Implementing stabilized co-simulation of strongly coupled systems using the Functional Mock-up Interface 2.0

Antoine Viel
LMS Imagine
7 Place des Minimes, 42300 Roanne, France
antoine.viel@lmsintl.com

Abstract

This paper addresses the main issue encountered with the co-simulation of coupled systems that exchange energy, i.e. the trade-off between computational performances and numerical stability. This property is first explained in details with the help of a simple generic test system for which a large oversampling with respect to the Nyquist frequency is required in order to keep a good level of accuracy. The linearly implicit stabilization method from [4] is then implemented and tested thanks to the directional directives computation capability of the FMI for Co-simulation 2.0 standard. Some minor extensions to the standard are proposed to efficiently implement the method. When applied to the test system, it is shown that large co-simulation steps can be taken, and hence significant computation time speed-ups are observed.

Keywords: Functional Mock-up Interface; co-simulation; linear system theory; stability

1 Introduction

Among the existing methods for coupling simulation models and software, co-simulation is used for performing transient simulations of coupled simulators[1]. The fundamental principle of co-simulation is to locally decouple in time simulators that are synchronized only through a limited set of coupling variables at scheduled time instants.

The most widespread numerical co-simulation scheme, is an explicit non-iterative Jacobi-type sequence of forward solving steps [7] done by each involved numerical solver (the stepping is described on figure 1 in the case of two simulators).

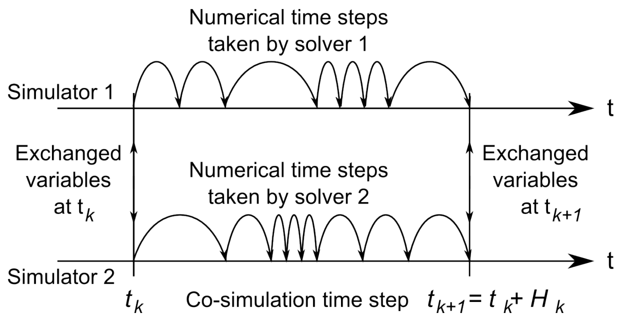
Numerical time steps taken by solver 1
Simulator 1
t
Exchanged variables at t_k
Numerical time steps taken by solver 2
Exchanged variables at t_{k+1}
t
Simulator 2
t_k
Co-simulation time step
$t_{k+1} = t_k + H_k$

Figure 1: The explicit modular stepping scheme applied to the co-simulation of two coupled subsystems. H_k is the size of the current co-simulation or macro step.

As a consequence of this stepping scheme, each simulator is seen as a discrete dynamical system from the outside, and although each simulator is able to reach convergence if taken alone, the co-simulation process of energetically coupled systems is conditionally stable, even if the system is devoid of algebraic loops. This means that the convergence of the co-simulation process depends on the size of the co-simulation step, also called macro-step. An absolute stability limit does exist beyond which divergence is rapidly reached, and slightly below this limit undamped numerical oscillations still propagate between the subsystems. Divergence and poor accuracy are usually avoided by taking small enough macro-steps, which prevents the numerical solvers from taking large numerical micro-steps and thus leads to reduced computational efficiency with respect to continuously coupled systems that are integrated with a unique variable-step solver.

This property leads to a trade-off between computational efficiency and numerical stability which can be studied on the simple test case of a strongly coupled system introduced in the next section. In a latter sec-

[1]A simulator being defined as the combination of a simulation model and mathematical libraries with a numerical solver, the latter being itself a combination of numerical integrators of ODE or DAE systems, error estimators, step size and order control heuristics, and discrete event schedulers.

tion, a numerical method is then implemented within the FMI for Co-simulation 2.0 framework, and tested with the same system, showing that this trade-off can be significantly enhanced.

2 A conditionally stable co-simulated system

2.1 Description of the test system

We consider a two degrees of freedom hydraulic system [1] obtained by connecting serially two elementary subsystems (see figure 2) made of:

1. a pipe modeled as a lumped-parameter nonlinear R-I element, with first-mode inertial effects and regular head losses

2. a volume modeled as a lumped-parameter nonlinear C-R element, with fluid-related compressibility effects, and singular head losses due to a leakage to the main circuit tank

The nonlinearities arise from the isothermal fluid properties that relate the density and compressibility to the system pressure, and from the laminar-turbulent friction models of the head losses. Some boundary conditions are introduced to model the surrounding environment: a constant pressure source on the left, and a transient flow rate source on the right.

2.2 Analysis of the continuously coupled system

The figure 3 shows the transient response of the continuously coupled system to the change of flow rate applied by the source on the right. The system parameters (pipe length and diameter, roughness, volume, head loss, ...) are chosen to be the same in the left and right subsystems. This choice is made to exemplify the nature of the coupling and its consequences on the dynamics of the coupled system.

Indeed, the dynamics of each subsystem can be studied by linearizing the system around some operating points, for example at the steady state following the first transient, for $0.5 \leq t < 1$. Instead of building the nonlinear state space and then evaluating the Jacobian matrix, a better understanding of the dynamics is obtained by analyzing the bond graph of the system.

Following the bond graph analysis of figure 4 and considering that the energy storage and dissipation elements are modeled using linear behaviour law, each elementary subsystem can be modeled by a first order

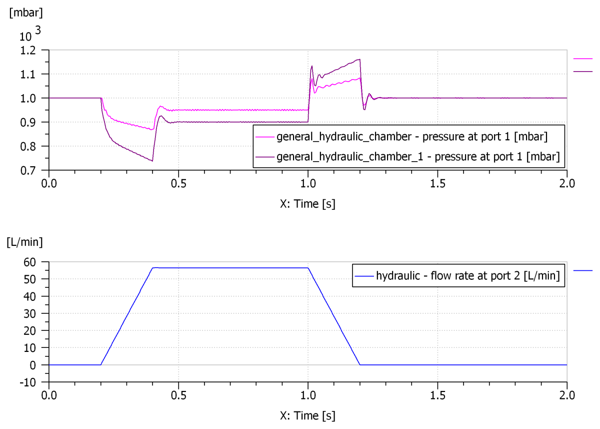

Figure 3: Transient pressure in the two hydraulic chambers (top) and source flow rate (bottom). Numerical simulation performed with the LSODA variable-step solver

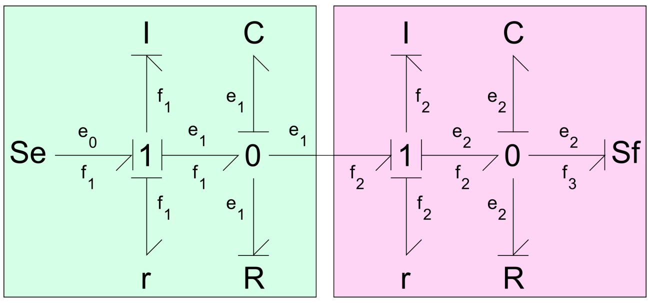

Figure 4: Bond graph of the system showing the partitioning in two subsystems. The coupling variables are e_1 (output effort from first subsystem on the left) and f_2 (output flow from second subsystem on the right).

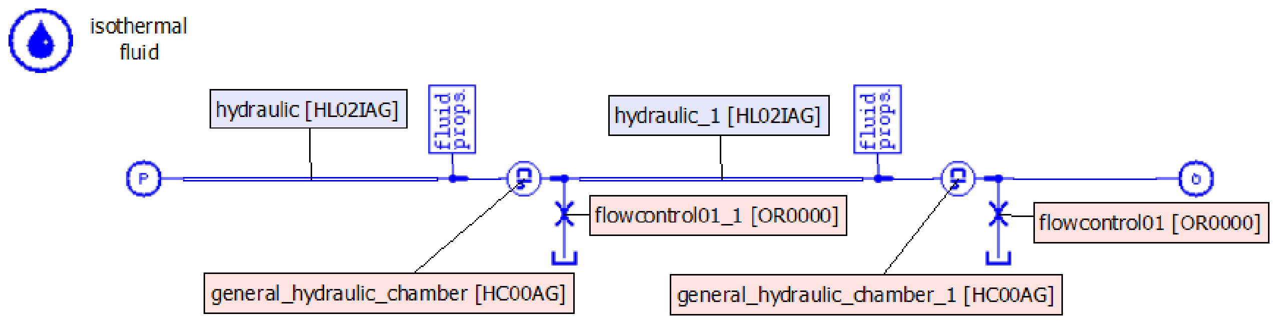

Figure 2: Sketch of an hydraulic two degrees of freedom system obtained by connecting two identical R-I-R-C subsystems built with LMS Imagine.Lab AMESim

linear system of ordinary differential equations in the state variables q_i and p_i associated with the C and I energy storage elements. For subsystem 1, the state-space equations are given by:

$$\begin{pmatrix} \dot{q}_1 \\ \dot{p}_1 \end{pmatrix} = \begin{pmatrix} -1/\tau & 1/I \\ -1/C & -2\zeta\omega_0 \end{pmatrix} \begin{pmatrix} q_1 \\ p_1 \end{pmatrix} + \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} e_0 \\ f_2 \end{pmatrix} \quad (1)$$

where $\omega_0 = 1/\sqrt{IC}$, $\zeta = r/(2\omega_0 I)$ and $\tau = 1/(RC)$ are the usual reduced parameters of a first degree of freedom linear system, and e_0, f_2 are respectively the effort source of the left subsystem, and the flow source of the right subsystem. The output relation of subsystem 1 provides the effort e_1 associated with the capacitive element:

$$e_1 = \begin{pmatrix} 1/C & 0 \end{pmatrix} \begin{pmatrix} q_1 \\ p_1 \end{pmatrix} + \begin{pmatrix} 0 & 0 \end{pmatrix} \begin{pmatrix} e_0 \\ f_2 \end{pmatrix} \quad (2)$$

For subsystem 2, it yields:

$$\begin{pmatrix} \dot{q}_2 \\ \dot{p}_2 \end{pmatrix} = \begin{pmatrix} -1/\tau & 1/I \\ -1/C & -2\zeta\omega_0 \end{pmatrix} \begin{pmatrix} q_2 \\ p_2 \end{pmatrix} + \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} e_1 \\ f_3 \end{pmatrix} \quad (3)$$

where f_3 is the flow source on the right of subsystem. The corresponding output relation gives the flow f_2 that is also the input of subsystem 1:

$$f_2 = \begin{pmatrix} 0 & 1/I \end{pmatrix} \begin{pmatrix} q_2 \\ p_2 \end{pmatrix} + \begin{pmatrix} 0 & 0 \end{pmatrix} \begin{pmatrix} e_1 \\ f_3 \end{pmatrix} \quad (4)$$

For any subsystem, defined by equations (1) or (3), and provided that the damping parameters ζ and $1/\tau$ are small, the eigenvalues of the Jacobian matrix are given by

$$\lambda_i = -\zeta\omega_0 (1 + \frac{1}{2\zeta\omega_0\tau}) \pm j\omega_0 \sqrt{1 - \zeta^2 (1 - \frac{1}{2\zeta\omega_0\tau})^2} \quad \text{for } i = 1,2 \quad (5)$$

whereas the eigenvalues of the whole system obtained by coupling the equations (1) and (3) through the output relations (2) and (4) are given by

$$\lambda_i = -\zeta\omega_0 (1 + \frac{1}{2\zeta\omega_0\tau}) \pm j\omega_0 \sqrt{\phi_i^2 - \zeta^2 (1 - \frac{1}{2\zeta\omega_0\tau})^2} \quad \text{for } i = 1,2 \quad (6)$$

where $\phi_i^2 = \frac{3\pm\sqrt{5}}{2}$ are the coupling coefficients. The non-unity ratio $\frac{\phi_1}{\phi_2} = \sqrt{\frac{3+\sqrt{5}}{3-\sqrt{5}}} \neq 1$ expresses the fact that the two subsystems are strongly coupled and that part of the dynamics lie in the coupling itself.

2.3 Analysis of the co-simulated system

The nonlinear hydraulic system of figure 2 being partitioned in the two subsystems shown on 5, the resulting coupled system is co-simulated by assigning to each subsystem a slave simulator which embeds a numerical solver. With the explicit modular stepping shown on figure 1, the output variables of each subsystem e_1 and f_2 are sampled at the communication points t_k, and are taken as input variables which are held constant on the duration H_k of the macro-step.

Based on the eigenvalues (6), it should be possible to schedule the macro-steps $(H_k)_{k=0,\cdots,M-1}$. A first approach consists in choosing a step size smaller than the Nyquist frequency, i.e. half the smallest time constant of the system, in order to ensure a proper sampling of

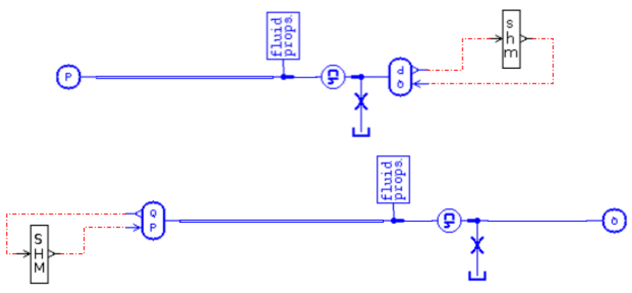

Figure 5: Sketch of the two nonlinear hydraulic subsystems being co-simulated within LMS Imagine.Lab AMESim. Blocks labeled *SHM* and *shm* are used for exchanging the coupling variables at the scheduled communication points. *SHM* stands for master simulator, whereas *shm* identifies the slave simulator.

the coupling variables:

$$H_k \leq \frac{\pi}{\max\limits_{i=1,2} |\lambda_i|}$$

Unfortunately, this bound is too high regarding stability. This can be shown empirically by performing co-simulation with macro step sizes slightly smaller than the Nyquist bound (about one tenth of the above limit period, for small damping factors of less than 1 %). In a few macro-steps, instabilities propagate between the two subsystems that rapidly lead to divergence. To understand this phenomena and be able to correctly schedule the macro step size, it is necessary to fully analyze the stability of the system with coupling variables subjected to a zero-order sample and hold process, as shown on figure 6.

Stability analysis of this type of loop sampled system is carried by following the methodology described in [8]. This analysis, which focuses on the dicretization of the coupling variables induced by the stepping, relies in other respects on the assumption that the subsystems can be exactly integrated by their respective numerical solvers[2]. Since there are only two subsystems connected through a unique loop, the analysis is done by considering the discrete-time transfer function associated with any of the two coupling variables which are obtained from the linearized[3] state-space

[2]or at least that these variable-step solvers are able to bound the truncation errors by any arbitrary tolerance

[3]around the same operating point reached in $0.5 \leq t < 1$

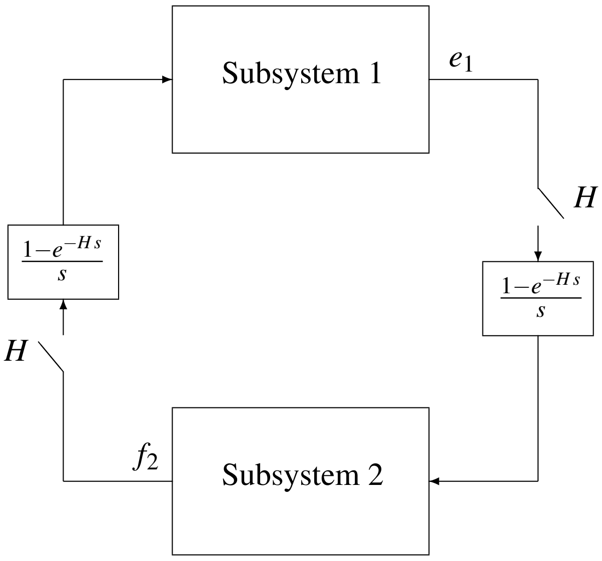

Figure 6: Bloc diagram of co-simulated system. *H* is the length of the current macro-step.

system (1)(2)

$$e_1(z) = -Z\left[\frac{1-e^{-Hs}}{s}\,\frac{G_1(s)}{C}\right] f_2(z) + Z\left[\frac{\omega_0^2\, G_1(s)}{s+\alpha_1} e_0(s)\right] \quad (7)$$

or (3)(4)

$$f_2(z) = Z\left[\frac{1-e^{-Hs}}{s}\,\frac{G_2(s)}{I}\right] e_1(z) + Z\left[\frac{\omega_0^2\, G_2(s)}{s+\alpha_2} f_3(s)\right] \quad (8)$$

with

$$G_i(s) = \frac{s+\alpha_i}{(s+\alpha_1)(s+\alpha_2)+\omega_0^2}$$

and

$$\alpha_1 = 2\,\zeta\,\omega_0, \qquad \alpha_2 = 1/\tau$$

By combining the transfer functions (7)(8) and noticing that $z = e^{Hs}$ by definition, the closed-loop transfer function is given by:

$$(1+G^\star(z))\,e_1(z) = \omega_0^2\, Z\left[\frac{G_1(s)}{s+\alpha_1} e_0(s)\right] - \frac{\omega_0^2}{C}\, G_1^\star(z)\, Z\left[\frac{G_2(s)}{s+\alpha_2} f_3(s)\right] \quad (9)$$

in which

$$G_i^\star(z) = (1-z^{-1})\, Z\left[\frac{G_i(s)}{s}\right] \quad (10)$$

and

$$G^{\star}(z) = \omega_0^2 \, G_1^{\star}(z) \, G_2^{\star}(z) \tag{11}$$

is the open loop discrete-time transfer function. Without giving here all the details about the calculation of (10) obtained using tables of Z-transforms, the Nyquist criterion [6] can be applied on the open loop transfer function (11) to evaluate the stability.

For given values of the reduced parameters, for example $\zeta = 0.1\,\%$, and $\omega_0\,\tau = 0.5$, the Nyquist plot (figure 7) shows that the system is unstable for macro-step sizes such that $H\,\frac{\omega_0\,\phi_1}{\pi} \simeq 0.1$.

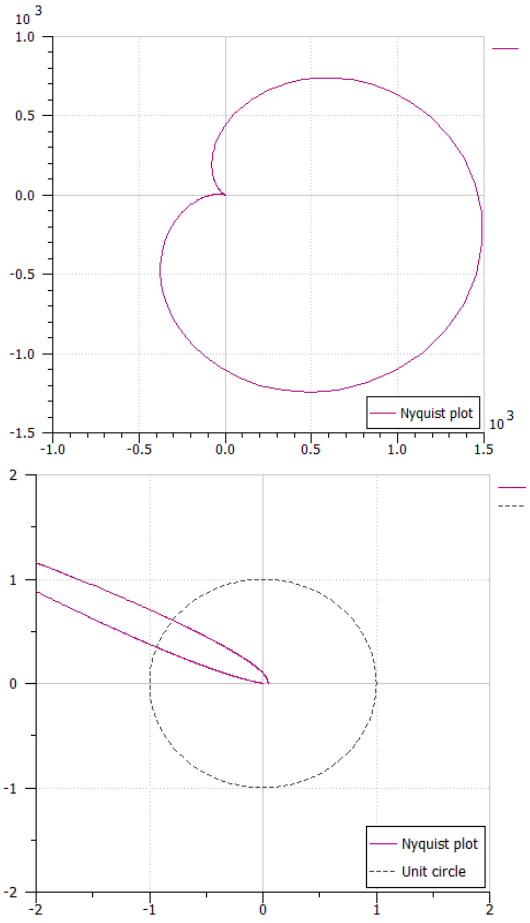

Figure 7: (top) Nyquist plot of the open loop transfer function for $\zeta = 0.1\,\%$, $\omega_0\,\tau = 0.5$, and $H \simeq 0.1\,\pi/\omega_0\,\phi_1$, which is a ten times oversampling of the Nyquist frequency. (bottom) Zoom on the unit circle showing the encirclement of the -1 point by the $z = e^{j\,\omega H}$ contour for $0 \le \omega H \le \pi$

With the same values of the damping parameters, it can be shown that the absolute stability limit is reached for $H\,\frac{\omega_0\,\phi_1}{\pi} \simeq 0.01$, and a phase margin of at least $2\,^{\circ}$ is obtained for a ratio less than 0.002, which means oversampling 500 times the Nyquist frequency. The effect of the damping coefficients on the phase margin is analyzed on figure 8. This oversampling requirement

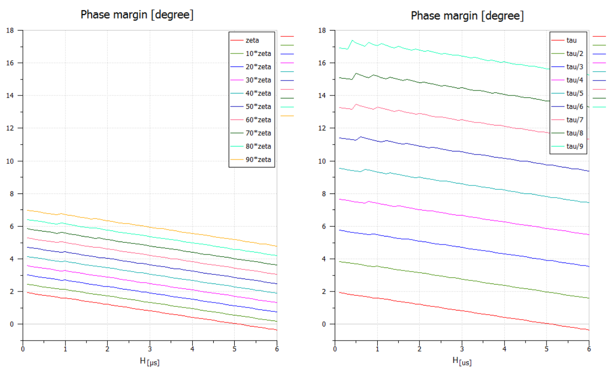

Figure 8: Plot of the phase margin versus macro-step size H, for different values of the the damping factor ζ (left), and of the decay time constant τ (right).

makes co-simulation unpractical for strongly coupled systems that are lightly damped. Indeed, comparison of the CPU time spent for performing the direct simulation of the continuously coupled system of figure 2, with the time needed for co-simulation with 500 times oversampling, exhibits a large slowdown, as shown on table 1. This is easily explained by looking at the number of micro-steps taken by the numerical solvers (last column of table 1). In the case of continuous simulation, the unique LSODA variable-step solver uses only 26000 steps, taking steps as large as 76 µs, which is only a four times oversampling of the Nyquist limit period $\frac{\pi}{\omega_0\,\phi_1}$. With co-simulation the local numerical solvers used for integrating each subsystem (DOPRI5 variable step solvers) cannot take large steps since the micro-step size is bounded by the macro-step size. It yields a large number (2×10^6) of dynamics function evaluation by the two numerical solvers, which is inefficient with respect to the frequencies of the coupled system.

3 Implementing the linearly implicit stabilization method

The rationale behind this method is to mitigate the stability-performance trade-off exemplified in the last section, by extending the phase margin. The linearly implicit stabilization method, first described by Arnold [4], makes use of the Jacobian matrices of the subsystems to build reduced linear models of the subsystems in state-space form that are exactly integrated locally in time using an unconditionaly stable method. This allows to take relatively large macro-steps, which in

Type of simulation	**CPU time** [s]	**Number of micro-steps**	**Maximum micro-step size** [µs]
Direct simulation with LSODA variable-step solver	< 1	26 000	76
Co-simulation with H = 1 µs ($H\frac{\pi}{\omega_0\phi 1} = 0.002$)	70	2 000 000	1

Table 1: Comparison of continuous time simulation with co-simulation

turn do not restrict the size of the micro-steps taken by the embedded numerical solvers.

3.1 Description of the method

According to the FMI specification [5] the external representation of a system contained in a slave corresponds to a system of ordinary differential equations. Consequently the mathematical model of the whole coupled system is described by the following set of ODEs:

$$\dot{x}(t) = f(x(t), u(t)) \tag{12}$$
$$y(t) = g(x(t), u(t)) \tag{13}$$
$$u(t) = K\,y(t) \tag{14}$$

where $x(t) = (x_1(t), x_2(t), \cdots, x_N(t))$ is the state vector obtained by gathering the state vectors of the $N \geq 2$ subsystems being involved, $u(t) = (u_1(t), u_2(t), \cdots, u_N(t))$ and $y = (y_1(t), y_2(t), \cdots, y_N(t))$ are the input and output variables of all subsystems.

The third equation, which is not specified in the slave subsystems but in the co-simulation master, is required to close the above system. It defines how the output variables are connected to the input variables through a K matrix, which verify the following properties:

- it is a square matrix, since it can be assumed with no loss of generality that the output of a subsystem is connected to exactly one input of another subsystem
- the elements of K take their value in $\{0, 1\}$
- there is exactly one 1 value per row and column of K

The FMI specification [5] for Co-simulation allows a slave subsystem to expose its Jacobian matrices related to the equations (12-13):

$$A = \nabla_x f(x,u) \quad B = \nabla_u f(x,u)$$
$$C = \nabla_x g(x,u) \quad D = \nabla_u g(x,u)$$

The linearly implicit stabilization method introduced by Arnold in [4] relies on the following three assumptions:

1. the product DK is assumed to be nilpotent [3], since the class of co-simulation methods considered here do not take into account algebraic loops on coupling variables
2. Inside a co-simulation macro-step, when $t \in [t_k, t_k + H_k[$, part of the system (12-13) can be approximated by a linear time invariant system obtained by linearizing the ODEs and the output relation around the point $x = x(t_k)$, $u = u(t_k)$
3. The linear approximate system is discretized using either the backward Euler method or the trapezoidal rule [2]

Assumption 2 leads to consider the following linear system:

$$\dot{\xi}(t) = \dot{x}(t_k) + A\,\xi(t) + B\,(w(t) - u(t_k)) \tag{15}$$
$$\eta(t) = y(t_k) + C\,\xi(t) + D\,(w(t) - u(t_k)) \tag{16}$$

where A, B, C, D are obtained at $t = t_k$ and ξ, η and w are the counterpart of x, y and u in the linear system. With this choice of variable, the corresponding initial condition is given by $\xi(t_k) = 0$.

The coupling equation (14) is thus rewritten to couple the dynamic system of each subsystem with the approximate linear system, inside a macro-step:

$$u(t) = K\,\eta(t) \tag{17}$$
$$w(t) = K\,y(t) \tag{18}$$

On the duration of a macro-step, the differential algebraic system made of equations (12), (15), (13), (17), (16), (18) holds. As there is no algebraic loop (assumption 1), this DAE can be reduced to a coupled set of ODEs by taking the derivate the last four equations.

Following assumption 3, the ODE (15) is first discretized using one of the two proposed methods:

$$(I - r(t-t_k)A)\,\xi(t) = (t-t_k)\,[\dot{x}(t_k) + rB\,(w(t) - u(t_k))]$$

where r depends on the discretization method and is either 1 for backward Euler or 0.5 for the trapezoidal rule. This equation provides an estimate for the state vector derivative:

$$\dot{\xi}(t) \simeq \mathscr{A}(t)\,[\dot{x}(t_k) + rB\,(w(t) - u(t_k))]$$

as well as for the output relation (16):

$$\begin{aligned}\dot{\eta}(t) &= C\dot{\xi}(t) + D\dot{w}(t)\\ &= C\mathscr{A}(t)\,[\dot{x}(t_k) + rB\,(w(t) - u(t_k))] + D\dot{w}(t)\end{aligned} \tag{19}$$

with

$$\mathscr{A}(t) = (I - r(t-t_k)A)^{-1}$$

In order to explicitly take into account the lack of algebraic loop, the equation (18) is approximated using the assumption 2. This means that the inputs w are obtained from a linear approximation of the outputs y of the subsystems around the point $t = t_k$:

$$w(t) - u(t_k) = K\,[C\,(x(t) - x(t_k)) + DK\,(\eta(t) - y(t_k))]$$

$$\dot{w}(t) = K\,[C\dot{x}(t) + DK\,\dot{\eta}(t)]$$

After substituting these two relations into (19) and noticing the nilpotency of DK due to assumption 1, an explicit differential equation for the outputs η is obtained:

$$\begin{aligned}\dot{\eta}(t) = C\mathscr{A}(t)\,\dot{x}(t_k) + DKC\,f(x(t), K\,\eta(t))\\ + rC\mathscr{A}(t)\,BK\,[C\,(x(t) - x(t_k)) + DK\,(\eta(t) - y(t_k))]\end{aligned} \tag{20}$$

The coupling condition (17) applied to (12) finally rewrites as:

$$\dot{x}(t) = f(x(t), K\,\eta(t)) \tag{21}$$

These two ODE (20) and (21) along with the initial condition $\eta(t_k) = y(t_k)$ explicitly define the dynamics of the system on the duration of a macro-step $[t_k, t_{k+1} = t_k + H_k[$. At the end of the step the outputs are evaluated and propagated among the subsystems using (13):

$$y(t_{k+1}) = g(x(t_{k+1}), K\,\eta(t_{k+1}))$$

3.2 Computational flow

The following notation is introduced to describe the submatrices obtained by restricting to the variables involved in the slave simulator numbered $s \in \{1, \cdots, N\}$:

- $K_{\bar{s},s}$ is the square submatrix of K obtained by taking the columns corresponding to the output variables of slave s, and the rows corresponding to the input variables of the other slave simulators that are connected to the outputs of slave s;

- $K_{s,\bar{s}}$ is the square submatrix of K obtained by taking the rows corresponding to the input variables of slave s, and the columns corresponding to the output variables of the other slave simulators that are connected to the inputs of slave s.

The computational flow is a two steps process, the first step taking place at the communication point of co-simulation, the second step being the continuous time solving of the coupled DOE system (20)(21) during one co-simulation macro-step.

At $t = t_k$: The slave subsystem $s \in \{1, \cdots, N\}$ computes:

- the derivative of its state vector

$$\dot{x}_s(t_k) = f_s(x_s(t_k), K_{s,\bar{s}}\,\eta_{\bar{s}}(t_k))$$

- its outputs

$$y_s(t_k) = g_s(x_s(t_k), K_{s,\bar{s}}\,\eta_{\bar{s}}(t_k))$$

- the Jacobian matrices

$$\begin{aligned}A_{s,s} &= \nabla_x f_s(x_s(t_k), K_{s,\bar{s}}\,\eta_{\bar{s}}(t_k))\\ B_{s,s} &= \nabla_u f(x_s(t_k), K_{s,\bar{s}}\,\eta_{\bar{s}}(t_k))\\ C_{s,s} &= \nabla_x g_s(x_s(t_k), K_{s,\bar{s}}\,\eta_{\bar{s}}(t_k))\\ D_{s,s} &= \nabla_u g_s(x_s(t_k), K_{s,\bar{s}}\,\eta_{\bar{s}}(t_k))\end{aligned} \tag{22}$$

- it also provides the updated value $x_s(t_k)$ of its state vector. If $k = 0$ this is the global initial condition of (12)

- it receives its inputs and set up the stepwise local initial condition for its extended state:

$$K_{s,\bar{s}}\,\eta_{\bar{s}}(t_k^+) = K_{s,\bar{s}}\,y_{\bar{s}}(t_k) \tag{23}$$

For $t_k < t \leq t_{k+1}$**:** The slave subsystem $s \in \{1, \cdots, N\}$ solves a subset of the coupled DOE system (20)(21):

$$\dot{x}_s(t) = f_s(x_s(t), K_{s,\bar{s}}\, \eta_{\bar{s}}(t)) \tag{24}$$

$$K_{s,\bar{s}}\, \dot{\eta}_{\bar{s}}(t) = f_{\bar{s}}(x_s(t), K_{s,\bar{s}}\, \eta_{\bar{s}}(t), t) \tag{25}$$

where $f_{\bar{s}}$ is evaluated by the master:

$$\begin{aligned} f_{\bar{s}}(x_s, K_{s,\bar{s}}\, \eta_{\bar{s}}, t) = {} & K_{s,\bar{s}}\, C_{\bar{s},.}\, \mathscr{A}(t)\, \dot{x}(t_k) \\ & + K_{s,\bar{s}}\, D_{\bar{s},\bar{s}}\, K_{\bar{s},s}\, C_{s,s}\, f_s(x_s, K_{s,\bar{s}}\, \eta_{\bar{s}}) \\ & + r\, K_{s,\bar{s}}\, C_{\bar{s},.}\, \mathscr{A}(t)\, B_{.,\bar{s}}\, K_{\bar{s},s}\, [C_{s,s}\, (x_s - x_s(t_k)) \\ & + D_{s,s}\, K_{s,\bar{s}}\, (\eta_{\bar{s}} - y_{\bar{s}}(t_k))] \end{aligned} \tag{26}$$

Practically, the derivative of the state vector $\dot{x}_s$ is provided by the slave to the master, which uses it for evaluating the second term of equation (26) instead of evaluating the function f_s that appears in the right-hand side of equation (24).

3.3 Implementation within the FMI framework

The guiding principle behind the organization of computation is a strict devolution of responsability between the slaves and the master in the co-simulation process. Each slave FMU is responsible locally for the system being solved and does not have any information about the coupling and the surrounding environment. It is up to the master algorithm to gather and assemble this information from the different slaves and to provide to the slaves ways for evaluating the additional ODE that represent the linearized part of the coupled system. This information is provided partly in the description[4] of the model structure of each slave, and partly at run-time through appropriate functions that evaluate the directional derivatives of the system enclosed in the FMU. With this structure-related information, as well as the variables exchanged at communication like the outputs of the models, state variables and their derivatives, a cooperation between the master simulator and slave simulators can be set up that keep the organization clean from the point of view of computational responsabilities.

In addition, the implementation of the stabilized co-simulation has to be compatible with the classical modular stepping specified by the FMI for Co-simulation [5]. This means that the stabilization method operates only if both the slaves FMU and the co-simulation master cooperate. If it is not supported by any part of the coupled systems, the classical modular stepping with stepwise extrapolation has to be applied.

[4]stored in the corresponding XML file distributed with the FMU

On the side of the slave FMU, it is mandatory to first enable the computation of the directional directives of all state variables and connected outputs, with respect to all state variables and connected inputs. This is done by declaring the flag *providesDirectionalDerivative* and by implementing the *fmiGetDirectionalDerivatives* function of the FMI specification. Moreover, the stabilization method being a model-based extrapolation, it is not compatible with the optional history-based extrapolation schemes that are allowed by the FMI specification. So the current *canInterpolateInputs* attribute has to be extended to specify which type of extrapolation is actually supported.

The way the slave handles its input variables has to be modified: the input variables that appear in the dynamics equation (24) are now considered as additional state variables, according to equation (25), and the input variables now act as initial conditions (equation (23)) for these state variables. The state vectors to be solved by the numerical integrator embedded in the slave is thus (x_s, u_s), where u_s is actually $K_{s,\bar{s}}\, \eta_{\bar{s}}$, the actual mapping between the outputs $\eta_{\bar{s}}$ of the linear system and the inputs of the slave being done by the master, since the structural information about connection (i.e. the elements of the K matrix) is only known by the master simulator.

If the master does not support stabilization, the right-hand side of equation (25) reduces to zero and so the actual inputs of the slave do not vary: a zero-order hold extrapolation is thus performed as defined in the FMI specification when the *canInterpolateInputs* flag is not set. In that case, the inputs of each slave is directly given by the initial condition (23), in which $y_{\bar{s}}$ are the outputs of the slaves that are fed to the inputs of the slave number s, through the *fmiGetReal* and *fmiSetReal* functions called by the master simulator at each communication point $(t_k)_{k=0,\cdots,M-1}$.

On the side of the master simulator, more tasks are required to implement the linearly implicit stabilization method. The master has to set up a callback function that is used to evaluate the dynamics associated with the inputs of the slaves. Before co-simulation, the model structure information of the slaves is used to prepare the matrices and the computations that appear in (26). Then, during co-simulation, these elements are updated at each communication step, through calls to the *fmiGetContinuousStates*, *fmiGetDerivatives* and *fmiGetDirectionalDerivatives* functions. Notice that the first two functions are originally defined only in the *Model Exchange* part of specification, so the Co-simulation specification has to be extended in the future. The same is true about the availability of the callback function associated with relation (26). This function is aimed at being called by the the numerical solvers of the slaves, when performing the numerical integration of (24)(25). A proposal for extending the FMI for Co-simulation is to define a function called for example *fmiGetStabilizedInputDerivatives* and having the following arguments:

- the current time of the slave numerical solver at which the right hand side of (26) is desired
- the current value of the additional state variables, i.e. the stabilized inputs $u_s = K_{s,\bar{s}}\,\eta_{\bar{s}}$
- the current values of the state variables and state derivatives $(x_s, f_s(x_s, u_s))$ of the slave
- practically, a reference to the master environment, declared as *componentEnvironment* during instantiation, may be needed to help the master simulator identify the calling slave.

This function should return the vector of derivatives of the stabilized inputs, according to relation (26). It has to be declared by the master environment in the *fmiCallbackFunctions* argument of the slave instantiation function.

The master algorithm is roughly sketched in Algorithm 1, in which the additional tasks required for stabilization are emphasized.

3.4 Test results

The hydraulic test case studied in section 2.1 and composed of two elementary hydraulic systems connected in serie is tested under various conditions of co-simulation:

- Reference implementation : transient simulation of the continously coupled system, using the LSODA variable-step solver [2]
- Co-simulation with native interfaces in the LMS Imagine.Lab AMESim simulation environment, or through a specially crafted prototype of a master simulator supporting the FMI 2.0 RC1 [5] specification with the proposed extensions described in section 3.3

```
**Require:** $N$ slave FMU $s \in \{1, \cdots, N\}$ and a sequence
  of $M$ macro-steps $\{H_0, \cdots, H_{M-1}\}$.
  read the model structure description of slaves and
  create the connection matrix $K$
  instantiate each slave $s$ and provide to it a callback
  function $f_{\bar{s}}$
  provide the initial conditions for all slave variables
  initialize the slaves
  initialize time $t = t_0$
  **for** $k = 0$ **to** $M - 1$ **do**
    get the outputs of slaves in $y$
    get the state variables and derivatives $x_s, \dot{x}_s$ of
    slaves
    get the Jacobian matrices $A_{s,s}, B_{s,s}, C_{s,s}, D_{s,s}$ of
    slaves
    set the inputs of slaves as $u = K\,y$
    perform one macro-step for the slaves from $t$ up
    to $t + H_k$
    update time $t \leftarrow t + H_k$
  **end for**
```

Algorithm 1: Master simulator algorithm. Lines in blue correspond to the additional tasks required by the stabilization method.

- Comparison of stabilized co-simulation with explicit modular stepping
- Comparison of a mixed C/Python implementation for the co-simulation master, or «direct» C implementation, both with DOPRI5 variable-step solver for the slave simulators

For comparison purpose, all tests are performed with the same values of the reduced damping coefficients as in section 2.3, namely $\zeta = 0.1\,\%$ and $\omega_0\,\tau = 0.5$. The limit period corresponding to the Nyquist frequency is $\pi/\omega_0\,\phi_1 \simeq 425\,\mu s$, the absolute stability limit is reached for $H \simeq 5\,\mu s$, and a 2° phase margin is obtained for $H = 1\,\mu s$.

The test results are summarized in table 2. The accuracy of co-simulation is measured by taking the root mean square error on the coupling variables e_1 and f_2 with respect to the reference implementation and the mixed tolerance for all the variable-step solvers is set to 10^{-5}. In all co-simulation tests, the CPU time is measured by setting the print interval (i.e. the sampling of result variables) to $50\,\mu s$, corresponding to the largest macro-step size used across the tests, in order to have comparable processing times regarding result storage and disk access.

The stabilization method allows to choose large macro-step sizes H, up to the size of the numerical

Tested implementation	**Macro-step size** H [µs]	**RMS error**	**CPU time** [s]
#1. Continously coupled system with variable step solver in AMESim		*reference*	1.0
#2. Explicit co-simulation, native AMESim interface	1	104	70.0
#3. Stabilized co-simulation, native AMESim interface	50	207	8.5
#4. FMI 2.0 explicit co-simulation, mixed Python/C prototype	1	123	40.0
#5. FMI 2.0 stabilized co-simulation, mixed Python/C prototype	50	237	36.0
#6. FMI 2.0 explicit co-simulation, direct C prototype	1	123	12.0
#7. FMI 2.0 stabilized co-simulation, backward Euler, direct C prototype	50	215	1.5
#8. FMI 2.0 stabilized co-simulation, trapezoidal rule, direct C prototype	50	152	1.5

Table 2: Summary of tests performed with different implementations, macro-step sizes H and discretization methods.

micro-steps taken by the variable-step solver in the reference case. Indeed, the absolute stability limit of co-simulation seems to be reached for macro-step sizes of about 76 µs, which is the maximum numerical step size reported in table 1. Consequently, a maximum value of 50 µs is used for comparison across the tests #3, #5, #7 and #8.

Clearly, the performances obtained for the maximum macro step size depend on the implementation. Additional operations are needed by the stabilization method like the computation of Jacobian matrices and integration of the additional state equations related to the inputs on the side of the slaves, and the evaluation of the approximate dynamics (26) on the side of the master. This overhead, if not efficiently implemented, may cancel out the gain over the number of micro-steps. This is the case with the mixed Python/C implementation #5 of the master simulator, which exhibits poor performances with respect to #3, due to the too many context switches between the two environments. On the contrary, the two other C-based implementations #3 and #7 (or #8), show a large speed-up factor of about 8 with respect to the corresponding explicit co-simulations #2 and #6.

Regarding the accuracy, the tests #3, #5 and #7, which are based on a backward Euler method, yield nearly the same level of accuracy, about twice the error obtained with explicit co-simulation at $H = 1$ µs. The increase of the RMS error with the macro-step size is depicted on figure 9 for tests #7 and #8. Clearly, the use of a second-order method like the trapezoidal rule provides more accurate results, for the same computational load.

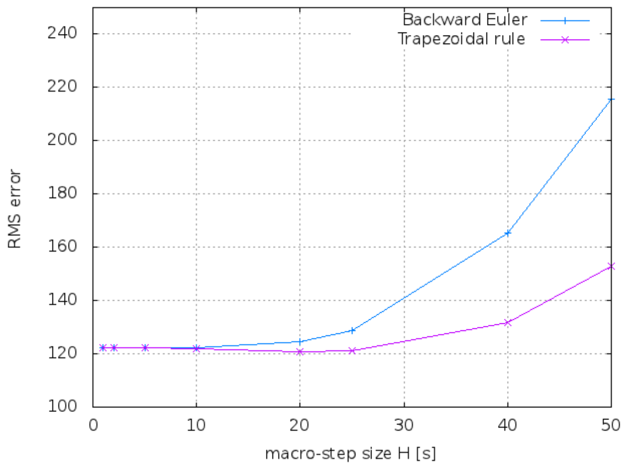

Figure 9: RMS errors obtained with respect to continuously coupled simulation versus the macro step size.

4 Conclusions

Although co-simulation is generally considered a robust method of simulator coupling, this paper presented the main issue that remains with the co-simulation of strongly coupled system, namely the trade-off between stability (or accuracy) and the computational performances. With the help of a simple yet representative example of this class of system, it showed how the stability issue may affect the computational load, since an oversampling factor as large as 500 is observed with respect to the highest dynamics of the system. The implementation of the linearly implicit stabilization method within the framework of the FMI for Co-simulation 2.0 standard then showed that significant speed-up can be regained at the price of a moderate loss of accuracy, provided that an efficient implementation is available as well as some minor extensions to the FMI for Co-simulation specification. With the advent of the FMI 2.0 specification, which one of its major enhancements is an interface for the directional derivative matrices, it seems that the efficient co-simulation of strongly coupled systems becomes feasible. Although this paper focused on the stabilization of the more common type of co-simulation, i.e. the explicit modular stepping, further performance gains are expected with more advanced types of co-simulation, for instance by combining the stabilization method with variable macro-step size heuristics or implicit (iterative) stepping, since these currently seldom used techniques are now enabled by the FMI specifications.

Acknowledgments

This work was in part supported by DGCIS in the ITEA2 MODRIO (11004) project under contract number 12.2.93.0126.

References

[1] Akers A., Gassman M., Smith R., Hydraulic Power System Analysis, CRC Press, 2006.

[2] Ascher U., Petzold L., Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations, Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[3] Arnold M., Clauss C., Schierz T., Error Analysis and Error Estimates for Co-simulation in FMI for Model Exchange and Co-simulation V2.0, The Archive of Mechanical Engineering, Vol. LX, No 1, 2013.

[4] Arnold M., Numerical stabilization of co-simulation techniques, the ODE case, Working document MODELISAR, sWP 200-203, September 5, 2011.

[5] Modelica Association Project "FMI", Functional Mock-up Interface for Model Exchange and Co-simulation 2.0 Release Candidate 1, October 18, 2013, available at `https://svn.modelica.org/fmi/branches/public/specifications/FMI_for_ModelExchange_and_CoSimulation_v2.0_RC1.pdf`

[6] Franklin G.F., Powell J.D., Workman M.L., Digital Control of Dynamic Systems, Addison Wesley, 1997.

[7] Kübler R., Schiehlen W., Two Methods of Simulator Coupling, Mathematical and Computer Modelling of Dynamical Systems, Vol. 6, No. 2, 2000.

[8] Viel A., Strong Coupling of Modelica System-Level Models with Detailed CFD Models for Transient Simulation of Hydraulic Components in their Surrounding Environment, 8th International Modelica Conference, Dresden, March 2011.