TLM-based Asynchronous Co-simulation with the Functional Mockup Interface

Robert Braun[1], Robert Hällqvist[2], Dag Fritzson[3]

[1] Div. of Fluid and Mechatronic Systems, Linköping University, Sweden,
robert.braun@liu.se,
[2] Dept. of Systems Simulation and Concept Analysis, Saab Aeronautics, Sweden,
robert.hallqvist@saabgroup.com,
[3] SKF Group Technology, AB SKF, Sweden,
dag.fritzson@skf.com

Abstract. Numerical stability is a key aspect in co-simulation of physical systems. Decoupling a system into independent sub-models will introduce time delays on interface variables. By utilizing physical time delays for decoupling, affecting the numerical stability can be avoided. This requires interpolation, to allow solvers to request input variables for the time slot where they are needed. The FMI for co-simulation standard does not support fine-grained interpolation using interpolation tables. Here, various modifications to the FMI standard are suggested for improved handling of interpolation. Mechanical and thermodynamic models are used to demonstrate the need for interpolation, as well as to provide an industrial context. It is shown that the suggested improvements are able to stabilize the otherwise unstable connections.

Keywords: co-simulation, FMI, TLM, numerical stability

1 Introduction

Numerical robustness is a key factor in simulation solver coupling. Using different solvers for different parts of a simulation model can be of great benefit in terms of increasing simulation speed and robustness. However, all variables shared by more than one solver will need to be delayed in time. This may result in numerical errors and instability. One solution is to decouple the model where significant physically motivated time delays are present. In this way, all time delays are a natural part of the model and no non-physical time delays will thus need to be inserted.

The Functional Mock-up Interface (FMI) is a tool-independent standardized interface for connecting simulation models from different modelling and simulation tools [3]. A model is exported as a Functional Mock-up Unit (FMU), containing executable C code and an XML description file. Co-simulation is conducted by importing multiple FMUs to a master simulation tool (MST) and connecting them in a *composite model*. FMUs can be either for co-simulation

(FMI CS) or for model exchange (FMI ME). With FMI CS, each FMU has its own internal solver and exchanges data only at pre-defined communication points. This enables discrete-time co-simulation, where each FMU internally may use continuous-time simulation. With FMI ME, the MST must provide a solver. Derivatives of state variables can then be updated at any time, which enables continuous-time co-simulation. Experiments have been conducted to investigate how the FMI standard can be combined with physically motivated model decoupling. Four different methods for extrapolation and interpolation of interface variables have been implemented and compared. Based on the results, some improvements to the FMI standard are suggested.

This paper uses a domain-independent notation of the exchanged variables. All physical connections use *intensity* and *flow* variables. For the mechanical and fluid domains, intensity would equal force and pressure while flow would equal velocity and volume flow.

1.1 Problem Description

Transmission Line Modelling (TLM) is a well-known technique for decoupling of simulation models [11]. The basic idea is that in reality information propagation speed is always limited. This includes, for example, stress waves in materials or pressure waves in fluids. Hence, every physical element has a natural time delay. By including physical time delays in model variables, equations can be separated without affecting numerical stability. A TLM element and its equations are shown in fig. 1. F is the force, v velocity, Δt the time delay and Z_c characteristic impedance. The delayed information traversing the element is denoted the *wave variable*. A co-simulation MST using TLM has been developed by SKF [14]. The implementation is based on asynchronous socket communication. Each slave tool has fully independent time variables and step sizes. Due to the physical time delays, input data is available not only at the beginning of the step but also during the step via interpolation. Slave tools use callback functions for receiving inputs and sending outputs for specific time instances. Input variables are thus kept up-to-date even during internal iterations performed by the slave.

The FMI standard aims to reduce overhead costs associated with simulation tool coupling by means of a standardized interface. FMI is stipulated to enhance and simplify model export and integration in both industry and academia. It is therefore desirable to make the MST compatible to the FMI standard. This would enable simulation of multiple connected Functional Mock-up Units (FMUs) exported from any simulation tool with FMI support. While FMI for model exchange can easily be adapted for the TLM-based co-simulation MST, FMI for co-simulation induces several challenges concerning numerical stability. Since input variables can only be updated at the beginning of each communication step, the stability benefits of TLM are lost.

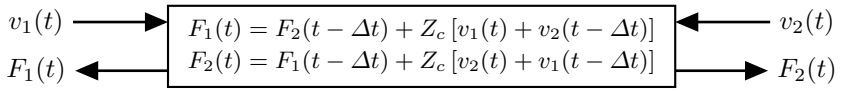


Fig. 1: A TLM element with its equations.

1.2 Related Work

A prototype of a master simulation tool for FMI-based co-simulation was presented in [2]. Master algorithms combining FMI-based co-simulation with the High-Level Architecture (HLA) standard have also been developed [8][1][12]. A TLM-based co-simulation MST with FMI support was implemented in the Hopsan simulation tool [5]. In [4] it was shown that it is possible to connect two simulation tools with a TLM element, using FMI, without a master simulation tool orchestrating the simulation. The MST used in this paper differs from the previous experiments in that it uses asynchronous data communication. Consequently, each TLM connection can have its own independent time delay, and each FMU can use its own independent simulation step size. With synchronous communication, all connections must have the same time delay and each sub-model must be able to provide output variables at this interval.

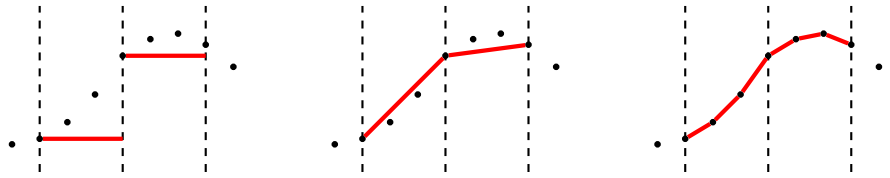
Solving two parts of a simulation model simultaneously on different solvers without using physical time delays requires incorporating numerical time delays, which may affect accuracy and numerical stability. Errors can usually be reduced by reducing the size of the delays, at the cost of a longer simulation time. One solution to reduce the resulting negative impact on simulation time is to use adaptive communication step-size [13]. In contrast to physically motivated decoupling, this method requires the FMU to support setting and getting FMU states.

2 Numerical Solutions

Several solutions, both with and without modifications to the FMI standard, are implemented and analysed. Three different variable estimation methods are used, see fig. 2. With constant extrapolation, variables are kept constant during each communication step. Coarse-grained interpolation means that the value at the beginning and at the end of the step, i.e. at the communication points, are used for interpolation. Fine-grained interpolation includes the values between communication points as well. Experiments are limited to linear interpolation. Higher order interpolation methods may be used to improve accuracy further, but has not been considered further in this paper.

2.1 Constant Extrapolation

The simplest method for exchanging variables is to use constant extrapolation, see fig. 2a. Input variables are updated at the beginning of each step and remain



(a) Constant extrapolation (b) Coarse-grained interpolation (c) Fine-grained interpolation

Fig. 2: Three variable estimation methods are used to stabilize the connecitons.

constant during the step. This solution is fully supported by the current FMI for co-simulation standard. However, numerical stability cannot be guaranteed. If the FMU supports saving and loading FMU states, it is possible to improve stability by using adaptive communication step size [13]. Unfortunately, this feature is optional in the standard and many exporting tools do not support it. Stability can then only be achieved by using a sufficiently small constant communication step size during the entire simulation. This induces a severe performane penalty. Furthermore, it is not possible to tell in advance whether or not the step size is small enough.

2.2 Coarse-grained Interpolation

It is possible to provide an FMU with the approximated values of input variable time derivatives. These can be used by the FMU for interpolation or extrapolation, see fig. 2b. For a delayed variable the value is known both at the beginning and at the end of the step. Hence, the first-order derivative can easily be approximated. The resolution of the interpolation, however, will be limited since all data in the interpolation table will not be used. Furthermore, the delayed information will consist of the wave variable and not the intensity variable. The FMU must therefore include the TLM boundary equations, and compute the intensity internally. While technically possible, it makes it harder to generalize the method for FMUs from arbitrary generation tools.

2.3 Fine-grained Interpolation Inside the FMU

Here, fine-grained interpolation means that all points in the interpolation table are used, as shown in fig. 2c. The actual interpolation can either be performed inside the FMU or provided by the master simulation tool from callback functions.

Interpolation inside the FMU with high resolution is possible only if the FMU is provided with the interpolation table. Interpolation data can be sent as normal variables, using the `fmi2SetReal()` function. However, this requires customized FMUs and manual adjustments will therefore be required, and it will not work for all simulation tools.

Furthermore, it requires a large amount of data exchange, which may affect simulation performance. If support for populating interpolation tables in FMUs could be incorporated into the standard, this could become a more general and computationally efficient solution. A proposed function for setting time-stamped variables is shown in listing 1.1. Like the coarse-grained interpolation, this approach also requires the FMU to include the TLM equations.

Listing 1.1: A proposal for extending the FMI API to support time-stamped variables

```
fmi2Status fmi2SetReal (fmi2Component c,
                        const fmi2ValueReference vr[],
                        size_t nvr,
                        const fmi2Real value[],
                        const fmi2Real time[]);
```

Callback functions for reading and writing input variables at specified times could be provided to the FMUs. Interpolation can then be handled by the MST. This would preserve the guaranteed stability provided by TLM, while exposing only a minimal interface consisting of intensity and flow variables. The exported models would not need any adaption for TLM. However, this method requires an extension to the current FMI standard. A proposal for such an extension is shown in listing 1.2.

Listing 1.2: A proposal for extending the FMI API with callback functions for requesting interpolated inputs

```
typedef fmi2Real (*fmi2GetRealCb) (fmi2ValueReference,
                                                fmi2Real,
                                                fmi2Real);

fmi2Status fmi2SetGetRealCb(fmi2Component c,
                            const fmi2GetRealCb cb);
```

An alternative solution to callback functions is to use pure discrete-event simulation using FMI for model exchange. The slave can have its own custom solver internally, without exposing any continuous state variables to the MST. In this way, FMI ME can be used for co-simulation with distributed solvers, in contrast to FMI ME for continous-time simulation, where the MST must provide the solver. Whenever an interpolated input variable is required or an output variable is available, the slave informs the MST by using time events. With the current API, however, it is not possible to distinguish input data events from output events. The API would thus have to be extended with an additional flag in the `fmi2EventInfo` structure, as shown in listing 1.3.

Listing 1.3: A proposal for extending the FMI event info class for events with only inputs

```
typedef struct{
  fmi2Boolean newDiscreteStatesNeeded;
  fmi2Boolean terminateSimulation;
  fmi2Boolean nominalsOfContinuousStatesChanged;
  fmi2Boolean valuesOfContinuousStatesChanged;
  fmi2Boolean outputsNotAvailable;
  fmi2Boolean nextEventTimeDefined;
  fmi2Real nextEventTime;
} fmi2EventInfo;
```

2.4 One-step Functions

With FMI for co-simulation, each FMU writes output variables only after every completed communication step. Interpolation accuracy can be improved by letting each FMU provide output variables as often as possible. The FMI standard provides a function called `fmi2doStep()`, which tells the slave to simulate to a specified stop time. If the MST could tell the FMU to take a step without specifying a stop time, the FMU could simulate until the next time it is able to provide output data and then return the actual stop time to the MST. This would make it possible for the MST to obtain output data as often as possible, which would enable more densely populated interpolation tables. Many numerical solvers, such as CVODE and IDA, already provide one-step execution [10]. It is, however, unclear how this function would work for FMUs not based on differential equations.

3 Numerical Experiments

Four example composite models are used to illustrate the need for and to verify the feasibility of the proposed solutions. The first model is a one-dimensional spring-mass system. The second two models are two-dimensional pendulums. 2D mechanics models are usually more stiff and put stricter requirements on numerical stability. Finally, a thermodynamic connection extracted from an industrial use case is investigated.

The first three examples are implemented using custom FMUs, written in plain C++. The FMUs support first order input derivatives for coarse-grained interpolation. Interpolation tables are provided to FMUs by repeatedly calling the standard `fmi2setReal()` function. The FMI API is extended to support a prototype of callback functions. Discrete event simulation is tested by (incorrectly) using the valuesOfContinuousStatesChanged flag to indicate whether output variables are available or not. Finally, the thermodynamic model is used

to compare simulations implementing constant extrapolation, coarse-grained and fine-grained interpolation using FMUs exported from the commercial Modelica tool Dymola. Interpolation is implemented as Modelica code. A function for coarse-grained interpolation is shown in listing 1.4. This makes it possible to use input derivatives even if the FMU does not support the native FMI function for this. Fine-grained interpolation is performed by an interpolation model and two functions, see listings 1.5 to 1.7. The FMU is provided with wave variables and corresponding time variables. Then, the actual value is obtained by using the built-in Modelica function `interpolate(t,c,tc)`.

Listing 1.4: A Modelica function for interpolating input variables in Dymola using first order time derivatives

```
function LinearInterpolation
  input Real t "Time variable";
  input Real c "Wave variable";
  input Real dCdt "Time derivative of wave variable";
  input Real tc "Current time";
  output Real ci "Interpolated wave variable";

algorithm
   ci:=c+dCdt*(tc-t);

end LinearInterpolation;
```

Listing 1.5: A Modelica model for interpolating input variables in Dymola

```
model Interpolate
  Modelica.Blocks.Interfaces.RealVectorInput c1[n];
  Modelica.Blocks.Interfaces.RealVectorInput t1[n];
  Modelica.Blocks.Interfaces.RealOutput pi;
  Modelica.Blocks.Interfaces.RealInput q1;
  Modelica.Blocks.Interfaces.RealInput Zc;
  parameter Integer n;

protected
  Real ci;

equation
  ci = PopulateInterpolate(t1,c1,time);
  pi = getPressure(ci,Zc,q1);

end Interpolate;
```

Listing 1.6: A Modelica function for interpolating the wave variable in Dymola

```
function PopulateInterpolate
  input Real t[:] "Time variable vector";
  input Real c[:] "Wave variable vector";
  input Real tc "Current time";
  output Real ci "Interpolated wave variable";

algorithm
  ci:=Modelica.Math.Vectors.interpolate(t,c,tc);

end PopulateInterpolate;
```

Listing 1.7: A Modelica function for requesting pressure variable at a specific time in Dymola

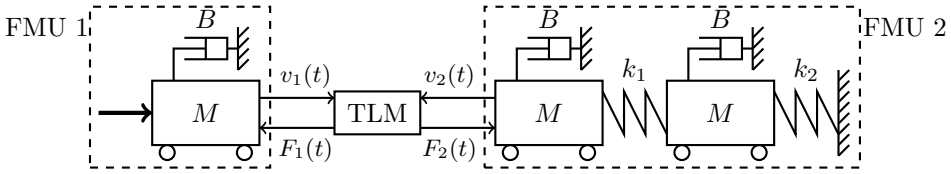
```
function getPressure
  input Real c "Wave variable";
  input Real q "Volume flow";
  input Real Zc "Characteristic impedance";
  output Real p "Pressure";

algorithm
  p:=Zc*q+c;

end getPressure;
```

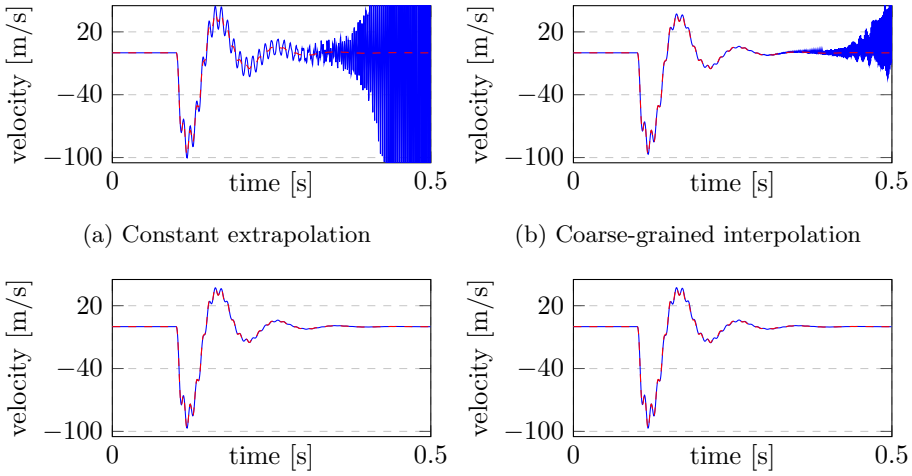
3.1 1D Three-mass System

An example composite model consisting of a three-mass system separated into two FMUs is presented in fig. 3. Simulation results verify the proposed method with second order dynamics in one dimension. One of the FMUs has internal dynamics with two different resonance frequencies. The composite model is simulated with a communication step size (i.e. TLM time delay) of 0.4 ms. After 0.1 s, a step force of 100 N is applied on the first mass. Parameters are intentionally chosen to make the simulation unstable when using constant extrapolation. Simulation results using constant extrapolation, coarse-grained interpolation, interpolation and callback functions are shown in fig. 4. Constant extrapolation and coarse-grained interpolation methods are unstable. Fine-grained interpolation is able to stabilize the coupling both when computed in MST and locally inside the FMU.



$M = 0.005$ kg $k_1 = 90$ N/m
$B = 0.2$ Ns/m $k_2 = 140$ N/m

Fig. 3: A test model consisting of a three-mass system is divided into two FMUs.



(a) Constant extrapolation

(b) Coarse-grained interpolation

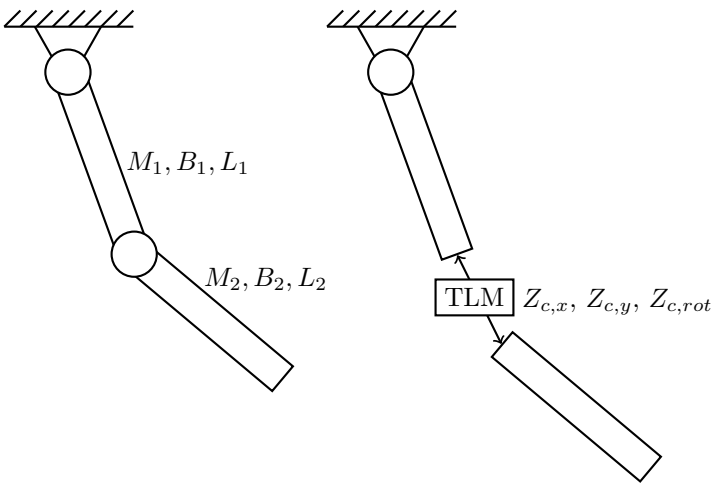
(c) Fine-grained interpolation inside FMU

(d) Fine-grained interpolation in MST

Fig. 4: Velocity against time at the left side of the TLM connection. Red dashed line is the exact reference solution.

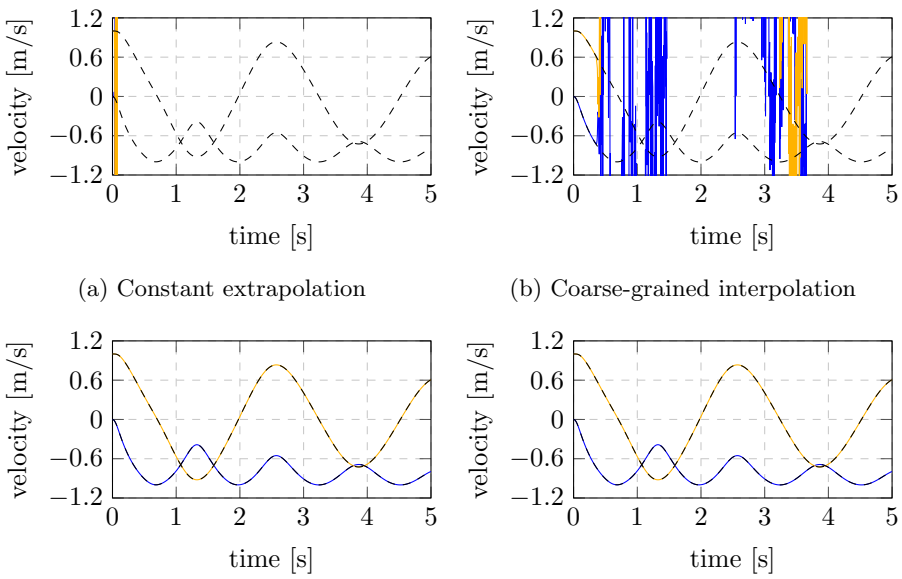
3.2 2D Double Pendulum

The second example composite model consists of a two-dimensional double pendulum, see fig. 5. Double pendulums exhibit chaotic motion and are sensitive to initial conditions. This makes it an interesting example for verifying simulation techniques. The two arms are connected through a TLM element with a time delay of $\Delta t = 1e-3$ s and a characteristic impedance of $Z_c = 1e5$ for X and Y directions. This corresponds to a spring of stiffness $K_s = 1e8$ N/m. Rotational impedance is set to zero. Hence, the TLM element represents a revolute joint with some flexibility. Simulation is initiated with both arms pointing horizontally. Results from simulations implementing the four different methods are shown in fig. 6. Blue and orange curves represent vertical and horizontal position, respectively. Black dashed curves are the exact reference solutions. Constant extrapolation and coarse-grained interpolation are both unstable. Fine-grained interpolation results in stable simulation.



$M_1 = 100$ kg $B_1 = 20$ Ns/m $L_1 = 0.5$ m
$M_2 = 100$ kg $B_2 = 10$ Ns/m $L_2 = 0.5$ m
$Z_{c,x} = 1e5$ Ns/m $Z_{c,y} = 1e5$ Ns/m $Z_{c,rot} = 0$ Nms

Fig. 5: A double pendulum is modelled as two arms connected by a TLM element. Impedance in the rotational dimension is set zero to allow free rotation.



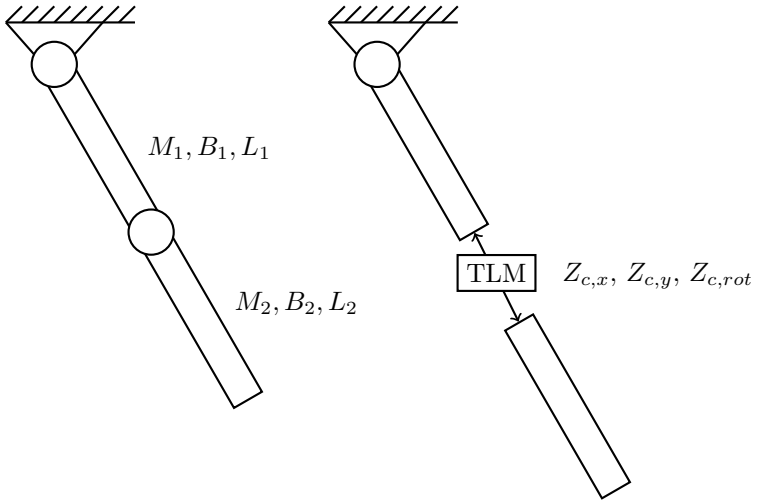
(a) Constant extrapolation (b) Coarse-grained interpolation

(c) Fine-grained interpolation inside FMU (d) Fine-grained interpolation in MST

Fig. 6: Vertical and horizontal positions of the lower end of the double pendulum model. Dashed lines are the exact reference solutions.

3.3 2D Single Pendulum

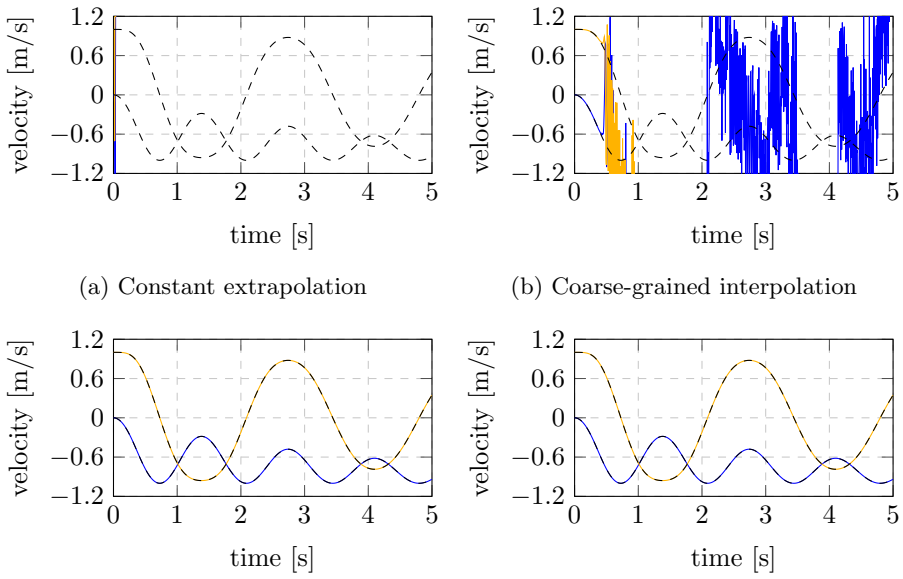
The final example composite model also consists of two pendulum arms connected through a TLM element. However, unlike the previous example model, the TLM element now has a rotational inertia of $Z_c = 1e4$. In this way the TLM element represents a fixed attachment rather than a joint. Hence, the two arms are attached to each other and will swing together like a single pendulum, see fig. 7. In general, single pendulums are less demanding to simulate compared to double pendulums. This model is included to verify the methods against a rigid TLM connection, which is locked in all dimensions.



$M_1 = 100$ kg $\quad B_1 = 20$ Ns/m $\quad L_1 = 0.5$ m
$M_2 = 100$ kg $\quad B_2 = 10$ Ns/m $\quad L_2 = 0.5$ m
$Z_{c,x} = 1e5$ Ns/m $Z_{c,y} = 1e5$ Ns/m $Z_{c,rot} = 1e4$ Nms

Fig. 7: A single pendulum is modelled as two arms connected by a stiff TLM element. Impedance is non-zero in all three dimensions.

Simulation is initiated with the two arms pointing horizontally. Results with the four different methods are shown in fig. 8. Blue and orange curves represent vertical and horizontal position, respectively. Black dashed curves are the exact reference solutions. In consistence with results from the previous composite models, extrapolation and coarse-grained interpolation are both unstable. Fine-grained interpolation results in stable simulation.



(a) Constant extrapolation

(b) Coarse-grained interpolation

(c) Fine-grained interpolation inside FMU

(d) Fine-grained interpolation in MST

Fig. 8: Vertical and horizontal positions of the lower end of the single pendulum model. Dashed lines are the exact reference solutions.

3.4 Thermodynamic connection

The presented industrial application consists of two connected FMUs for co-simulation generated from one Modelica model. The two FMUs are connected via a TLM element with a characteristic impedance of $Z_c = 700000$ sPa/m^3 and $\Delta t = 0.24$ ms. These transmission line settings stem from a pipe with an approximate length of 0.1 m in which an ideal and incompressible gas flows. The physical quantities not accounted for in the TLM connection, in this case specific enthalpy and two phase water content, are passed in both directions directly between the two FMUs as delayed signal connections.

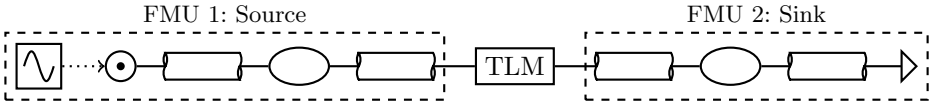
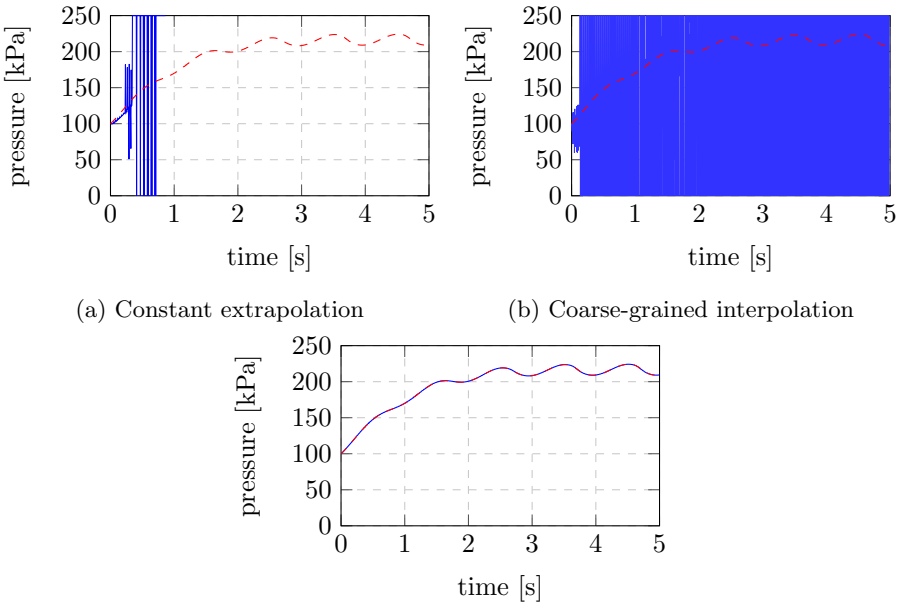


Fig. 9: Schematic description of industrial use-case. Two FMUs are connected via a TLM element. The FMUs originate from one Modelica model that is parametrized such that it can represent a source and a sink. The FMU is exported from the commercial Modelica-based modelling tool Dymola [6].

In fig. 9, the left-hand side FMU instantiation acts as a source generating an oscillating mass flow to the second instantiation which acts as a sink. The FMU sink and source specific characteristics are specified via input parameters. The Modelica model in the application example is developed in the Modelica library Modelica Fluid Light (MFL) [7]. MFL is particularly used for modelling of different industry-grade aircraft cooling and aircraft coolant distribution systems such as the aircraft vehicle systems simulator presented in [9]. The use-case in fig. 9 is relevant as its FMU interface is principally equivalent to its more complex counterparts. MFL is an in-house library developed by Saab where information of mass flow, pressure, water content, and enthalpy are passed between resistive and capacitive components. The application example comprises several different differential and algebraic equations (DAEs). The resistive components, in this example the pipes, describe the mass flow as a function of component pressure drop by means of an algebraic equation. The capacitive components, in this case the volumes, express the system dynamics by means of a first order differential equation relating net mass flow to the pressure time derivative.



(a) Constant extrapolation

(b) Coarse-grained interpolation

(c) Fine-grained interpolation

Fig. 10: Pressure in the TLM connection of the thermodynamic model. Dashed line is the Modelica simulation reference solution.

The composite model presented in fig. 9 is simulated using constant extrapolation, coarse grained interpolation, and fine-grained interpolation inside the FMU. The latter two are achieved by means of modifying the Modelica model prior

to FMU export. Two different adaptors are added to the Modelica model. The adaptor presented in listing 1.5 receives information on wave variables from the MST. These values are equally distributed in time across the macro step. An interpolation table in the adaptor is populated at the start of each communication step; the pressure input to the original model is then available at all necessary internal times by means of linear interpolation. The adaptor presented in listing listing 1.4 receives information on wave variables and their corresponding time derivatives from the master at the beginning of each communication step. The composite model input pressure can then be estimated for any necessary internal times by means of the forward Euler method. Figure 10 clearly visualizes the advantage of having interpolated input information available as numerical stability is maintained when using fine-grained interpolation. Coarse-grained interpolation and constant extrapolation results in obvious stability issues. The simulation is terminated prematurely as a result of the severe stability issues resulting from constant extrapolation.

4 Conclusions

It is shown that fine-grained interpolation is required to achieve stable connections for all the presented composite models. Variables can either be interpolated locally inside the FMU, or be handled by the master simulation tool. The first method requires the FMU to be provied with the complete interpolation table. This leads to a large amount of data exchange, which may reduce simulation performance. Meanwhile, the second method would require a callback function from where the slaves can request interpolated data from the master simulation tool. Three possible improvements to the FMI standard that would facilitate asynchronous data exchange have been identified:

- Improved support for exchanging interpolation tables
- Support for callback functions
- Support for one-step execution mode

Fine-grained interpolation would be facilitated by the first two suggestions. Support for simple exchange of interpolation tables would enable interpolation of input variables inside an FMU. Callback functions on the other hand will provide the FMU with access to variables interpolated in the master simulation tool. An advantage with a callback function is that it is not limited to pure interpolation. In addition, it can include simple expressions, for example the TLM boundary equations. Having the interpolation table in the MST also significantly reduces the amount of data transfer compared to what is necessary if the tables are located inside the FMU. On the other hand, adding support for exchanging interpolation tables would constitute a far less comprehensive modification to the current standard. Finally, one-step execution mode will enable more densely populated interpolation tables and thereby improve accuracy in the interpolated variables.

References

1. Muhammad Usman Awais, Peter Palensky, Wolfgang Mueller, Edmund Widl, and Atiyah Elsheikh. Distributed hybrid simulation using the HLA and the functional mock-up interface. *Industrial Electronics Society, IECON*, pages 7564–7569, 2013.
2. Jens Bastian, Christoph Clauß, Susann Wolf, and Peter Schneider. Master for co-simulation using FMI. In *8th International Modelica Conference, Dresden*. Citeseer, 2011.
3. T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf. The Functional Mockup Interface for tool independent exchange of simulation models. In *8th International Modelica Conference 2011*, Como, Italy, September 2009.
4. Robert Braun, Liselott Ericsson, and Petter Krus. Full vehicle simulation of forwarder with semi active suspension using co-simulation. In *ASME/BATH 2015 Symposium on Fluid Power and Motion Control*, October 2015.
5. Robert Braun and Petter Krus. Tool-independent distributed simulations using transmission line elements and the Functional Mock-up Interface. In *SIMS 54th Conference*, October 2013.
6. Dag Brück, Hilding Elmqvist, Sven Erik Mattsson, and Hans Olsson. Dymola for multi-engineering modeling and simulation. In *Proceedings of modelica*, volume 2002, 2002.
7. Magnus Eek, Hampus Gavel, and Johan Ölvander. Definition and implementation of a method for uncertainty aggregation in component-based system simulation models. *Journal of Verification, Validation and Uncertainty Quantification*, 2(1):011006, 2017.
8. Atiyah Elsheikh, Muhammed Usman Awais, Edmund Widl, and Peter Palensky. Modelica-enabled rapid prototyping of cyber-physical energy systems via the functional mockup interface. In *Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2013 Workshop on*, pages 1–6. IEEE, 2013.
9. Robert Hällqvist, Robert Braun, and Petter Krus. Early insights on fmi-based co-simulation of aircraft vehicle systems. In *The 15th Scandinavian International Conference on Fluid Power*.
10. Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.
11. P Krus. Robust system modelling using bi-lateral delay lines. In *Proceedings of the 2nd Conference on Modeling and Simulation for Safety and Security*, Linköping, Sweden, 2005.
12. Himanshu Neema, Jesse Gohl, Zsolt Lattmann, Janos Sztipanovits, Gabor Karsai, Sandeep Neema, Ted Bapty, John Batteh, Hubertus Tummescheit, and Chandraseka Sureshkumar. Model-based integration platform for fmi co-simulation and heterogeneous simulations of cyber-physical systems. In *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*, pages 235–245. Linköping University Electronic Press, 2014.
13. Tom Schierz, Martin Arnold, and Christoph Clauß. Co-simulation with communication step size control in an FMI compatible master algorithm. In *9th Int. Modelica Conf., Munich, Germany*, pages 205–214, 2012.

14. Alexander Siemers, Dag Fritzson, and Iakov Nakhimovski. General meta-model based co-simulations applied to mechanical systems. *Simulation Modelling Practice And Theory*, 17(4):612–624, 2009.