be done in parallel.

The classical approach for handling inputs inside each FMU is to keep any inputs constant throughout the communication interval (zero-order hold). Then, at the next communication point a jump is made to the newly received input values.

We refer to the above algorithm as *classical co-simulation* with each FMU using *classical input handling*.

During a `doStep` call an integrator is employed inside the FMU. The here considered integrators CVODE, IDA, and DASSL are all variable step-size solvers. That is, they have their own step-size sequence. Normally, but not necessarily, there are several integrator steps during each communication interval.

When using classical input handling (zero-order hold) the input signal (as seen by the integrator) is piecewise constant with jumps at the communication points. Therefore, the integrator needs to restart at each such point. This is costly for any variable step-size solver, but especially so for multistep methods.

2.2 Input smoothing

In contrast to classical input handling, the purpose of the proposed input smoothing is to construct input signals which are continuous over communication points. The idea is to do this inside the FMU such that no modifications to the co-simulation master or the exported models are required, cf. Figure 1.

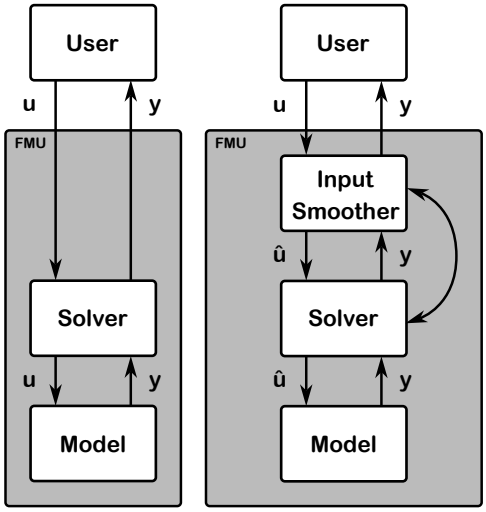

Figure 1. On the left is the abstract structure of a Classical FMU, on the right the modified variant. The input smoothing can be visualized as an additional block between user and model processing the inputs. Input u provided by the user is converted to the interpolated $\hat{u}$ signal. Additionally, an interaction of the input smoothing block and the solver has to be implemented.

Therefore, at each communication point, the input values provided from the master are not immediately set. Instead, we ramp up to these values such that they are reached at the end of the communication interval. In this way, continuous, piece-wise linear input signals are achieved, c.f. Figure 2. This must be done with care to also handle the case of variable communication interval lengths.

DOI 10.3384/ecp218653

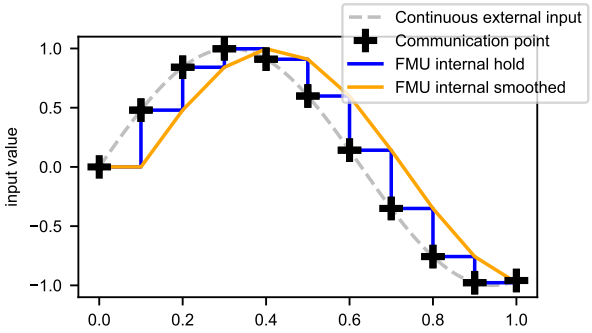

Figure 2. The orange smoothed input signal instead of the blue hold signal is used by the model

As the integrator inside the FMU only sees this smoothed input, then we can allow the integrator to continue integration without restarts at communication points.

An important assumption here is that the integrator does not step past any communication points before receiving new inputs. Otherwise, it would use the previous piecewise linear segment of the input interpolation in the new interval. This is incorrect and also gives a discontinuity in the input signal at the end of the integrator step. An implementation of the input smoothing must make sure that the integrator, at each communication point, has a coinciding internal step. This restriction may lead to an increased number of successful solver steps and Jacobian updates.

The input smoother is enabled in Dymola by setting the flag

`Advanced.FMI.UseInputSmoother`,

and then translating your model to a co-simulation FMU. As algorithm select either CVODE, IDA, or the Dymola solver DASSL. Input smoothing for FMI 2 is supported from Dymola 2025x. FMI 3 support is available from Dymola 2025x Refresh 1.

We have identified two potential problems that may be encountered when using the proposed input smoothing. Firstly, the smoother introduces additional input delays. Secondly, when the integrator uses orders higher than one, mere continuity of the input signals is not enough for optimal performance. These problems are discussed in the upcoming subsections together with proposals for how to handle them.

2.3 Delayed inputs

Already classical input handling introduces input delays as the inputs are set at the communication points and then held throughout the communication interval. The input smoothing makes the situation worse by further delaying the input. This additional delay is effectively half a communication interval as we ramp up to the new inputs, reaching them at the end of the interval. Therefore, input smoothing must be used with care when the co-simulation setup is sensitive to delays.

For such setups, a potential solution may be to slightly reduce the communication interval to compensate for the additional delays. For co-simulation problems where the majority of the CPU time is spent inside the FMUs, then using the input smoother may still be faster, even with the smaller communication interval. This idea is further pursued with the mechanical example model in Section 3.1.

2.4 Discontinuous input derivatives

The input smoothing guarantees that all smoothed inputs seen by the integrator are continuous. But, already the first input derivative is generally discontinuous. When the solver uses any order higher than one, these discontinuities may still cause problems for the integrator.[3] This may result in reduced step size, reduced order, and Jacobian re-evaluations, meaning that we lose some of the benefits from the input smoothing.

The main problem is that the integrator makes bad predictions. The multistep methods considered in this article use old solution values to predict the next step. The predictor can generally be described by a polynomial of the same order as the current order of the BDF method used. This polynomial is evaluated at the time of the next step to get the prediction. For details see Brenan, Campbell, and Petzold (1996) and Radhakrishnan and Hindmarsh (1993).

The prediction is then used as a start guess in the corrector iteration that solves for the actual next step. A bad start guess makes it more difficult for the quasi-Newton method to solve the corrector equations, which may cause extra model evaluations. In the worst case, the Newton method fails, meaning reduced step size, or even Jacobian re-evaluations.

Further, the multistep methods use their predictor to estimate the local error. The estimate is based on the difference between the predictor and the solution of the corrector iterations. When the predictor is bad the error estimate will typically be far too large. The result is that steps are often discarded when they should not be. Even if no steps are discarded, the large error estimates typically result in reduced step sizes and reduced order while the integrator recovers and improves its error estimates.

2.5 Predictor compensation

To alleviate the problem of discontinuous input derivatives, we tailor the predictor compensation of Andersson (2016, Chapter 7) to our input smoother and extend it to the IDA and DASSL solvers. The zeroth- and first-order terms of the predictor polynomial are already reasonable, the first-order term due to the continuity of the input achieved by the input smoothing. Thus, the predictor compensator implemented in Dymola 2025x targets the second-order term. Higher order terms are not considered as they would be computationally too complex to handle with the employed technique, offsetting any benefits.

The predictor compensation is employed as needed at each communication point. It requires an approximation of how the second state derivative is affected by jumps in

[3] In contrast, note that this is not a problem for classical input handling due to the re-initialization done at every communication point.

the input derivative. As the dependency has to be traced through the model, we need to approximate the Jacobian of the state derivatives with respect to the inputs. If the FMU supports directional derivatives (Modelica Association 2022), these can be used for this task.

Dymola FMUs support directional derivatives and require two model evaluations to compute them. This means that the predictor compensation comes with an additional cost that offsets some of the benefits.

To enable the predictor compensation in Dymola, follow the steps in Section 2.2 to enable the input smoothing. In addition, enable the flag

```
Advanced.FMI.UsePredictorCompensation.
```

As algorithm select either CVODE, IDA, or the Dymola solver DASSL. Predictor compensation is supported in the same Dymola versions as input smoothing, except for DASSL where support starts with Dymola 2026x.

3 Applications

We present a simple mechanical model along with two different thermo-fluid examples to highlight the benefit of input smoothing. As mentioned in Section 2.3, the delayed inputs can introduce errors in the simulation. The mechanical example is especially sensitive to delays, and a method to handle it is presented in Section 3.1. The thermo-fluid models studied in Section 3.2 and Section 3.3 are also affected by delays, but for these examples the focus lies on the performance gain of the input smoother.

3.1 Simple mechanical example

To illustrate the effects of input smoothing and predictor compensation we start with a simple mass-spring-damper system. The example is selected not only to showcase the improved simulation statistics, but also to see the effects of input smoothing on a model famously sensitive to co-simulation delays.

3.1.1 Setup

Consider the mass-spring-damper system in Figure 3. We split the model at the connection between the mass component and the spring-damper component. Then we export each submodel into an FMU. From the mass FMU, position and velocity is sent to the spring-damper FMU. The latter sends the force to the former. Please see Table 1 for non-default physical and numerical parameters.

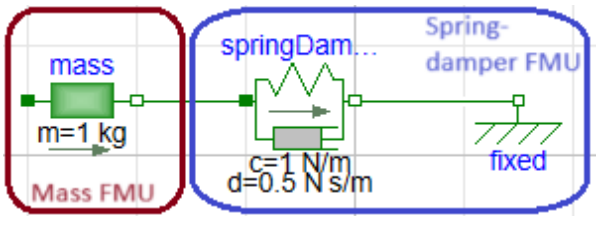

Figure 3. Simple mass-spring-damper model split into two co-simulation FMUs.

Table 1. Physical and numerical parameters of the mass-spring-damper co-simulation problem. Only non-default values are listed.

Parameter	Value
Mass start position	1 m
Mass (m)	1 kg
Spring constant (c)	1 N/m
Damping constant (d)	0.5 Ns/m
Simulation stop time	100 s
Communication interval	0.2 s

The co-simulation is performed using a classical non-iterative master. For the mass FMU we include either CVODE, IDA, or DASSL as solver inside the FMU to integrate its two states: position s and velocity v. For this FMU we either use classical input handling or input smoothing with or without predictor compensation. The spring-damper FMU has no states so it uses no integrator and only uses classical input handling.

3.1.2 Result and analysis

Select simulation statistics for all combinations of solvers and input options are listed in Table 2. In each case, the number of f-evaluations is significantly reduced by input smoothing. The additional evaluations required by the predictor compensation is more than repaid by the further reductions in f-evaluations made by the solvers. Especially for CVODE and IDA. On top of that, the more expensive Jacobian-evaluations are significantly fewer.

When using classical input handling, CVODE constructs one Jacobian at the start of each communication interval (due to the integrator restart) and then keeps it throughout the interval. Enabling input smoothing reveals that these Jacobians actually can be kept for a longer time, resulting in only 40 Jacobian evaluations throughout the co-simulation. Also enabling the predictor compensation helps CVODE's nonlinear solver and reduces the number of required Jacobians to only 14.

For IDA, the reduction in Jacobian-evaluations given by the predictor compensation is not significant. However, when only using input smoothing, then IDAs nonlinear solver fails to converge seven times during the co-simulation. With the help of predictor compensation to improve the start guesses, all of these failures are eliminated revealing the numerical soundness of this technique.

The position of the mass (s) is plotted over time in Figure 4 for the co-simulations using IDA in the mass FMU. A direct simulation of the full system is provided as reference. As expected the classical non-iterative co-simulation attenuates the oscillations. This is made worse by the extra delays introduced by the input smoothing. As previously discussed, the predictor compensation has no direct effects on the simulation results, but rather only helps the solver. Since the result of the input smoother with predictor compensation coincides with that without compensation, we do not include it in the figure.

DOI
10.3384/ecp218653

Table 2. Simulation statistics for the mass FMU from the mass-spring-damper co-simulation problem. Three different solvers are tested with classical input handling or input smoothing (IS) with or without predictor compensation (PC). The statistics are the number of times the model is evaluated by the solver (f-evals), the number of times the solver Jacobian is approximated (Jac-evals), and the number of times the predictor compensation evaluates the model (PC f-evals).

	CVODE			IDA			DASSL		
Option	f-evals	Jac-evals	PC f-evals	f-evals	Jac-evals	PC f-evals	f-evals	Jac-evals	PC f-evals
Classical	5471	499		5958	4960		5042	4543	
IS	2708	40		4715	1030		2653	534	
IS & PC	853	14	996	2692	916	996	1481	259	996

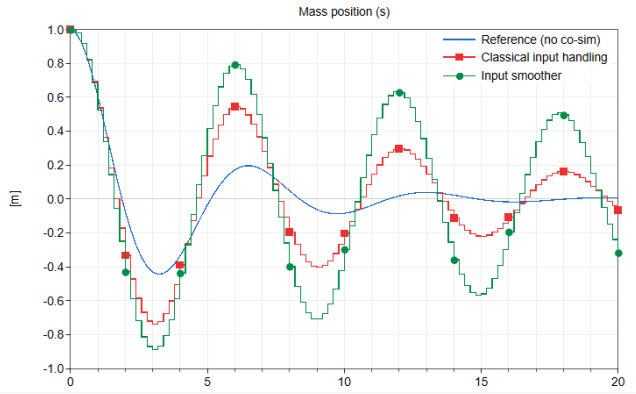

Figure 4. The mass position (s) over the first 20 s of simulation of the mass-spring-damper co-simulation problem with communication interval $H = 0.2$ s, with or without input smoothing. The reference solution is given by simulating the full model directly in Dymola.

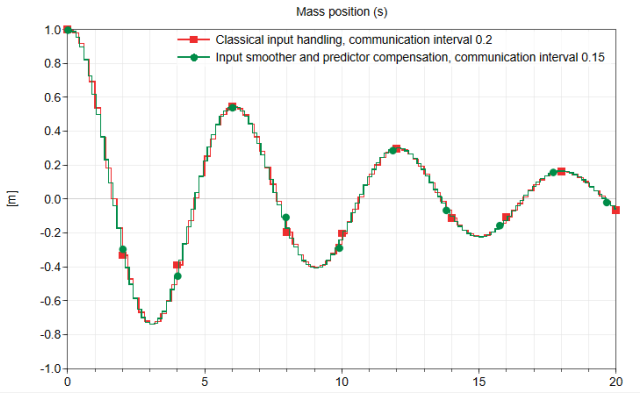

Figure 5. The mass position (s) over the first 20 s of simulation of the mass-spring-damper co-simulation problem. The simulation with input smoothing and the predictor compensation has a reduced communication interval to compensate for the additional co-simulation delay. Under these settings the results are similar.

Table 3. Simulation statistics for the mass FMU from the mass-spring-damper co-simulation problem. The co-simulations are run using IDA. The simulation statistics for communication interval $H = 0.2$ s is as in the previous experiments, cf. Figure 4 and Table 2. To compensate for the additional delay, the interval has been reduced to $H = 0.15$ s for the simulations with input smoothing, cf. Figure 5.

		IDA		
Option	H [s]	f-evals	Jac-evals	PC f-evals
Classical	0.2	5958	4960	
IS	0.2	4715	1030	
IS & PC	0.2	2692	916	996
IS	0.15	2380	471	
IS & PC	0.15	1506	357	1330

As a final experiment we reduce the communication interval to $H = 0.15$ s for the simulations with input smoothing to compensate for the additional delay. With this choice of H, the result is similar to the result when using classical input handling with $H = 0.2$ s, cf. Figure 5. From Table 3 we can see that even with this smaller communication interval the simulations with input smoothing are cheaper in terms of f-evaluations and Jacobian-evaluations inside the mass FMU. Indeed, the statistics are even better compared to Table 2.

A disadvantage is that the master has to handle inputs and outputs more often. Thus, in a larger system, reducing the communication interval could increase the number of function evaluations in other part of the coupled system. Further, in practice, it is not always possible to arbitrarily change the communication interval, e.g. due to real-time requirements.

3.2 HIL simulation of a heat pump

In this section the model of a heat pump as used for Hardware-in-the-Loop (HiL) is examined. On the HiL environment the model is simulated with a (usually) fixed synchronization rate. The input signals of the model are updated in each step. The input signal often has a small noise. In this section we only replicate the boundary conditions by applying a constant input signal with a small noise. The example uses a single FMU, the surrounding environment is represented by the input signals. This example is focussed on the overall performance gain, it does not examine the effect on the results due the modified input trajectory, or the introduced delay.

3.2.1 Setup

The domestic heat pump used in this example is based on the HiL use case of (Schulze, Gräber, and Huhn 2011) and uses the TIL-Suite (TLK-Thermo GmbH 2024). It is a simple vapor compression cycle with efficiency-

based compressor, Bernoulli expansion valve, two 1D-discretized TubeAndTube HX (heat exchangers) and an ideal separator model. The refrigerant is R-407C.

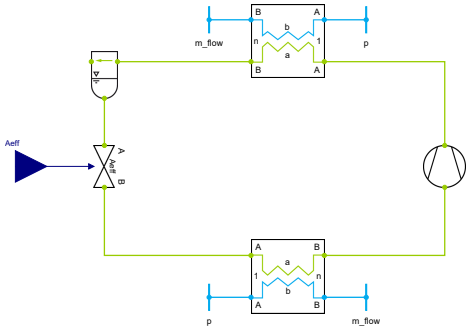

Figure 6. The valve opening area is input of this domestic heat pump model.

A large simulation study with almost 17000 single simulations was conducted to analyze the CPU time performance. The cumulative CPU time exceeded 5 days and 16 hours. The model is exported as FMU 2.0. The solver tolerance is set to 10^{-5}. Five independent "dimensions" have been varied: The solver (CVODE, DASSL and IDA), the smoothing method (Classical, IS, and IS & PC), model variant, the boundary condition (constant, small noise, large perturbation), the `doStep` interval length.

The model variants are:

- Baseline (1 s): Low number of HX volumes with num. ODE-Jacobian and 1 s stop time, 83 continuous-time states, Helmholtz fundamental equation of state.
- Baseline (100 s): Low number of HX volumes with num. ODE-Jacobian and 100 s stop time, 83 continuous-time states, Helmholtz fundamental equation of state. Used to display what happens if the time constants of the model are relatively smaller compared to the `doStep` interval lengths.
- High-Resolution (1 s): High number of HX volumes with num. ODE-Jacobian and 1 s stop time, 403 continuous-time states, Helmholtz fundamental equation of state. Used to display the influence of the model complexity.
- Analytical Jacobian (1 s): Low number of HX volumes with analytical ODE-Jacobian and 1 s stop time, 83 continuous-time states, spline-based fluid properties. Used to display the influence of the computation of the ODE-Jacobian.

There are three types of boundary conditions for the effective valve opening area:

- Constant input: 1mm^2
- Small noise: 1mm^2 plus a small random number varying at each time step with an amplitude of 0.1%, simulating signal noise.
- Large stochastic perturbation: 1mm^2 plus a large random number varying at each time step with an amplitude of 50%, replicating abrupt controller output changes.

The time constants of the Baseline model with 83 continuous-time states range from 0.1 s to 65 s with a mean of 2.1 s and a median of 0.58 s. The time constants of the High-Resolution model with 403 continuous-time states range from 0.023 s to 52 s with a mean of 0.6 s and a median of 0.1 s.

3.2.2 Result and analysis

Figure 7 shows the performance gain of CPU time relative to the reference case plotted against the number of `doStep` calls. The Classical FMU is the fastest scenario if the inputs are constant (best-case scenario). If the input of the Classical FMU is updated, then the solver is reset and the CPU time increases significantly (worst-case scenario). The IS variant has a limited solver step size, and an increased number of Jacobian evaluations compared to the Classical variant with constant input signal, as mentioned in Section 2.2.

The Baseline (100 s) demonstrates smaller performance gains if the input is subject to large perturbations. The model responds comparatively quickly because the `doStep` interval length is larger compared to the model time constants. The model has to be brought closer to the steady state, and hence the computational effort is higher. The effect of input smoothing diminishes. Using constant input values (dotted line, best-case scenario) is significantly faster, but also the results are significantly different.

The High-Resolution model demonstrates that as the number of continuous-time states of the model increases, the computational cost of the f- and Jacobian-evaluation also rises. Consequently, the speed-up factor of the IS and IS & PC method becomes more significant. The model variant with analytic Jacobian is generally faster than the Baseline with num. Jacobian. In all cases, the speed-up factor with analytic Jacobian is at a lower level, but the overall picture is unchanged.

The minimum and maximum speed-up summarized in Table 4 of the variants shown in Figure 7 illustrate that input smoothing is beneficial, if inputs have to be updated and the synchronization step size is low.

Augmenting the input smoother with predictor compensation is occasionally very beneficial. In Figure 7 we see the benefits when the input is altered by large perturbation and IDA or DASSL is used. When only the input smoother is used, these represent the cases where the solver runs into severe problems due to the large changes in the derivative of the smoothed input signal. Investing a few extra model evaluations at communication points to help the solver predictor is repaid many times over. In the best cases, we see about 10 times faster simulation compared to only using input smoothing (about 100 times faster compared to the reference). The most striking is that the number of Jacobian evaluations is reduced from about 1000 (IS) to

DOI
10.3384/ecp218653

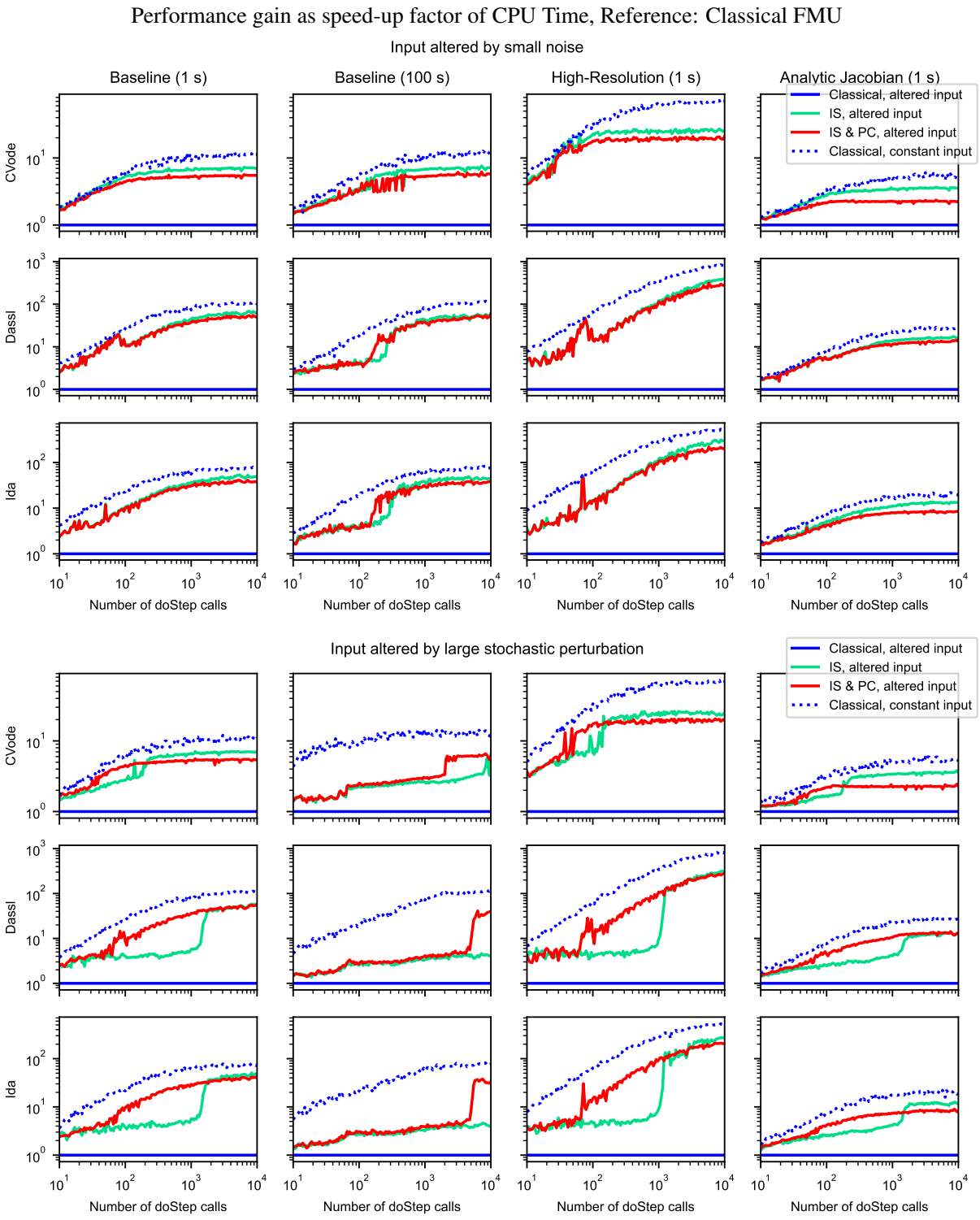

Figure 7. The figure illustrates the CPU time performance gain across various simulations, represented as a speed-up factor relative to the solid blue reference scenario. The x-axis indicates the number of `doStep` calls required to reach the fixed stop time. The upper half of the figure represents simulations with small noise in the boundary condition, while the lower half shows results under large stochastic perturbations. The reference scenario, defined as the Classical FMU with altered input (regardless of perturbation amplitude), represents the worst-case in terms of performance. Conversely, the Classical FMU with constant input exhibits the fastest execution and serves as the best-case scenario. As the number of `doStep` calls with the same stop time increases, the performance gain improves in all plots. The Baseline (100 s) shows smaller performance gains under large perturbation, as the model responds comparatively quickly. With increasing model complexity (High-Resolution), the performance gain becomes more pronounced. Utilizing an analytic Jacobian reduces the impact of input smoothing while preserving the overall trend. Adding predictor compensation is sometimes very beneficial with large perturbation boundary conditions using DASSL or IDA, reaching about a factor of 10 times faster simulations compared to when only using input smoothing.

Table 4. The table provides a summary of Figure 7, presenting the minimum and maximum speed-up factors across selected number of `doStep` calls. The range reflects the differences among model variants and boundary conditions. The rows compare the IS and IS & PC performance with updated inputs to the best case scenario. Across all scenarios, IS consistently improved performance. However, the speed-up factor is highly influenced by time constants, model complexity and the input trajectory.

		Speed-up factor			
Solver	**Variant**	Number of doStep calls			
		10	100	1000	10000
CVODE	Classical with constant input	1.3 - 5.5	3.1 - 34.0	4.8 - 61.4	5.2 - 70.1
	IS, altered input	1.2 - 4.5	1.7 - 22.8	2.6 - 25.7	3.6 - 26.2
	IS & PC, altered input	1.2 - 4.1	2.1 - 18.0	2.2 - 20.3	2.3 - 21.0
DASSL	Classical with constant input	1.8 - 7.8	7.6 - 66.5	21.3 - 338.9	25.8 - 847.2
	IS, altered input	1.4 - 4.3	2.4 - 14.6	3.4 - 122.0	4.1 - 403.0
	IS & PC, altered input	1.5 - 4.9	2.7 - 14.6	4.1 - 113.2	13.1 - 282.1
IDA	Classical with constant input	1.8 - 8.8	7.1 - 67.8	17.2 - 272.5	19.0 - 532.2
	IS, altered input	1.3 - 4.1	2.5 - 14.9	3.2 - 119.6	4.1 - 299.2
	IS & PC, altered input	1.4 - 3.8	3.1 - 14.9	3.8 - 105.9	8.3 - 206.2

about 50 (IS & PC), showcasing the numerical soundness of the predictor compensation.

With that said, we can also see that predictor compensation is not always needed. When the input is altered by small noise, then the changes in input are small enough for the integrator to function well, even without the help of predictor compensation. This is evident by looking at the internal step-size sequence of the integrator inside the FMU. Very often the integrator takes the largest possible step, i.e. the length of the communication interval. In these cases, the predictor compensation only contributes its additional f-evaluations and therefore worsens performance. This is also the case in the experiments with CVODE when the input is altered by large perturbation and the number of `doStep` calls is large.

It is important to note that the results of the simulations performed are not identical. The user must evaluate whether the results are still within tolerance, even with the altered input trajectory effectively causing additional time delay.

3.3 Heat pump simulation data for training surrogate models

To show the possible improvements for industry-relevant applications, we investigate the application for a detailed model of an air-source heat pump. The used model is a demonstrator of the *OpenSCALING* research project, delivered by the industry partner *Bosch*. It consists of a refrigeration cycle with controller and is modeled using the TIL-library of *TLK-Thermo*. The outdoor heat exchanger is especially detailed; apart from the liquid-vapor behavior of the R-290 refrigerant, it models the moist air flow including condensation and frost formation on the heat exchanger surface. The diagram view of the R-290 refrigeration cycle (or just model) can be seen in Figure 8.

In practice, system models including refrigerant cycles can be computationally expensive, due to the detailed representation of the physical phenomena (evaporation, condensation, ice formation, ...). Required simulation times

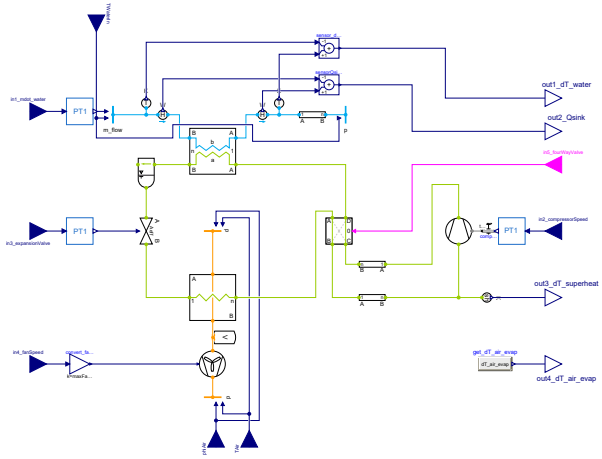

Figure 8. Diagram of the R-290 refrigerant with 7 inputs.

and computational resources set limits to repeated evaluation of the models. This restricts how the model can be used, for example in virtual testing, controller design or setpoint optimization. Here, data-driven surrogate modeling comes into play: By sampling input signals to the physical simulation model, a dataset is created, with which surrogate models are trained. This means that one has to accept a high computational upfront cost for generating the surrogate model; but once it is trained, its evaluation becomes really cheap and can be easily used for millions of evaluations. For example, *University of Augsburg* examines in the research project *OpenSCALING* Balanced Neural ODEs for generating surrogates. This method has been demonstrated to decrease computation times by a factor up to 100 (Aka et al. 2025).

3.3.1 Operating points

Evaluation times of the heat pump model can considerably differ depending on the operation point; it is especially high for reverse-cycle operation to defrost the outdoor heat exchanger, or for oscillating controller behavior, that frequently arises during virtual controller devel-

DOI
10.3384/ecp218653

opment. On the other hand, if steady state operating conditions are reached, adaptive-step size solvers quickly solve the model. To evaluate the impact of the input smoother for these different cases, we include it with (1) tuned controllers that avoid oscillating behavior and (2) controller with high gains to enforce oscillations. Additionally, the model is operated at air-conditions that avoid frost formation.

3.3.2 Setup

The heat pump system consisting of the refrigeration cycle and controllers is exported as a co-simulation FMU 2.0 with DASSL as solver. The inputs are defined in a co-simulation master. Moreover, the heat-pump model with setup for controllers is also implemented in pure Modelica and simulated for comparison.

3.3.3 Result and analysis

The results for the simulation statistics related to the solver are found in Table 5, while computational time measured is presented in Table 6.

Table 5. Simulation statistics of the solver for the heat pump co-simulation FMU exported with DASSL, from the controller model for the two different operation points. Simulated to $T = 500$ s with communication interval $H = 1$. In addition the model is simulated using pure Modelica, with applicable statistics for f-evals and Jac-evals. The model with tuned controllers that avoid oscillating behavior in the output is labeled *Low Oscillations* while the model with controllers with high gains to enforce oscillations in the output is labeled *High Oscillations*.

Low Oscillations				
Option	f-evals	Jac-evals	Events	PC f-evals
Classical	8251	4546	506	
IS	1950	209	6	
IS & PC	1910	222	6	998
Modelica	1038	140		

High Oscillations				
Option	f-evals	Jac-evals	Events	PC f-evals
Classical	71939	9073	514	
IS	70734	7011	16	
IS & PC	70442	6936	16	926
Modelica	35343	6113		

When the controller is tuned to avoid oscillating behavior (Low Oscillations) the input smoother performs well, reaching over 40 times faster co-simulation. This is clear from looking at the reduced number of events in Table 5. The Classical FMUs generate events every time the integrator re-initializes. Thanks to the continuity of the input these re-initializations are not needed, drastically speeding up the simulation. On the other hand, adding predictor compensation slightly worsens performance for the same reason as described in Section 3.2.2 when the input is altered by small noise, c.f. Figure 7.

Although still having a positive impact, input smoothing is less effective for the model containing high oscillations. The reason is that the simulation inside the heat pump FMU is challenging for the integrator given the used communication interval. The work is dominated by a lot of internal steps in each interval, overshadowing the performance gained by avoiding restarts at communication points. Adding predictor compensation improves performance slightly.

Table 6. CPU-time statistics of the solver for the heat pump co-simulation FMU exported with DASSL, from the controller model for the two different operation points. CPU-time gain is calculated with classical FMU as baseline. Simulated to $T = 500$ s with communication interval $H = 1$. In addition the model is simulated using pure Modelica. The model with tuned controllers that avoid oscillating behavior in the output is labeled *Low Oscillations* while the model with controllers with high gains to enforce oscillations in the output is labeled *High Oscillations*.

Low Oscillations		
Option	CPU-time [s]	CPU-time gain
Classical	129	1
IS	3.11	41.5
IS & PC	3.27	39.5
Modelica	2.35	54.9

High Oscillations		
Option	CPU-time [s]	CPU-time gain
Classical	321	1
IS	273	1.18
IS & PC	255	1.26
Modelica	183	1.75

In all cases it is most efficient to simulate the model using pure Modelica, but employing input smoothing techniques decreases computational effort to be more in level with the Modelica implementation.

4 Conclusion

In this article we have introduced techniques that may speed up co-simulation by orders of magnitude, namely input smoothing and predictor compensation. The main benefit is their ease-of-use. No special co-simulation master is required, nor any manipulations of the models that are being exported to FMUs. Rather, the options are easily enabled in Dymola 2025x and the logic is handled entirely by the generated code.

By smoothing the input to the FMU, integrator re-initializations at communication points can be avoided. For multistep solvers like CVODE, IDA, and DASSL, this may heavily reduce the number of model and Jacobian evaluations. The benefits of smoothing the input were observed for the examples studied in this article, with some configurations seeing over 400 times faster co-simulation.

However, the smoothed input may still have discontinuous derivatives. To handle this problem we proposed the use of predictor compensation to help the integrator inside

the FMU. When enabled, additional model evaluations are performed to improve the guess for the next internal state. In the best cases we observed performance gains of a factor 10 compared to only using input smoothing.

Based on the experiments with the thermo-fluid applications we conclude that input smoothing gives the greatest benefits when the co-simulation FMU is computationally heavy but the communication interval is small compared to the dynamics of the model inside the FMU. This is a commonn case in applications as the communication interval often is set by external factors, such as stability and accuracy of the overall co-simulation problem or real-time requirements, c.f. Sections 3.2.

Predictor compensation is most beneficial when, additionally, the integration with input smoothing is not too trivial. Thereby, predictor compensation performs best within a sweet spot where the integration inside the FMU is not too simple, nor too difficult. In our experiments IDA and DASSL are more often helped by predictor compensation.

Although, the results presented in this article are very promising, further work is needed to study the effect of input smoothing and predictor compensation for co-simulation setups containing several coupled FMUs.

5 Acknowledgments

The authors would like to thank Vaillant and Airbus for their invaluable contributions in developing and testing the input smoothing technology.

Research for the input smoother was funded by the German Federal Ministry for Education and Research (BMBF) under the research project “MODAK” (Grant 01IS17006C).

The inclusion of the air-source heat pump example for training surrogate models has been supported by VINNOVA (grant number 2023-00969) and by the ITEA 4 project OpenSCALING (Open standards for SCALable virtual engineerING and operation) N°22013 under grant number 01IS23062H.

SPONSORED BY THE

References

Aka, Julius et al. (2025). “Balanced Neural ODEs: nonlinear model order reduction and Koopman operator approximations”. In: *The Thirteenth International Conference on Learning Representations*.

Andersson, Christian (2016). “Methods and Tools for Co-Simulation of Dynamic Systems with the Functional Mock-up Interface”. Doctoral Thesis (monograph). Mathematics (Faculty of Engineering, Lund University). ISBN: 978-91-7623-698-7.

Arnold, Martin (2010-05). “Stability of Sequential Modular Time Integration Methods for Coupled Multibody System Models”. In: *Journal of Computational and Nonlinear Dynamics* 5.3, p. 031003. ISSN: 1555-1415. DOI: 10.1115/1.4001389.

Benedikt, Martin and Edo Drenth (2019). “Relaxing Stiff System Integration by Smoothing Techniques for Non-iterative Co-simulation”. In: *IUTAM Symposium on Solver-Coupling and Co-Simulation*. Ed. by Bernhard Schweizer. Cham: Springer International Publishing, pp. 1–25. ISBN: 978-3-030-14883-6.

Braun, R. and P. Krus (2013). “Tool-Independent Distributed Simulations Using Transmission Line Elements And The Functional Mock-up Interface”. In: *53rd SIMS conference on Simulation and Modelling*. Linköping University Electronic Press.

Braun, Robert and Dag Fritzson (2022). “Numerically robust co-simulation using transmission line modeling and the Functional Mock-up Interface”. In: *SIMULATION* 98.11, pp. 1057–1070. DOI: 10.1177/00375497221097128.

Brenan, Kathryn E., Stephen L. Campbell, and Linda R. Petzold (1996). *Numerical Solution of Initial-Value Problems in Differential–Algebraic Equations*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics.

Busch, Martin (2016). “Continuous approximation techniques for co-simulation methods: Analysis of numerical stability and local error”. In: *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik* 96.9, pp. 1061–1081.

Gardner, David J et al. (2022). “Enabling new flexibility in the SUNDIALS suite of nonlinear and differential/algebraic equation solvers”. In: *ACM Transactions on Mathematical Software (TOMS)*. DOI: 10.1145/3539801.

Modelica Association (2022-11). *Functional Mock-up Interface for Model Exchange and Co-Simulation Version 2.0.4*. Tech. rep. Modelica Association. URL: https://fmi-standard.org.

Modelica Association (2023-07). *Functional Mock-up Interface for Model Exchange and Co-Simulation Version 3.0.1*. Tech. rep. Modelica Association. URL: https://fmi-standard.org.

Radhakrishnan, K and A C Hindmarsh (1993-12). “Description and use of LSODE, the Livemore Solver for Ordinary Differential Equations”. In: DOI: 10.2172/15013302.

Schierz, Tom, Martin Arnold, and Christoph Clauß (2012). “Co-simulation with communication step size control in an FMI compatible master algorithm”. In: *Proceedings of the 9th International Modelica Conference*. Linköping University Electronic Press, pp. 205–214. DOI: 10.3384/ecp12076205.

Schulze, Christian, Manuel Gräber, and Michaela Huhn (2011-06). “Real-Time Simulation of Vapour Compression Cycles”. In: DOI: 10.3384/ecp1106348.

TLK-Thermo GmbH (2024). *TIL Suite*. Version 2024.1. URL: https://www.tlk-thermo.com/software/til-suite.