Notes on Co-Simulation

Francisco González, Borja Rodríguez, Alejandro Zar, Miguel Ángel Naya

Laboratorio de Ingeniería Mecánica - LIM

Universidade da Coruña

August 3, 2021

1 Introduction

Since its introduction in industrial environments in the 1960s, computer simulation of engineering systems has proved itself a valuable tool to reduce R&D costs and shorten product development cycles. In particular, multibody systems dynamics (MBS) has been successfully used to this end in a wide range of industrial applications. During the latest decades, hardware and software improvements have made it possible to effectively represent complex phenomena such as contacts and flexibility in MBS simulations [11, 12]. Better computational capabilities, however, have also resulted in growing expectations regarding the performance and results to be delivered by predictive simulation software tools. While early MBS simulations were limited to dealing with mechanical systems composed of a few rigid bodies, nowadays the goal is to accurately describe the dynamics of complex engineering applications, which often include non-mechanical components such as electronics or hydraulics, in an efficient way. Real-time performance is demanded in many cases, such as Human/Hardware-in-the-Loop (HiL) environments.

This transition from component-level to system-level simulation has motivated the need to develop formalisms that are able to handle multiphysics systems, in which the governing dynamics equations and time scales are likely to differ across its components. The methods that have been proposed to address this task can be categorized into two main groups: ***monolithic simulation*** and ***co-simulation***, as discussed in Section 1.2. These two approaches differ in the way in which they consider and handle the dynamical systems that describe the application to be simulated.

1.1 Dynamical systems

A ***dynamical system*** is a model that describes a physical system by means of a series of variables, named the ***state*** (in the case of a mechanical system, its generalized coordinates and velocities), and a set of ***evolution rules*** that define how this state changes as time progresses [14]. For most mechanical systems, these evolution rules are given in the form of a set of ordinary differential equations (ODEs) or differential algebraic equations (DAEs).

Example: continuous dynamical system

As an example, consider the point-mass, simple pendulum moving under gravity effects, shown in Fig. 1. If the distance between points O and P is L, the mass at point P is m, and the acceleration of gravity along the negative y-axis is g, then the dynamics of this one-degree-of-freedom mechanical system can be given by the following equation of motion

$$mL^2\ddot{\varphi} - mgL\sin\varphi = 0 \tag{1}$$

which is a second-order ODE expressed in terms of the angle φ from the vertical axis to the pendulum rod.

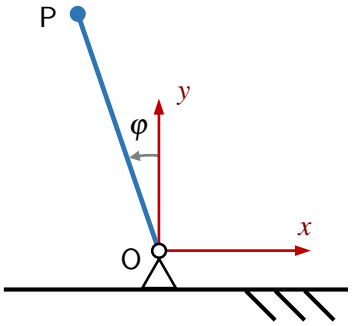


Figure 1: A point-mass simple pendulum.

If the position and velocity of the pendulum, φ_0 and $\dot{\varphi}_0$, are known for a given time $t = t_0$, then the motion of the pendulum can be obtained solving the initial value problem defined by these initial conditions and Eq. (1). The pendulum state $\mathbf{x}$, in this case, includes angle φ and its derivative with respect to time, $\dot{\varphi}$

$$\mathbf{x} = \begin{bmatrix} \varphi \\ \dot{\varphi} \end{bmatrix} \tag{2}$$

The evolution rule for this system would be defined by the differential equation of motion (1) and the

initial conditions, as follows

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\varphi} \\ \ddot{\varphi} \end{bmatrix} = \begin{bmatrix} \dot{\varphi} \\ g \sin\varphi / L \end{bmatrix}; \qquad \mathbf{x}_0 = \begin{bmatrix} \varphi_0 \\ \dot{\varphi}_0 \end{bmatrix} \tag{3}$$

The dynamical representation of a mechanical system is not unique. The motion of the pendulum in Fig. 1 can be characterized by means of the x and y coordinates of point P with a system of two differential equations

$$\begin{bmatrix} m & 0 \\ 0 & m \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} + \begin{bmatrix} 0 \\ mg \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{4}$$

The pendulum state would be in this case $\mathbf{x} = \left[x,\, y,\, \dot{x},\, \dot{y} \right]^{\mathrm{T}}$. However, the two coordinates x and y are not independent; they are related through a kinematic constraint that enforces that the distance between O and P is always L

$$\Phi = x^2 + y^2 - L^2 = 0 \tag{5}$$

In this case, the motion of the pendulum is described by both the differential equations of motion (4) and the constraint equation (5), together with the initial conditions $\mathbf{x}(0) = \mathbf{x}_0$. This means that the evolution rule of the mechanical system is given as an initial value problem with a system of DAEs instead of ODEs. The solution of the dynamics of mechanical systems defined by means of either ODEs or DAEs is one of the key topics addressed in the Multibody Systems (MBS) dynamics literature [4, 13].

Example: discrete dynamical system

Not all dynamical systems are described using differential equations. Some can be expressed using a sequence of vectors $\{\mathbf{q}_n\}_{n=0}^{\infty}$ generated by an iteration rule. For instance

$$\mathbf{q}_{n+1} = a \sin \mathbf{q}_n, \qquad \mathbf{q}_0 = \mathbf{Q} \tag{6}$$

where a is a scalar and $\mathbf{Q}$ is the initial value of the vector series. The transition from one state to the next one can be determined by the reaching of a certain instant in time (time-driven system), or by the occurrence of an ***event*** that triggers the transition (event-driven system).

Mathematical definition of a dynamical system

The mathematical definition of a dynamical system can be found in [33], in the form of rules that generate sequences of vectors in a general metric space. Consider a sequence $\{\mathbf{q}_n\}_{n=0}^{\infty}$ that satisfies

$$\mathbf{q}_{n+1} = \mathbf{g}(\mathbf{q}_n), \qquad \mathbf{q}_0 = \mathbf{Q} \in \mathbb{R}^p \tag{7}$$

where $\mathbb{R}^p$ is an Euclidean metric space of dimension p, and function $\mathbf{g}$ satisfies $\mathbf{g}: D \to \mathbb{R}^p$, where $D \subseteq \mathbb{R}^p$ is the domain of definition of $\mathbf{g}$. Vector $\mathbf{Q}$ is known as the initial data, initial value, or initial condition and Eq. (7) is the map of iteration. Equation (7) defines a dynamical system on a subset $E \subseteq \mathbb{R}^p$ if, for every $\mathbf{Q} \in E$, there exists a unique solution $\{\mathbf{q}_n\}_{n=0}^{\infty}$ of Eq. (7) defined and remaining in E for all $n \geq 0$.

Note that this definition is only valid for dynamical systems defined using a discrete sequence as evolution rule. However, it can also represent continuous dynamical systems defined by ODEs or DAEs when numerical integration formulas are used to step forward in time the system dynamics.

Simulators

A ***simulator*** or ***solver*** is an algorithm that computes the evolution of the state and outputs of a dynamical system [14]. The set of trajectories followed by the state and outputs is known as the ***behaviour trace*** of the system.

1.2 Monolithic simulation and co-simulation

Nowadays, most applications of industrial interest can only be described using rather complex dynamical systems. Often, such applications include subsystems of disparate natures, with dissimilar physical behaviours and time scales. An electric road vehicle is a good representative of these multiphysics assemblies. While many subsystems in the car fall within the definition of a conventional mechanical system, like the suspension and the steering mechanism, others like the electronics components in the e-powertrain and the ADAS, or the hydraulic behaviour of the power brake, cannot be appropriately represented with conventional MBS formulations. Moreover, while the component-level simulation of these components is still useful and necessary during the development cycle, the interactions between dissimilar components need to be understood in detail to ensure that the behaviour of the final product will match the design specifications. For this reason, full-vehicle models geared towards system-level simulation are nowadays required at several stages in the development cycle of a new car.

System-level simulation can be addressed following several approaches; most of them can be catego-

rized into two major groups: monolithic simulation and co-simulation. The difference between these lies in the way in which dynamical systems are formulated, solved, and integrated.

Monolithic simulation

Monolithic simulation consists in describing the response of all the components in the system to be studied with a single, all-encompassing set of equations. When this approach is followed, a single solver takes care of the integration of the dynamics of every component in the assembly. This solver typically has access to the details that describe the dynamic model of each component, as these are often required to build and solve their dynamics equations.

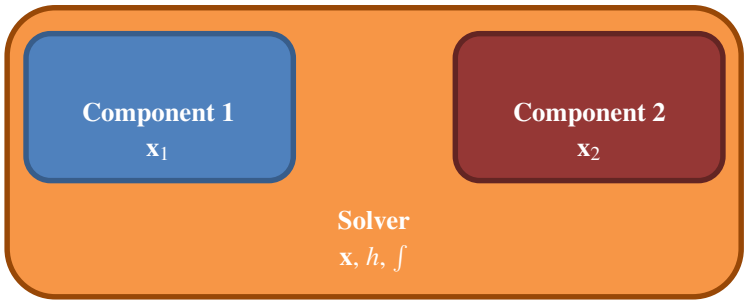


Figure 2: Scheme of a monolithic simulation.

Figure 2 shows the generic scheme of a monolithic simulation environment with two subsystems. Each component can be understood as a dynamical system with its own state, $\mathbf{x}_1$ for component 1 and $\mathbf{x}_2$ for component 2. A single solver, with a given integration method $\int$, is responsible for the assembly of the equations of motion and their numerical integration, often using a common integrator for all the components. The overall assembly is regarded as a single dynamical system; its state $\mathbf{x}$ includes the states $\mathbf{x}_1$ and $\mathbf{x}_2$ of the individual components, and a common evolution rule governs the change of all of them as time progresses.

When properly designed, monolithic integration methods tend to be robust and efficient. They have been successfully used in a wide range of applications, including mechatronics [29] and hydraulically actuated mechanical systems [21, 27]. Among their drawbacks is the need to develop a dynamic formulation that is able to deal with the behaviour of every subsystem, which can become a complicated task when these are governed by very dissimilar physical equations. Besides, in many cases the solver needs to have full access to the implementation details of each component to assemble the dynamics equations of the whole system. Because of confidentiality reasons, this poses a problem in industrial applications that combine components from different vendors. An additional difficulty in industry stems from the fact that, sometimes, companies use for their simulation tasks software models that have been developed over the years with particular modelling tools. Repeating the development of these models to integrate them in a

new monolithic simulation software would be costly and time-consuming. Co-simulation is an alternative to monolithic implementations that offers a possible answer to the above mentioned problems.

Co-simulation

Instead of viewing the application under study as a single process, co-simulation envisions it as a set of dynamical systems, each with its own solver, whose integration proceeds independently in time.

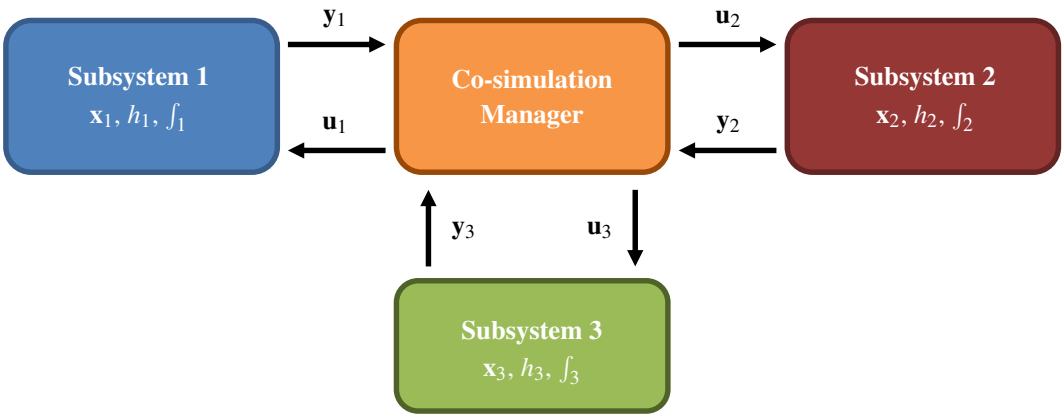


Figure 3: Scheme of a generic co-simulation environment.

A generic co-simulation scheme with three subsystems is shown in Fig. 3. Each subsystem is an independent solver, with its own state $\mathbf{x}_i$ $(i = 1,2,3)$, integration method $\int_i$, and step-size h_i. The interaction of each solver with its environment, i.e., the rest of subsystems in the assembly, is represented by a set of ***coupling variables*** that includes its input $\mathbf{u}_i$ and output $\mathbf{y}_i$. The exchange of coupling variables between the subsystems is coordinated by a ***co-simulation manager*** or ***orchestrator***, responsible for the synchronization of the integration processes of the solvers.

The exchange of coupling variables between the subsystems takes place at discrete-time ***communication points***, at which they receive their input and send their output to the manager. Between two consecutive communication points, each subsystem proceeds with the integration of its state without any interaction with its environment. This means that information about the evolution of the rest of subsystems is generally not available until the next communication point is reached. The time interval between two communication points is known as a ***macro time-step***.

Co-simulation processes usually consist in two stages. During the ***initialization*** phase, a consistent initialization of the simulators must be ensured. The initial states of the different subsystems may have to satisfy one or more algebraic constraints, which generally involve the inputs and outputs of the system and can be expressed as

$$\mathbf{\Phi}\left(\mathbf{x}_{1,0},\cdots\mathbf{x}_{n,0},\mathbf{y}_{1,0},\cdots\mathbf{y}_{n,0},\mathbf{u}_{1,0},\cdots\mathbf{u}_{n,0}\right) = \mathbf{0} \tag{8}$$

where n is the total number of subsystems in the setup. Solving Eq. (8) may require an iterative procedure to ensure the consistency of the initial states of all the subsystems. Once the initialization is complete, the ***execution*** phase can begin.

Example: linear oscillator

A two-degree-of-freedom mass-spring-damper linear oscillator, shown in Fig. 4, will be used here as example to illustrate the difference between monolithic implementations and co-simulation. This simple mechanical system has been used several times in the co-simulation literature as benchmark problem, e.g., [16, 31, 9, 15].

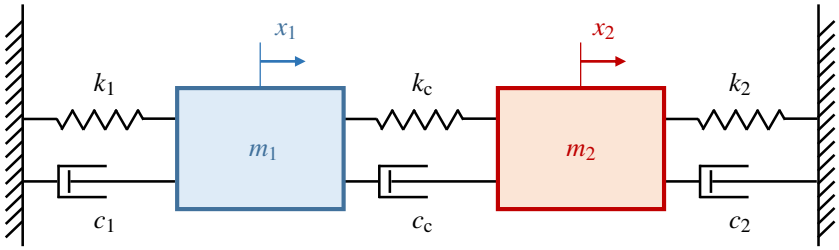


Figure 4: A two-degree-of-freedom linear oscillator.

The dynamics equations for the linear oscillator can be formulated as follows

$$\begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix} \begin{bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix} + \begin{bmatrix} c_1 + c_c & -c_c \\ -c_c & c_2 + c_c \end{bmatrix} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} + \begin{bmatrix} k_1 + k_c & -k_c \\ -k_c & k_2 + k_c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \tag{9}$$

where m_1 and m_2 are the values of the oscillator point masses, k_1, k_2, and k_c represent the stiffness of the springs that connect the masses to the ground and to each other, and c_1, c_2, and c_c are their damping coefficients. Variables x_1 and x_2 denote the displacement of each mass with respect to the equilibrium configuration, and f_1 and f_2 contain the forces applied on each point mass. Equation (9) can be expressed in a more compact form as

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{C}\dot{\mathbf{x}} + \mathbf{K}\mathbf{x} = \mathbf{f} \tag{10}$$

where $\mathbf{x} = [\ x_1 \quad x_2\ ]^{\mathrm{T}}$ is the set of generalized coordinates that define the position of the oscillator, and $\mathbf{M}$, $\mathbf{C}$, and $\mathbf{K}$ are its mass, damping, and stiffness matrices, respectively. Defining some initial conditions

$$\mathbf{x}(t=0) = \mathbf{x}_0\,, \qquad \dot{\mathbf{x}}(t=0) = \dot{\mathbf{x}}_0\,, \tag{11}$$

one obtains a dynamical system, formed by Eqs. (10) and (11), whose motion can be integrated using one of the many numerical integration formulas that exist in the literature. The resulting forward-dynamics

integration is an example of monolithic simulation, in which the dynamics of the whole system is formulated together and handled by a single integrator.

It is also possible to formulate the system dynamics equations following a co-simulation approach. Figure 5 shows the linear oscillator divided into two subsystems, $\mathcal{M}_1$ and $\mathcal{M}_2$. The way in which this division is conducted is not unique, and many options exist to carry it out, as will be shown in Section 2.

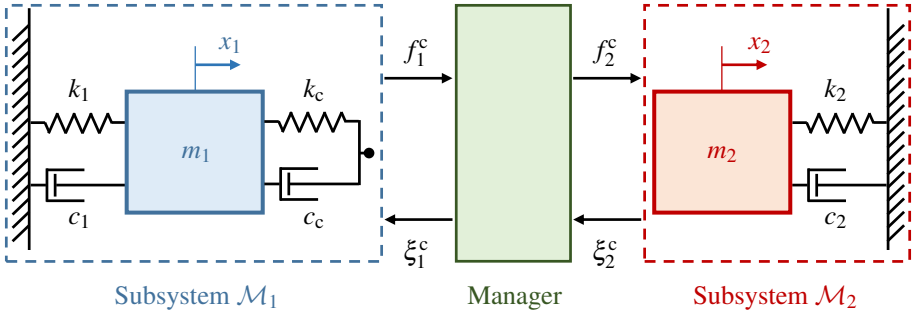


Figure 5: Division of the linear oscillator into subsystems.

In this particular example, subsystem $\mathcal{M}_1$ evaluates the coupling force between the two point masses and sends it to subsystem $\mathcal{M}_2$, which in turn returns its position to $\mathcal{M}_1$. The dynamics of each subsystem can be formulated separately. For subsystem $\mathcal{M}_1$

$$m_1\ddot{x}_1 + (c_1 + c_c)\dot{x}_1 - c_c\dot{\xi}_1^c + (k_1 + k_c)x_1 - k_c\xi_1^c = f_1 \tag{12}$$

where ξ_1^c is the position of the point mass in subsystem $\mathcal{M}_2$, received by $\mathcal{M}_1$ as an external input. The coupling force, f_1^c is evaluated by subsystem $\mathcal{M}_1$ as

$$f_1^c = k_c\left(x_1 - \xi_1^c\right) + c_c\left(\dot{x}_1 - \dot{\xi}_1^c\right) \tag{13}$$

The dynamics of subsystem $\mathcal{M}_2$, in turn, can be written as

$$m_2\ddot{x}_2 + c_2\dot{x}_2 + k_2x_2 = f_1 + f_2^c \tag{14}$$

where f_2^c is the coupling force, received from $\mathcal{M}_1$. The output of subsystem $\mathcal{M}_2$ is its position with respect to the equilibrium configuration $\xi_2^c = x_2$. The block labelled as "Manager" in Fig. 5 is responsible for scheduling and synchronizing the execution of the numerical integration processes of the two subsystems, and coordinating the exchange of coupling variables, and maybe performing some manipulations on them to improve the accuracy or stability of the overall process. For this reason, in general, $f_1^c \neq f_2^c$ and $\xi_1^c \neq \xi_2^c$.

With this approach, each subsystem becomes a dynamical system on its own. The dynamics equations of $\mathcal{M}_1$ and $\mathcal{M}_2$, (12) and (14), can now be integrated separately, using different integrator formulas and

step-sizes if needed. On the other hand, it is necessary to define the management methods that will take care of the coordination of these integration procedures.

2 Co-simulation schemes

Solver coupling can be carried out following a wide variety of approaches. The most commonly used co-simulation schemes and configuration options are summarized in this Section. It must be noted that the nomenclature and notation in this area have not fully converged yet [17], and so the names given to the methods here described may not match those used in some publications in the literature.

2.1 Continuous and event-driven co-simulation

When a subsystem in a co-simulation environment has its evolution rule described by differential equations it is called a ***continuous subsystem***. Such a system can be described with the following initial value problem

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x},\mathbf{u}) \\ \mathbf{y} &= \mathbf{g}(\mathbf{x},\mathbf{u}) \\ \mathbf{x}(t=0) &= \mathbf{x}_0 \end{aligned} \tag{15}$$

where $\mathbf{x}$ is the system state, $\mathbf{u}$ and $\mathbf{y}$ stands for the inputs and the outputs, respectively, and $\mathbf{f}$ and $\mathbf{g}$ are, in general, nonlinear functions. Simulation tools often integrate the differential equation in (15) by means of a time discretization. For instance, with a Taylor series expansion

$$\mathbf{x}(t+h) = \mathbf{x}\,t + h\,\mathbf{f}(\mathbf{x}(t),\mathbf{u}(t)) + \boldsymbol{\eta} \tag{16}$$

where h is the integration step-size and $\boldsymbol{\eta}$ is the approximation error incurred by the discretization with respect to time. The behaviour trace obtained this way is an approximation of the one that the system would theoretically have. From this point of view, most time-driven subsystems can be considered continuous subsystems.

A ***discrete-event subsystem***, on the other hand, has its change of state triggered by an external event. These subsystems feature a limited set of possible states, input events, and output events. While continuous subsystems have states that evolve continuously over time and outputs that have to obey the physical laws of continuity, discrete-event subsystems can have states that assume multiple values at the same time (***transiency***) and discontinuous outputs [14].

The term ***hybrid co-simulation*** refers to co-simulation environments in which some subsystems can be considered continuous and others are discrete-event ones. There are two general approaches to tackling these problems. A possibility is formulating the continuous subsystems as discrete-event ones, and

using a discrete-event manager to orchestrate the simulation. Conversely, the discrete-event subsystems can be converted into continuous subsystems; in this case a continuous manager will be used to coordinate the numerical integration.

2.2 Jacobi and Gauss-Seidel schemes

The order in which subsystems are evaluated during execution has an important effect on the accuracy and stability of the results.

The ***Jacobi scheme*** executes the subsystems in parallel. At each communication instant t_n the manager receives the outputs from the subsystems and sends them their corresponding inputs. Then, every subsystem moves forward in time until the next communication point at time t_{n+1} without further interaction with its environment.

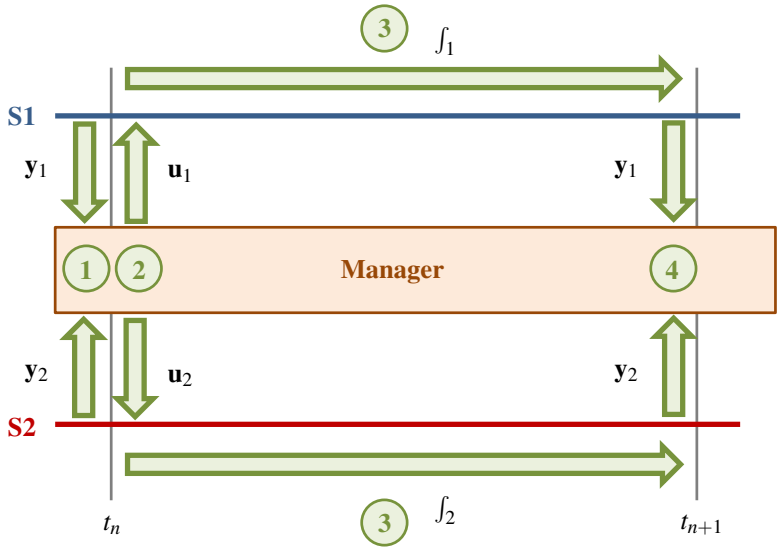


Figure 6: Diagram of a Jacobi coupling scheme.

Figure 6 shows a diagram of the Jacobi coupling scheme applied to the co-simulation of two subsystems $S1$ and $S2$. At time t_n both systems send their outputs $\mathbf{y}_1$ and $\mathbf{y}_2$ to the manager (step 1). The manager processes the information and sends the inputs $\mathbf{u}_1$ and $\mathbf{u}_2$ to the subsystems (step 2). Then, both subsystems advance in time until the next communication point at time t_{n+1} (step 3). This step can be performed in parallel, because each subsystem does not require any additional information from its environment until the macro-step is completed. The process begins again at time t_{n+1} with the subsystems sending their outputs to the manager at the new communication point (step 4).

The ***Gauss-Seidel scheme***, also called zigzag approach, executes the subsystems sequentially, so that the outputs of one of them obtained upon completion of a macro time-step are available when the other starts its integration. The scheme is illustrated in Fig. 7. At communication point t_n, one of the subsystems

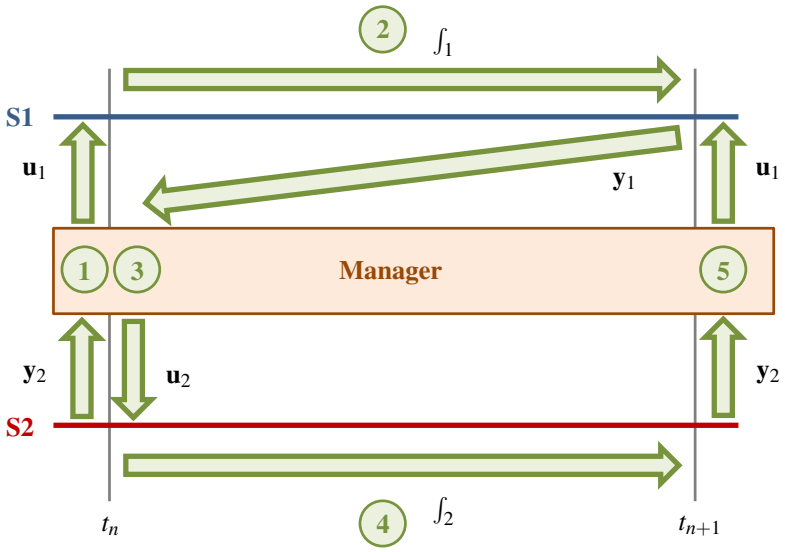


Figure 7: Diagram of a Gauss-Seidel coupling scheme.

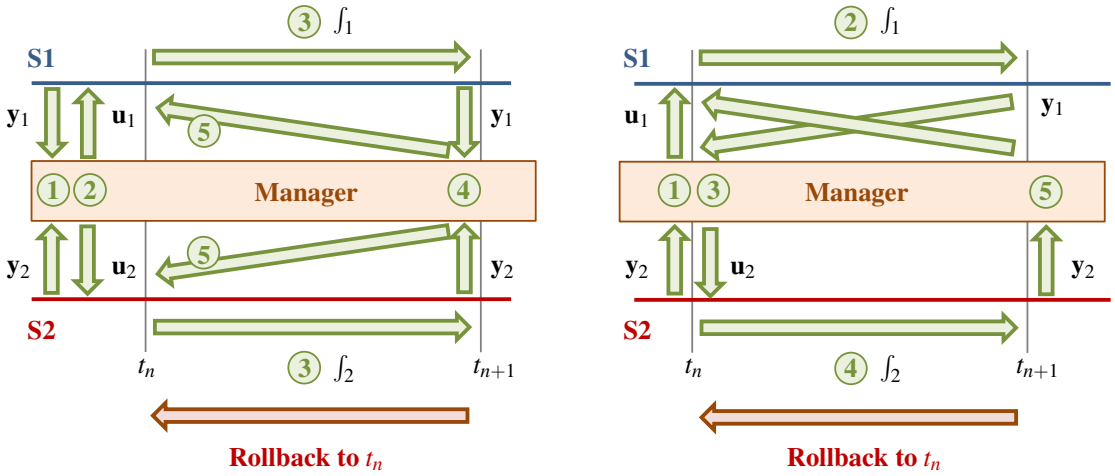
–$S2$, in this case– sends its outputs $\mathbf{y}_2$ to the manager, which processes the information and sends the corresponding inputs $\mathbf{u}_1$ to the other subsystem (step 1). Then, subsystem $S1$ proceeds with its integration until the next communication point, at time t_{n+1} (step 2) and evaluates its outputs $\mathbf{y}_2$ at this time. These and can now be used to evaluate (step 3) the inputs $\mathbf{u}_2$ that subsystem $S2$ will use during its integration from t_n to t_{n+1} (step 4). Finally, once the integration of $S2$ is complete, the procedure starts again for the next macro time-step when $S2$ sends its outputs and $S2$ receives its inputs (step 5).

In general, Gauss-Seidel schemes tend to be more stable than Jacobi ones; on the other hand, they hinder the parallel execution of the simulators for the different subsystems. The coupling scheme selection will depend on the efficiency, accuracy, and stability needs of each particular application.

2.3 Iterative and non-iterative co-simulation

In the coupling schemes described in Section 2.2, the time integration from t_n to t_{n+1} is performed only once in each subsystem. It is also possible, however, to retake either or both integration steps, using the now known subsystem outputs at time t_{n+1} to obtain a more accurate solution in a predictor-corrector fashion. This way, both Jacobi and Gauss-Seidel schemes can be turned into iterative coupling algorithms [30], as shown in Fig. 8. Rollback here refers to the restart of the integration steps in the subsystems, after their outputs at t_{n+1} have been obtained.

Iterative co-simulation schemes are also referred to as ***implicit coupling*** or ***waveform relaxation***. Less frequently, they are also called strong coupling, although this may lead to confusion with monolithic schemes. ***Non-iterative co-simulation schemes***, on the other hand, are also called ***explicit coupling***. They



(a) Jacobi.

(b) Gauss-Seidel.

Figure 8: Diagrams of iterative coupling schemes.

can also be labelled as weak coupling, although this term is often employed to denote co-simulation in general. In some publications, ***semi-implicit coupling*** means an iterative coupling scheme in which the iteration procedure is interrupted after a fixed number of repetitions of the macro step, as opposed to iterating until a certain convergence error below a given tolerance is attained.

In general, implicit schemes are more stable than explicit ones [18]. Iterating over the macro-step until convergence removes most of the problems associated with explicit co-simulation, such as the need to perform input extrapolation (see Section 2.5). On the other hand, implicit schemes cannot be used with subsystems that do not allow rollback –for instance real-time components– or in some applications that impose limitations on the computational resources that can be used.

2.4 Selection of coupling variables

In most cases, the selection of coupling variables for a given co-simulation application can be done in many different ways. Frequently, there are several possibilities to divide a system and represent the connection between the resulting subsystems. For instance, in the simulation of mechanical systems, the interaction between components is commonly represented by forces. However, sometimes the selection of other kinds of variables for the exchange of information may be more convenient.

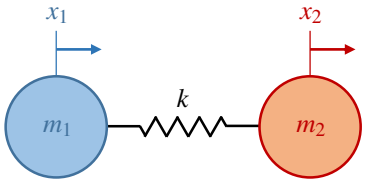


Figure 9: A two-mass mechanical system.

Consider the mechanical system composed of two masses, m_1 and m_2, connected by a spring with stiffness constant k, shown in Fig. 9. It is possible to co-simulate its dynamics dividing it into two subsystems, each containing one of the two masses, and defining their inputs and outputs in different ways.

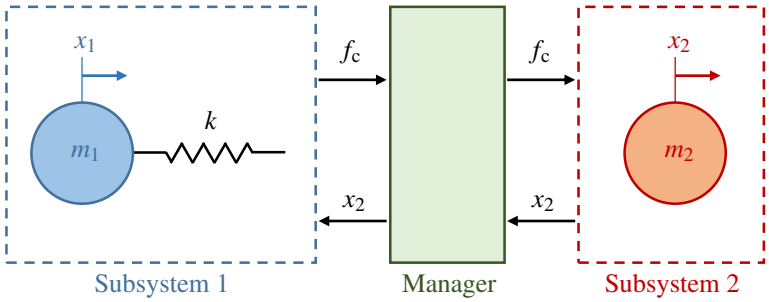


Figure 10: A two-mass mechanical system in a force-displacement co-simulation setup.

Figure 10 shows a setup in which subsystem 1 contains information about mass m_1 and the stiffness k of the spring. Subsystem 2 only contains the information about mass m_2. The equation of motion for subsystem 1 is

$$\ddot{x}_1 = \frac{k(x_2 - x_1)}{m_1} \tag{17}$$

and its output is the coupling force, $f_c = k(x_1 - x_2)$. On the other hand, for subsystem 2

$$\ddot{x}_2 = \frac{f_c}{m_2} \tag{18}$$

and its output is the position of the mass, x_2. Here, for the sake of simplicity, we are assuming that the manager does not modify at all the information contained in the coupling variables. The inputs required by each subsystem in Eqs. (17) and (18) are highlighted red. The arrangement in Fig. 10 is a ***force-displacement coupling***.

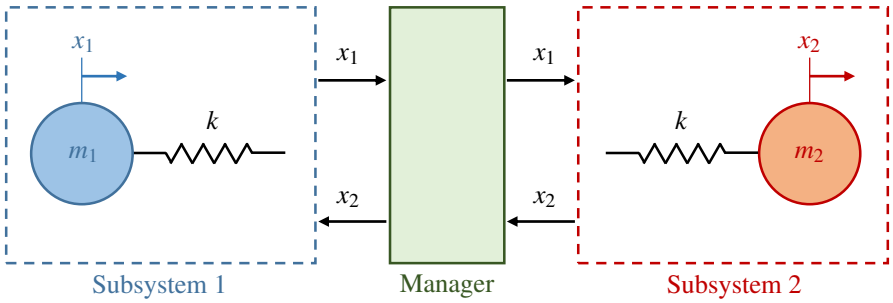


Figure 11: A two-mass mechanical system in a displacement-displacement co-simulation setup.

Other selections of coupling variables, however, can be used as well. Consider the arrangement shown in Fig. 11, a ***displacement-displacement coupling***. Now, the subsystems exchange their positions x_1 and

x_2 as output variables. Their dynamics equations are in this case

$$\ddot{x}_1 = \frac{k(x_2 - x_1)}{m_1}; \qquad \ddot{x}_2 = \frac{k(x_1 - x_2)}{m_2} \tag{19}$$

The expression of Eqs. (19) requires that information about the stiffness k of the spring is available for both subsystems. Additional selections of coupling variables are also possible, such as ***force-force coupling*** arrangements, in which the coupling force would be evaluated externally, e.g., by the co-simulation manager as in Fig. 12, and both subsystems would receive it as input. In this case, the dynamics equations would be

$$\ddot{x}_1 = \frac{-f_c}{m_1}; \qquad \ddot{x}_2 = \frac{f_c}{m_2} \tag{20}$$

and the coupling force would be computed by the manager as $f_c = k(x_1 - x_2)$. This approach requires that information regarding the stiffness properties of the coupling interface is available to the co-simulation orchestrator.

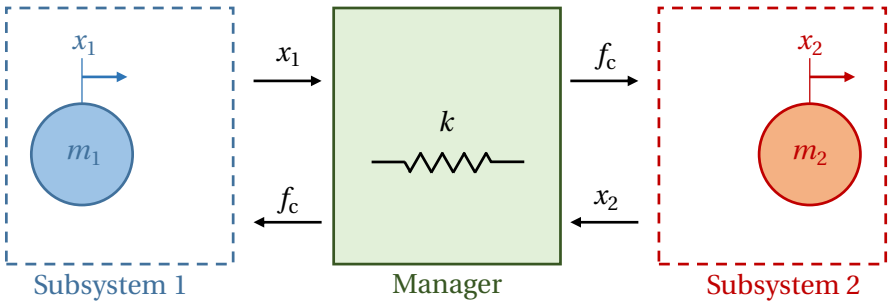


Figure 12: A two-mass mechanical system in a force-force co-simulation setup, where the coupling force is evaluated by the co-simulation manager.

It is difficult to make an all-encompassing classification of the existing options for the selection of coupling variables, because the subsystems coupled in co-simulation setups can represent a very wide variety of physical entities, and their inputs and outputs can be pretty much any imaginable physical quantity, from angular velocities and torques to temperatures and heat flows. However, it is possible to make some generally valid statements about variable selection.

Advantages and disadvantages of some coupling variable choices

Using a force-displacement approach –or an equivalent one, such as torque-angular speed– can be advantageous because it expresses the connection between subsystems in terms of a pair of coupling variables that provide information about the energy exchanged at the co-simulation interface. For instance, in the simple mechanical system shown in Fig. 10, the product $P = f_c \cdot \dot{x}_2$ is directly the power exchanged between the two subsystems through the coupling interface. Even if x_2 is the variable actually exchanged between

subsystems, it is possible to obtain the value of its derivative, $\dot{x}_2$, by differentiation of the inputs of subsystem 1. The evaluation of the power exchanged at the coupling interface is the basis for the development of indicators for co-simulation accuracy and correction methods to keep the numerical integration precise and stable, e.g., [28], [15]. These will be discussed in further detail in Section 3.1.

However, displacement-displacement coupling schemes can also be beneficial in certain cases. Let us consider the two-mass example in Fig. 9 and formulate the equations of its subsystems in state-space form

$$\begin{bmatrix} \dot{\mathbf{z}} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{z} \\ \mathbf{u} \end{bmatrix} \tag{21}$$

where $\mathbf{z}$ is the subsystem state, $\mathbf{y}$ includes the outputs, $\mathbf{u}$ is the subsystem inputs, and $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ are the state, input, output, and feedthrough matrices, respectively. If the co-simulation takes place according to a non-iterative Jacobi scheme, in which both subsystems are stepped forward in time with the same communication macro-step, the numerical integration process of a subsystem between communication points can be summarized in four stages, as shown in Fig. 13.

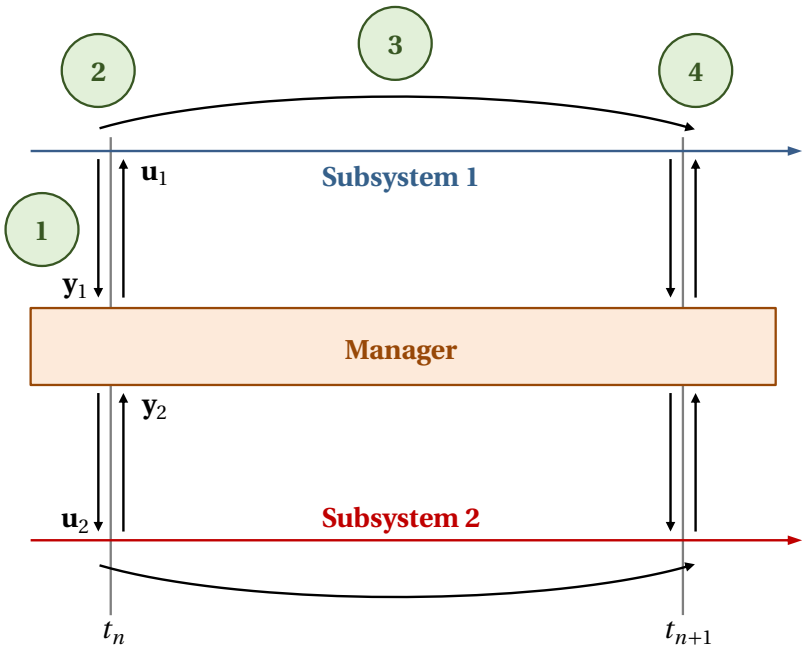


Figure 13: Stages in the integration of a subsystem between two communication points in a single-step Jacobi scheme

At the beginning of macro time step (t_n, t_{n+1}), both subsystem receive their inputs $\mathbf{u}_1$ and $\mathbf{u}_2$ from the manager (stage 1). Before, at the end of the previous macro step, the subsystems had sent their outputs $\mathbf{y}_1$ and $\mathbf{y}_2$ to the manager. The next steps for each subsystem are the evaluation of the derivatives of its state, $\dot{\mathbf{z}}$, (stage 2) and their integration to step forward in time the subsystem from t_n to t_{n+1} (stage 3). Once the subsystem has received its inputs at time t_n, $\mathbf{u}_n$, this is equivalent to the conventional numerical

integration of a dynamic system. The derivatives at time t_n are obtained as

$$\dot{\mathbf{z}}_n = \mathbf{A}\mathbf{z}_n + \mathbf{B}\mathbf{u}_n \tag{22}$$

which is an exact equation, because $\mathbf{z}$ and $\mathbf{u}$ are known at time t_n. The derivatives $\dot{\mathbf{z}}_n$ can now be used to integrate the state of the system until t_{n+1} to obtain $\mathbf{z}_{n+1}$. So far, the only numerical error in the process has been introduced by the time discretization of the system dynamics and the use of a numerical integrator during stage 3. At stage 4, the subsystem must evaluate its outputs at time t_{n+1} before sending them to the manager

$$\mathbf{y}_{n+1} = \mathbf{C}\mathbf{z}_{n+1} + \mathbf{D}\mathbf{u}_{n+1} \tag{23}$$

The problem is that $\mathbf{u}_{n+1}$ are not known yet. They will remain undetermined until the next macro step begins and the manager sends them to the subsystems. Usually, their value is extrapolated from the previously known values of the subsystem inputs, so Eq. (23) actually becomes

$$\mathbf{y}_{n+1} = \mathbf{C}\mathbf{z}_{n+1} + \mathbf{D}\tilde{\mathbf{u}}_{n+1} \tag{24}$$

where $\tilde{\mathbf{u}}_{n+1}$ is the approximated value of the inputs at time t_{n+1}. Equation (24) highlights that, besides the numerical integration, the need to extrapolate the inputs introduces an additional source of errors in the co-simulation process. However, in those cases in which the feedthrough matrix $\mathbf{D}$ is empty, the subsystem outputs can be evaluated without knowing the inputs $\mathbf{u}_{n+1}$

$$\mathbf{y}_{n+1} = \mathbf{C}\mathbf{z}_{n+1} \tag{25}$$

which would reduce again the total error of the co-simulation process to that of the numerical integrator. This explains the deterioration of error convergence properties observed in [3] for co-simulation arrangements that involve subsystems with $\mathbf{D} \neq \mathbf{0}$.

In a Jacobi scheme with a force-displacement coupling like the one in Fig. 10, the subsystem that returns force as an output (subsystem 1) needs to use Eq. (24), because the coupling force $f_c = k(x_1 - x_2)$ depends on input x_2. In this case, the feedthrough matrix $\mathbf{D}$ is not zero. However, if a displacement-displacement coupling is used, the outputs of both subsystems are just identical to their states. In this case, $\mathbf{D} = \mathbf{0}$ and there is no need to extrapolate subsystem inputs, because they can be calculated with Eq. (25). It can be shown that, in examples like those introduced in Sections A.1 and A.5, selecting the coupling variables such that $\mathbf{D} = \mathbf{0}$ results in better energy conservation and stability properties. On the other hand, such an approach usually requires that both subsystems have access to information about the physical properties of the coupling interface, which is not always possible. For example, both subsys-

tems in the two-mass problem would need to know the stiffness k of the coupling spring, as evidenced by Eqs. (19).

Coupling via algebraic constraints

Up to this point, only coupling through constitutive relations has been considered in this Section. The interface between subsystems has a certain flexibility, in the sense that the coupling variables are used to determine a term that will enter the differential equations of at least one of the subsystems. For example, in the case of the linear oscillator in Section 1.2, the interface between the two subsystems is compliant and can described by a spring.

It is also possible to use a rigid coupling between subsystems. Mathematically, this is done using algebraic constraints to characterize the interface. In this case, explicit co-simulation schemes cannot be directly used without modifications. Coupling via algebraic constraints requires either the use of implicit co-simulation methods or the knowledge of partial derivatives of the system states with respect to the coupling variables, as shown for instance in [31, 32]. In some cases, algebraic couplings can be replaced with compliant spring-damper systems, thus obtaining an approximation of the dynamics of the overall system. This approach, however, introduces artificial dynamics in the solution and can result in the need to decrease the macro step-size to obtain accurate enough results [19].

2.5 Extrapolation and interpolation of coupling variables

Explicit co-simulation schemes frequently need to have recourse to extrapolation methods to evaluate subsystem inputs. The Jacobi scheme diagram shown in Fig. 13 illustrates this issue. In step 4, subsystem 1 must evaluate its outputs at time $n+1$ without having access to the value of its inputs at this instant, $\mathbf{u}_{n+1}$, as confirmed by Eq. (23). An approximated value of the inputs, $\tilde{\mathbf{u}}_{n+1}$, must be used instead; several ways exist to determine this value.

Polynomial extrapolation is a popular way to approximate unknown input values in co-simulation applications, e.g., [22, 25]. ***Constant extrapolation***, also known as ***zero order hold (ZOH)***, is commonly used because of its simplicity. With ZOH, the unknown inputs at time $n+1$ are simply

$$\tilde{\mathbf{u}}_{n+1}^{\mathrm{ZOH}} = \mathbf{u}_n \tag{26}$$

Higher extrapolation orders, such as ***linear extrapolation*** or ***first order hold (FOH)*** and ***quadratic extrapolation*** or ***second order hold (SOH)*** are often used in an attempt to make the integration process

more stable [8], especially in multi-rate co-simulation environments (see Section 2.6). Higher order polynomials are expected to increase the coupling bandwidth [6] and accuracy, although they are also more sensitive to discontinuities in subsystem dynamics. The use of FOH to evaluate the inputs at time $n+1$ would result in the following approximated values

$$\tilde{\mathbf{u}}_{n+1}^{\mathrm{FOH}} = \mathbf{u}_n + (t_{n+1} - t_n)\frac{\mathbf{u}_n - \mathbf{u}_{n-1}}{t_n - t_{n-1}} \tag{27}$$

Similarly, SOH would require to adjust the coefficients of a second order polynomial to perform the extrapolation, using the known values of the inputs at time steps t_n, t_{n-1}, and t_{n-2}. Other methods, such as least square approximation [16] or polynomial approximations that result in smooth input evolution [9] have been proposed as well. Extrapolation methods do not need any information other than the previously received input values to operate; they are also generally easy to implement and can be used with almost any type of subsystem in a co-simulation setup, regardless of its nature or internal behaviour.

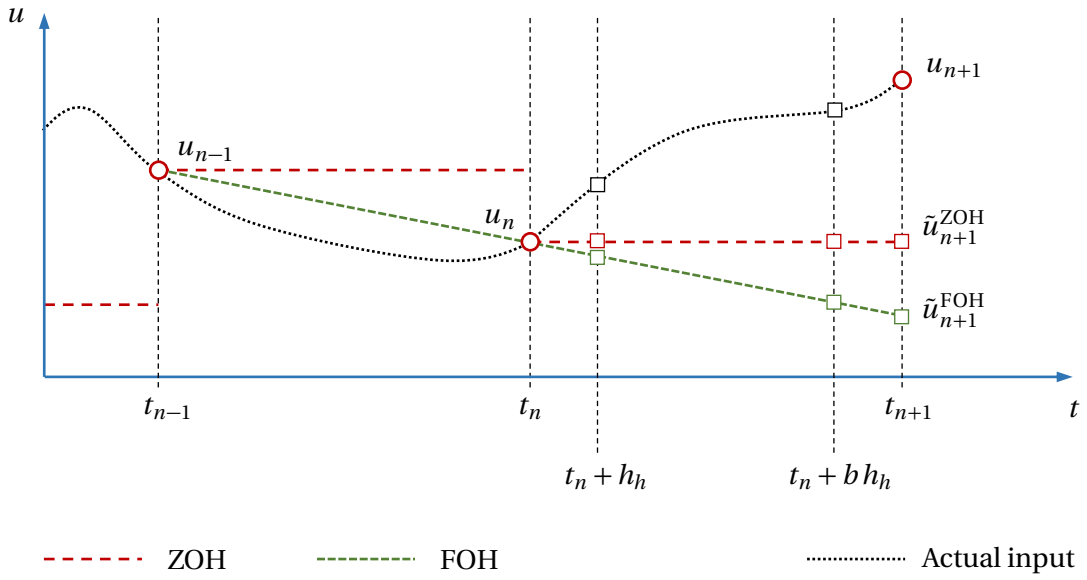


Figure 14: Effect of ZOH and FOH extrapolation on the prediction of subsystem inputs

It is important to note that all the above mentioned approximation and extrapolation methods are signal reconstruction techniques that build on previously received input values, but do not use information from the subsystem dynamics to predict future input values. As such, they may deliver unrealistic predictions, especially when applied to co-simulation environments subjected to sudden changes in the dynamics, e.g., impacts and discontinuities. Figure 14 highlights the fact that extrapolation may lead to inaccurate input prediction: the extrapolated values at time t_{n+1} may not follow the actual evolution of the input variable, and increasing the polynomial order does not necessarily improve the results. Along these lines, as noted in [16], the selection of the extrapolation method often has a critical and strongly case-dependent impact on the accuracy and robustness of the co-simulation process, and it is difficult to select the most favourable extrapolation method for a given application without a previous evaluation

of its behaviour [25]. This has led to the development of methods that adaptively select the polynomial degree to be used for extrapolation during runtime [5].

2.6 Time grids

The concept ***time grid*** denotes the time discretization obtained in each subsystem as a result of setting its internal integration step-size, and also those that result from establishing the communication intervals between the manager and the subsystems. Time grids can be equally spaced, if all their time intervals are of the same size, or not. Co-simulation time grids are ***matching grids*** if the communication intervals of all the subsystems are multiples of each other. In this document, only constant and matching grids are discussed; however, non-matching grids can also be found in co-simulation applications [16].

The concepts of single-rate and multirate co-simulation are related to the time grids used in the coupling scheme. In a ***single-rate*** co-simulation algorithm, all the subsystems use the same macro step-size H to communicate with the manager. In ***multirate*** co-simulation, conversely, at least two subsystems have different macro step-sizes. Using multirate schemes is advantageous sometimes when the co-simulated subsystems have very different time scales. In such cases, slow subsystems can be simulated using large integration steps while using a smaller step-size for subsystems with faster dynamics. This often results in a more efficient execution from a computational point of view: if small communication steps were to be used for every subsystem, this would increase the number of integration steps taken to solve the slow subsystem dynamics, consequently causing a computational overhead.

In spite of making it possible to decrease the overall computational effort required to integrate the dynamics, multirate co-simulation schemes have the disadvantage of needing input extrapolation in cases in which their single-rate equivalent would not.

Figure 15 shows a multirate co-simulation scheme in which subsystem 1 (SS1) and subsystem 2 (SS2) use macro step-sizes H_1 and $H_2 = H_1/2$. Subsystem 1 exchanges information with the manager at communication points t_n and t_{n+1}; subsystem 2, besides, has an intermediate communication point t_b. The integration of the dynamics of SS2 between t_n and t_{n+1} is performed in two consecutive integration steps: from t_n to t_b and from t_b to t_{n+1}. This multirate scheme decreases the number of integration steps required to solve the dynamics of SS1, compared to a single-rate scheme in which $H = H_2$. On the other hand, however, input extrapolation is likely to be required at time t_b in SS2. Let us assume that the dynamics of this subsystem can be expressed in state-space form as

$$\begin{bmatrix} \dot{\mathbf{x}}_2 \\ \mathbf{y}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{A}_2 & \mathbf{B}_2 \\ \mathbf{C}_2 & \mathbf{D}_2 \end{bmatrix} \begin{bmatrix} \mathbf{x}_2 \\ \mathbf{u}_2 \end{bmatrix} \tag{28}$$

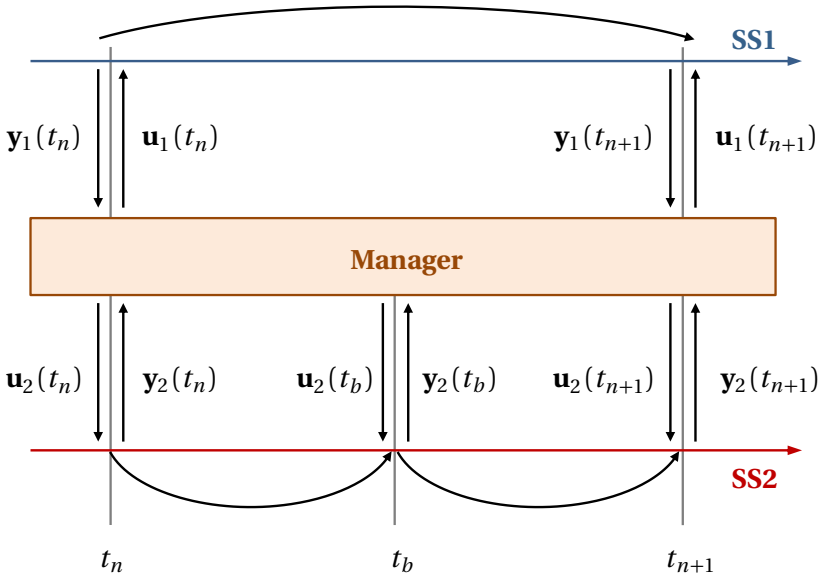


Figure 15: Example of explicit multi-rate co-simulation scheme

At time t_b, input $\mathbf{u}_2$ is required to evaluate the output y_2 if SS2 has direct feedthrough, i.e., $\mathbf{D}_2 \neq \mathbf{0}$. However, even in systems without direct feedthrough, $\mathbf{u}_2(t_b)$ is still required to evaluate $\dot{\mathbf{x}}_2(t_b)$, necessary in turn to complete the integration step between t_b and t_{n+1}. It must be stressed that subsystem 1 only needs to extrapolate its inputs if $\mathbf{D}_1 \neq \mathbf{0}$, but not to evaluate the derivative of its state.

In fact, this problem exists whenever the integration step-sizes within the subsystems do not match the macro step-size used to communicate with the co-simulation manager. Input extrapolation will be required to evaluate the state derivatives at intermediate times between communication points, except in the rather infrequent case in which $\mathbf{B} = \mathbf{0}$.

2.7 Real-time co-simulation

The term ***real-time (RT) co-simulation*** is used to describe those applications in which all computations and information transfers between two consecutive communication points t_n and t_{n+1} must be conducted in a real-world (***wall-clock time***) time span shorter than the length of the macro time step, H. The simulation outputs not only have to be correct: they also have to be delivered at the right time [23]. Fast code execution is essential for RT co-simulation, but so is predictability regarding the duration of the computation intervals.

RT co-simulation setups make it possible to interface simulated environments to physical components to deliver physical-virtual applications, such as test benches. ***Haptic*** simulators, for instance, can be regarded as a particular type of RT co-simulation setup, in which a human being specifies the desired motion of a part of a mechanical system in a virtual environment and receives the simulated force that should be experienced if the operation took place in the physical world. Other examples of RT co-simulation in-

clude HiL simulators, like testing microprocessors with vehicle ADAS (Advanced Driver-Assistance Systems) with a virtual model of the vehicle on which they will be mounted, and ***System-in-the-Loop*** (SITL) test benches, in which the physical component under test is interfaced to a simulated, virtual environment [1]. The latter is the case of physical-virtual test benches, which are gaining momentum in automotive applications, for instance test benches for e-powertrain components. Simulators that interact with real-world subsystems are very often subjected to RT execution constraints, because physical systems cannot "pause" or "slow down" their execution.

Besides the pointed out limitation regarding the available time for computations, RT co-simulation setups frequently present the following features:

- They use **constant communication macro step-sizes** H and **matching time grids**. Physical components are often sampled at constant rates that cannot be modified. Even though variable step integrators can still be used within the virtual subsystems, it is rather complicated to modify the macro step-size.
- **Explicit co-simulation schemes** are almost exclusively used. Physical components cannot perform rollback and re-take their "integration steps" between communication points. Among these, Jacobi schemes are the most frequent, although Gauss-Seidel algorithms can also be used.

[1] The term ***hybrid simulation*** is often used to refer to the integration of physical testing and numerical simulations. Please note that ***hybrid co-simulation*** has a different meaning, as noted in Section 2.1.

3 Non-iterative co-simulation

As highlighted in Section 2.7, many applications of industrial interest require the use of non-iterative (explicit) co-simulation. The following points must be taken into consideration in these cases:

- As pointed out in Section 2.3, explicit co-simulation schemes are prone to become unstable, especially when one or more subsystems have direct feed-through. This can be alleviated with a proper selection of coupling variables, see Section 2.4.
- It is possible to apply correction methods to enhance simulation stability. Besides conventional polynomial extrapolation, energy-based and frequency analysis methods have been employed in the literature.
- Rollback is not possible and macro steps cannot be re-taken in the majority of cases.
- Multirate co-simulation schemes can be used.
- In RT applications, the additional requirement that the communication macro step-size H is kept constant is often enforced. Besides, all the computations required to complete a macro time-step, including communication overheads, must be completed in less time than the duration of the macro step, H.

A common concern in applications that employ explicit co-simulation is being able to predict when the numerical integration will become unstable or unreliable. This is critical for RT applications that include physical components, as unstable behaviour can develop into a hazard for the machinery and also people. A second desirable ability is being able to correct these instabilities and inaccuracies once they are detected.

3.1 Error monitoring in explicit co-simulation

Ideally, error indicators in non-iterative co-simulation should be derived exclusively from the information contained in the coupling variables exchanged between the subsystems and the co-simulation manager; in most cases, it cannot be assumed that the subsystems internals are known.

The ***residual power*** indicator δP was introduced by Sadjina in [28] to quantify the energy error caused by the time-discrete exchange of coupling variables at the co-simulation interface. Consider a co-simulation setup with two subsystems 1 and 2 that exchange energy with the rest of the co-simulation environment through their coupling interfaces. In theory, the total sum of the energy that flows through the co-simulation interface should be zero, because energy is neither generated nor removed there. As a con-

sequence, at any time t, the total power exchanged between the subsystems through the co-simulation interface should amount to zero

$$P_1 + P_2 = 0 \tag{29}$$

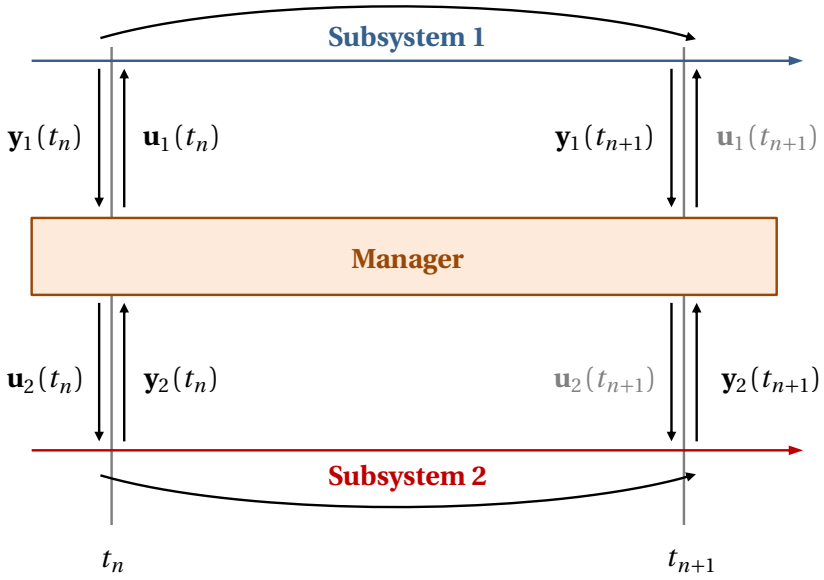


Figure 16: Exchange of coupling variables during an explicit co-simulation macro-step

In practice, however, this is not the case, because each subsystem evaluates its interface power in a different way. Let us assume that the product of the coupling variables exchanged at the interface has power units, e.g., as in a force-velocity coupling. At the end of a macro-step like the one shown in Fig. 16, at time t_{n+1}, subsystem 1 will evaluate the power that it exchanges through the interface as

$$P_1(t_{n+1}) = -(\tilde{\mathbf{u}}_1(t_{n+1}))^{\mathrm{T}}\,\mathbf{y}_1(t_{n+1}) \tag{30}$$

where notation $\tilde{\mathbf{u}}_1$ highlights the fact that inputs $\mathbf{u}_1$ at time t_{n+1} are unknown for subsystem 1 and must be approximated somehow, e.g., by means of polynomial extrapolation, to complete the evaluation of $\mathbf{y}_1(t_{n+1})$. Similarly, for subsystem 2

$$P_2(t_{n+1}) = -(\tilde{\mathbf{u}}_2(t_{n+1}))^{\mathrm{T}}\,\mathbf{y}_2(t_{n+1}) \tag{31}$$

The addition of the two power values in Eqs. (30) and (31) is no longer zero

$$\delta P = -(P_1 + P_2) \neq 0 \tag{32}$$

and this deviation from zero becomes the residual power, which indicates the co-simulation quality. The closer the residual is to zero, the more accurate the co-simulation is. The integral over a given time interval

of δP is the ***residual energy***

$$\delta E(t_{n+1}) = \int_{t_n}^{t_{n+1}} \delta P(t)\,\mathrm{d}t \tag{33}$$

and both quantities, δP and δE, can be used as an indicator of the coupling error caused by the time-discrete interface [25, 26]. It is noteworthy that these indicators were originally defined assuming that every subsystem extrapolated its inputs using ZOH extrapolation.

If the coupling variables include information that makes it possible to determine the overall energy of the system, then a more precise energy balance can be established [15]. This can be used to monitor energy errors at the coupling interface. This method, though, requires a computational model of each subsystem involved in the co-simulation; if physical components exist, e.g., in a SiTL test bench, then an all-encompassing energy balance cannot be evaluated, unless means exist to estimate the energy of the physical subsystem components. This would require either precise and comprehensive sensor readings or the use of digital twins of these components.

4 Implementation

Implementation is a crucial step in co-simulation applications. The schemes and correction methods described in the previous Sections need to be coded in an effective, robust, and easy to use way to be used to their full potential. Conversely, a poor implementation may overweight the advantages of a given co-simulation approach, rendering it useless for its intended application.

In general, a certain co-simulation scheme can be implemented using a wide range of implementation techniques, which will have an impact on the overall performance and stability of the numerical integration. The different solvers in a co-simulation environment may be executed in the same computer or over a network, employing homogeneous or heterogeneous computer architectures. In certain co-simulation applications, some components are physical, non-simulated elements, that interact with the virtual ones represented by the simulators. The implementation techniques to be used will need to address specific requirements depending on the configuration of the simulation environment.

In all cases, a necessary requirement to couple different solvers in a coordinated integration process is that the exchange of information between them takes place through a protocol that all of them can understand and use to communicate with their environment.

4.1 The FMI standard

When coupling subsystems in a co-simulation environment, it is desirable that the communication between solvers takes place via an unambiguous interface with minimal specifications and impact on the internals of the subsystems. This interface must be general enough to be used with subsystems of all natures and physical behaviours, so that it can be widely accepted both in academic and industrial environments.

The ***Functional Mock-up Interface (FMI)*** was conceived to address these needs: it is a tool independent standard to support both model exchange and co-simulation of dynamic models using a combination of XML files and compiled C code [1]. Its first version was published in 2010; FMI 2.0 was released in 2014. The development of the standard was started by Daimler AG to ease the exchange of simulation models between suppliers and manufacturers, and is currently fostered by the Modelica Association [2].

5 Selection of co-simulation configurations

As shown in Section 2, many different options to couple subsystems in a co-simulation setup exist. Selecting the right ones for a given application is not straightforward; it often requires prior knowledge of the subsystems behaviour or some trial and error parameter tuning [25]; moreover, generally valid guidelines are difficult to arrive at, and exceptions to the general rules exist in many occasions. When setting up a co-simulation environment, it is necessary to answer the following questions.

- **How to define the subsystems?**

  In some cases, it is straightforward to draw the boundaries between subsystems in a co-simulation setup. When the dynamics of the parts of an engineering application are governed by dissimilar equations, it is natural to assign different solvers to each of them. Often, however, several splitting options are available. In general, it is advisable to avoid coupling subsystems via algebraic constraints, especially in explicit schemes, as explained in Section 2.4. Coupling subsystems by means of a constitutive relation, e.g., a spring-damper element, works better from a stability and solvability standpoint, especially for moderate and low values of the coupling stiffness.

- **Continuous or event-driven co-simulation?**

  This depends on the nature of the subsystems in the setup. If there is a mix of continuous and event-driven subsystems is used, the user must select an approach and adapt the dynamics of the subsystems that do not comply with the selected paradigm. Some authors affirm that a discrete-event framework may just be enough to appropriately manage any co-simulation environment [20].

- **Implicit or explicit co-simulation?**

  Implicit co-simulation schemes are more stable than their explicit counterparts [18]. They are often more accurate too. However, they impose on the subsystems the need to be able to repeat their integration process between communication points, which is not always possible. Moreover, iterative procedures can be time consuming and thus incompatible with the RT requirements of some applications.

- **Jacobi or Gauss-Seidel schemes?**

  Jacobi schemes enable the parallelization of subsystem integration between communication points. Gauss-Seidel ones, on the other hand, require the sequential execution of the integration procedures.

  While implicit co-simulation algorithms often lead to the convergence of Jacobi and Gauss-Seidel schemes to the same solution, this is not the case with explicit coupling schemes. Regarding co-simulation stability and accuracy, the performance of explicit Jacobi schemes can be severely degraded if one or more subsystems present direct feedthrough. In these cases, correction algorithms

at the coupling interface are necessary to prevent the numerical integration from going unstable. Gauss-Seidel approaches seem to benefit from the stabilizing properties of the sequential execution of the integration of the subsystems, at the cost of introducing small variations in the system energy as a consequence.

- **Extrapolation method**

  When input extrapolation is required, the selected approach has a noticeable impact on the accuracy and stability of the results. Studies exist that show that higher order polynomials result in a higher usable bandwidth of the coupling signal when compared to ZOH extrapolation [6, 7]. However, the actual performance of an extrapolation approach is case-dependent and also modified by the selection of the communication step-size; extrapolation orders are often chosen following trial-and-error procedures or adjusted during operation based on system behaviour [5, 25]. In most cases, prior knowledge of the simulated system is required to achieve effective co-simulation configurations.

- **Selection of coupling variables**

  The way in which the application under study is split into subsystems has an important impact on co-simulation behaviour [26]. As a general guideline, whenever possible these should be selected to avoid having subsystems with direct feedthrough.

- **Single-rate or multirate co-simulation?**

  The use of multirate schemes can be used to improve computational efficiency. However, as shown in Section 2.6, this may work against the accuracy of the results, as it may introduce the need to perform input extrapolation in subsystems that do not have to use it in single-rate schemes.

References

[1] FMI - Functional Mock-up Interface (2019). URL https://fmi-standard.org/

[2] Modelica (2019). URL https://www.modelica.org/

[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. Archive of Mechanical Engineering **60**(1), 75–94 (2013). DOI 10.2478/meceng-2013-0005

[4] Bauchau, O.A.: Flexible Multibody Dynamics. Springer, Dordrecht, The Netherlands (2011). DOI 10.1007/978-94-007-0335-3

[5] Ben Khaled-El Feki, A., Duval, L., Faure, C., Simon, D., Ben Gaid, M.: CHOPtrey: contextual online polynomial extrapolation for enhanced multi-core co-simulation of complex systems. Simulation **93**(3), 185–200 (2017). DOI 10.1177/0037549716684026

[6] Benedikt, M., Watzenig, D., Hofer, A.: Modelling and analysis of the non-iterative coupling process for co-simulation. Mathematical and Computer Modelling of Dynamical Systems **19**(5), 451–470 (2013). DOI 10.1080/13873954.2013.784340

[7] Benedikt, M., Watzenig, D., Zehetner, J., Hofer, A.: Macro-step-size selection and monitoring of the coupling errof for weak coupled subsystems in the frequency-domain. In: International Conference on Computational Methods for Coupled Problems in Science and Engineering - Ibiza, Spain (2013)

[8] Burger, M., Steidel, S.: Local extrapolation and linear-implicit stabilization in a parallel coupling scheme. In: IUTAM Symposium on Solver-Coupling and Co-Simulation, pp. 43–56. Springer International Publishing (2019). DOI 10.1007/978-3-030-14883-6_3

[9] Busch, M.: Continuous approximation techniques for co-simulation methods: Analysis of numerical stability and local error. ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik **96**(9), 1061–1081 (2016). DOI 10.1002/zamm.201500196

[10] Cardona, A., Geradin, M.: Modeling of a hydraulic actuator in flexible machine dynamics simulation. Mechanism and Machine Theory **25**(2), 193–207 (1990). DOI 10.1016/0094-114X(90)90121-Y

[11] Dopico, D., Luaces, A., González, M., Cuadrado, J.: Dealing with multiple contacts in a human-in-the-loop application. Multibody System Dynamics **25**(2), 167–183 (2011). DOI 10.1007/s11044-010-9230-y

[12] García de Jalón, J.: Twenty-five years of natural coordinates. Multibody System Dynamics **18**(1), 15–33 (2007). DOI 10.1007/s11044-007-9068-0

[13] García de Jalón, J., Bayo, E.: Kinematic and Dynamic Simulation of Multibody Systems. The Real–Time Challenge. Springer–Verlag, New York, USA (1994). DOI 10.1007/978-1-4612-2600-0

[14] Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: A survey. ACM Computing Surveys **51**(3), 49:1–49:33 (2018). DOI 10.1145/3179993

[15] González, F., Arbatani, S., Mohtat, A., Kövecses, J.: Energy-leak monitoring and correction to enhance stability in the co-simulation of mechanical systems. Mechanism and Machine Theory **131**, 172–188 (2019). DOI 10.1016/j.mechmachtheory.2018.09.007

[16] González, F., Naya, M.Á., Luaces, A., González, M.: On the effect of multirate co-simulation techniques in the efficiency and accuracy of multibody system dynamics. Multibody System Dynamics **25**(4), 461–483 (2011). DOI 10.1007/s11044-010-9234-7

[17] Hafner, I., Popper, N.: On the terminology and structuring of co-simulation methods. In: Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools - EOOLT'17, pp. 67–76. Weßling, Germany (2017). DOI 10.1145/3158191.3158203

[18] Kübler, R., Schiehlen, W.: Modular simulation in multibody system dynamics. Multibody System Dynamics **4**(2), 107–127 (2000). DOI 10.1023/A:1009810318420

[19] Lacoursière, C., Härdin, T.: FMI Go! A simulation runtime environment with a client server architecture over multiple protocols. In: Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017 (2017). DOI 10.3384/ecp17132653

[20] Lacoursière, C., Sjöström, T.: A non-smooth event-driven, accurate, adaptive time stepper for simulating switching electronic circuits. Tech. Rep. UMINF 16.15, Umeå University (2014)

[21] Naya, M.A., Cuadrado, J., Dopico, D., Lugrís, U.: An efficient unified method for the combined simulation of multibody and hydraulic dynamics: Comparison with simplified and co-integration approaches. Archive of Mechanical Engineering **58**(2), 223–243 (2011). DOI 10.2478/v10180-011-0016-4

[22] Oberschelp, O., Vöcking, H.: Multirate simulation of mechatronic systems. In: Proceedings of the IEEE International Conference on Mechatronics, ICM'04. Istanbul, Turkey (2004). DOI 10.1109/icmech.2004.1364473

[23] Pastorino, R., Cosco, F., Naets, F., Desmet, W., Cuadrado, J.: Hard real-time multibody simulations using ARM-based embedded systems. Multibody System Dynamics **1**(37), 127–143 (2016). DOI 10.1007/s11044-016-9504-0

[24] Peiret, A., González, F., Kövecses, J., Teichmann, M.: Multibody system dynamics interface modelling for stable multirate co-simulation of multiphysics systems. Mechanism and Machine Theory **127**, 52–72 (2018). DOI 10.1016/j.mechmachtheory.2018.04.016

[25] Rahikainen, J., González, F., Naya, M.Á.: An automated methodology to select functional co-simulation configurations. Multibody System Dynamics **48**(1), 79–103 (2020). DOI 10.1007/s11044-019-09696-y

[26] Rahikainen, J., González, F., Naya, M.Á., Sopanen, J., Mikkola, A.: On the cosimulation of multibody systems and hydraulic dynamics. Multibody System Dynamics **50**(2), 143–167 (2020). DOI 10.1007/s11044-020-09727-z

[27] Rahikainen, J., Mikkola, A., Sopanen, J., Gerstmayr, J.: Combined semi-recursive formulation and lumped fluid method for monolithic simulation of multibody and hydraulic dynamics. Multibody System Dynamics **44**(3), 293–311 (2018). DOI 10.1007/s11044-018-9631-x

[28] Sadjina, S., Kyllingstad, L.T., Skjong, S., Pedersen, E.: Energy conservation and power bonds in co-simulations: non-iterative adaptive step size control and error estimation. Engineering with Computers **33**(3), 607–620 (2017). DOI 10.1007/s00366-016-0492-8

[29] Samin, J.C., Brüls, O., Collard, J.F., Sass, L., Fisette, P.: Multiphysics modeling and optimization of mechatronic multibody systems. Multibody System Dynamics **18**(3), 345–373 (2007). DOI 10.1007/s11044-007-9076-0

[30] Schweizer, B., Li, P., Lu, D.: Explicit and implicit cosimulation methods: Stability and convergence analysis for different solver coupling approaches. Journal of Computational and Nonlinear Dynamics **10**(5), 051,007 (2015). DOI 10.1115/1.4028503

[31] Schweizer, B., Lu, D.: Semi-implicit co-simulation approach for solver coupling. Archive of Applied Mechanics **84**(12), 1739–1769 (2014). DOI 10.1007/s00419-014-0883-5

[32] Schweizer, B., Lu, D.: Stabilized index-2 co-simulation approach for solver coupling with algebraic constraints. Multibody System Dynamics **34**(2), 129–161 (2015). DOI 10.1007/s11044-014-9422-y

[33] Stuart, A.M., Humphries, A.R.: Dynamical Systems and Numerical Analysis. Cambridge University Press (1998)