



IFOSMONDI: A Generic Co-simulation Approach Combining Iterative Methods for Coupling Constraints and Polynomial Interpolation for Interfaces Smoothness

Yohan Éguillon¹^a, Bruno Lacabanne² and Damien Tromeur-Dervout¹^b

¹*Institut Camille Jordan, Université de Lyon, UMR5208 CNRS-U.Lyon1, Villeurbanne, France*

²*Siemens Industry Software, Roanne, France*

Keywords: Solver Coupling, Iterative Co-simulation, Rollback, Polynomial Interpolation, Fixed-point Method.


Abstract: This paper introduces IFOSMONDI co-simulation algorithm that combines iterative coupling methods and a smooth representation of interface variables. In explicit (i.e. non-iterative) coupling methods, representing smooth interface variables requires the introduction of a delay (Busch, 2016) because the values of the interface variables at the end of a given macro-step are not known when the co-simulation only reached the beginning of this very macro-step. One of the advantages of implicit co-simulation (i.e. iterative coupling methods) is that the values of the interface variables can be known at the end of a macro-step with the possibility to replay the integration on this very macro-step. Combining this with a polynomial representation of the interface variables enables to use interpolation instead of extrapolation across the macro-steps (Kübler and Schiehlen, 2000). Taking into account time-derivatives of interface variables makes it possible to ensure C^1 smoothness even with no history of the past exchanged data: then, no delay is introduced. A new possibility then arises: the solvers of each subsystem may take into account this smoothness and be less restrictive on their restarts due to the communication times. The results obtained on the test case of the two mass oscillators (Busch, 2016) show the advantage of IFOSMONDI coupling in terms of trade-off between elapsed time and accuracy.


1 INTRODUCTION

Industrial applications of simulations have reached a point where modular models are not only convenient but mandatory. Indeed, model providers are often specialized in a precise range of physical fields such as thermodynamic, mechanic, fluids or electrical circuits. One of the consequences of this is the necessity of dealing with modular systems built by connecting several subsystems where each of them embeds a part of the physics of the global model. In the case where the subsystems also embed a solver (which may differ between different subsystems), the simulation of the global system can be run by simulating each subsystem separately with regular communications of data between subsystems. This is co-simulation.

The co-simulation method (or co-simulation algorithm) is the rule which deals with this meta-simulation level by defining the times of the data communications, the inputs that each subsystem should

use at each step, the way the outputs are used and so on. Many co-simulation algorithms have been established until now (Kübler and Schiehlen, 2000) (Arnold and Unther, 2001) (Gu and Asada, 2004) (Bartel et al., 2013) (Sicklinger et al., 2014) (Li et al., 2014) (Busch, 2016) (Schweizer et al., 2016), with various complexity of implementation, subsystems capabilities requirements, or physical field-specific principles (see also the recent state of art on co-simulation of (Gomes et al., 2018)). It may appear in some co-simulation algorithms that a time interval should be integrated more than once on one or more subsystems. This is called iterative co-simulation. The systems that need to do so must have the capability to replay a simulation over this time-interval with different data (time to reach, input values, ...). Depending on the model provider, this capability called "rollback" is not always available on every subsystem. Besides the rollback, other platform-dependent capabilities may lead to an impossibility of use of a given co-simulation method on certain subsystems. Amongst them, the ability to provide inputs with n^{th}

^a <https://orcid.org/0000-0002-9386-4646>

^b <https://orcid.org/0000-0002-0118-8100>

order time-derivatives and the ability to obtain the time-derivatives of the outputs at the end of a macro-step can be mentioned.

In this paper we present a synchronous and iterative co-simulation algorithm called IFOSMONDI coupling as presented on figure 1 where the dynamical sub-systems are solved iteratively on a macro-step before exchanging data between sub-systems to start the solving on a new macro-step. We propose a standardisation on the multiple principles present in co-simulation algorithms so that fields-specificities vanish and that it could hence be applied on any model. The two salient features needed by our algorithm are the rollback capability and the ability to obtain the time-derivatives of the outputs.

This algorithm is inspired by two main methods previously developed in (Kübler and Schiehlen, 2000) and (Busch, 2016). (Kübler and Schiehlen, 2000) introduces the modular description of dynamic systems on the mathematical model description level. They presented an iterative method based on quasi-Newton on the system of nonlinear algebraic equations on the outputs, after having proceeded to the elimination of the inputs between subsystems. Their simulator coupling with iteration on the global integration step uses interpolation for the inputs in the integration procedure based on the history of inputs at past macro-steps and the updated inputs given by the quasi-Newton. In (Busch, 2016), an approximation method denoted as extrapolated interpolation is proposed in order to circumvent the discontinuity issue of the input at the end of each macro-step but only in the context of Gauss-Seidel and parallel Jacobi schemes (Skelboe, 1992) (non iterative procedure within a macro step). This paper proposes an algorithm that takes advantage of these two approaches which, once combined, allow us to have a more numerically efficient method than each independently.

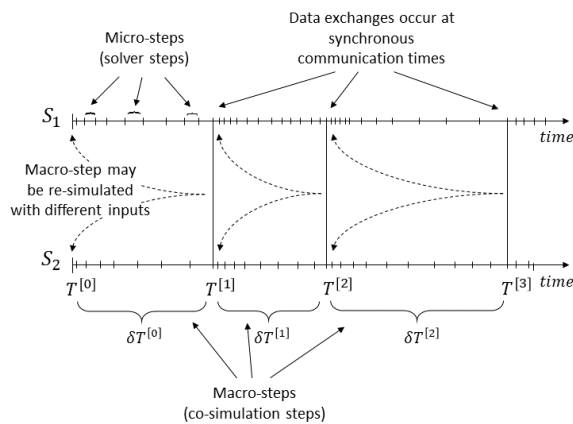


Figure 1: Visualization of the behavior of a synchronous and iterative co-simulation method on 2 subsystems.

The plan of the paper is as follows. Section 2 presents the mathematical formalism we used to develop the co-simulation algorithm which is constituted of subsystems framework, link function giving the data connectivities between subsystems and coupling constraint. Then section 3 gives the IFOSMONDI coupling algorithm showing how the combining of the iterative method and the continuity constraints interact. The macro step-size selection is also given. Section 4 gives the results of IFOSMONDI coupling on the classical linear two-mass oscillator test case (see (Schweizer et al., 2016) for example) where the parameters spring and damping constants are chosen to have more stiff dynamics. Comparison of IFOSMONDI coupling with the algorithm of (Kübler and Schiehlen, 2000) and the algorithm of (Busch, 2016) is also achieved. Section 5 gives the conclusion.

2 MATHEMATICAL FORMALISM AND MOTIVATIONS

A subsystem that communicates with other subsystems to form a co-simulation configuration will be represented by its equation. As the context is about time integration, these equations will be time differential equations. First of all, we present the general form of a monolithic system, in other words a closed system (which neither has inputs nor outputs) represented by a 1st order differential equation which covers, among others things, ODE, DAE and IDE as follows:

$$\begin{cases} F\left(\frac{d}{dt}x, x, t\right) = 0 \\ x(t^{\text{init}}) = x^{[0]} \end{cases} \quad (1)$$

where

$$\begin{aligned} n_{st} \in \mathbb{N}^*, x^{[0]} \in \mathbb{R}^{n_{st}}, t \in [t^{\text{init}}, t^{\text{end}}], \\ [t^{\text{init}}, t^{\text{end}}] \subset \mathbb{R} \text{ so that } t^{\text{end}} - t^{\text{init}} \in \mathbb{R}^*, \\ F : \mathbb{R}^{n_{st}} \times \mathbb{R}^{n_{st}} \times [t^{\text{init}}, t^{\text{end}}] \rightarrow \mathbb{R}^{n_{st}}, \end{aligned} \quad (2)$$

are given, and

$$x : [t^{\text{init}}, t^{\text{end}}] \rightarrow \mathbb{R}^{n_{st}} \quad (3)$$

is the solution state vector with its n_{st} components called the **state variables**.

This paper will only cover the ODEs case, which are of the form (4). The terms "equation", "equations", and "differential equations" will refer to the ODE of a system in the whole document.

$$\frac{d}{dt}x = f(t, x) \quad (4)$$

where

$$f : [t^{\text{init}}, t^{\text{end}}] \times \mathbb{R}^{n_{st}} \rightarrow \mathbb{R}^{n_{st}} \quad (5)$$

2.1 Subsystems Framework

In a cosimulation context, the principle to link the derivatives of the states to them (and potentially the time) is the same as in the monolithic case, yet the inputs and outputs have to be considered.

Let $n_{\text{sys}} \in \mathbb{N}^*$ be the number of subsystems. Please note that the case $n_{\text{sys}} = 1$ corresponds to a monolithic system. The cases that will be considered here are connected subsystems, that is to say $n_{\text{sys}} \geq 2$ subsystems that need to exchange data to one another: each input of each subsystem has to be fed by an output of another subsystem. A subsystem will be referenced by its index $k \in \llbracket 1, n_{\text{sys}} \rrbracket$ so that subsystem-dependent functions or variables will have an index indicating the subsystem they are attached to.

Considering the inputs and the outputs, the co-simulation version of (4) - (5) for subsystem $k \in \llbracket 1, n_{\text{sys}} \rrbracket$ is:

$$\begin{cases} \frac{d}{dt}x_k &= f_k(t, x_k, u_k) \\ y_k &= g_k(t, x_k, u_k) \end{cases} \quad (6)$$

where

$$\begin{aligned} x_k &\in \mathbb{R}^{n_{\text{st},k}}, u_k \in \mathbb{R}^{n_{\text{in},k}}, y_k \in \mathbb{R}^{n_{\text{out},k}} \\ f_k &: [t^{\text{init}}, t^{\text{end}}] \times \mathbb{R}^{n_{\text{st},k}} \times \mathbb{R}^{n_{\text{in},k}} \rightarrow \mathbb{R}^{n_{\text{st},k}} \\ g_k &: [t^{\text{init}}, t^{\text{end}}] \times \mathbb{R}^{n_{\text{st},k}} \times \mathbb{R}^{n_{\text{in},k}} \rightarrow \mathbb{R}^{n_{\text{out},k}} \end{aligned} \quad (7)$$

Equations (6) - (7) are the equations representing a given subsystem. This is the minimal data required to entirely characterize any subsystem, yet it is sufficient on its own. Please note that it does not define the whole co-simulation configuration (connections are missing and can be represented by extra data, see 2.2).

To stick to precise concepts, we should write x as $x: [t^{\text{init}}, t^{\text{end}}] \rightarrow \mathbb{R}^{n_{\text{st}}}$ (respectively for y , and u) as it is a time-dependent vector. That being said, we will keep on using x , y and u notations, as if they were simple vectors. Thus, we consciously assimilate x (resp. y and u) to $x(t)$ (resp. $y(t)$ and $u(t)$) for a given time $t \in [t^{\text{init}}, t^{\text{end}}]$.

This representation enables us to evaluate derivatives at a precise point that has been reached. It is compliant to methods that need only the subsystems' equations, such as *Decoupled Implicit Euler Method* and *Decoupled Backward Differentiation Formulas* in (Skelboe, 1992).

At a known state and a known time, that is to say when t and $x_k(t)$ are known, we can define the S_k function which is the simplified writing of g_k .

$$S_k : \begin{cases} \mathbb{R}^{n_{\text{in},k}} &\rightarrow \mathbb{R}^{n_{\text{out},k}} \\ u_k(t) &\mapsto y_k(t) \end{cases} \quad (8)$$

This notation is not proper as t and x_k are not explicitly known (they are not arguments), yet it will be used

at a given state, when t and $x_k(t)$ are known so that the representation of the outputs-inputs relation is easy to figure out.

Another concept that can be included in a system representation is a way to get the state x_k and the output y_k values at a given time according to given inputs u_k and states x_k values at a previous time. In other words, the previously described element is the following \tilde{S} function :

$$\tilde{S}_k : \begin{cases} \mathbb{R}^2 \times \mathbb{R}^{n_{\text{st},k}} \times \mathcal{L}(\mathbb{R}, \mathbb{R}^{n_{\text{in},k}}) &\rightarrow \mathbb{R}^{n_{\text{st},k}} \times \mathbb{R}^{n_{\text{out},k}} \\ ((t'), x_k(t), u_k) &\mapsto (x_k(t'), y_k(t')) \end{cases} \quad (9)$$

where $t' > t$.

It is common to consider u_k as a constant function on $[t, t']$ which makes us able to pass only $u_k(t) \in \mathbb{R}^{n_{\text{in},k}}$ instead of the u_k function. However, this only allows *zero order hold* (ZOH) as u_k can be seen as $n_{\text{in},k}$ polynomials of order 0. This is too restrictive, hence we will not consider this simplification here.

When the focus is on the interfaces only, an extracted version of \tilde{S}_k can be considered: \hat{S}_k . It will only retrieve the outputs according to the inputs. The states will be computed, but hidden. It is not a properly defined mathematical function anymore as it is assumed that the states $x_k(t)$ at time t are known for the evaluation, and that at the end of the evaluation the states $x_k(t')$ at time t' will have been computed. Here is the following function:

$$\hat{S}_k : \begin{cases} \mathbb{R}^2 \times \mathbb{R}^{n_{\text{in},k}} &\rightarrow \mathbb{R}^{n_{\text{out},k}} \\ ((t'), u_k) &\mapsto y_k(t') \end{cases} \quad (10)$$

This function will be called alternatively the *step function*, the *do-a-step function*, the *simulation function*, or by its name \hat{S}_k . An evaluation of \hat{S}_k will imply a simulation of a macro-step of (6) - (7).

It may come out that such a function is called a "simulation unit" in articles such as (Gomes et al., 2018).

We will also sometimes need to retrieve the time-derivatives of the outputs y_k at t' . Out of formalism concerns, we will write an alternative of \hat{S}_k that feeds this need.

$$\hat{\hat{S}}_k : \begin{cases} \mathbb{R}^2 \times \mathbb{R}^{n_{\text{in},k}} &\rightarrow \mathbb{R}^{n_{\text{out},k}} \\ ((t'), u_k) &\mapsto \dot{y}_k(t') \end{cases} \quad (11)$$

If considered subsystems are not able to produce this information (*e.g.* for technical reasons), an approximation with left finite differences on macro-steps can be used.

$$\tilde{\tilde{S}}_k : \begin{cases} \mathbb{R}^2 \times \mathbb{R}^{n_{\text{in},k}} &\rightarrow \mathbb{R}^{n_{\text{out},k}} \\ ((t'), u_k) &\mapsto \frac{1}{t'-t} (\hat{S}_k((t'), u_k) - y_k(t)) \end{cases} \quad (12)$$

Please consider that the use (12) instead of (11) should be done on small macro-steps so that the time derivative of y_k does not vary too much, otherwise inconsistent data may be introduced.

2.2 Link Function

Many ways to represent the connections between subsystems have been proposed in co-simulation related articles. We introduce the *link function* that we will use here. It can either be seen as a sparse storage of a permutation matrix, a non-rectangle matrix of couples, a binding of couples, *etc.*

This approach is more general than using permutation matrices for each subsystem because all subsystems may have a different number of inputs and outputs and an output may be used for several subsystem inputs. Another approach using matrices on the entire set of interface variables (inputs and outputs) is presented in (Petridis and Clauß, 2015).

For $k \in \llbracket 1, n_{\text{sys}} \rrbracket$, we can define the k -subsystem links function L_k :

$$L_k : \begin{cases} \llbracket 1, n_{in,k} \rrbracket & \rightarrow \mathbb{N}^2 \\ i & \mapsto (l, j) \end{cases} \quad (13)$$

where input i of system k is connected to output j of system l , $l \in \llbracket 1, n_{\text{sys}} \rrbracket$, and $j \in \llbracket 1, n_{out,l} \rrbracket$.

It is now possible to define the *global link function* or simply the *link function* L :

$$L : (k, i) \mapsto L_k(i) \quad (14)$$

where $k \in \llbracket 1, n_{\text{sys}} \rrbracket$ and $i \in \llbracket 1, n_{in,k} \rrbracket$.

It is important to notice that all these approaches are strictly equivalent to one another. Indeed, paper calculations and code writing make it mandatory to have a consistent way of linking interface variables, but whatever the method is considered (L function, matrices, base change, ...), the results will and must be the same.

An advantage of the link function as defined in (14) is that the total amount of data to store the information of the connections is $\sum_{k=1}^{n_{\text{sys}}} n_{in,k}$ couples of integers, that is to say $2 \cdot \sum_{k=1}^{n_{\text{sys}}} n_{in,k}$ integers, yet defining matrices of 0 and 1 binding the set of all outputs to the set of all inputs uses $\sum_{k=1}^{n_{\text{sys}}} (n_{in,k}) \cdot \sum_{k=1}^{n_{\text{sys}}} (n_{in,k})$ integers. With the link function, less data is used to retrieve the same final information.

2.3 Constraint Function

The *constraint function* or *coupling function* is related to the link function. It defines the coupling error on interfaces and thus one of the main challenges of co-simulation methods is to make it as close as possible to *zero*.

To lighten the notations, we will use double indices so that $y_{L(k,i)_1, L(k,i)_2}$ can be simply written $y_{L(k,i)}$.

$$\begin{aligned} \mathbb{R}^{\sum_{k=0}^{n_{\text{sys}}} n_{in,k}} \ni g_\lambda \left((u_{k,i})_{\substack{k \in \llbracket 1, n_{\text{sys}} \rrbracket \\ i \in \llbracket 1, n_{in,k} \rrbracket}} \right) &= \\ = \left((u_{k,i} - y_{L(k,i)})_{i \in \llbracket 1, n_{in,k} \rrbracket} \right)_{k \in \llbracket 1, n_{\text{sys}} \rrbracket} &= \\ = \begin{pmatrix} u_{1,1} & - & y_{L(1,1)} \\ \vdots & \vdots & \vdots \\ u_{1,n_{in,1}} & - & y_{L(1,n_{in,1})} \\ \vdots & \vdots & \vdots \\ u_{n_{\text{sys}},1} & - & y_{L(n_{\text{sys}},1)} \\ \vdots & \vdots & \vdots \\ u_{n_{\text{sys}},n_{in,n_{\text{sys}}}} & - & y_{L(n_{\text{sys}},n_{in,n_{\text{sys}}})} \end{pmatrix} \end{aligned} \quad (15)$$

Satisfying the coupling constraint will refer to the equation (16).

$$g_\lambda \left((u_{k,i})_{\substack{k \in \llbracket 1, n_{\text{sys}} \rrbracket \\ i \in \llbracket 1, n_{in,k} \rrbracket}} \right) = 0 \quad (16)$$

3 IFOSMONDI COUPLING

We introduce here IFOSMONDI coupling which stands for Iterative and Flexible Order, SMOoth and Non-Delayed Interfaces.

The idea of the method is to combine an iterative approach of co-simulation and a smooth representation of interface variables. In explicit (i.e. non-iterative) coupling methods, representing smooth interface variables requires the introduction of a delay (Busch, 2016) because the values of the interface variables at the end of a given macro-step are not known when the co-simulation only reached the beginning of this very macro-step. One of the advantages of implicit co-simulation (i.e. iterative coupling methods) is that the values of the interface variables can be known at the end of a macro-step with the possibility to replay the integration on this very macro-step. Combining this with a polynomial representation of the interface variables enables to use interpolation instead of extrapolation across the macro-steps (Kübler and Schiehlen, 2000). Taking into account time-derivatives of interface variables makes it possible to ensure C^1 smoothness even with no history of the past exchanged data: then, no delayed is introduced. A new possibility arises: the solvers of each subsystems may take into account this smoothness and be less restrictive on their restarts due to the communication times.

3.1 Regard of the Coupling Constraint

An intuitive solution for bringing the coupling constraint to zero is to feed the inputs with their connected outputs. This is exactly what explicit co-simulation does.

The inputs can then be represented as constants (ZOH, *zero order hold*) on the next macro-step (see figure 2), or with constant time-derivatives on the macro-step (FOH, *first order hold*) which can be determined by extrapolation on the passed exchanged values (except for the first macro-step) (see figure 3). Higher extrapolation orders are also possible, yet some problems may arise. Indeed, we can notice that feeding the inputs with their corresponding outputs at communication times only satisfies the coupling constraint (16) at the very beginning of each macro-step. At the end of them, (16) is no longer satisfied.

Moreover, the update due to the outputs-inputs communication may introduce huge discontinuities which are not always physically coherent depending on the model.

First order extrapolation (figure 3) may improve convergence around the communication times (on the right), yet it created a risk to reach non-physical values. *E.g.*, if the average slope of an input over one macro-step is strongly negative (system (S_1) , figure 3) the corresponding variable may be represented on the upcoming macro-step by a 1st order polynomial reaching negative values. If the corresponding quantity is a pressure, this is forbidden (as it is a non-sense) and the simulation will crash due to impossible values.

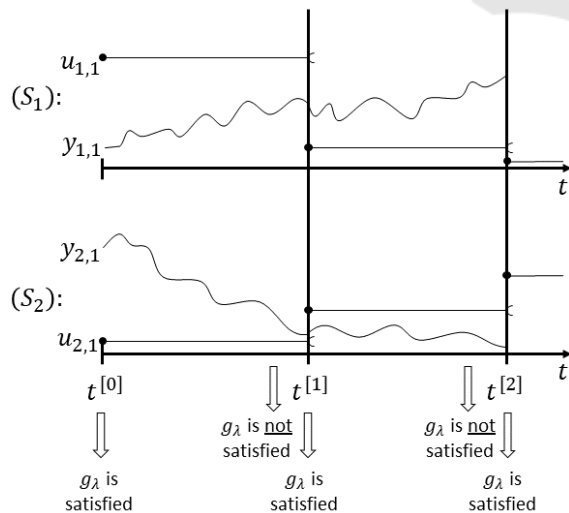


Figure 2: Explicit (i.e. non-iterative) co-simulation with ZOH on two subsystems with $L(1, 1) = (2, 1)$ and $L(2, 1) = (1, 1)$.

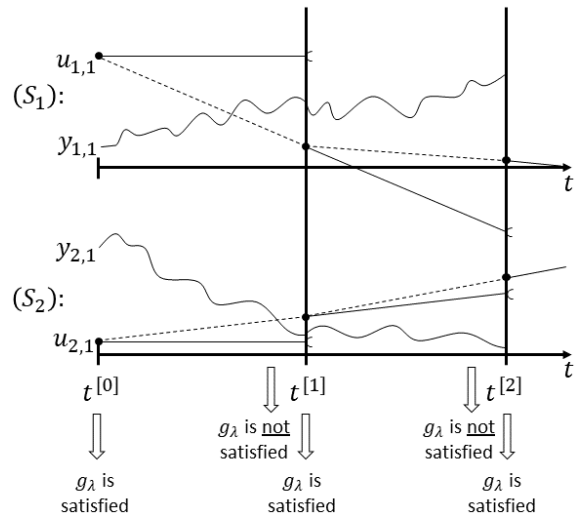


Figure 3: Explicit (i.e. non-iterative) co-simulation with FOH on two subsystems with $L(1, 1) = (2, 1)$ and $L(2, 1) = (1, 1)$; dashed lines representing the extrapolation polynomial on past exchanged data, full lines being the used part of the extrapolation polynomials for u_k representation across the macro-steps.

Alternatives have been proposed so that the values that can be reached are always in an interval bounded by already reached values (Busch, 2016), but this introduces a delay of at least the size of 1 macro-step.

Another approach consists in calling the simulation functions of each subsystem sequentially for each macro-step so that the inputs of the last ones are fed with the outputs of the first ones. This method, known as Gauss-Seidel scheme (Arnold, 2010) (Skelboe and Zlatev, 1997) satisfies one part of the coupling constraint at the beginning of the macro-steps and the other part at the end. Although this enables to keep parts of (16) satisfied either at the beginnings and ends of the macro-steps, it has drawbacks such as the sequential implementation and the fact that there is no time at which the coupling constraint is guaranteed to be fully satisfied.

The idea of IFOSMONDI coupling is to use iterative schemes so that the coupling constraint is satisfied at the end of the current macro-step. Moreover, if this is done at every macro-step, using at least 1st order polynomial representation of the inputs may keep this regard at the beginning of the step, and furthermore avoid the introduction of discontinuities (see figure 4). If additionally \hat{S} is available (see (11)), it is possible to have C^1 inputs even while crossing communication times with 3rd order Hermite interpolation polynomials (TOH) (see figure 5).

Due to conditioning issues that may appear on small macro steps $[t^{[N]}, t^{[N+1]}]$, it may be necessary to process the Hermite polynomial computation through

a variable change on $[0, 1[$. It implies a scaling on the derivatives at $t^{[N]}$ and $t^{[N+1]}$ (respectively becoming the derivatives at 0 and 1), but it may avoid failures when $t^{[N]} \gg 0$ and $t^{[N+1]} - t^{[N]}$ is very small (because of the $\frac{1}{(t^{[N-1]} - t^{[N]})^2}$ factor in the coefficients of the polynomial).

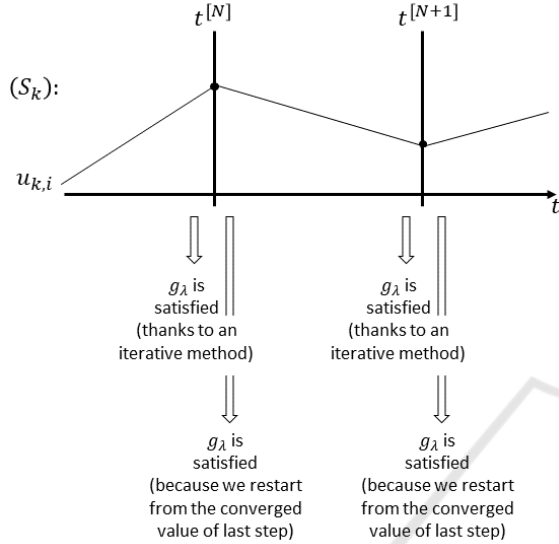


Figure 4: IFOSMONDI results on k^{th} subsystem with FOH. The drawn u are the one of the representation on last iteration of iterative method for each macro-step.

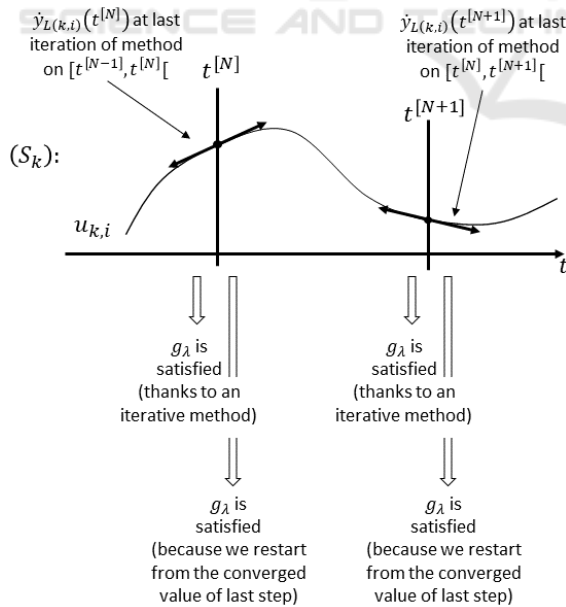


Figure 5: IFOSMONDI results on k^{th} subsystem with TOH. The drawn u are the ones of the representation on last iteration of iterative method for each macro-step.

3.2 Iterative Methods

IFOSMONDI offers two iterative methods that can alternatively be chosen in order to ensure the coupling condition (16) at the end of the macro-steps.

These iterative methods are Newton's algorithm and fixed-point method. Due to space concerns, the Newton version of IFOSMONDI coupling will be described in an extended version of this paper, still it can be seen as a generalization of methods such as (Sicklinger et al., 2014) or (Schweizer and Lu, 2015). It will be detailed in a further paper so that the analysis can really be linked to (Sicklinger et al., 2014) and (Schweizer and Lu, 2015).

Let $t^{[N]}$ be the N^{th} discretization time. For the sake of readability, we will write the global input vector (or non-rectangular matrix) at the time $t^{[N]}$ as in (17).

$$u^{[N]} := (u_{k,i}^{[N]})_{\substack{k \in \llbracket 1, n_{\text{sys}} \rrbracket \\ i \in \llbracket 1, n_{\text{in},k} \rrbracket}} \quad (17)$$

Now, the assumption is that $t^{[N]}$ has been successfully reached, and that the coupling condition is satisfied at this time (18).

$$g_\lambda(u^{[N]}) = 0 \quad (18)$$

The aim of an iterative method is to find $u^{[N+1]} = (u_{k,i}^{[N+1]})_{\substack{k \in \llbracket 1, n_{\text{sys}} \rrbracket \\ i \in \llbracket 1, n_{\text{in},k} \rrbracket}}$ so that the coupling condition is satisfied at the left of $t^{[N+1]}$, that is to say at the end of the macro-step $[t^{[N]}, t^{[N+1]})$.

$$g_\lambda(u^{[N+1]}) = 0 \quad (19)$$

Let Ψ be defined as in (20).

$$\Psi : \begin{cases} \mathbb{R}^{n_{\text{in},\text{tot}}} & \rightarrow \mathbb{R}^{n_{\text{in},\text{tot}}} \\ u & \mapsto u - g_\lambda(u) \end{cases} \quad (20)$$

$$n_{\text{in},\text{tot}} = \sum_{k=0}^{n_{\text{sys}}} n_{\text{in},k}$$

We can notice that

$$\Psi(u) = u \iff g_\lambda(u) = 0 \quad (21)$$

which means that (19) is equivalent to

$$\Psi(u^{[N+1]}) = u^{[N+1]} \quad (22)$$

that is to say

$$\Psi((u_{k,i}^{[N+1]})_{\substack{k \in \llbracket 1, n_{\text{sys}} \rrbracket \\ i \in \llbracket 1, n_{\text{in},k} \rrbracket}}) = (u_{k,i}^{[N+1]})_{\substack{k \in \llbracket 1, n_{\text{sys}} \rrbracket \\ i \in \llbracket 1, n_{\text{in},k} \rrbracket}} \quad (23)$$

It is thus possible to implement a fixed-point algorithm to find $(u_{k,i}^{[N+1]})_{\substack{k \in \llbracket 1, n_{\text{sys}} \rrbracket \\ i \in \llbracket 1, n_{\text{in},k} \rrbracket}}$. As an initial guess, it is possible to use $(u_{k,i}^{[N]})_{\substack{k \in \llbracket 1, n_{\text{sys}} \rrbracket \\ i \in \llbracket 1, n_{\text{in},k} \rrbracket}}$.

Let denote the fixed-point iteration by a left exponent. We therefore set

$$\begin{aligned} \forall k \in \llbracket 1, n_{\text{sys}} \rrbracket, \forall i \in \llbracket 1, n_{\text{in},k} \rrbracket, \\ [0]u_{k,i}^{[N+1]} &:= u_{k,i}^{[N]} \\ [m+1]u_{k,i}^{[N+1]} &:= \left(\Psi \left(\left([m]u_{l,j}^{[N+1]} \right)_{\substack{l \in \llbracket 1, n_{\text{sys}} \rrbracket \\ j \in \llbracket 1, n_{\text{in},l} \rrbracket}} \right) \right)_{k,i} \end{aligned} \quad (24)$$

Regarding the last line of (24), it is possible to simplify the expression while using the definition of Ψ (20) and g_λ (15).

$$\begin{aligned} \forall k \in \llbracket 1, n_{\text{sys}} \rrbracket, \forall i \in \llbracket 1, n_{\text{in},k} \rrbracket, \\ [m+1]u_{k,i}^{[N+1]} &:= \left(\Psi \left(\left([m]u_{l,j}^{[N+1]} \right)_{\substack{l \in \llbracket 1, n_{\text{sys}} \rrbracket \\ j \in \llbracket 1, n_{\text{in},l} \rrbracket}} \right) \right)_{k,i} \\ &= \left(\left([m]u_{l,j}^{[N+1]} \right)_{\substack{l \in \llbracket 1, n_{\text{sys}} \rrbracket \\ j \in \llbracket 1, n_{\text{in},l} \rrbracket}} - g_\lambda \left(\left([m]u_{l,j}^{[N+1]} \right)_{\substack{l \in \llbracket 1, n_{\text{sys}} \rrbracket \\ j \in \llbracket 1, n_{\text{in},l} \rrbracket}} \right) \right)_{k,i} \\ &= [m]u_{k,i}^{[N+1]} - \left(g_\lambda \left(\left([m]u_{l,j}^{[N+1]} \right)_{\substack{l \in \llbracket 1, n_{\text{sys}} \rrbracket \\ j \in \llbracket 1, n_{\text{in},l} \rrbracket}} \right) \right)_{k,i} \\ &= [m]u_{k,i}^{[N+1]} - \left(\left([m]u_{l,j}^{[N+1]} - [m]y_{L(l,j)}^{[N+1]} \right)_{\substack{l \in \llbracket 1, n_{\text{sys}} \rrbracket \\ j \in \llbracket 1, n_{\text{in},l} \rrbracket}} \right)_{k,i} \\ &= [m]u_{k,i}^{[N+1]} - \left([m]u_{k,i}^{[N+1]} - [m]y_{L(k,i)}^{[N+1]} \right) \\ &= [m]y_{L(k,i)}^{[N+1]} \end{aligned} \quad (25)$$

Please note that in the equation above (25), $[m]y_{L(k,i)}^{[N+1]}$ denotes the result of the simulation of system $(L(k,i))_1$ with inputs worthing $\left([m]u_{(L(k,i))_1,j}^{[N+1]} \right)_{j \in \llbracket 1, n_{\text{in},(L(k,i))_1} \rrbracket}$ at the end of step $[t^{[N]}, t^{[N+1]}[$.

Consequently, if ZOH is used, the inputs of subsystem $(L(k,i))_1$ have to be set to constant $\left([m]u_{(L(k,i))_1,j}^{[N+1]} \right)_{j \in \llbracket 1, n_{\text{in},(L(k,i))_1} \rrbracket}$ over the integration step $[t^{[N]}, t^{[N+1]}[$ in order to produce $[m]y_{L(k,i)}^{[N+1]}$ and thus $[m+1]u_{k,i}^{[N+1]}$. This way, smoothness cannot be respected and it is impossible to get a satisfied coupling constraint at left and right of every communication time such as in figures 4 and 5, but we get at least a satisfied coupling constraint at left of each of these times as far as the fixed-point method converges. Figure 6 shows an example of this case on two subsystems: it is exactly the iterative Jacobi co-simulation

scheme. When we want to force an input to worth a value at the end of a macro-step, we inject it at the beginning (as it is constant in ZOH): figure 6 indeed shows $[N+1]$ exponents on inputs at time $t^{[N]}$.

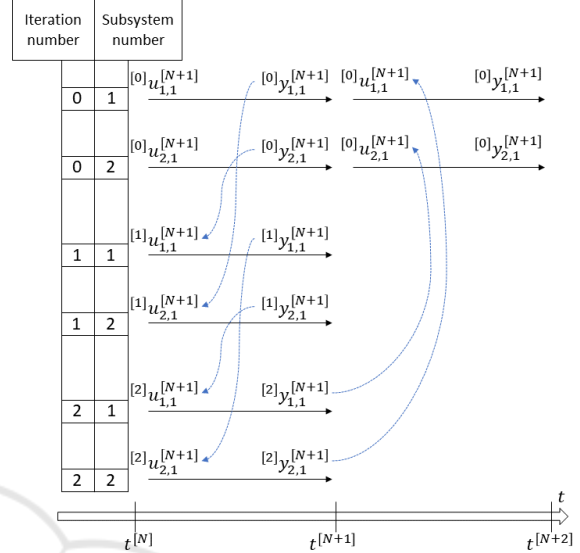


Figure 6: Jacobi scheme (IFOSMONDI with ZOH in fixed-point mode) with supposed convergence of fixed-point method after 3 iterations, on two subsystems with $L(1, 1) = (2, 1)$ and $L(2, 1) = (1, 1)$.

As far as FOH is usable, it is possible to change the scheme into a variant which is compliant with figure 4. Here are the steps :

- At $t^{[N]}$, a converged value of $(u_{k,i}^{[N]})_{\substack{k \in \llbracket 1, n_{\text{sys}} \rrbracket \\ i \in \llbracket 1, n_{\text{in},k} \rrbracket}}$ is known and satisfies the coupling constraint.
- We set $\forall k \in \llbracket 1, n_{\text{sys}} \rrbracket, \forall i \in \llbracket 1, n_{\text{in},k} \rrbracket,$
 $[0]u_{k,i}^{[N+1]} := u_{k,i}^{[N]}$.
- Until convergence, we integrate over $[t^{[N]}, t^{[N+1]}[$ with input i of system k represented by an affine function worthing $u_{k,i}^{[N]}$ at $t^{[N]}$ and $[m]u_{k,i}^{[N+1]}$ at $t^{[N+1]}$.

This way, the first fixed-point iteration will necessarily use ZOH, yet the following ones won't. The coupling condition reached at the end of $[t^{[N-1]}, t^{[N]}[$ will not be broken at the beginning of $[t^{[N]}, t^{[N+1]}[$. Moreover, the continuity of the inputs will also be guaranteed since the fixed-point method converged on every macro-step.

The principle can be extended to keep C^1 inputs if time-derivatives of inputs can be obtained at the end of every macro-step on every subsystem. TOH is then used (Hermitian polynomial with 4 constraints). In this case (corresponding to figure 5), the first

fixed-point iteration of every macro-step needs to be smoother than ZOH in order to avoid time-derivative continuity break. Indeed, figure 7 shows that at least FOH should be used to keep C^1 inputs. However, using FOH at the first fixed-point iteration on a given macro-step can only be done with extrapolation (as no other information can be known), so the dangers showed on figure 3 may appear. To reduce (but not annihilate) the risk of out-of-range values, SOH (standing for *second-order-hold*) can be done by forcing the polynomial to rejoin at the end of the macro-step the value it had at the beginning.

Current implementation of IFOSMONDI coupling uses SOH for first iteration of new macro-step when TOH is used overall (to have C^1 inputs).

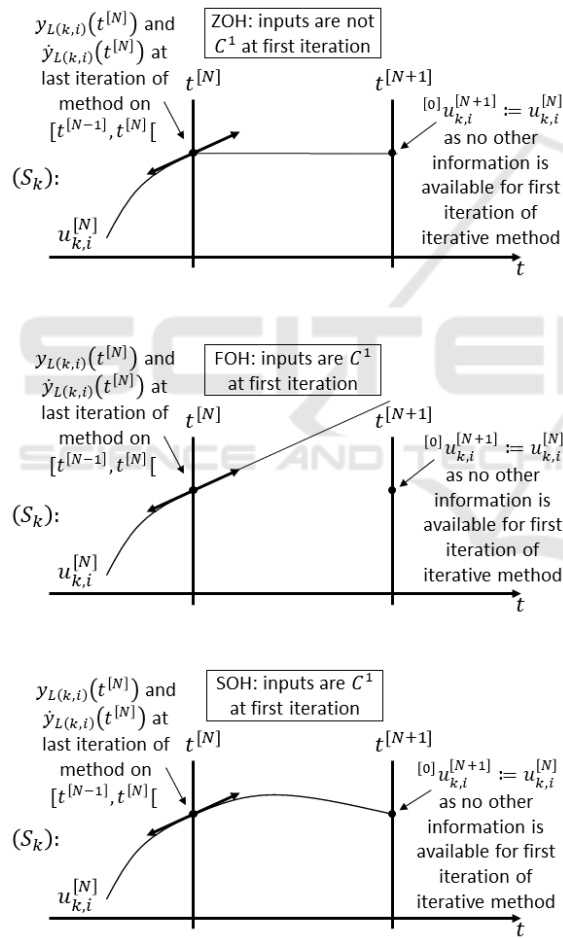


Figure 7: IFOSMONDI with TOH in fixed-point mode: visualisation of 3 strategies for first iteration of a new macro-step following a converged one.

3.3 Step Size Control

When the fixed-point method does not converge after $m_{\max} + 1$ iterations, it is possible to restart the

macro-step with reduced size (to try to integrate over $[t^{[N]}, \frac{t^{[N]} + t^{[N+1]}}{2}]$ [e.g.]). However, this is not a real error-based criterion of step-size adaptiveness. Such criteria can be established for methods which allow estimation of local error (Busch, 2016), yet here, IFOSMONDI coupling computes the fixed-point method until convergence.

Researches about the use of such a criterion are currently in progress. However, this paper only presents fixed-step version of IFOSMONDI coupling. The currently used rule-of-thumbs is:

- Define an initial synchronous macro-step size δt .
- Define m_{\max} , a maximum number of iterations to reach (m should never exceed m_{\max}).
- At the beginning of any new macro-step, try to integrate over a step of size $\min(\delta t, 1.3 \delta t^{[N-1]})$ where $\delta t^{[N-1]}$ was the previous macro-step size for which the iterative method converged.
- As soon as the iterative method converged, start the next macro-step.
- If the iterative method does not converge, while m reaches m_{\max} without convergence, divide δt by two and restart from the last reached converged time.

Further investigations will overwrite this rule. The convergence of the iterative method is defined by a criterion based on two values to provide: ϵ_{abs} and ϵ_{rel} . For subsystem $k \in \llbracket 1, n_{\text{sys}} \rrbracket$ input $i \in \llbracket 1, n_{\text{in},k} \rrbracket$, the convergence is assumed when

$$\frac{|u_{k,i} - y_{L(k,i)}|}{\epsilon_{\text{rel}} |u_{k,i}| + \epsilon_{\text{abs}}} < 1 \quad (26)$$

The convergence of the iterative method on a macro-step is assumed when every input has converged, that is to say when

$$\forall k \in \llbracket 1, n_{\text{sys}} \rrbracket, \forall i \in \llbracket 1, n_{\text{in},k} \rrbracket, |u_{k,i} - y_{L(k,i)}| < \epsilon_{\text{rel}} |u_{k,i}| + \epsilon_{\text{abs}} \quad (27)$$

When $\epsilon_{\text{abs}} = \epsilon_{\text{rel}}$, these values may be denoted by ϵ .

3.4 Algorithm

The fixed-point iterative method (subsection 3.2) is used to satisfy the coupling constraint (subsection 3.1) defined in (16) (subsection 2.3) on a 2-side neighborhood of each communication time. The macro-step will be adapted until the fixed-point method converges (subsection 3.3) on every macro-step. The IFOSMONDI coupling combines all the previously mentioned aspects. Its pseudo-code is given below.

Algorithm 1 : IFOSMONDI coupling method in "fixed-point" mode for $n_{sys} \in \mathbb{N}^*$ subsystems.

Require: t^{init}, t^{end} initial and final times
 δt base macro-step
 $(y_k^{[0]})_{k \in \llbracket 1, n_{sys} \rrbracket}$ initial outputs
 $(\hat{S}_k)_{k \in \llbracket 1, n_{sys} \rrbracket}$ simulation functions
 L link function
 $\epsilon_{abs}, \epsilon_{rel} > 0$ convergence criteria
 m_{max} iteration max

- 1: $N := 0$
- 2: $t^{[0]} := t^{init}$
- 3: $\delta t^{[0]} := \delta t$
- 4: $\forall k \in \llbracket 1, n_{sys} \rrbracket, \forall i \in \llbracket 1, n_{in,k} \rrbracket, u_{k,i}^{[0]} := y_{L(k,i)}^{[0]}$
- 5: **while** $t^{[N]} < t^{end}$ **do**
- 6: $t^{[N+1]} := t^{[N]} + \delta t^{[N]}$
- 7: $m = 1$ \triangleright interation of iterative method
- 8: **while** $(m < m_{max}) \wedge ((27) \text{ is false})$ **do**
- 9: **for** $k \in \llbracket 1, n_{sys} \rrbracket$ **do** \triangleright in parallel
- 10: **if** $m = 1$ **then**
- 11: for every input,
build $^{[0]}u^{[N]}$ as SOH in figure 7
- 12: **else**
- 13: for every input, build
 $^{[m]}u^{[N]}$ as TOH in figure 5
using $^{[m-1]}y^{[N]}$ and
corresponding time-derivatives
- 14: **end if**
- 15: $(^{[m]}y_{k,j}^{[N]})_{j \in \llbracket 1, n_{out,k} \rrbracket} :=$
 $\hat{S}_k((^{[m]}u^{[N]})_{k,i})_{i \in \llbracket 1, n_{in,k} \rrbracket}$
- 16: **end for**
- 17: $m \leftarrow m + 1$
- 18: **end while**
- 19: **if** $m = m_{max}$ **then** \triangleright convergence test failed
- 20: $\delta t^{[N]} \leftarrow \frac{\delta t^{[N]}}{2}$ \triangleright erase
- 21: **else** \triangleright convergence test succeeded
- 22: $\delta t^{[N+1]} := \min(\delta t, 1.3 \delta t^{[N]})$
- 23: $N \leftarrow N + 1$
- 24: **end if**
- 25: **end while**

4 RESULTS ON BENCHMARK MODEL

4.1 Benchmark Model

The model that will be considered is the two masses oscillator (Busch, 2016) presented on figure 8 with the chosen unbalanced parameters to avoid instantaneous simulations, and to enable comparisons of execution time.

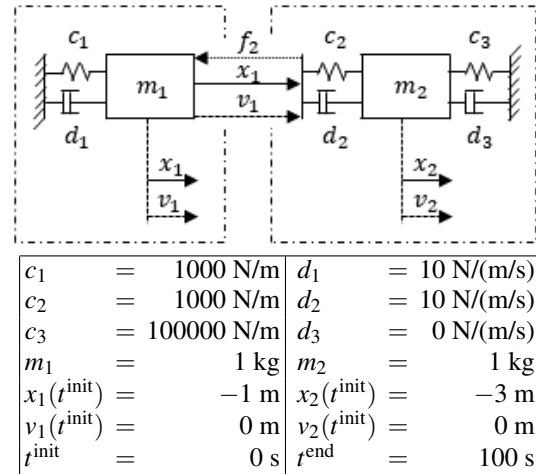


Figure 8: Two masses model with coupling on force, displacement and velocity and the chosen unbalanced parameters.

Let the model of figure 8 be decoupled in two subsystems: the $k = 1$ denoting the left part (carrying them mass m_1) and the $k = 2$ denoting the right part (carrying the mass m_2). The interfaces are:

$$(S_1) : \begin{cases} n_{in,1} = 1 \\ u_{1,1} = f_2 \\ n_{out,1} = 2 \\ y_{1,1} = x_1 \\ y_{1,2} = v_1 \end{cases} \quad (S_2) : \begin{cases} n_{in,2} = 2 \\ u_{2,1} = x_1 \\ u_{2,2} = v_1 \\ n_{out,2} = 1 \\ y_{2,1} = f_2 \end{cases} \quad (28)$$

$$L(1,1) = (2,1)$$

$$L(2,1) = (1,1)$$

$$L(2,2) = (1,2)$$

4.2 Measures Environment

All measures have been made on the same virtual machine with 4 real cores. The co-simulations have been launched one by one so that the performances of each code were fully dedicated to the current processes. Each co-simulation was made of 3 processes (1 for each subsystem and 1 for the orchestrator).

Either the two subsystems and the monolithic reference have been built with Simcenter Amesim, embedding the LSODA internal variable step solver (called for an integration on multiple micro-steps for each evaluation of \hat{S}_k , see (10)) on a macro-step.

4.3 Results

Results are presented on tables 1 and 2 in order to notice the benefits of combining the iterative method and enhanced smoothness of interfaces.

Table 1: Results on 2 masses oscillator benchmark on non-iterative methods: Explicit (denoting synchronous ZOH co-simulation) and (Busch, 2016) in its C^0 configuration using FOH for inputs.

δt	Explicit	(Busch, 2016) with C^0 interfaces
$1 \cdot 10^{-5}$	0.1% 1868s	0.2% 1813s
$5 \cdot 10^{-5}$	0.7% 379s	0.9% 378s
$1 \cdot 10^{-4}$	1.2% 203s	1.5% 179s
$5 \cdot 10^{-4}$	2.6% 40s	2.8% 36s
$1 \cdot 10^{-3}$	3.2% 18s	3.3% 20s
$5 \cdot 10^{-3}$	3.7% 5.4s	3.7% 5.1s
$1 \cdot 10^{-2}$	3.7% 3.9s	3.7% 4.1s
$5 \cdot 10^{-2}$	4.3% 1.1s	9.0% 1.4s

The algorithm denoted as "Explicit" in table 1 is the most used in the industry due to its strong link between the needed accuracy and the macro-step size δt . Indeed, target precision may be reached by reducing this macro-step size, however this substantially increases the computation time.

Enhancing smoothness of the interface variables by having C^0 inputs (Busch, 2016) leads to a similar behavior.

The benefits on accuracy produced by the fixed-point method of the second coupling algorithm presented in (Kübler and Schiehlen, 2000) enable to reach higher precisions than non-iterative methods for bigger macro-steps (see left column of table 2). However, IFOSMONDI coupling presents the lowest computation time to reach the accuracy of 0.2% of mean relative error: 7.6s. It should be noted that the macro-step δt only gives a trend: indeed, it will not necessarily stay constant as explained in subsection 3.3.

Nevertheless, when the macro-step becomes too small¹, the third-order polynomial inputs in the case of IFOSMONDI coupling behave similarly to first-order polynomial inputs in the case of (Kübler and Schiehlen, 2000). The consequence is that the computation time and accuracy reach the same values for these methods (see first line of table 2). However, it can be investigated to implement a smarter restart of the internal solvers (with bigger micro-steps just af-

¹In this precise case, *too small* stands for: too small to produce a significative difference between the first and third order polynomials.

ter the communication times) on each subsystem in the case where the C^1 smoothness of inputs is guaranteed: this may lead to a smaller computation time on IFOSMONDI coupling even with a small macro-step (top-right case of table 2).

Finally, in the results presented above, the time-derivative are computed at the end of each macro-step with a least-square method where a decentered Taylor method will be more appropriate. The latter will be tackled in further papers.

Table 2: Results on 2 masses oscillator benchmark on iterative methods: (Kübler and Schiehlen, 2000) (denoting its second simulator coupling method with $p_E = 1$) and IFOSMONDI coupling in fixed-point mode and using TOH to have C^1 interfaces ; in each case, parameters are fixed to $\epsilon_{abs} = \epsilon_{rel} = 10^{-3}$ and $m_{max} = 10$.

δt	Kübler & al, 2000 with $p_E = 1$ for continuous interfaces	IFOSMONDI coupling algorithm
$1 \cdot 10^{-3}$	0.1% 27s	0.1% 27s
$5 \cdot 10^{-3}$	1.0% 8.3s	0.2% 7.6s
$1 \cdot 10^{-2}$	3.7% 6.0s	0.6% 5.3s

5 CONCLUSIONS

The IFOSMONDI coupling algorithm allows a good trade-off between accuracy and elapsed time by combining iterative coupling method and smooth representations of interface variables. It gives similar to better results than each approach taken separately.

Investigations on further enhancements are made possible in a context where IFOSMONDI coupling is used, both upstream and downstream. Upstream in the sense of enhancements on computations of data needed by IFOSMONDI, and downstream in the sense of enhancements regarding technical aspects of the subsystems. Regarding the upstream enhancements, the way the time-derivative are computed (currently done with a least squares method, a more robust implementation based on Taylor truncated developments is currently being integrated) can be cited. Regarding the downstream enhancements, we can point a faster restart of embedded solvers inside of the subsystems (as mentionned in subsection 4.3) when C^1 smoothness of inputs is guaranteed at the communication times: the latters no more have to be seen as discontinuities by the solvers.

We limited our presentation to the fixed-point iterative coupling and in a further paper we will fo-

cus on Newton's iterative coupling with our mathematical formalism. Moreover, this formalism allows us to design a generically usable method covering domain-specific methods such as (Schweizer and Lu, 2015) (originally deriving from a predictor-corrector approach).

REFERENCES

- Arnold, M. (2010). Stability of Sequential Modular Time Integration Methods for Coupled Multibody System Models. *Journal of Computational and Nonlinear Dynamics*, 5(3):031003.
- Arnold, M. and Unther, M. G. (2001). Preconditioned Dynamic iteration for coupled differential-algebraic systems. *Bit*, 41(1):1–25.
- Bartel, A., Brunk, M., Günther, M., and Schöps, S. (2013). Dynamic iteration for coupled problems of electronic circuits and distributed devices. *SIAM J. Sci. Comput.*, 35(2):B315–B335.
- Busch, M. (2016). Continuous approximation techniques for co-simulation methods: Analysis of numerical stability and local error. *ZAMM Zeitschrift für Angewandte Mathematik und Mechanik*, 96(9):1061–1081.
- Gomes, C., Thule, C., Broman, D., Larsen, P. G., and Vangheluwe, H. (2018). Co-simulation : State of the art. *ACM Computing Surveys*, 51(3):49:1—49:33.
- Gu, B. and Asada, H. H. (2004). Co-Simulation of Algebraically Coupled Dynamic Subsystems Without Disclosure of Proprietary Subsystem Models. *Journal of Dynamic Systems, Measurement, and Control*, 126(1):1.
- Kübler, R. and Schiehlen, W. (2000). Two Methods of Simulator Coupling. *Mathematical and Computer Modelling of Dynamical Systems*, 6(2):93–113.
- Li, P., Meyer, T., Lu, D., and Schweizer, B. (2014). Numerical stability of explicit and implicit co-simulation methods. *J*, 10(5):051007.
- Petridis, K. and Clauß, C. (2015). Test of Basic Co-Simulation Algorithms Using FMI. *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*, 118:865–872.
- Schweizer, B., Li, P., and Lu, D. (2016). Implicit co-simulation methods: Stability and convergence analysis for solver coupling approaches with algebraic constraints. *ZAMM Zeitschrift für Angewandte Mathematik und Mechanik*, 96(8):986–1012.
- Schweizer, B. and Lu, D. (2015). Predictor/corrector co-simulation approaches for solver coupling with algebraic constraints. *ZAMM Zeitschrift für Angewandte Mathematik und Mechanik*, 95(9):911–938.
- Sicklinger, S., Belsky, V., Engelman, B., Elmqvist, H., Olsson, H., Wüchner, R., and Bletzinger, K.-U. (2014). Interface Jacobian-based Co-Simulation. *International Journal for Numerical Methods in Engineering*, 98:418–444.
- Skelboe, S. (1992). Methods for Parallel Integration of Stiff Systems of ODEs. *BIT Numerical Mathematics*, 32(4):689–701.
- Skelboe, S. and Zlatev, Z. (1997). Exploiting the natural partitioning in the numerical solution of ODE systems arising in atmospheric chemistry. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1196.