D2.4	**Adapting FMI for Co-simulation to Numerically Stable TLM Couplings**
Access[1]:	**PU**
Type[2]:	**Report**
Version:	**1.0**
Due Dates[3]:	**M36**

Open Cyber-Physical System Model-Driven Certified Development

Executive summary[4]:

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, adverse effects on the numerical stability can be avoided. This requires asynchronous communication, where variables are interpolated for the time where they are needed. The current FMI for co-simulation standard have no support for interpolating input variables inside an FMU. Some modifications to the FMI standard for improved handling of interpolation are suggested. One- and two-dimensional mechanical models are used to demonstrate the problem and the need for interpolation. It is shown that the suggested improvements are able to stabilize the otherwise unstable connections. A proposal for extending FMI with callbacks for intermediate inputs and outputs has been submitted.

[1] Access classification as per definitions in PCA; PU = Public, CO = Confidential. Access classification per deliverable stated in FPP.
[2] Deliverable type according to FPP, note that all non-report deliverables must be accompanied by a deliverable report.
[3] Due month(s) according to FPP.
[4] It is mandatory to provide an executive summary for each deliverable.

Deliverable Contributors:

	Name	Organisation	Primary role in project	Main Author(s)[5]
Deliverable Leader[6]	Robert Braun	AB SKF	T2.4 Leader	X
Contributing Author(s)[7]	Dag Fritzson	AB SKF		
Internal Reviewer(s)[8]	Robert Hällqvist	Saab AB		
	Lennart Ochel	LiU		
	Adeel Asghar	LiU		
	Jan Hartford	AB SKF		

Document History:

Version	Date	Reason for Change	Status[9]
0.1	2018-11-19	First Draft Version	Draft
1.0	2018-11-30	Final Version	Released

[5] Indicate Main Author(s) with an “X” in this column.
[6] Deliverable leader according to FPP, role definition in PCA.
[7] Person(s) from contributing partners for the deliverable, expected contributing partners stated in FPP.
[8] Typically person(s) with appropriate expertise to assess deliverable structure and quality.
[9] Status = “Draft”, “In Review”, “Released”.

CONTENTS

ABBREVIATIONS

List of abbreviations/acronyms used in document:

Abbreviation	**Definition**
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
MST	Master Simulation Tool
TLM	Transmission-Line Method

1 INTRODUCTION

Numerical robustness is a key factor in simulation solver coupling. Using different solvers for different parts of a simulation model will cause variables that are shared by multiple solvers to be delayed in time. This may result in numerical errors and instability. One solution to is to introduce physically motivated time delays in the model equations. In this way, all time delays become a natural part of the model. Hence, no non-physical time delays will need to be inserted. Experiments have been conducted to investigate how the Functional Mockup Interface (FMI) standard [1] 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.

2 PROBLEM DESCCRIPTION

Transmission Line Modelling (TLM) is a well-known technique for decoupling of simulation models [2]. The basic idea is that, in real physical systems, information always propagates with finite speed. 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 the model, equations can be separated without affecting numerical stability. In this way, sub-models can be numerically isolated from each other. As shown in Figure 1, this enables independent solvers, independent time variables and independent step-size control. A mechanical TLM element with its equations is shown in Figure 2: A TLM element with its equations.Figure 2. F is the force, v velocity, Δt the time delay and Z_c characteristic impedance. The corresponding equations apply to other physical domains as well. An existing co-simulation framework using TLM has been developed by SKF [3]. This will be merged with the OMSimulator and used for the TLM connections. The implementation is based on asynchronous socket communication. Each slave tool has 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 can also be interpolated during the step. Slave tools use callback functions for receiving inputs and sending outputs for specified time instances. Thus, input variables are kept up-to-date even during internal iterations performed by the slave.

For compatibility reasons, it is desirable to make the framework compatible with FMI by importing Functional Mock-up Units (FMUs) from other simulation tools. While FMI for model exchange easily can be adopted for TLM-based co-simulation, 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.

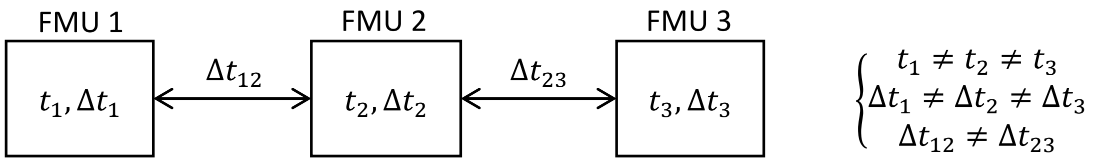

FMU 1
FMU 2
FMU 3
$t_1, \Delta t_1$
$t_2, \Delta t_2$
$t_3, \Delta t_3$
Δt_{12}
Δt_{23}
$\begin{cases} t_1 \neq t_2 \neq t_3 \\ \Delta t_1 \neq \Delta t_2 \neq \Delta t_3 \\ \Delta t_{12} \neq \Delta t_{23} \end{cases}$

Figure 1: TLM connections enable independent solvers, time variables and step-size control.

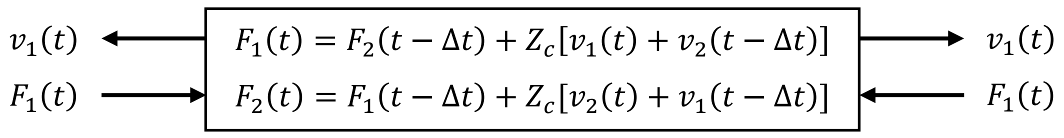

$v_1(t)$
$F_1(t)$
$F_1(t) = F_2(t - \Delta t) + Z_c[v_1(t) + v_2(t - \Delta t)]$
$F_2(t) = F_1(t - \Delta t) + Z_c[v_2(t) + v_1(t - \Delta t)]$
$v_1(t)$
$F_1(t)$

Figure 2: A TLM element with its equations.

2.1 Related Work

A prototype of a master algorithm for FMI-based co-simulation was presented in [4]. Master algorithms combining FMI-based co-simulation with the High-Level Architecture (HLA) standard have also been developed [5] [6] [7]. A TLM-based co-simulation framework with FMI support was implemented in the Hopsan simulation tool [8]. In [9] it was shown that connecting two simulation tools with TLM elements using FMI without a master simulation tool is possible. The framework 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.

Decoupling a model without using physical time delays requires a numerical time delay, which may affect accuracy and numerical stability. Errors can usually be reduced by reducing the size of the delay, at the cost of a longer simulation time. One solution is to use adaptive communication step-size [10]. In contrast with physically motivated decoupling, this method requires the FMU to support saving and restoring FMU states. This is a feature that few FMI export tools support.

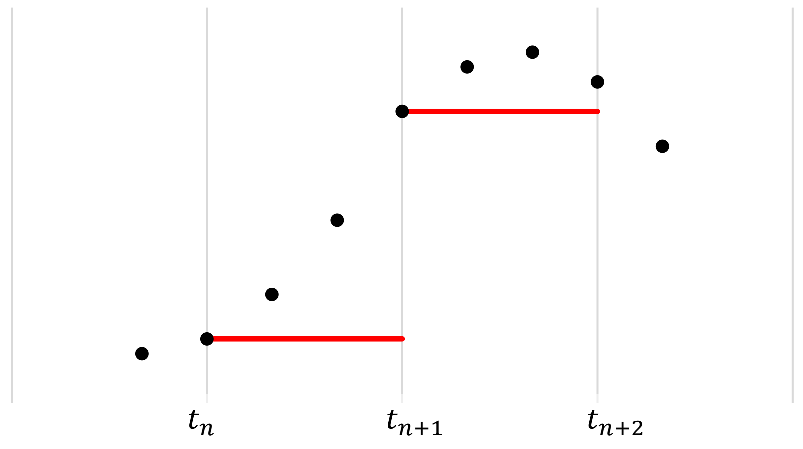
3 PROPOSED SOLUTIONS

Several solutions, both with and without modifications to the FMI standard, are implemented and analyzed. Three different variable estimation methods are used. With constant extrapolation, variables are kept constant during each communication step. Coarse-grained interpolation means that the value before and after 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 but are not expected to improve numerical stability. Detailed stability analyses of the methods are provided in appendix B and C.

Notice that annex C is yet to be submitted as a journal article **and is therefore temporarily considered as confidential** until the paper is accepted for publication.

3.1 Constant Extrapolation

The simplest method for exchanging variables is to use constant extrapolation, also known as zero-order hold, see Figure 3. Input variables are updated at the beginning of each step and remain constant during the step. This solution is fully supported by the current FMI for co-simulation standard. However, numerical stability cannot be guaranteed. It is possible to improve stability by using adaptive communication step size [10], but this assumes that the exporting tool supports saving and loading FMU states.

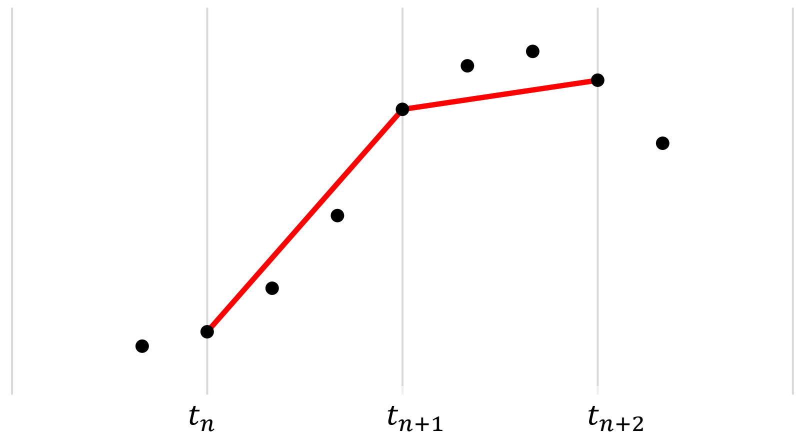


$$x(t_n \leq t \leq t_{n+1}) = x(t_n)$$

Figure 3: Constant extrapolation of input variables. Black dots represent available time-stamped data in the interpolation table.

3.2 Coarse-Grained Interpolation

It is possible by provide an FMU with approximated time derivatives of input variables. These can be used by the FMU for interpolation or extrapolation, see Figure 4. 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. However, the resolution of the interpolation will be limited since all data in the interpolation table will not be used. Also, the delayed variable is not the force, but the wave variable. Therefore, the FMU must include the TLM boundary equations, and compute the force internally.

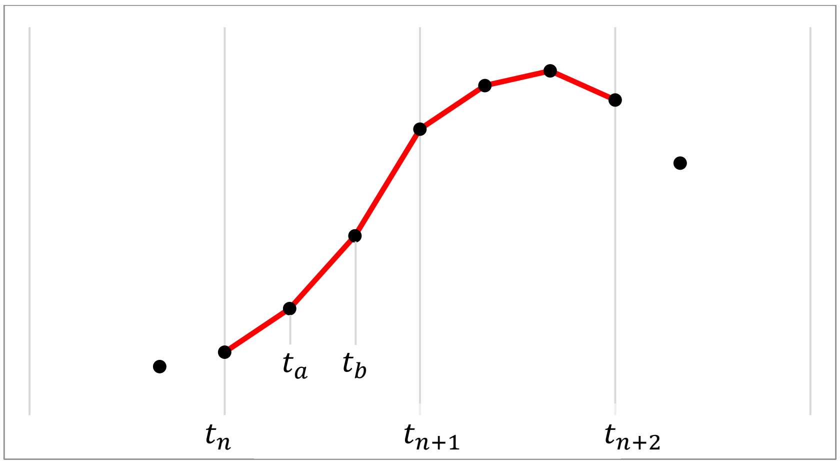


$$x(t_n \leq t \leq t_{n+1}) = x(t_n) + \frac{x(t_{n+1}) - x(t_n)}{t_{n+1} - t_n}(t - t_n) = x(t_n) + \hat{\dot{x}}(t - t_n)$$

Figure 4: Coarse-grained interpolation of input variables. Black dots represent available time-stamped data in the interpolation table.

3.3 Fine-grained Interpolation

Here, fine-grained interpolation means that all points in the interpolation table are used, as shown in Figure 5. The actual interpolation can be performed either inside the FMU, or be provided by the master simulation tool from callback functions.



$$x(t_n \le t \le t_{n+1}) = x(t_a) + \left(x(t_b) - x(t_a)\right)\frac{t - t_a}{t_b - t_a}, \quad \begin{cases} t_n \le t_j \le t_{n+1} \\ t_n \le t_k \le t_{n+1} \end{cases}$$

Figure 5: Fine-grained interpolation of input variables. Black dots represent available time-stamped data in the interpolation table.

3.4 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 master 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 master. This would make it possible for the master 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. There are always two conditions for step size. One is the local error estimate. The other one is the maximum step size. If we have a solver which does not have a variable step size, we can always use the constant maximum step size.

5 IMPLEMENTATION IN OMSIMULATOR

Both coarse-grained and fine-grained interpolation have been implemented in OMSimulator. For now, a decision was taken to support the current FMI standard without any custom extensions. Consequently, neither callback functions nor one-step execution has been implemented. This ensures that the master simulation tool will be usable by third-party organizations using standard FMUs.

Coarse-grained interpolation utilizes the FMI support for providing time derivatives of input variables. An estimated derivative is computed by drawing a straight line from the values at the beginning and at the end of the step. The FMU is then responsible for interpolating values using the value and its derivative. Only first order derivatives have been used. Higher order derivatives is neither expected to improve stability, nor supported by most tools.

Fine-grained interpolation is implemented by providing the FMU with 10 data samples and 10 corresponding time stamps for each wave variable. The FMU can then use these variables to interpolate internally. Fixed-size vectors must be used because the FMI standard does not support arrays of arbitrary length. Using 10 samples per communication step can be assumed to be stable when the time constants for two connected sub-models differs at most one order of magnitude.

For FMUs without interpolation support, OMSimulator supports sub-stepping, i.e. using a step size inside the FMU which is smaller than the TLM communication delay. The FMU will then simulate multiple steps during each master communication step. This makes it possible to update input values more frequently, at the cost of reduced simulation performance.

FMUs with interpolation support have been developed from OpenModelica and Dymola. Custom hand-coded FMUs have also been written in C++ for detailed evaluation. The Modelica code for fine-grained interpolation is using the built-in interpolation method from the Modelica Standard Library as shown in Listing 1. A 1D rotational TLM interface using fine-grained interpolation is shown in Listing 2.

```
function TLMGetInterpolatedEffort
    input Real t "Simulation time";
    input Real v "Flow";
    input Real ci[10] "Wave vector";
    input Real ti[10] "Time vector";
    input Real Zc "Characteristic impedance";
    output Real f "Computed force";
    protected Real c "Interpolated wave";
algorithm
    c := Modelica.Math.Vectors.interpolate(ti,ci,t);
    f := -c + Zc*v;
end TLMGetInterpolatedEffort;
```

Listing 1: Interpolation for fine-grained interpolation using the Modelica Standard Library.

```
model FMITLMInterfaceRotationalFineGrained1D
    Modelica.Mechanics.Rotational.Interfaces.Flange_a flange_a;
    parameter String interfaceName = "fmitlm";
    parameter Boolean debugFlg = false;
    output Real w;
    input Real c[10](start = zeros(10));
    input Real ti[10](start=linspace(-1.0,0.0,10));
    input Real Zcr;
    output Real phi;
    Real tau(start = 0);
algorithm
    phi := flange_a.phi;
    w := der(flange_a.phi);
    tau := FMITLM_Functions.TLMGetInterpolatedEffort(time,w,c,ti,Zcr);
    flange_a.tau := tau;
end FMITLMInterfaceRotationalFineGrained1D;
```

Listing 2: A Modelica interface for a 1D rotational TLM connection using fine-grained interpolation.

6 PROPOSED CHANGES TO FMI STANDARD

A change proposal has been submitted to the FMI design group, see appendix A: "FCP 015: Callback Functions for Numerical Stability". It was noted that some of the proposed changes overlaps with changes previously proposed in "FCP 010: Intermediate Output Values", which was a result from the ACOSAR project. In short, the suggestion is to extend the "fmi2CallbackFunction" struct with two more functions: "setFlow" and "getComputedEffort".

It was first investigated how fine-grained interpolation can be supported more efficiently by the FMI standard. 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 would not work with a general simulation tool. 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 solution. A proposal of a function for setting time-stamped variables are shown in Listing 3.

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

Listing 3: A proposal of a function for setting time-stamped variables.

Like the coarse-grained interpolation, this approach would also require the FMU to include the TLM equations. Due to this restriction, this extension has not been further investigated. Instead, a proposal was made based on callback functions for reading and writing input variables at specified times. Interpolation can then be handled by the master tool. This would preserve the guaranteed stability provided by TLM, while exposing only a minimal interface consisting of intensity and flow variables. The exporting tool would not need any adaption for TLM.

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. 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 4. This method has not been further investigated, as it is considered more complex compared to callback functions while not adding any additional benefits.

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

Listing 4: A proposal to add an additional event info flag for events when output variables are not available.

7 CONCLUSIONS

It is shown that fine-grained interpolation is required to achieve stable connection in all investigated models. Variables can be interpolated either locally inside the FMU or be handled by the master simulation tool. The first method requires the FMU to be provided 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. 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 from FMU to MST
- Event flag for only updating input variables

Fine-grained interpolation would be facilitated by the first two suggestions. Support for easily exchanging 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. Finally, one-step execution mode will enable more densely populated interpolation tables, and thereby improve accuracy in the interpolated variables.

A change proposal for extending the FMI standard with callback functions for intermediate inputs and outputs has been submitted.

REFERENCES

[1] 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*, Dresden, Germany, 2011.

[2] 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.

[3] A. Siemers, D. Fritzson and I. Nakhimovski, "General meta-model based co-simulations applied to mechanical systems," *Simulation Modelling Practice And Theory*, vol. 17, no. 4, pp. 612-624, 2009.

[4] J. Bastian, C. Clauß, S. Wolf and P. Schneider, "Master for co-simulation using {FMI}," in *8th International Modelica Conference*, Dresden, 2011.

[5] A. Elsheikh, M. U. Awais, E. Widl and P. Palensky, "Modelica-enabled rapid prototyping of cyber-physical energy systems via the functional mockup interface," in *Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, 2013.

[6] M. U. Awais, P. Palensky, W. Mueller, E. Widl and A. Elsheikh, "Distributed hybrid simulation using the {HLA} and the Functional Mock-up Interface," *Industrial Electronics Society, IECON*, pp. 7564-7569, 2013.

[7] H. Neema, J. Gohl, Z. Lattmann, J. Sztipanovits, G. Karsai, S. Neema, T. Bapty, J. Batteh, H. Tummescheit and C. Sureshkumar, "Model-based integration platform for fmi co-simulation and heterogeneous simulations of cyber-physical systems," in *Proceedings of the 10th International Modelica Conference*, Lund, Sweden, 2014.

[8] R. Braun and P. Krus, "Tool-Independent Distributed Simulations Using Transmission Line Elements And The Functional Mock-up Interface," in *SIMS 54th Conference*, Bergen, Norway, 2013.

[9] R. Braun, L. Ericsson and P. Krus, "Full Vehicle Simulation of Forwarder with Semi Active Suspension using Co-Simulation," in *ASME/BATH Symposium on Fluid Power and Motion Control*, 2015.

[10] T. Schierz, M. Arnold and C. Clauß, "Co-simulation with communication step size control in an FMI compatible master algorithm," in *9th International Modelica Conferance*, Munich, Germany, 2012.