Politecnico di Milano

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING
Master of Science – Aerospace Engineering

Thesis Title

Supervisor
Title Name SURMANE

Co-Supervisor
Title Name SURNAME

Candidate
Claudio CACCIA – 820091

Academic Year 2019 – 2020

Acknowledgements

Thanks

Abstract

This thesis describes bla bla...

Keywords— one, two, three, four

Sommario

In questa tesi bla bla...

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Fluid-Structure Interaction (FSI) describes the mutual interaction between a moving or deformable object and a fluid in contact with it, surrounding or internal. It is present in various forms both in nature and in man-made systems: a leaf fluttering in the wind, water flowing underground or blood pumping in an artery are typical examples of fluid-structure interaction in nature. FSI occurs in engineered systems when modeling the behavior of turbomachinery, the flight characteristics of an aircraft, or the interaction of a building with the wind, just to name a few examples.

All the aforementioned problems go under the same category of FSI, even if the nature of the interaction between the solid and fluid is different. Specifically, the intensity of the exchanged quantities and the effect in the fluid and solid domains vary among different problems.

There can be multiple ways to classify FSI problems, based on the flow physics and on the behavior of the body. Incompressible flow assumption is always made for liquid-solid interaction, while both compressible and incompressible flow assumptions are made when a gas interacts with a solid, depending on the flow properties and the kind of simulation. The main application of air-solid interaction considers the determination of aerodynamic forces on structures such as aircraft wings, which is often referred to as *aeroelasticity*. Dynamic aeroelasticity is the topic that normally investigates the interaction between aerodynamic, elastic and inertial forces. Aerodynamic *flutter* (i.e. the dynamic instability of an elastic structure in a fluid flow) is one of the severe consequences of aerodynamic forces. It is responsible for destructive effects in structures and a significant example of FSI problem.

The subject may also be classified considering the behavior of the structure interacting with the fluid: a solid can be assumed rigid or deforming because of the fluid forces. Examples where rigid body assumption may be used include internal combustion engines, turbines, ships and offshore platforms. The rigid body–fluid interaction problem is simpler to some extent, nevertheless the dynamics of rigid body motion requires a solution that reflects the fluid forces. Within the deformable body–fluid interaction, the nature of the deforming body may vary from very simple linear elastic models in small strain to highly complex nonlinear deformations of inelastic materials. Examples of deforming body–fluid interaction include aeroelasticity, biomedical applications and poroelasticity.

The interaction between fluid (incompressible or compressible) and solid (rigid or deformable) can be *strong* or *weak*, depending on how much a change in one domain influences the other. An example of weakly coupled problem is aeroelasticity at high Reynolds number, while incompressible flow often leads to strongly coupled problems. This distinction can lead to different solution strategies, as briefly described below.

Physical models aren't the only way in which FSI problems can be classified. The solution

procedure employed plays a key role in building models and algorithms to solve this kind of problems. The two main approaches are: the *monolithic approach* in which both fluid and solid are treated as one single system and the *partitioned approach* in which fluid and solid are considered as two separated systems coupled only through an interface. This latter approach is often preferred when building new solution procedures as it allows to use solvers that have been already developed, tested and optimized for a specific domain. The solvers only need to be linked to a third component, which takes care of all the interaction aspects.

The partitioned approach can be further classified considering the coupling between the fluid and solid: the solution may be carried out using a *weakly coupled approach*, in which the two solvers advance without synchronization, or a *strongly coupled approach*, in which the solution for all the physics must be synchronized at every time step. Although the weakly coupled approach is used in some aerodynamic applications, it is seldom used in other areas due to instability issues. A strongly coupled approach is generally preferred, even though this leads to more complex coupling procedures at the interface between fluid and solid.

This work describes the implementation and the validation of what is called an *adapter*, that is the "glue-code" needed to interface a solver to a coupling software library, thus adopting a *partitioned approach* to solve FSI problems. The *adapter* presented here connects the software code *MultiBody Dynamics analysis software (MBDyn)* to the multiphysics coupling library *precise Code Interaction Coupling Environment (preCICE)*.

Interfacing MBDyn with preCICE has multiple advantages: on one side it extends the capabilities of MBDyn to be used in FSI simulations by connecting it with a software library which has been already connected to widely used CFD solvers; on the other side, it allows to describe and simulate FSI problems with a suite of lumped, 1D and 2D elements (i.e. rigid bodies, *beams*, *membranes*, *shells*, etc.) decoupling the shape of the object (the interface with the fluid) from its structural properties, which can be described by different models and constitutive laws.

The thesis is structured as follows:

- Section 2 introduces the reader to FSI problems and their complexity, with particular attention to the physical description of the fluid and solid domains and the interface.
- Section 3 focuses on numerical methods, describing how to computationally deal with these kind of problems: details regarding the different coupling approaches are given here.
- Section 4 explains the features of preCICE that the adapter needs to support and gives a short introduction to MBDyn, explaining the main functionalities of interest.
- Section 5 presents the adapter developed in this work, its most important features and how to configure a FSI simulation with it.
- Section 6 describes the successful validation of the adapter, the comparison of the results with some well-known benchmarks and an example of real world application.
- Section 7 summarizes the the findings and outcome of this work and gives an outlook to future work on this topic.
- Finally XX appendices give further information on...

Chapter 2

Physical aspects of Fluid-Structure Interaction problems

Dynamic models of solids or fluids aim at describing the evolution of an initial configuration through time. Structural mechanics and fluid dynamics use different perspectives when describing the motion of respectively a solid or a fluid particle. When dealing with FSI problems the two approaches need to be combined in order to obtain a suitable description of the two domains and their interface: this aspect is treated in 2.1.

As outlined in the introduction, the fluid and the solid domain of a FSI problem might be described by means of many different models: some of them are outlined in section 2.2. *Dimensional analysis* and the use of dimensionless numbers is a powerful tool used to classify fluid dynamics problems: some of the principles used there can be applied to FSI problems in order to classify them: this can help define and classify FSI problems, as described in section 2.3.

2.1 Description of motion

In a FSI model, the fluid in motion deforms the solid because of the forces exerted to the structure. The change in the shape of the solid modifies the fluid domain, causing a different flow behavior. For this reason it is necessary to describe formally the kinematics and the dynamics of the whole process. Classical continuum mechanics considers the motion of particles by means of two different perspectives [2]: the *Eulerian description*, briefly described in section 2.1.1, and the *Lagrangian description*, outlined in section 2.1.2. Those two perspectives are typically combined into the *arbitrary Lagrangian-Eulerian (ALE)* method, described in section 2.1.3.

2.1.1 Eulerian perspective

The *Eulerian perspective* observes the change of quantities of interest (e.g. density, velocity, pressure) at spatially fixed locations. In other words: the observer does not vary the point of view during different time steps. Thus, quantities can be expressed as functions of time at fixed locations. This is represented by the following notation:

$$\Theta = \tilde{\Theta}(x, y, z, t) \tag{2.1}$$

where Θ is a quantity of interest and $\tilde{\Theta}$ denotes the same quantity form an Eulerian point of view; (x, y, z) represent a fixed location in the euclidean space.

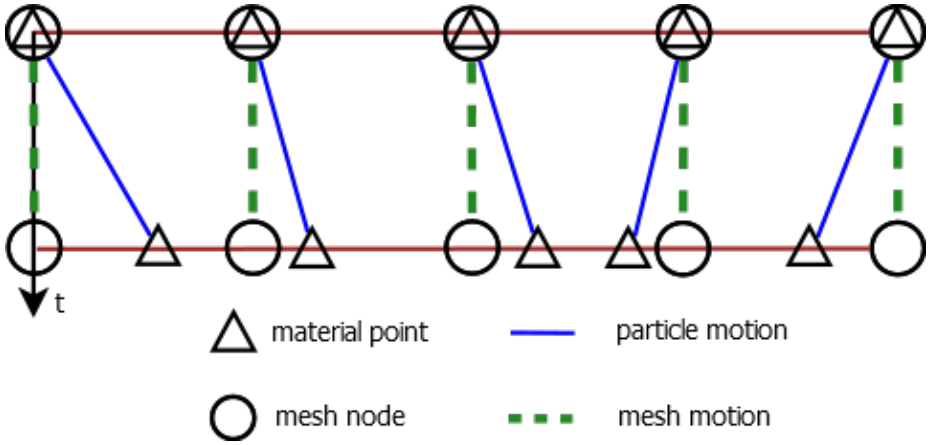


Figure 2.1. Eulerian perspective

A computational mesh can be interpreted as a number of observers distributed across the domain of interest and connected to each other in order to form a a grid with nodes. If the particles of the domain move, a purely euclidean mesh does not move and the position other nodes remain fixed at any instance of time [3]. This behavior is represented in Figure 2.1 (adapted from [3]). The mesh is independent of particles movement, resulting in a convenient choice for Computational Fluid Dynamics (CFD) problems, where fluid flows throughout the whole computational domain. Within this approach, proper mesh refinement is crucial for computational accuracy as it defines to what extent small scale movement can be modeled and resolved [4].

2.1.2 Lagrangian perspective

A *lagrangian observer* focuses on a single particles and follows it throughout the motion, as depicted in Figure 2.2. Changes in the quantities of interest are observed at different spatial locations.

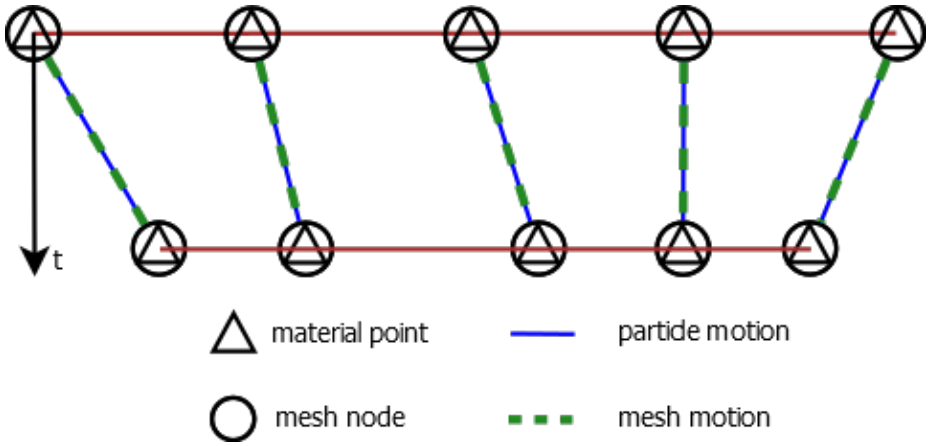


Figure 2.2. Lagrangian perspective

The motion of the particle and the other quantities of interest can be described by reference coordinates (or *material coordinates*) in Euclidean space (X, Y, Z), uniquely identifying the observed particle at a reference configuration [5]. Usually $t = 0$ is chosen as reference but this

is not mandatory. The Lagrangian observer only registers changes concerning one specific particle as time advances. Thus, quantities of interest can be described as:

$$\Theta = \hat{\Theta}(X, Y, Z, t) \tag{2.2}$$

In contrast to the Eulerian perspective (Equation 2.1), the obtained information is strictly limited to a single material particle (implied by the usage of the capital reference coordinate variables). Information about a fixed point in space is not directly available and no convective fluxes appear in a Lagrangian description.

This perspective is again translated into computational meshes: at a reference instance of time, mesh nodes are attached to material particles. As these move, the mesh nodes move with them causing the mesh to deform. Figure 2.2 describes the situation. The mesh nodes always coincide with their respective particles.

In this situation large-scale and irregular motions and more importantly deformation lead to distortions of the computational mesh, which yields smaller accuracy in simulations requiring to apply techniques to keep the desired accuracy [6].

Lagrangian perspective is the usual method of choice for Computational Solid Mechanics (CSM) simulations.

Eulerian and Lagrangian descriptions are related [7]. A mapping between them can described by the *motion* function ϕ such that:

$$\vec{x}(t) = \phi(\vec{X}, t) \tag{2.3}$$

Equation 2.3 tells that he Eulerian, spatial position $\vec{x}$ of a particle at time t is the mapping of the particle at its reference configuration $\vec{X}$: the mapping must be bijective.

2.1.3 ALE method

As outlined above, CSM and CFD problems adopt different perspectives. The ALE approach, a combination of the two points of view, is used for FSI problems. As the name implies, an ALE observer moves arbitrarily with respect to a specific material particle. Figure 2.3 depicts such a situation.

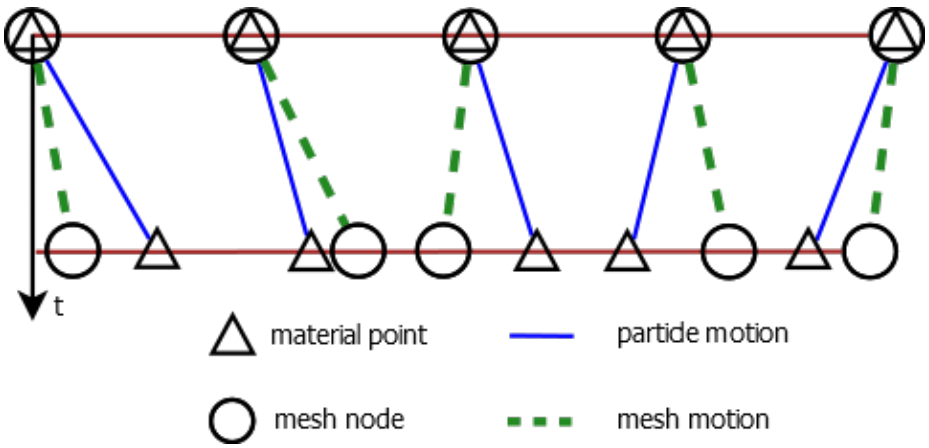
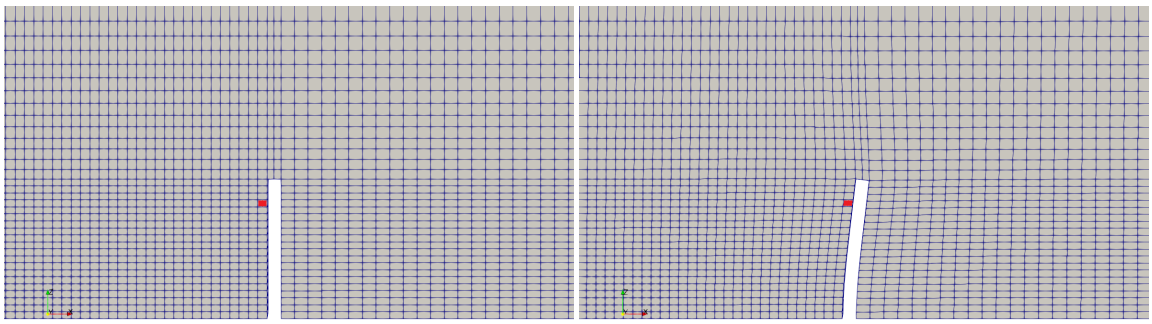


Figure 2.3. ALE perspective

When dealing with computational meshes, an ALE mesh is considered as it can move almost arbitrarily with respect to the motion of the underlying particles, as shown in Figure 2.4. The only constraint is that node movements should not distort the mesh too much

as this leads to computational inaccuracy. Many algorithm exist to implement suitable quality criteria and keep the mesh motion reasonable and to allow the nodes to follow moving particles up to a certain extent [8].

Since the mesh motion and material particle motion are not directly linked, a new unknown is introduced: the relative movement between the ALE mesh and the material domain. This approach is particularly useful in FSI problems: fluid and solid must follow the moving interface between them for physical reasons. Since the solid domain is usually described in a Lagrangian perspective, the solid mesh is kept attached to the FSI interface. However, also the fluid domain must deform to avoid formation of gaps between the meshes. Therefore, in ALE methods the fluid mesh nodes at the interface move with it. Fluid mesh nodes follow the fluid particles sticking to the interface (for viscous flows), while the rest of the fluid mesh is allowed to move in such way that mesh distortions are kept minimal, to preserve computational accuracy [9].



(a) undistorted mesh (b) distorted mesh

Figure 2.4. ALE mesh

2.2 Domains and interface

Fluid-structure interaction implies that the overall model is determined by models defining the fluid behavior and the solid behavior, briefly described in sections 2.2.2 and 2.2.1. A short overview of beam models is given in section 2.2.3 as it is relevant for the model developed in this work. Finally a formal definition of the interface is given in section 2.2.4, as it is necessary to define suitable coupling conditions at the common boundary of the solid and the fluid.

2.2.1 Fluid domain

An exhaustive description of all possible fluid models is far beyond the scope of this work. A quite general model is the viscous compressible one described by the Navier Stokes Equations (NSE).

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0 \tag{2.4a}$$

$$\frac{\partial}{\partial t}(\rho \vec{v}) + \nabla \cdot (\rho \vec{v} \otimes \vec{v}) + \nabla p - \nabla \cdot \tau - \rho \vec{g} = 0 \tag{2.4b}$$

$$\frac{\partial}{\partial t}(\rho e_0) + \nabla \cdot (\rho e_0 \vec{v}) + \nabla \cdot (\vec{v} p + \vec{q} - \vec{v} \cdot \boldsymbol{\tau}) - \vec{v} \cdot \rho \vec{g} = 0 \tag{2.4c}$$

where:

- ρ denotes density
- $\vec{v}$ is flow velocity in all dimensions
- p denotes pressure
- $\boldsymbol{\tau}$ is the viscous stress tensor
- $\vec{g}$ represents the sum of all body forces
- e_0 is the total energy per unit mass
- $\vec{q}$ is the heat flux by conduction

They consist in the mass conservation equation (2.4a), the conservation of momentum equation (2.4b) and the energy conservation equation (2.4c). For a Newtonian fluid, the viscous stress tensor is given by:

$$\boldsymbol{\tau} = -\frac{2}{3}\mu\left(\nabla \cdot \vec{v}\right) I + 2\mu S \tag{2.5}$$

with μ being the dynamic viscosity and S the rate of deformation tensor (i.e. the symmetric part of the velocity gradient $\nabla\vec{v}$):

$$\mathbf{S} = \frac{1}{2}\left(\nabla\vec{v} + \nabla\vec{v}^T\right) \tag{2.6}$$

A detailed derivation of such equations and the theory beyond can be found for example in [10] and [11] or in [12].

The set of equations above, even with a Newtonian fluid model, lack some other information in order to form a closed set of Partial Differential Equations (PDE). A conductive heat flux model is needed (e.g. Fourier's Law), the caloric and thermodynamic equations of state have to be chosen, a proper turbulence model if needed (see [12]) and finally, the appropriate initial and boundary conditions for the problem [13] must be defined.

Simplifications can be done to obtain less sophisticated models such as: adiabatic, inviscid, incompressible, and many others. Dimensional Analysis is a powerful tool to determine to what extent some reduced models are meaningful, and it is widely used in fluid dynamics, as described in section 2.3.1. Most CFD software codes allow to set up simulations with the most suitable model which can be coupled with a solid model to build a FSI problem. Some further details are given in section 3.1.

2.2.2 Solid domain

In solid mechanics, particles do not travel as much as they do in fluid dynamic problems, as described in 2.1.2. For this reason, a Lagrangian perspective is generally used.

The de-Saint Venant-Kirchhoff model [14] is very commonly used when describing the movement of a solid: it is also often used in FSI problems ad it is capable of handling large deformation. The material is considered:

- *homogeneous*: the material properties do not depend on the position of the particle
- *linear elastic*: the stress-strain relationship is linear
- *isotropic*: the stress-strain relationship is independent from the direction of the load

A general expression of the dynamic equation can be derived from the Virtual Work Principle (VWP) applied to an arbitrary control volume:

$$\frac{\partial^2 \vec{u}}{\partial t^2} = \nabla \cdot \mathbf{T} + \rho \vec{f} \tag{2.7}$$

In equation 2.7:

- ρ: is the material density
- $\vec{u}$: is the particle displacement
- $\mathbf{T}$: is the *second Piola-Kirchhoff* stress tensor
- $\vec{f}$: is the sum of body forces

In order to close the dynamic equation, a constitutive law which must be considered to relate stress and strain:

$$\mathbf{T} = \lambda \mathbf{I} \mathrm{tr}\left[\boldsymbol{\varepsilon_G}\right] + 2\mu \boldsymbol{\varepsilon_G} \tag{2.8}$$

where $\boldsymbol{\varepsilon_G}$ is the Green-Lagrange strain tensor:

$$\boldsymbol{\varepsilon_G} = \frac{1}{2}\left(\mathbf{F}^T\mathbf{F} - \mathbf{I}\right) \tag{2.9}$$

and $\mathbf{F}$ is the deformation gradient. λ and μ are material properties and are named Lamé constants. These relate to the Young modulus E and the Poisson ratio ν which are more commonly used in practice. The relationship among the various parameters is the following:

$$E = \frac{\mu(3\lambda + 2\mu)}{\lambda + \mu} \tag{2.10}$$

$$\nu = \frac{\lambda}{2(\lambda + \mu)} \tag{2.11}$$

The set of parameters (E, ν) or (λ, μ), together with the density ρ fully define the material, under the assumptions of linear elasticity, isotropy and homogeneity.

The set of PDEs is completed when suitable initial and boundary conditions are defined.

2.2.3 Models with reduced dimensionality: beams

The equations introduced in section 2.2.2 may be a tough task to solve even of the case of isotropic hyperelasticity, when considering a 3-D domain. Even with today's computers and using finite elements techniques, it is not always feasible or convenient to treat a solid as a three-dimensional continuum. Body with particular geometric features can be seen as lower dimension bodies, with respect to the governing equations [15]. Such bodies are called *beams* (one dimension) , *plates* or *shells* (two dimensions).

The *beam* model splits the description of the geometry into two subproblems:

1. a beam is defined by its *reference line* and the movement (displacement and rotation) of the solid is completely defined by it (see Figure 2.5),

2. the beam *cross section* is considered as a whole, its movement depends on the movement of the reference line, stresses are generalized into *resultants* (axial, bending, shear, torsional) which represent the aggregate effect of all of the stresses acting on the cross section. The constitutive properties of the section (axial, shear, torsion and bending stiffness) allow to relate stresses and deformations (by means of VWP) and close the problem.

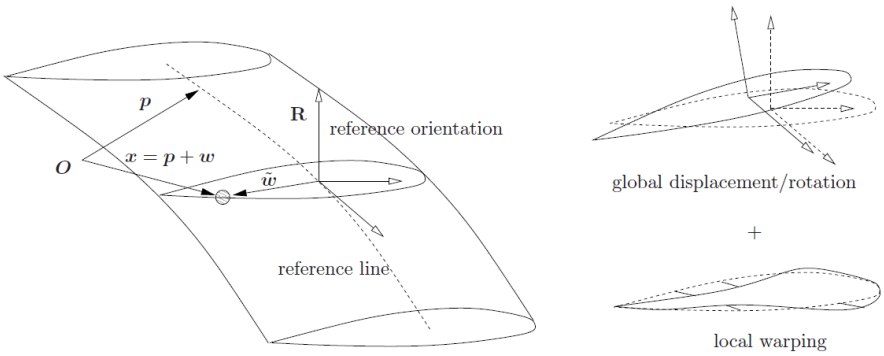


Figure 2.5. beam model, taken from [1]

The beam model can be used to build elements of a Finite Element Method (FEM). For example, the beam element can be modeled by means of a Finite Volume approach, as described in [16], which computes the internal forces as functions of the straining of the reference line and orientation at selected points along the line itself, called evaluation points.

This approach is particularly interesting for FSI problems in which slender structures are involved. A mapping is needed between the fluid-solid interface and the reference line movement, which will be described in Sections 4.2.4 and 4.2.7.

2.2.4 Interface and interaction

Since FSI problems are centered on the interaction of the fluid and solid domain, their common interface needs to be described properly. A simple representation of the situation at the so called *wet surface* is shown in Figure 2.6. Quantities related to the solid use S subscript, while fluid domain and the interface are labeled with F and FS, respectively.

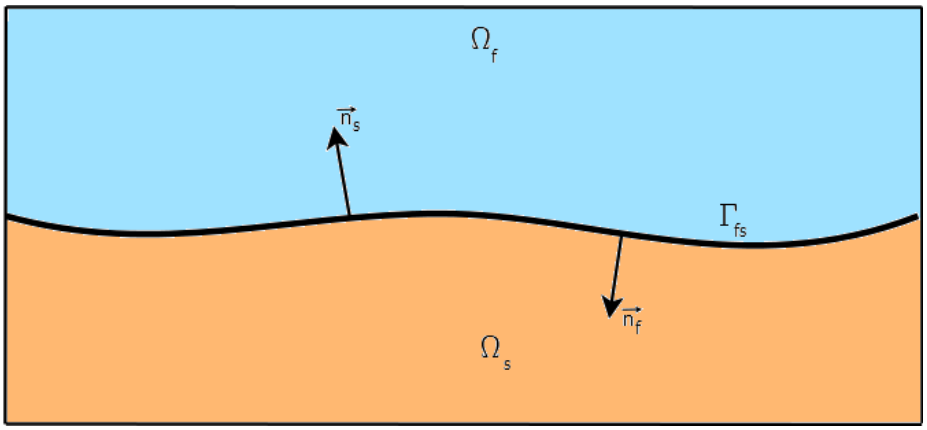


Figure 2.6. fluid solid interface

In order to have a physically correct behavior, some conditions have to be met [17]:

- solid and fluid domains should not overlap nor separate,
- for a viscous fluid model, the flow velocity at the interface must equal the boundary velocity (*no-slip* condition),
- for an inviscid fluid model, only velocity components normal to the wet surface have to be equal to the structural velocity as the fluid may slip freely in tangential direction at any boundary,
- forces exchanged at the interface must at equilibrium.

The first conditions result in the *kinematical requirement* that the displacements of fluid and solid domains, as well as their respective velocities have to be equal at the wet surface (denoted by Γ_{FS}):

$$\Delta\vec{x}_F = \vec{u}_S \tag{2.12}$$

$$\vec{v}_F = \frac{\partial\vec{u}_S}{\partial t} \tag{2.13}$$

The last condition results in the equilibrium requirement. Force vectors are computed from the stresses at the interface and the outward normal vectors of fluid and solid domain, respectively. They have to be equal and opposite leading to the dynamic coupling condition:

$$\boldsymbol{\sigma_F} \cdot \vec{n}_F + \boldsymbol{\sigma_S} \cdot \vec{n}_F = 0 \tag{2.14}$$

$\boldsymbol{\sigma} \in \mathbb{R}^{3\times3}$ represents the stress tensor (note that for the fluid it comprises pressure and viscous stresses), while $\vec{n} \in \mathbb{R}$ is the outward normal unit vector.

2.3 Classification of FSI problems

In the previous chapters we have seen that there exist a lot of models that can describe fluid flow and solid mechanics. In FSI problems we need to couple two of them: the variety of coupled problems seems to be so large that single FSI model that is applicable to every problem appears to be unfeasible. For this reason it is useful to classify FSI problems and look for specific properties in each class. The first step is to switch from dimensional quantities to dimensionless ones.

2.3.1 Dimensional analysis

We use the principle that a physical law should only relate to dimensionless quantities. There exist a rather general theorem called the Π Theorem or the *Vaschy-Buckingham Theorem* [18], which tells how many dimensionless quantities are needed to rewrite a model in dimensionless fashion. This theorem states that the number of dimensionless quantities, P, is equal to that of the dimensional ones describing the problem, N, minus R, which is the rank of the matrix of dimension exponents. This matrix is formed by the columns of the dimension exponents of all variables [19]. An example is given in the following Section 2.3.2, Table 2.1.

2.3.2 Dimensional analysis in fluid domain

Dimensional analysis is widely used in fluid dynamics. In order to keep the analysis simple,, we consider the adimensionalization of the incompressible Navier-Stokes momentum equation for a Newtonian fluid [20]:

$$\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla)\,\vec{v} = -\frac{\nabla p}{\rho} + \nu \nabla^2 \vec{v} + \vec{g} \tag{2.15}$$

The variables involved in equation 2.15 are the following:

- t: time
- $\vec{x}$: coordinates
- $\vec{v}$: velocity field
- p: pressure field
- ρ: fluid density
- ν: fluid kinematic viscosity
- $\vec{g}$: gravity
- L: reference dimension
- V_0: reference velocity

	t	$\vec{x}$	$\vec{v}$	p	ρ	ν	$\vec{g}$	L	V_0
L	0	1	1	-1	-3	2	1	1	1
M	0	0	0	1	1	0	0	0	0
T	1	0	-1	-1	0	-1	-2	0	-1

Table 2.1. fluid matrix of dimension exponents

The rank of the above matrix is 3 so 6 dimensionless parameters are needed to rewrite the equation 2.15:

- *length*: $\vec{x}^* = \frac{\vec{x}}{L}$
- *velocity*: $\vec{v}^* = \frac{\vec{v}}{V_0}$
- *time*: $t^* = \frac{V_0 t}{L} = \frac{t}{T_{fluid}}$
- *pressure*: possible choices: $p^* = \frac{p}{\rho V_0^2}$ or, if viscous forces are dominant, $p* = \frac{pL}{\rho \nu V_0}$
- *Reynolds number*: $Re = \frac{V_0 L}{\nu}$. It defines the ratio between inertia and viscous forces.
- *Froude number*: $Fr = \frac{V_0}{\sqrt{gL}}$. It defines the ratio between the flow inertia to the body field forces

The adimensionalized momentum equation becomes:

$$\frac{\partial \vec{v}^*}{\partial t^*} + (\vec{v}^* \cdot \nabla)\,\vec{v}^* = -\nabla p^* + \frac{1}{Re}\nabla^2 \vec{v}^2 + \frac{1}{Fr^2}\vec{g} \tag{2.16}$$

From Equation 2.16 a lot of models might be derived: from *Stokes regime* when viscosity is dominant, to *Euler regime* when viscosity is negligible with respect to inertia forces.

2.3.3 Dimensional analysis in solid domain

Even if it is seldom used, Dimensional Analysis can be made also for the solid domain [21].

The variables involved in a solid dynamics equation are:

- t: time
- $\vec{X}$: coordinates
- $\vec{u}$: displacement field
- ρ_S: solid density
- E: elastic modulus
- $\vec{g}$: gravity
- L: reference dimension
- U_0: reference displacement

From the variables above the following parameters can be derived:

- *length*: $\frac{\vec{X}}{L}$: dimensionless coordinate
- *displacement*: $\frac{\vec{u}}{L}$: dimensionless displacement
- *time*: $\frac{t\sqrt{\frac{E}{\rho_S}}}{L} = \frac{t}{T_{solid}}$ dimensionless time
- entity of displacements: $\frac{U_0}{L} = \delta$: *displacement number*
- gravity: $\frac{\rho_S g L}{E}$: elastogravity number

T_{solid} can be seen as $\frac{L}{c}$ with $c = \sqrt{\frac{E}{\rho_S}}$ which is the scale of elastic wave velocity. The displacement number δ tells how big the structure displacement are related to the overall dimension, and defines the *large displacements* region. Finally, the *elastogravity number* combines gravity (or body forces in general), density and stiffness: when large the deformation induced by body forces in the solid are large.

2.3.4 Dimensional analysis of coupled problems

It is now possible to undertake the dimensional analysis of a fully coupled fluid and solid interaction problem. Some of the parameters are only defined in the fluid side or in the solid side (e.g. viscosity or stiffnes). Some parameters are common to both domains (e.g. lenght scale or gravity). The variables of interest are now the velocity in the fluid and the displacements in the solid. Each of them can be related to all the parameters without separation. For example, the fluid velocity relationship is of the kind:

$$g(\vec{v}; \vec{x}, t; \rho, \mu, V_0; \rho_S.E; \vec{g}, L) = 0 \tag{2.17}$$

Equation 2.17 is composed of 11 dimensional parameters. Applying π theorem, the total number of independent dimensionless parameter expected is 8. Starting from the ones derived in the previous sections:

- $\vec{x}^* = \frac{\vec{x}}{L}$: dimensionless coordinates
- $\vec{v}^* = \frac{\vec{v}}{V_0}$: dimensionless fluid velocity
- $t*_f = \frac{V_0 t}{L}$: dimensionless time
- $Re = \frac{V_0 L}{\nu}$: Reynolds number
- $Fr = \frac{V_0}{\sqrt{gL}}$: Froude number
- $\delta = \frac{U_0}{L}$: displacement number
- $\frac{\rho_S g L}{E}$: elastogravity number

The 7 quantities above derive from the separated problem. The last one necessarily mixes things from the fluid and the solid side otherwise it would have been found in one uncoupled case. There is no unique choice for this parameter, the following are the most common ones.

Mass number

The simplest, but arguably most important parameter is the ratio of the two densities: the *Mass Number* M.

$$M = \frac{\rho}{\rho_S} \tag{2.18}$$

This can range from $\mathcal{O}\left(10^{-4}\right)$ in air-steel interaction to $\mathcal{O}\left(1\right)$ when both media have about the same density. This parameter is particularly significant for the so called *added mass* stability problem, described in Section 3.5.

Reduced velocity

Another possible choice is the reduced velocity:

$$U_R = \frac{V_0}{\sqrt{\frac{E}{\rho_S}}} \tag{2.19}$$

It is the ratio between the fluid free velocity and the velocity of elastic waves in a solid, c. It contains information on the way the two dynamics are related and it can range different orders of magnitude.

Cauchy number

Another possible parameter combines stresses or stiffness: the Cauchy number, as defined in [22]:

$$C_Y = \frac{\rho V_0^2}{E} \tag{2.20}$$

It is the ratio between the fluid inertial forces, quantified by the dynamic pressure and the stiffness of the solid E. The higher it is, the more the solid is elastically deformed by the flow.

These are actually the most important parameters involving FSI problems. Among them, there is no universally better choice. But there are efficient choices, that would be more helpful in solving a given problem.

Chapter 3

Computational aspects of Fluid-Structure Interaction problems

This section deals with the computational aspects of FSI problems. The first possible categorization of solution techniques distinguishes between monolithic and partitioned approach, as discussed in Section 3.1. This work is based on the latter approach, so the two different coupling strategies, namely strong and weak, are discussed in Section 3.2. As strong coupling is generally needed for accurate solution, ad overview of strong coupling algorithms is given in Section 3.3. Section 3.4 focuses on aspects concerning the interface mesh, and how the solid and the fluid exchange data between them. Finally 3.5 briefly describes a common issue arising in strongly coupled problems: the added mass effect (AME).

3.1 Monolithic and Partitioned Approach

Analytical solutions are impossible to obtain for the large majority of FSI problems; on the other hand, laboratory experiments may be costly, unfeasible or limited. For those reasons, numerical simulations may be employed to analyze the physics involved in the interaction between fluids and solids. With the current capabilities of computer technology, simulations of scientific and engineering models have become increasingly detailed and sophisticated.

The numerical methods used to solve FSI problems may be roughly classified into two classes: the *monolithic approach* and the *partitioned approach*. There is no exact distinction between the two approaches, as they might be seen differently among fields of applications. The idea here is to consider how many solvers are used to find a solution.

In the *monolithic approach*, the whole problem is treated as a unique entity and solved simultaneously with a specialized ad hoc solver (see Figure 3.1). The fluid and structure dynamics form a single system of equations for the entire problem, which is solved simultaneously by a unified algorithm. The interface conditions are implicit in the solution procedure [23], [24].

This approach can potentially achieve better accuracy, as they solve the system of equations exactly the interface conditions are implicit in the model [25], but it may require more resources and expertise to develop from scratch a specialized code (it solves a very specific model) that can be cumbersome to maintain.

On the other hand, in the *partitioned approach*, the fluid and the solid domains are treated as two distinct computational fields, with their respective meshes, that have to be solved separately (see Figure 3.2: how data are passed between solvers is detailed in Section 3.2). The interface conditions are used explicitly to communicate information between the

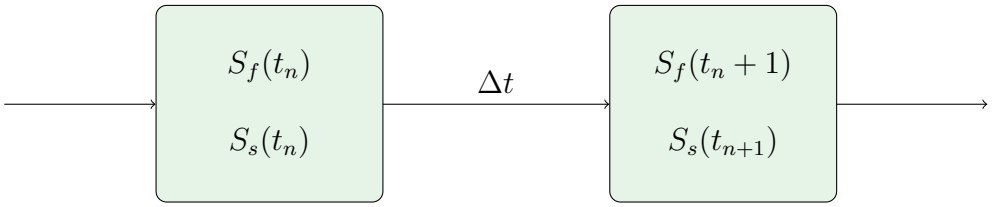


Figure 3.1. monolithic approach: S_f, S_s denote the fluid and the structure solutions

fluid and structure solutions. This implies that the flow does not change while the solution of the structural equations is calculated and vice versa [26]. The partitioned approach thus requires a third software module (i.e. a coupling algorithm) to incorporate the interaction aspects. It communicates the boundary conditions described in Section 2.2.4: that is forces or stresses (dynamic data) calculated by the fluid solver at the wet surface are passed to the solid component and displacements or velocities (kinematic data) computed by the solid solver at the interface are sent to the fluid component in return. Finally, fluid and structural solutions together yield the FSI solution.

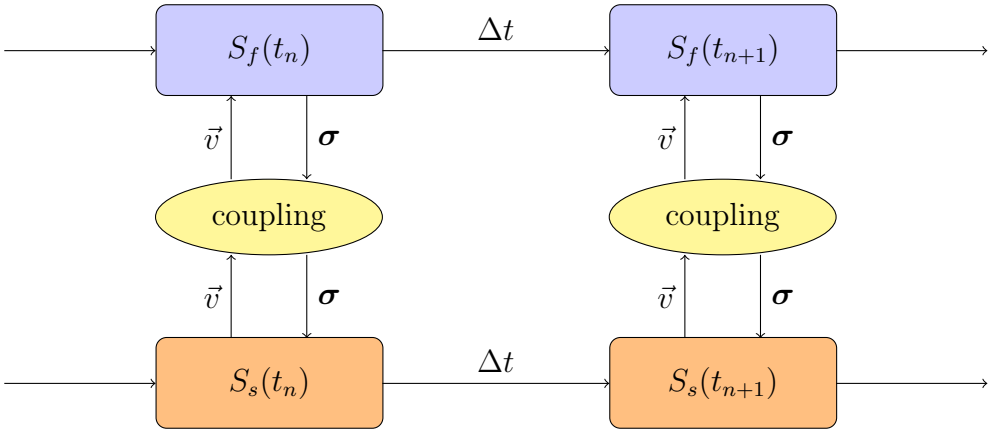


Figure 3.2. partitioned approach: S_f, S_s denote the fluid and the structure solutions, while $\boldsymbol{\sigma}$ and $\vec{v}$ represent coupling data

A big advantage of this approach is that software modularity is preserved: different and efficient solution techniques can be used for the flow equations and structural equations. Provided that they can exchange data, existing solvers for the fluid and solid problem can be reused, ranging from commercial to academic and open-source codes. Those solvers are usually well-validated. Besides, compared to monolithic procedures, the programming efforts are lower for partitioned approaches, as only the coupling of the existing solvers has to be implemented rather than the solvers themselves. The challenge of this approach is, however, to define and implement algorithms to achieve accurate and efficient fluid-structure interaction solution with minimal code modification. Particularly, the interface location that divides the fluid and the structure domains changes in time. The partitioned approach requires that the fluid solver has ALE capabilities, as introduced in Section 2.1.3. More detailed and practical explanations about the coupling component used in this work are given in Section 4.1.

3.2 Coupling Strategies

Because of the modularity, the partitioned approach has gained much attention in research. The structure sketched in Figure 3.2 needs to be detailed and specialized in function of the coupling strategies.

In an interface multi-physics coupling like FSI, the boundary surface is in common between the two sides of the simulation. The results make sense and are numerically stable only if the two sides of the interface are in agreement, since the output values of the one simulation become input values for the other (and vice-versa). The solution strategies can be roughly divided into weakly and strongly coupled approaches. They are often referred to as *explicit* and *implicit* methods in the literature. When the fluid and solid solutions are computed iteratively until some convergence criteria within the same time step, the scheme is called *implicit coupling*. The faster, simpler but less precise *explicit coupling* consists in executing a fixed number of iterations (typically one per time step) and exchange coupling values without convergence checks.

3.2.1 Explicit coupling schemes

As in the previous Section, S_f represents the fluid solver, which computes the pressures (named d_f here) at the deformable boundary and S_s is the structure solver, which uses these forces to compute the displacement and velocity of the boundary (named d_s). In a *serial-explicit* (or *conventional staggered*) coupling scheme, the solver S_f uses the old time step boundary values $d_s^{(n)}$ to compute the values of $d_f^{(n+1)}$ for the next time step:

$$d_f^{(n+1)} = S_f\left(d_s^{(n)}\right) \tag{3.1}$$

When the fluid solver completes the time step, data are passed to the structural solver:

$$d_s^{(n+1)} = S_s\left(d_f^{(n+1)}\right) \tag{3.2}$$

Note that Equation 3.1 uses values computed at t^n, while Equation 3.2 uses values computed at $t^{(n+1)}$. The order of execution might be inverted.

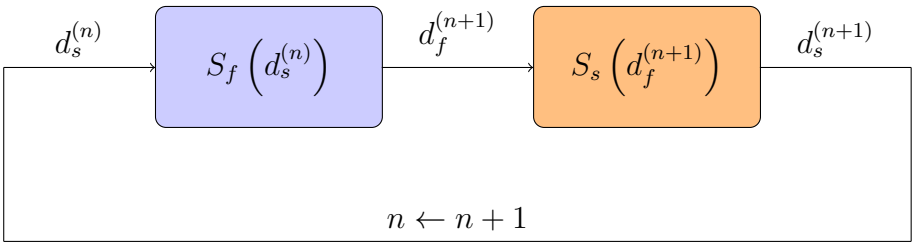
In order reduce execution time, the solvers might run in parallel, using data from the same time step (*parallel-explicit coupling*:

$$d_f^{(n+1)} = S_f\left(d_s^{(n)}\right) \tag{3.3a}$$

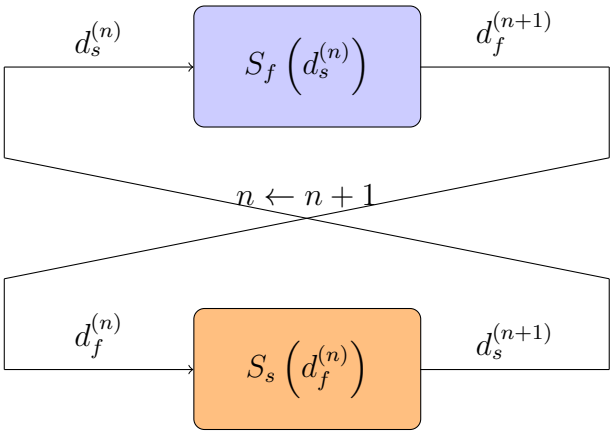
$$d_s^{(n+1)} = S_s\left(d_f^{(n)}\right) \tag{3.3b}$$

The two explicit schemes are shown schematically in Figures 3.3a and 3.3b.

In general, an explicit coupling is not enough to regain the exact (as in the monolithic approach) solution of the problem as the matching coupling conditions between the solvers is not enforced within each time step: no balance between fluid and structural domain with respect to forces and displacements at the interface can be guaranteed ([17], [26]). Nevertheless, explicit coupling yields good results if the interaction between fluid and solid is weak as in aeroelastic simulations, where in general the simulations show small displacements of the structure within a single time step and the flow field isn't much influenced by the structural displacements ([27]).



(a) serial explicit coupling



(b) parallel explicit coupling

Figure 3.3. Explicit coupling schemes

3.2.2 Implicit coupling schemes

On the other hand, strongly (implicit) coupling techniques require an iterative method to solve the fixed-point equation that derives from enforcing the agreement of the interface variables. The coupling conditions at the wet surface are enforced in each time step up to a convergence criterion. If the criterion is not met, another subiteration within the same time instance is computed. Therefore, the solution can approximate the monolithic solution to an arbitrary accuracy.

As in the explicit case, solvers may run in a sequential mode: the coupling is then named *serial* (or staggered) and the solvers wait for each other.

$$d_f^{(n+1),i+1} = S_f\left(d_s^{(n+1),i}\right) \tag{3.4a}$$

$$d_s^{(n+1),i+1} = S_s\left(d_f^{(n+1),i+1}\right) \tag{3.4b}$$

Equations 3.4 show that, in contrast with explicit coupling, both solvers use interface values at time step $n+1$, but one of them uses data from previous iteration. If run in parallel mode [28], the system becomes:

$$d_f^{(n+1),i+1} = S_f\left(d_s^{(n+1),i}\right) \tag{3.5a}$$

$$d_s^{(n+1),i+1} = S_s\left(d_f^{(n+1),i}\right) \tag{3.5b}$$

At convergence, the following relation holds of serial (or *Gauss-Seidel*) coupling:

$$d_s^{(n+1)} = S_s\left(S_f\left(d_s^{(n+1)}\right)\right) \tag{3.6a}$$

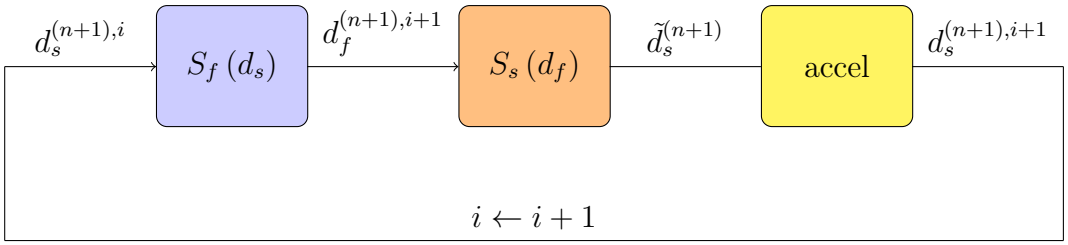
$$d_s^{(n+1)} = S_s \circ S_f\left(d_s^{(n+1)}\right) \tag{3.6b}$$

and the following relation holds for parallel (or *Jacobi*) coupling:

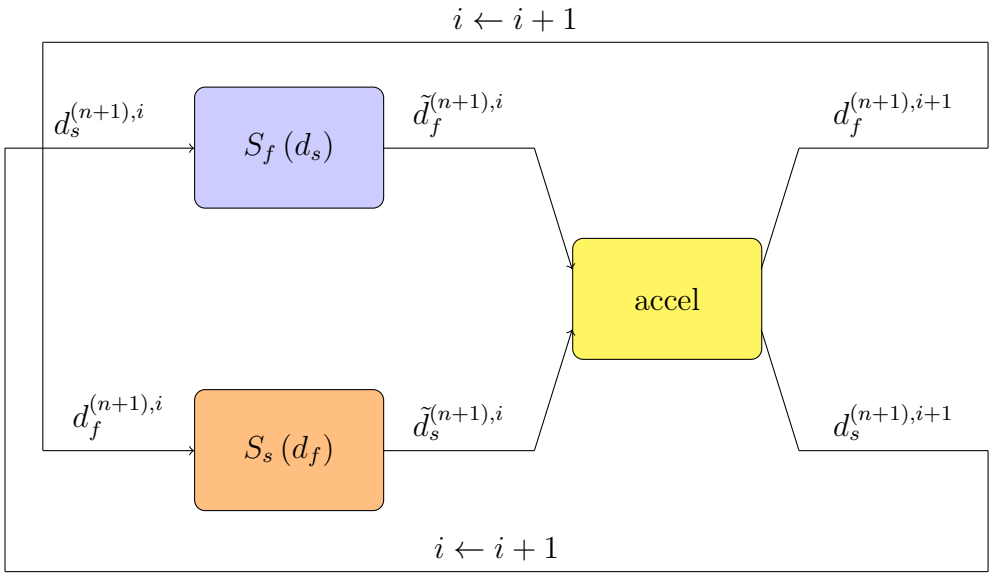
$$\begin{pmatrix} d_s^{(n+1)} \\ d_f^{(n+1)} \end{pmatrix} = \begin{pmatrix} 0 & S_f \\ S_s & 0 \end{pmatrix} \begin{pmatrix} d_s^{(n+1)} \\ d_f^{(n+1)} \end{pmatrix} \tag{3.7}$$

Acceleration techniques are necessary to bring fixed point equation 3.6b or 3.7 to convergence. Those techniques are described in Section 3.3.

The two implicit schemes are shown schematically in Figures 3.4a and 3.4b: *accel* refers to the post-processing step implemented to speedup convergence. After every non-converged iteration, the latest stored state of the solver (*checkpoint*) is reloaded and coupling iteration i for the current time step is incremented. When the solution converges, the time step n is incremented.



(a) serial implicit coupling



(b) parallel implicit coupling

Figure 3.4. Implicit coupling schemes

Implicit methods are generally applicable to any kind of FSI problems, in contrast with explicit methods. When fluid and structure are strongly coupled, explicit coupling can be

subject to numerical instabilities, a problem that cannot always be solved by reducing the coupling time step size [29]. These instabilities can be overcome by implicit methods, even if several coupling iterations may be executed every time step, until the values on both sides of the interface converge.

3.3 Strong coupling algorithms

As mentioned in the previous section, implicit methods require some post-processing (generally called *acceleration*) techniques to to make the solution of the single time step of the coupled partitioned FSI problem converge. This requires to solve a a *fixed-point equation*, in fact:

$$H(d_s) := S_s \circ S_f(d_s) \tag{3.8a}$$
$$d_s = H(d_s) \tag{3.8b}$$
$$R(d) := H(d) - d = 0 \tag{3.8c}$$

Equation 3.8a represents the composition of the solid and the fluid solution, while Equation 3.8b represents the resulting fixed point equation. As the order of execution can be switched, in Equation 3.8c, where the *residual* is defined, the input data d_s is generically substituted with d.

The basic approach to solve the fixed point equation is to perform the corresponding fixed point iteration (FPI):

$$x_{i+1} = H(x_i) \quad i = 1, 2, \ldots \tag{3.9}$$

which is known to converge if the mapping H is a contraction, but this is not the general case in FSI computations [28].

3.3.1 under-relaxation

The way to stabilize the iterations is to perform a FPI with *under-relaxation* as illustrated in the following algorithm:

Algorithm 1: FPI with relaxation

Result: d_k
initialization of d_0;
k = 0;
$\tilde{d}_1 = S_s \circ S_f(d_0)$;
$r_0 = \tilde{d}_1 - d_0$;
while $\|r_k\| > \varepsilon$ **do**
    compute d_k by relaxation;
    k = k +1;
end

The under-relaxation is defined by:

$$d_{k+1} = d_k + \omega \left(H(d_k) - d_k \right) \tag{3.10}$$

Where ω in Equation 3.10 is the *relaxation factor*. The relaxation parameter has to be small enough to keep the iteration from diverging, but as large as possible in order to use as

much of the new solution as possible [30]. The optimal ω value is problem specific and not known a priori. A suitable dynamic relaxation parameter, is a better choice, like the *Aitken under-relaxation* [31] which adapts the factor at each iteration with the following relation:

$$\omega_i = -\omega_{i-1} \frac{r_{i-1}^T (r_i - r_{i-1})}{\|r_i - r_{i-1}\|^2} \tag{3.11}$$

Aitken under-relaxation can be a good choice for strong interaction with a fluid solvers that does not fully converge in every iteration or for compressible fluid solvers.

3.3.2 Quasi Newton Least Squares schemes

Under-relaxation is a good choice for easy stable problems, but is outperformed by more sophisticated quasi-Newton coupling schemes. Equation 3.8c could be solved iteratively with a Newton method [32]:

$$R(d_k) \coloneqq r_k \tag{3.12a}$$

$$R(d_k) + \left.\frac{\partial R}{\partial d}\right|_{d_k} (d_{k+1} - d_k) = 0 \tag{3.12b}$$

$$d_k + \left(\left.\frac{\partial R}{\partial d}\right|_{d_k}\right)^{-1} (-r_k) = d_{k+1} \tag{3.12c}$$

The *residual* at iteration k is defined in Equation 3.12a, if the Jacobian matrix of the equation is known, a Newton iteration can be performed as in Equation 3.12b. The updated values can be computed using Equation 3.12c.

In situations where:

- *black-box* systems are considered (i.e. the Jacobian is unknown),
- the cost of a function evaluation is sufficiently high that numerical estimation of the Jacobian is prohibitive,

there exist a number of matrix-free methods that use only information derived from the consecutive iterates and that build an approximation based on those values. This approach is known as *quasi-Newton method* [33]. Input and output data of H and R are used to approximate the solution of 3.12c. Algorithm 12 (taken from [34]) shows the basics steps to

estimate data at next step using the *Quasi Newton Least Squares Method*:

Algorithm 2: Quasi Newton Least Squares method

Result: d_{k+1}

1 initial value d_0;
2 $\tilde{d}_0 = H(d_0)$ and $R_0 = \tilde{d}_0 - d_0$;
3 $d_1 = d_0 + \omega r_0$;
4 **for** $k = 1 \dots$ **do**
5 $\quad \tilde{d}_k = H(d_k)$ and $r^k = \tilde{d}_k - d_k$;
6 $\quad V^k = \left[\Delta r_0^k, \dots, \Delta r_{k-1}^k\right]$ with $\Delta r_i^k = r^i - r^k$;
7 $\quad W^k = \left[\Delta \tilde{d}_0^k, \dots, \Delta \tilde{d}_{k-1}^k\right]$ with $\Delta \tilde{d}_i^k = \tilde{d}_i - d_k$;
8 $\quad$ decompose $V^k = Q^k U^k$;
9 $\quad$ solve the first k lines of $U^k \alpha = -{Q^k}^T R^k$;
10 $\quad \Delta \tilde{d}^k = W^k \alpha$;
11 $\quad d_{k+1} = \tilde{d}_k + \Delta \tilde{d}_k$;
12 **end**

In algorithm 12 the matrices V^k and W^k are constructed from the previous iterations and the known values of $d_0, \dots d_k$ and $\tilde{d}_0, \dots, \tilde{d}_k$. $\Delta \tilde{d}^k$ is constructed in the column space of W^k (line 10). For this reason a least squares problem is solved:

$$\alpha = \underset{\beta \in \mathbb{R}^k}{\operatorname{argmin}} \|V^k \beta + R(d_k)\| \tag{3.13}$$

The least squares problem is solved computing the decomposition of V^k into an orthogonal matrix $Q^k \in \mathbb{R}^{k \times k}$ and an upper triangular matrix $U^k \in \mathbb{R}^{n \times k}$ (line 8). Then α is computed in line 9.

When building matrices V^k and W^k (lines 6-7), it is possible to use information from previous time steps.

Finally, to ensure linear independence of columns in the multi-secant system for Jacobian estimation, a filter can be used [35], in order to drop nearly dependent columns of Q^k and avoid singularity of the approximated Jacobian.

The above algorithm is usually denominated in FSI interface quasi Newton with inverse Jacobian from a least squares model (IQN-ILS) (or Anderson acceleration). There exist other algorithms, like generalized Broyden (IQN-IMVJ) or manifold mapping to solve the problem. A complete desciption of those methods goes beyond the scope of this work: a description of the most common algorithms can be found in [36], while a comparison of the performances can be found in [37].

3.3.3 Convergence criteria

At each time step, the coupling algorithm enforce matching conditions at the wet surface up to a convergence criterion. If not sufficiently met, another iteration within the same time step is performed. The fixed point formulation itself induces a criterion based on the residual itself r_{k+1}.

A scalar, *absolute convergence criterion* can de defined as in Equation 3.14: it is useful for close to zero values of the coupling quantities, when rounding errors become important:

$$\|r_{k+1}\| \leq \epsilon_{abs} \tag{3.14}$$

A more common *relative convergence criterion*, defined in Equation 3.15 is particularly useful when different quantities (e.g. forces and displacements) are compared together to evaluate convergence:

$$\frac{\|r_{k+1}\|}{\|\tilde{d}_{k+1}\|} \leq \epsilon_{rel} \tag{3.15}$$

3.4 Interface Mesh and Data Mapping

FSI methods can also be classified considering how the fluid and solid meshes are treated. The *conforming mesh methods* consider the interface as a physical boundary condition (see Section 3.4.2), while *non-conforming mesh methods* treat the boundary location as a constraint imposed on the model equations (see Section 3.4.1) [17].

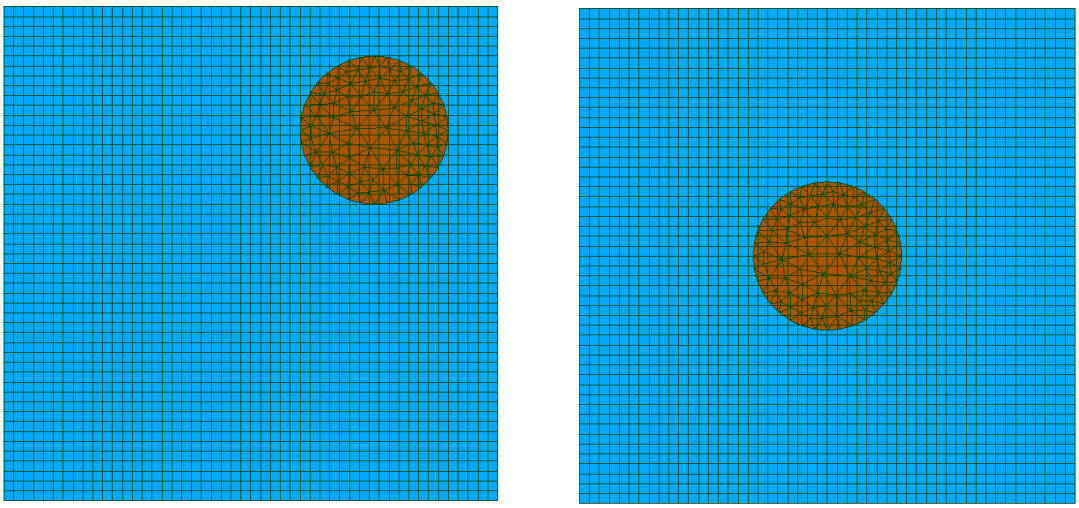
3.4.1 Non-conforming Mesh methods

In non-conforming mesh strategies all interface conditions are imposed as constraints on the flow and structural governing equations. It is possible to use non-conforming meshes for fluid and solid domains as they remain geometrically independent from each other (see Figure 3.5).

This approach is mostly used in *immersed boundary* methods [38]. Coupling is imposed by means of additional force terms appearing in the model equations of the fluid, which impose the kinematic and dynamic conditions. The forces represent the effects of a boundary or body being immersed in the fluid domain. A purely Eulerian mesh (see Section 2.1.1) can be used for the whole computational domain, since the force terms are dynamically added at specific locations to represent the structure.

The fluid forces applied on the solid at the wet surface are computed and used as input for the structural solver, which employs a standard Lagrangian mesh (see Section 2.1.2).

Immersed boundary methods are particularly innovative and are useful to overcome some issues in CFD computations, on the other hand most of the current implementations of FSI problem implement a conforming mesh strategy.



(a) mesh at $t = t_1$ (b) mesh at $t = t_2$

Figure 3.5. non conforming mesh example

3.4.2 Conforming Mesh methods

Conforming mesh methods adapt very well to the partitioned approach described in Section 3.1 as they usually consists in the computational steps described above, namely: computation in the fluid, computation in the solid, enforcing of interface condition and mesh movement (see Figure 3.6).

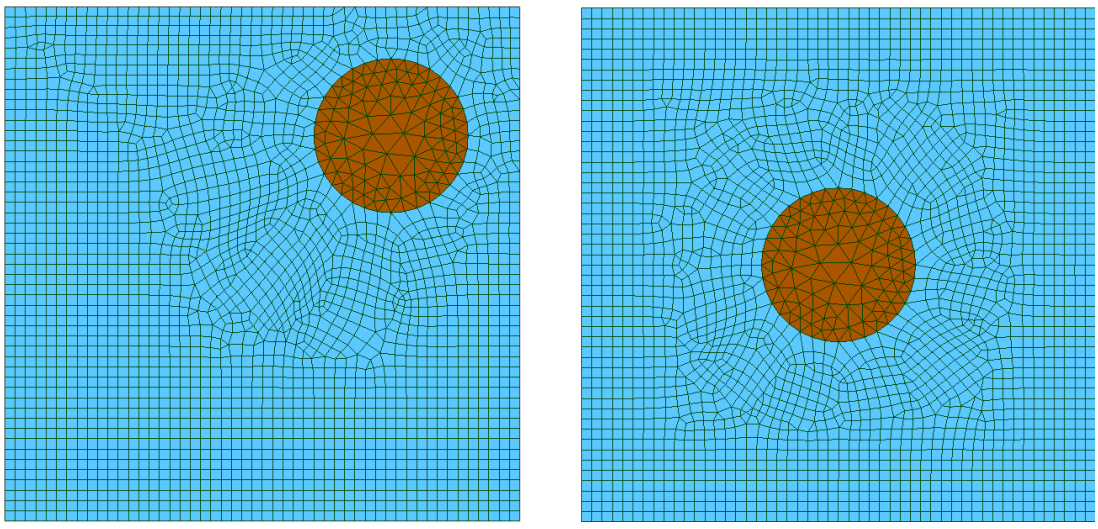
Fluid and structure meshes need share the boundary of the wet surface, as the coupling conditions are enforced by applying kinematic or dynamic conditions to those boundaries. Node-to-node matching of fluid and structure meshes at the interface is not required, as long as a suitable mapping between the interface nodes is performed (see Section 3.4.3).

The match between the interfaces must hold at each time step: this implies that both solid and fluid domains need to deform. Deformation is easily expressed in the solid domain as the structural mesh is usually represented in Lagrangian perspective (see Section 2.1.2). ALE perspective (Section 2.1.3) for the fluid domain becomes necessary in this case.

Mesh deformation can turn out to be a complicated task as in general the fluid mesh is deformed during motion (see Figure 2.4). Mesh smoothing techniques need to be applied in order to keep a good mesh quality in terms of distorted elements which can lead to accuracy loss in simulations. (the following video shows high distorted fluid elements during FSI motion: video).

Mesh smoothing is generally applied to keep the fluid mesh as uniform and undistorted as possible during movement. There is a wide variety of mesh updating procedures [39]. The *torsional spring analogy* [40] is a fairly simple technique that computes mesh movement considering mesh edges as springs and solving the subsequent Laplace equation that derives from the mesh movement.

Some other references about mesh motion alternatives can be found in [41].



(a) mesh at $t = t_1$ (b) mesh at $t = t_2$

Figure 3.6. conforming mesh example

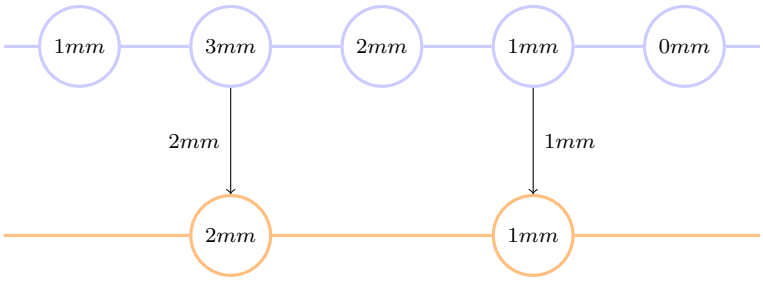
3.4.3 Data Mapping

When partitioned coupling is involved and the meshes are conforming but not node-to-node coincident, the challenge is to correctly map the data between the solid and the fluid sides.

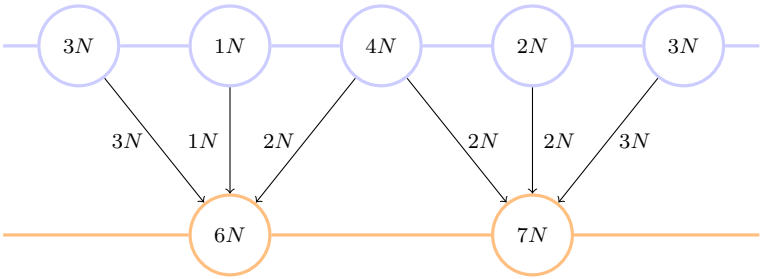
This is a common situation as fluid and solid require a different mesh refinement at the interface.

The mapping procedure needs not only to find the closest available mesh point (or points) on the opposite mesh, but also to preserve mass and energy balance. Variables are basically mapped in two ways: *consistent* and *conservative* forms.

In the *consistent mapping* a value on a node of one grid has the same value of the corresponding node on another grid: that is, it reproduces the values on both meshes. In the *conservative form*, integral values are preserved between meshes. In an FSI problem, nodal forces are mapped in conservative form, while velocities or displacements are mapped in consistent form. An example is shown in Figure 3.7.



(a) consistent mapping



(b) conservative mapping

Figure 3.7. Examples of mapping data between non-coincident meshes: consistent (a) and conservative (b) schemes.

different mapping strategies can be implemented [42]:

- *Nearest Neighbor*: finds the closest point on the source mesh and uses its value for the target mesh. It does not require any topological information and is first-order accurate. It is the computationally easiest implementation and it is useful when interface meshes are coincident.
- *Nearest Projection*: projects the points of the target mesh on the source mesh, interpolates the data linearly and assigns the values to the target mesh. It requires topology information for the source mesh. The interpolation on the mesh elements is second order accurate.
- *Radial Basis Function (RBF)*: this method does not requires topological information and works well on general meshes. The mapping uses radial basis functions centered at the grid points of the source mesh [43].

3.5 Stability: Added Mass Effect

When a solid moves or vibrates in a fluid domain, the interaction changes the way in which the structure behaves. There exist a vast variety of literature (e.g.: [44], [45], [46], [47]) describing the effects of FSI in terms of *added mass*, *added damping* or even *added stiffness* on a vibrating structure in function of fluid properties (e.g. density or viscosity), or flow properties (e.g. velocity), or geometry.

Besides the physical aspects of the interaction, some numerical issues arise when trying to simulate this kind of problems.

The numerical issue named *AME* is introduced here, as it is relevant for both strongly and weakly coupled partitioned approaches in the solution of FSI problems.

Weakly coupled algorithms give good results in aeroelasticity studies, but they are known to become unstable under certain conditions, in particular when fluid and structure densities are comparable ($M \approx 1$, see Section 2.3.4) and when the structure is particularly slender [48]. Under the same conditions, strongly coupled algorithms exhibit convergence problems.

For this reason, the mass ratio M is a suitable indicator to determine the kind of interaction between solid and fluid: when $M \ll 1$ (i.e. when the solid is much denser than the fluid), the interaction is weak, while when densities are comparable the interaction is strong and imposes some limits on the partitioned solution techniques.

The problem has been analyzed in literature by means of "toy FSI models" (in [48], [49] or [28]). Even though the number of parameters affecting stability is big and not completely understood in complex scenarios, all of the studies point out the mass-ratio M as the most relevant parameter.

A simplistic explanation of the phenomenon stems from the idea that at the interface, fluid and structure have no gaps (Section 2.2.4). For this reason, if the structure moves, also the fluid particles around it have to move: the acceleration of the surrounding fluid results in more inertial forces and the structure appears more inert.

Added mass effect appears both in incompressible and in compressible fluid models, but with slightly different effects and implications.

3.5.1 AME in compressible regime

In compressible regime, using a weakly coupled algorithm which doesn't enforce mass and energy balance at the interface (see Section 3.2.1) imposes a limit to the mass ratio above which the simulation becomes unstable and the algorithm fails to find a solution [50].

A strongly coupled algorithm (Section 3.2.2) does not become unstable, but converges slowly: many subiterations are needed at each time step in order to reach the required convergence criteria (Section 3.3.3).

It can be shown that, in the compressible case, changing the time step of the partitioned simulation can be beneficial to the solution. The time step reduction to an arbitrarily small value cannot stabilize a weakly coupled algorithms when the stability criterion on the mass ratio is not met, but has an effect on strongly coupled ones [29].

A step size reduction can compensate a higher mass-ratio value [51]: the convergence of a strongly coupled algorithm improves proportionally to the time step reduction. At the theoretical limit of vanishing time step, the monolithic solution could be found.

3.5.2 AME in incompressible regime

AME is a much greater issue in incompressible regime than in compressible flows. A simple physical explanation of this fact could be as follows: a deformation of the structure results

in a perturbation of the fluid domain close to the structure, which then propagates through the rest of the fluid domain. In compressible models, the speed at which a perturbation propagates (speed of sound) has a finite value. For this reason the the effect of a perturbation is spatially limited during a time step. In contrast, in an incompressible model, the speed at which the aforementioned perturbation propagates through the domain is infinitely large. A change in the geometry affects the whole domain without delay and impacts the whole domain at each time step [48].

It has been observed that loose coupling of fluid and structural part in the context of incompressible flow and slender structures frequently yields unstable computations [51]. However, strict stability limits exist also for strongly coupled algorithms. Those limits have a different relation to the time step with respect to compressible models.

Simulations show that reducing of the time step may result in an increased instability. The AME is inherent in the coupling itself: in sequentially staggered schemes the fluid forces depend upon predicted structural interface displacements rather than the correct ones and thus contain a portion of incorrect coupling forces. This contribution yields the instability [49].

Provided that the stability limit is not exceeded, implicit methods behave differently n incompressible regime: the number or subiterations required to reach convergence during a single time step increases when the time step decreases. Besides, achieving the monolithic solution limit is not guaranteed ([51], [29]).

These observations are consistent with the aforementioned physical explanation.

Chapter 4

Software Packages used in this work

This chapter illustrates the main software tools used in this work. First, the coupling library *preCICE* is introduced in Section 4.1. Then, the multibody dynamics solver *MBDyn* being connected to preCICE is shortly presented in Section 4.2.

4.1 preCICE

The main information concerning preCICE is taken from the official documentation: that is [52] and [42]. The preCICE website is also a source of documentation[1].

The open-source[2] software library preCICE provides the components to connect traditional single-physics solvers and create a partitioned multi-physics simulation (e.g. fluid-structure interaction, conjugated heat transfer, solid-solid interaction, etc.). It aims at coupling existing solvers in a partitioned black-box manner (see Section 3.2): only minimal information abut the solver is available and connection involves just interface nodes. In order to be flexible and easily implemented, the impact on the solvers should be as minimal as possible: for this reason, preCICE offers a high level application programming interface (API) (Section 4.1.5) different languages, such as C/C++, Fortran and Python. The ability to switch among different solvers is advantageous as it provides a lot of flexibility in developing and testing new coupled components.

In a nutshell, preCICE simply affects the input and observes the output of the solvers (called *participants*). The required data and control elements are accessed using an *adapter*, i.e. a "glue code" that is attached to the corresponding solver and communicates the information with the library.

preCICE performs all the actions required to perform a coupled simulation: implements the coupling strategy and convergence criteria (Section 4.1.1), the communication between the participants (Section 4.1.2), the mapping of data between meshes (Section 4.1.3). preCICE is configured by means of an extensible markup language (XML) file (Section 4.1.4).

4.1.1 Implemented coupling strategies

The *partitioned approach* (Section 3.2) is obviously the coupling strategy adopted by preCICE. It allows both *explicit* (Section 3.2.1) and *implicit* coupling (Section 3.2.2). The possible variants are four:

[1] www.precice.org

[2] The code can be accessed via Github: github.com/precice/precice

- `serial-explicit`: a serial, weakly coupled algorithm (Figure 3.3a). The first solver uses the second solver solution at the last time step to compute its current solution. In contrast, the second solver needs the current first solution to compute its solution at the same time instance. The order of execution is user defined.
- `parallel-explicit`: both solvers advance in parallel (Figure 3.3b) and exchange data at the end of each time step, resulting in a less stable procedure. The bottleneck of this procedure is related to the most time consuming solver.
- `serial-implicit`: a serial strongly coupled algorithm (Figure 3.4a). The user can define the order of execution, the coupling algorithm (basically all the algorithms described in Section 3.3) and its parameters in a section of the configuration file named `acceleration`.
- `parallel-implicit`: again both solvers execute in parallel (Figure 3.4b). An implicit scheme modifies the result of the fixed-point iteration on both data [28].

4.1.2 Communication strategies

All the participants need to communicate with each other, in order to share coupling data. Each solver might be executed in multiple processes or on different nodes of a cluster (*intrafield parallelism*). This form of parallelization requires efficient forms of communication between the solver in order to avoid that data transfer becomes a bottleneck during a simulation.

preCICE implements a fully parallel process-to-process communication approach [53] using:

- *message passing interface (MPI)*: available on most scientific computers, it may be necessary to adapt/change the MPI versions of the respective single-physics solvers or of preCICE.
- *Transmission Control Protocol/Internet Protocol (TCP/IP)*: popular means of network communication and free of incompatibilities between versions.

As to performances, MPI is the best technique especially when a high numbers of nodes is present. Anyway, socket communication is quite as fast, such that both techniques are very well-suited for larger-scale simulations [52].

In each solver, executed in parallel, one "master" process is defined to manage the progress of the simulation. No central node is required. The participating processes use asynchronous point-to-point (M:N) communication. The channels are static and defined in the beginning of the simulation. This sets a limit in using preCICE with dynamically adaptive meshes or immersed boundaries.

4.1.3 Data mapping

Even if volume coupling is possible, preCICE is mainly designed to couple simulations that share a common surface boundary (namely *conforming meshes*, in the terminology used in Section 3.4.2). The meshes don't need to be node-to-node coincident, so it is necessary to map variables at the interface, preserving the geometry (i.e. no gaps or superpositions at the interface) and the mass and energy balances.

The user defines which data are shared by each *participant* (i.e. solver) and the way data is shared in the configuration section named `mapping`. As described in Section 3.4.3, two kinds of mapping are available:

- `consistent`: the value of a node at the one grid is the same as the value of the corresponding node (or nodes) at the other grid. In general the number of fluid nodes is at least the same or, more often, exceeds the nodes of the structure, so a single structural node is associated to several fluid nodes. The mapping of displacements is consistent: in the simplest case, all fluid nodes experience the same displacement of a single solid node, otherwise an interpolation is performed (see Figure 3.7a).

- `conservative`: in the same conditions as before, forces are mapped from multiple fluid nodes to a single solid node in an additive manner (see Figure 3.7b).

Along with the mapping strategy, a method must be defined (see Section 3.4.3): `nearest-neighbor`, `nearest-projection`, `rbf`. For the latest method, preCICE implements a wide variety of basis functions, with Gaussian and thin plate splines being the most widely used.

4.1.4 Configuration

In order to run a multi-physics simulation with preCICE, all the participating, *adapted* solvers have to be started (the order is irrelevant). Some configuration files are needed:

- each adapter in general needs its own configuration file. It normally contains information about the boundaries (wet-surface) used for the coupling, names of exchanged data, mesh and the name of the common preCICE configuration file, plus other parameters, specific to the adapter. The one for the MBDyn adapter will be described in more detail in Chapter 5 and in Appendix B.

- preCICE configuration file: This is an XML file and each participant points at it. It defines all the information relevant to the simulation:

  - type and name of exchanged data and meshes over which those data are passed
  - which solvers participate in the simulation, which data produce or consume, and how the mapping is performed
  - how solvers communicate among each other
  - coupling scheme and all the necessary information

  The structure of a preCICE configuration file is illustrated in Appendix A.

4.1.5 Application Program Interface

A solver, in order to be coupled to preCICE, must either provide a way to access its core functions (e.g. initialize, set input data, read output, advance...) from outside the code (via API, socket, etc...) or it has to be slightly modified in order to perform all the operations required by the preCICE library.

The result is an *adapter*, which can be the modified and recompiled original solver, or a standalone piece of code that communicates with the original unmodified solver on one side and with the preCICE library on the other. The adapter groups together all the calls to the preCICE methods from its API (a list of the API calls, taken from the official documentation, can be found in Appendix C.1).

While preCICE is written in C++, there exist APIs also for other languages, so that the adapter can be written also in C, Fortran or Python.

A coupling consists of a configuration and an initialization phase, multiple coupling advancements and a finalization phase: the general structure of an adapter can be found in C.2.

4.1.6 Official Adapters

This work introduces an adapter to preCICE for MBDyn. It is based on previous MBDyn adapters and on the examples given on the preCICE website[3]. Official adapters are currently available for several free solvers, e.g. CalculiX, Code-Aster and SU2. Also some closed-source software packages are supported. A (maybe outdated) list of official adapters can be found in [54], while the current status of coupled codes can be found at the following link: preCICE adapters.

4.2 MBDyn

MBDyn is free and open-source[4] general purpose Multibody Dynamics analysis software developed at the *Dipartimento di Scienze e Tecnologie Aerospaziali* of the University Politecnico di Milano.

Most of the information concerning MBDyn is taken from the official documentation given in the software website[5] and form the input manual[6].

MBDyn allows to build a simulate multibody system, which[7] is the study of the dynamic behavior of interconnected rigid or flexible bodies, each of which may undergo large translational and rotational displacements.

MBDyn can simulate linear and non-linear dynamic or rigid and flexible bodies (including geometrically exact and composite-ready beam and shell finite elements, component mode synthesis elements, lumped elements) subjected to kinematic constraints, external forces and control subsystems [55]. MBDyn has been developed to serve as an analysis tool for rotorcraft research and cab simulate essential fixed-wing and rotorcraft aerodynamics.

As explained more in detail later, MBDyn is open to be connected to other software components to perform multiphysics simulations. In particular, it is possible to pass externally computed forces and to steer the multibody simulation from an external API: this feature has been particularly useful in building and *adapter* to couple MBDyn with the library preCICE (see Chapter 5).

In this section we give a short introduction to some of the relevant features of MBDyn, starting form basic information on the input file syntax (Section 4.2.1), and the output files (Section 4.2.8). Then some of the elements relevant for the description of the models used in the this work are presented: nodes (Section 4.2.2), beam elements (Section 4.2.4), bodies (Section 4.2.5) and forces (Section 4.2.7).

4.2.1 Basic syntax

MBDyn is a command line tool and can be generally started from a terminal passing an input file containing all the information required to perform a simulation.

[3] github.com/precice/precice/wiki/Adapter-Example
[4] the software is available through a public git repository gitlab.polimi.it/Pub/mbdyn
[5] MBDyn website: mbdyn.org
[6] a copy can be found at the following link [input manual] or in the code repository
[7] see Wikipedia entry: Multibody system

The input files is structured in blocks and each block has a syntax described in Backus Naur form in the input manual.

```
begin : data ;
    # select a problem
    problem : initial value ;
end : data ;

begin : initial value ;
    # problem-specific data
end : initial value ;

begin : control data ;
    # model control data
end : control data ;

begin : nodes ;
    # nodes data
end : nodes ;

begin : elements ;
    # elements data
end : elements ;
```

Listing 4.1. MBDyn input file structure

As illustrated in listing 4.1, statements are logically divided in blocks. Each block is opened by a `begin` statement and it is closed by an `end` statement.

The sequence of relevant valid blocks is:

- `data`: defines the kind of problem to be solved by the analysis. The most significant one is `initial value` and is already defined in the example.
- type of problem: block takes the name of the problem defined in the data block (`initial value` in this case) and contains all the information required by the integration method to perform the desired simulation (e.g. simulation type, time step, number of iterations, tolerance ...).
- `control data`: mostly contains information about the problem that is required to ensure that a consistent model will be generated (i.e. the number of nodes, elements, forces...).
- `nodes`: The nodes block contains all the nodes required by the simulation. They are are defined as the entities that make degrees of freedom available to the simulation, so they must exist before any element is generated.
- `elements`: contains all the elements. They are defined as the entities that generate equations using the degrees of freedom provided by the nodes.

We focus now on the main entities of a MBDyn simulation, namely nodes and elements. A detailed example of the input file used in most simulations in this work can be found in Appendix D.1.

4.2.2 Nodes

Nodes are the basic blocks of a model: they instantiate kinematic degrees of freedom and the corresponding equilibrium equations. There can be different type of nodes in MBDyn, we focus here on `structural nodes`.

Structural nodes can have 3 degrees of freedom, thus describe the kinematics of point mass motion in space (position) or 6 DoFs (position and orientation), and thus describe the kinematics of rigid-body motion in space.

Each node has a unique label and can be specified in different ways:

- `static`: this keyword instantiates only equilibrium equations (force for 3 DoF nodes or force and moment for 6 DoF nodes)
- `dynamic`: this also instantiates momentum (3 and 6 DoFs) and momenta moment (6 DoFs)

Also `modal` and `dummy` nodes exist, but are beyond the scope of this work.
Nodes are the starting point to define the elements of a simulation.

4.2.3 Elements

Elements constitute the components of the multibody model. Each has a unique numerical label and is connected to one or more nodes. They write contributions to nodes equations and represent *connectivity* and *constitutive properties*.

There exist many types of elements in MBDyn, we focus here on the ones used in the FSI simulation: beams, bodies, joints and forces.

4.2.4 Beam elements

As briefly introduced in Section 2.2.3, when simulations involve slender bodies, it is particularly interesting to use a 1D finite element model together with a form of mapping between the interface (wet surface) and the model to exchange kinematics and dynamics information. This can be performed in MBDyn using `beam` elements (described in this section) and the `external structural mapping` element (described in Section 4.2.7).

MBDyn models slender deformable components by means of finite volume `beam` elements with a high level of flexibility.

The beam element is defined by its nodes and a reference line; although 2 and 3 nodes beam elements are implemented, only `beam3` are considered here (Figure 4.1). Each node of the beam is related to a `structural node` (node 1 to 3 in Figure 4.1) by an offset (o_1 to o_3) and a relative orientation.

The Finite Volume approach described [16] is used to model the beam element. It computes the internal forces as functions of the reference line strain and as functions of the orientation at the *evaluation points* (i.e. integration points, point I and II in Figure 4.1) that are between nodes 1 and 2, and between nodes 2 and 3 (at $\xi = -1/\sqrt{3}$ and $\xi = 1/\sqrt{3}$ of a non-dimensional abscissa $-1 \leq \xi \leq 1$ ranging from node 1 to node 3).

A 6D constitutive law is defined at each evaluation point: it relates the strains and the curvatures of the beam (and their time derivatives) to the internal forces and moments at

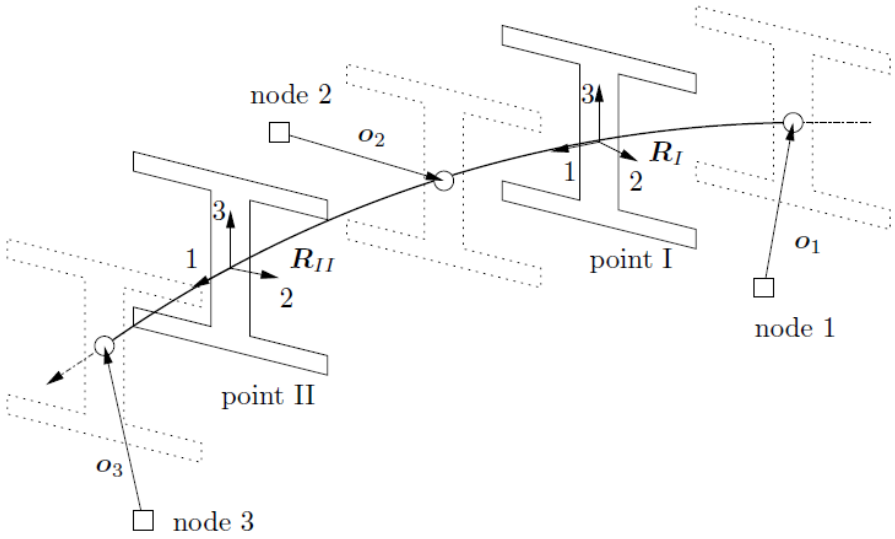


Figure 4.1. MBDyn beam model, taken from the input manual

the evaluation points in the form:

$$\begin{Bmatrix} F_x \\ F_y \\ F_z \\ M_x \\ M_y \\ M_z \end{Bmatrix} = f\left(\begin{Bmatrix} \epsilon_x \\ \gamma_y \\ \gamma_z \\ \kappa_x \\ \kappa_y \\ \kappa_z \end{Bmatrix}, \begin{Bmatrix} \dot{\epsilon_x} \\ \dot{\gamma_y} \\ \dot{\gamma_z} \\ \dot{\kappa_x} \\ \dot{\kappa_y} \\ \dot{\kappa_z} \end{Bmatrix} \right) \tag{4.1}$$

Using the convention of *x-axis* as beam axis we have:

- F_x: axial force component,
- F_y and F_z: shear force components,
- M_x: torsional moment component,
- M_y and M_z: bending moment components,
- ϵ_x: axial strain component,
- γ_y and γ_z: shear strain components,
- κ_x: torsional curvature component,
- κ_y and κ_z: bending curvature components,
- f: constitutive law.

Beam section Constitutive Law

In dynamic simulations, linear elastic or viscoelastic laws are generally used, even though nonlinear laws can be used. Focusing on linear laws, MBDyn allows the user to define every constitutive laws, going from an isotropic beam section to a fully anisotropic one: in fact the

entire 6×6 constitutive matrix can be provided. It is up to the user to define a valid law as the matrix must satisfy some constraints, e.g. it must be symmetric.

In the context of FSI simulations, this feature allows the user to define a section constitutive law that is independent from the shape of the beam itself. Thus the aerodynamic aspects and the structural aspects are handled by two distinct elements of the model: i.e. the interface mesh defines the aerodynamic forces and the beam constitutive law defines the structural properties.

Some studies about the definition of general beam section constitutive properties (*composite beam section characterization*) are available in the literature: an early work can be found in [56], a review in [57] or an application to wind turbine blades in [58].

4.2.5 Bodies

The `body` element describes a lumped rigid body when connected to a regular, 6 DoF structural node, or a point mass when connected to a rotationless, 3 DoF structural node. It can be used in connection with a structural element to give inertial properties: for example, in a `beam` element (see Section 4.2.4), 2 bodies are added to the evaluation points of the beam to account for lumped inertia of each portion in which the beam is divided.

4.2.6 Joints

`structural` nodes can be constrained by means of `joint` elements. Many different joints are available. In the FSI model the following types of joints are used:

- `clamp`: grounds all 6 DoFs of a node in an arbitrary position and orientation
- `total joint`: allows to arbitrarily constrain specific components of the relative position and orientation of two nodes [59]

4.2.7 Forces

The `force` element is a general means to introduce a right-hand side to the equations. Structural forces are specific to structural nodes and have three components that may depend on arbitrary parameters and a location in space.

MBDyn allows to communicate with an external software that computes forces based on information on the kinematics of the model. This feature is at the basis of the development of the *adapter*. The following elements can be used.

External Structural

The `External Structural` element allows to communicate with an external software that computes forces applied to a pool of `nodes` and may depend on the kinematics of those nodes. In this case forces are applied directly to the nodes. In a FSI model, this would require that each interface mesh node has a correspondent MBDyn structural node.

External structural mapping

This element is similar to the previous one, but the nodes where forces are applied and the kinematics is computed depend on structural nodes through a linear mapping. This element has been used in building the adapter. In order to use the `external structural mapping` elements, the following steps have to be performed:

1. a set of points is defined for each `structural node` according to a specified offset. Those points are used to compute the kinematics of the interface points, originating from the rigid-body motion of the structural nodes
2. before the simulation, a linear mapping matrix H is generated starting from the position of the above points and the interface mesh points (this is performed by means of an `Octave` script which is part of MBDyn). The matrix is stored in sparse form
3. the mapping matrix is used during simulation to map forces and kinematics between the interface nodes and the structural nodes

The constant matrix mapping allows to compute the position and the velocity of the *interface* points as function of the points rigidly offset from structural nodes:

$$x_{interf} = Hx_{mbdyn} \tag{4.2a}$$
$$\dot{x}_{interf} = H\dot{x}_{mbdyn} \tag{4.2b}$$

The same matrix is used to map back the forces onto structural nodes based on the preservation of the work done in the two domains:

$$\delta x^T_{mbdyn} \cdot f_{mbdyn} = \delta x^T_{interf} \cdot f_{interf} = \delta x^T_{mbdyn} \cdot H^T \cdot f_{interf} \tag{4.3}$$

which implies

$$f_{mbdyn} = H^T f_{interf} \tag{4.4}$$

When performing an FSI simulation with strong coupling (see Section 3.3), MBDyn may need to compute multiple iterations of the same time step in order to reach global convergence. This is performed by using the keyword `tight` in the coupling of the `external structural mapping`. The computation and the communication pattern is the following:

1. MBDyn sends the predicted kinematics for time step k
2. MBDyn receives a set of forces sent by the external peer; those forces are computed based on the kinematics at iteration j
3. MBDyn continues iterating until convergence using the last set of forces until, while reading the forces, it is informed that the external peer converged. this implies that MBDyn solves the kinematics for time step k at iteration j using the forces evaluated by the external solver for iteration $j - 1$

The communication with the external software, in our case the adapter itself, is performed by means of a local unix socket.

4.2.8 Simulation output

There are a bunch of output files regarding a simulation performed with MBDyn. the name of those files is specified with the option `-o` (otherwise the name is the same as the input file) and the extensions are:

- `.out`: for miscellaneous output
- `.mov`: for kinematic output of the nodes

- `.ine`: for the dynamic output of the nodes
- `.frc`: for the output of force elements
- `.act`: for the output of beam elements
- `.jnt`: for the output of joint elements

The `.out` file contains information regarding the simulation iterations residuals while other files are described in more detail in Appendix D.2.

Chapter 5

MBDyn Adapter and its integration

To prepare an existing simulation code for coupling, preCICE has to be integrated with the solver, using API described in Section 4.1.5 and in Appendix C.1. The "glue-code" required for this operation is called *adapter*, as depicted in Figure 5.1.

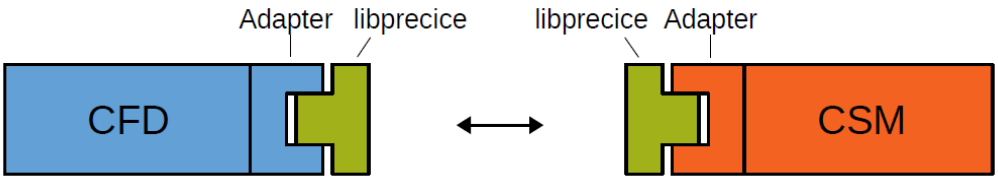


Figure 5.1. Coupling CFD to CSM via preCICE.The existing solver code, the adapter and the linked library are highlighted (image taken from [54]).

5.1 Design of the adapter structure

In order to couple MBDyn with preCICE a C++ adapter has been implemented within the scope of this work. The *adapter* needs to be integrated with both the MBDyn solver and the coupling library. The two connections are distinct but strictly interconnected. The adapter has the advantage of being completely independent from both the preCICE library and MBDyn. The first connection is achieved via the API given by the library `libprecice.so`, the second connection exploits the API given by MBDyn through its library `libmbc.so`.

5.2 Structure of the code

The code for the adapter is available through a public git repository[1]. The code is conceptually divided in two classes, as illustrated in Figure 5.2.

The main class is `MBDynAdapter`, which implements the functions given by the preCICE interface. It has access to the class `MBDynConnector` which takes care of all the aspects regarding MBDyn. Attributes, methods and operations of each class are briefly described in the following sections.

[1] mbdyn-beam-adapter

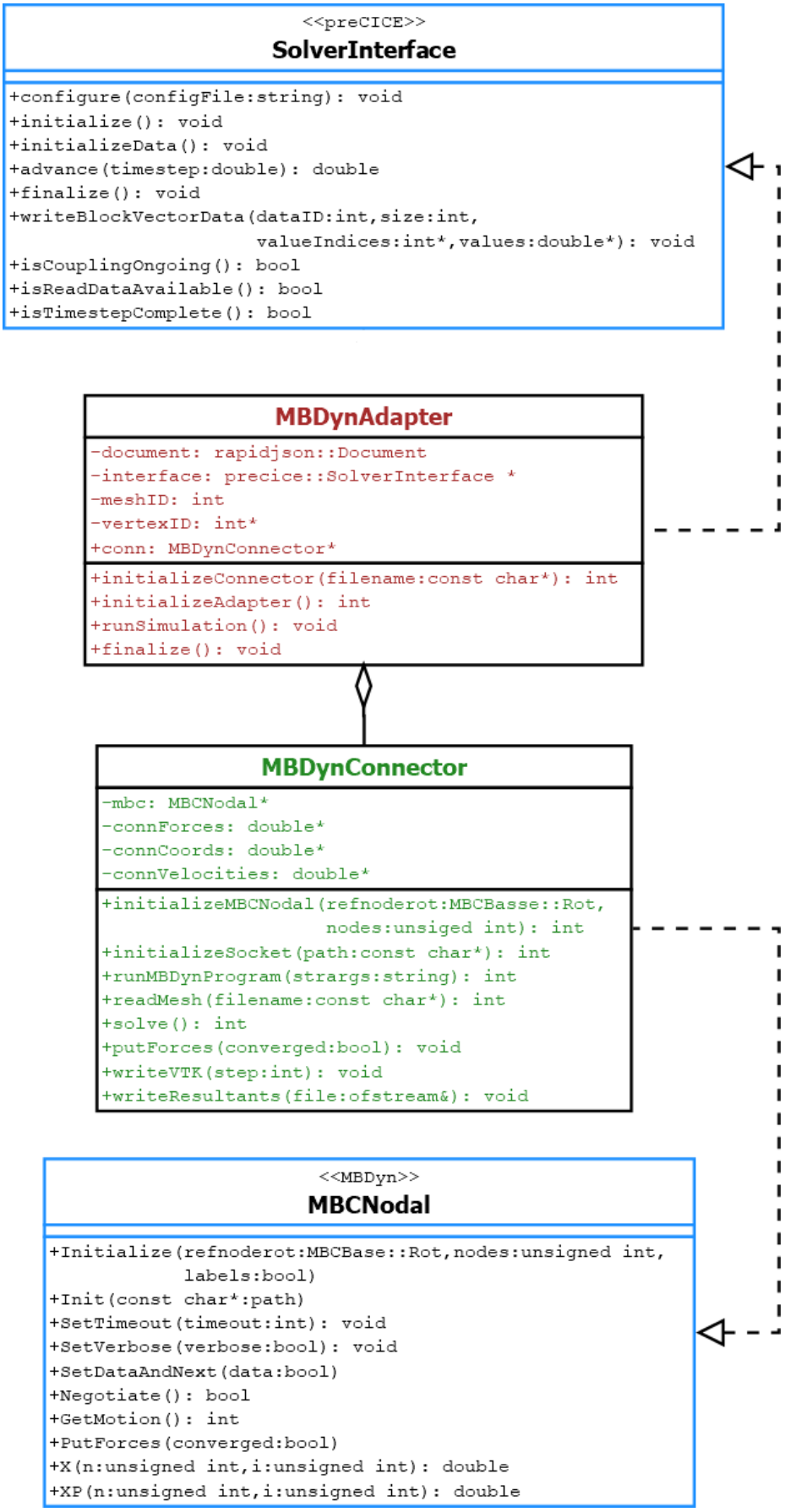


Figure 5.2. MBDyn adapter class structure

5.2.1 Class MBDynAdapter

The file `MBDynAdapter.h` and its source file `MBDynAdapter.cpp` implements all the methods required to perform and FSI simulation with MBDyn as the solid solver. The basic steps are:

1. prepare the MBDyn solver,
2. prepare the interface,
3. provide access to the mesh and initialize the coupling data,
4. steer the coupled simulation,
5. finalize the simulation.

Initialization

In the initialization phase, the instance of `MBDynAdapter` gets a `json` file (see Section 5.3) that contains all the parameters useful for the simulation. Then it instantiates the `MBDynConnector` (see Section 5.2.2) which takes care of all the operations concerning MBDyn: in particular starting the simulation the creating an instance of `MBCNodal` in order to have access to the simulation.

In the next step an instance of `precice::SolverInterface` is initialized and configured with all the relevant information data:

- preCICE configuration file (see Section 4.1.4 and Appendix A).
- *participant* (i.e. solver) name
- information regarding the data to be read and written

The next initialization step concerns the definition of the interface mesh. The data concerning the vertices is stored in the `MBDynConnector` to be used to plot the output and is passed to the `SolverInterface` to define the wet surface nodes. The mesh nodes are stored in the same text file that is used by MBDyn to build the `external structural mapping` information (see Section 4.2.7). This means that the MBDyn mapped points coincide with the interface mesh on the structural side (note that it doesn't have to be the same mesh of the fluid side, as preCICE can map non identical meshes, as described in 4.1.3). The suitable size of memory is then initialized to contain the coupling information: mainly *displacements*, to be written on the preCICE interface, and *forces*, to be read from the interface.

Execution

Finalization

In the finalization phase all the object used during the simulation are closed and memory released.

5.2.2 Class MBDynConnector

- mbdynConnector: connection to MBDyn

5.3 Input parameters

input: json config file
- simulation parameters

5.4 Output results

output: VTK and resultants

Chapter 6

Validation Test-cases

Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Chapter 7

Conclusions

Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt

ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt

ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur.

Appendix A

preCICE configuration file

```
<?xml version="1.0"?>
<precice-configuration>
<solver-interface dimensions="3">

  <data:vector name="Forces" />
  <data:vector name="Displacements" />

  <mesh name="FluidMesh">
    <use-data name="Forces" />
    <use-data name="Displacements" />
  </mesh>
  <mesh name="StructureMesh">
    <use-data name="Forces" />
    <use-data name="Displacements" />
  </mesh>

  <participant name="FluidSolver">
    <use-mesh name="FluidMesh" provide="yes" />
    <use-mesh name="StructureMesh" from="StructureSolver" />
    <write-data name="Forces" mesh="FluidMesh" />
    <read-data name="Displacements" mesh="FluidMesh" />
  </participant>
  <participant name="StructureSolver">
    <use-mesh name="StructureMesh" provide="yes"/>
    <write-data name="Displacements" mesh="StructureMesh" />
    <read-data name="Forces" mesh="StructureMesh" />
  </participant>

  <m2n:sockets from="FluidSolver" to="StructureSolver" />

  <coupling-scheme:serial-implicit>
    <participants first="FluidSolver" second="StructureSolver" />
    <max-time-windows value="10" />
    <time-window-size value="1.0" />
    <max-iterations value="15" />
```

```
36     <relative-convergence-measure limit="1e-3" data="Displacements" mesh="
           StructureSolver"/>
37     <exchange data="Forces" mesh="StructureMesh" from="FluidSolver" to="
           StructureSolver" />
38     <exchange data="Displacements" mesh="StructureMesh" from="
           StructureSolver" to="FluidSolver"/>
39   </coupling-scheme:serial-implicit>
40 </solver-interface>
41 </precice-configuration>
```

Listing A.1. preCICE configuration file example

Appendix B

MBDyn adapter configuration file

```
{
  "readDataName": "Displacements0",
  "writeDataName" :"Forces0",
  "participantName" :"Solid",
  "meshName": "Solid-Mesh",
  "mesh": "./mesh/root0f.dat",
  "root-coords": [0.0, 0.0, 0.0],
  "mbdyn-input": "./map_10n_3x_21j.mbd",
  "mbdyn-output": "./out_mbd",
  "vtk-output": "./output/MBDyn-OF_",
  "precice-config": "../precice-config.xml",
  "displacement-delta": false,
  "write-interval": 10,
  "iterstart": 500,
  "coeff0": 0.05,
  "period": 2000,
  "ramp-type": "linear",
  "resultant-file": "./output/resultant.txt",
  "node-socket": "/tmp/mbdyn.node.sock",
  "pre-iteration": 5,
  "read-coords": true,
  "every-iteration": false
}
```

Listing B.1. MBDyn adapter configuration file example

Appendix C

preCICE API

C.1 preCICE API calls

Each participating solver needs to be modified to link to the preCICE library and call methods from its application programming interface. Usually, the calls to the API are grouped together in a preCICE adapter. While preCICE is written in C++, it provides an API also for C, Fortran and Python. An excerpt from the C++ API is shown in Listing 2.1., as drawn from the preCICE source code documentation

```
1  class SolverInterface
2  {
3  public:
4  SolverInterface(
5  %const std::string& solverName,
6  int solverProcessIndex,
7  int solverProcessSize);
8
9  %void configure(const std::string& configurationFileName);
10
11 double initialize();
12 void initializeData();
13 double advance(double computedTimestepLength);
14 void finalize();
15
16 %int getMeshID(const std::string& meshName);
17 int setMeshVertex(int meshID, const double* position);
18 void setMeshVertices(int meshID, int size, double* positions, int* ids);
19
20 void writeScalarData(int dataID, int valueIndex, double value);
21 void writeVectorData(int dataID, int valueIndex, const double* value);
22 void writeBlockScalarData(
23 int dataID,
24 int size,
25 int* valueIndices,
26 double* values);
27
28 bool isCouplingOngoing();
```

```
29 bool isReadDataAvailable();
30 bool isWriteDataRequired(double computedTimestepLength);
31 bool isTimestepComplete();
32
33 %bool isActionRequired(const std::string& action);
34 %void fulfilledAction(const std::string& action);
35
36 // ...
37 };
```

Listing C.1. preCICE API methods

C.2 preCICE adapter structure

First, a SolverInterface object needs to be created. The constructor expects the rank of the process and the size of the communicator in the parallel execution environment. The configure() method sets the name of the preCICE configuration file, reads and validates it and sets up the communication inside the solver. The methods that follow in Listing 2.1. are called "steering methods". The initialize() method sets up the data structures and communication channels to other participants. Here the first communication happens, as the participants exchange meshes and, if necessary, re-partition them. It returns the maximum time step size that the solver is allowed to execute next. It is followed by the initializeData(), which is optional and transfers any initial non-zero coupling data values among participants. The equation coupling, the data mapping and the communication are all hidden inside the advance() method. This method also returns the maximum time step size allowed. The last method to call is finalize(), which destroys the data structures and closes the communication channels. The solvers need to define their interface meshes. Only some of the methods for mesh definition are listed in Listing 2.1.. Each mesh and each node on the mesh are assigned an integer ID. The getMeshID() gets the ID of the mesh over which the coupling data of the specific kind are exchanged. The setMeshVertex() creates a vertex on the specified mesh and position and returns the id of the vertex. For performance reasons, multiple vertices can be defined at once with setMeshVertices(). Additionally, topological information, such as edges or triangles can be passed to preCICE with additional methods which are not listed here. After defining the meshes, they need to be assigned data values. This is done by methods with names write*Data(), which fill the "buffers" with data from the solver's mesh or with methods read*Data(), which read data from the buffers into the solver's mesh. Since preCICE distinguishes between scalar and vector data, both kinds of methods are available. Again, for performance reasons, blocks of data can be processed together using the respective writeBlock*Data() and readBlock*Data(). Please note that these data are not communicated among participants before the next advance() (or initializeData()). A number of auxiliary methods allow to access information important for the coupling, such as whether the coupled simulation is still running, if writing or reading data is expected or if the current coupling time step has finished successfully. The solver can also inquire if it is required to execute a specific action (e.g. to write or to read a checkpoint) and it can inform preCICE that it fulfilled these tasks [16.]. An example of an adapted solver is shown in Listing 2.2., as published in the presentation article of preCICE [6.]. A more detailed example is presented in the wiki of preCICE6 ..

..

```
turnOnSolver(); //e.g. setup and partition mesh

precice::SolverInterface precice("FluidSolver","precice-config.xml",rank,
    size); // constructor

%const std::string& coric = precice::constants::
    actionReadIterationCheckpoint();
%const std::string& cowic = precice::constants::
    actionWriteIterationCheckpoint();

int dim = precice.getDimension();
int meshID = precice.getMeshID("FluidMesh");
int vertexSize; // number of vertices at wet surface
// determine vertexSize
double* coords = new double[vertexSize*dim]; // coords of vertices at wet
    surface
// determine coordinates
int* vertexIDs = new int[vertexSize];
precice.setMeshVertices(meshID, vertexSize, coords, vertexIDs);
delete[] coords;

int displID = precice.getDataID("Displacements", meshID);
int forceID = precice.getDataID("Forces", meshID);
double* forces = new double[vertexSize*dim];
double* displacements = new double[vertexSize*dim];

double dt; // solver timestep size
double precice_dt; // maximum precice timestep size

precice_dt = precice.initialize();
while (precice.isCouplingOngoing()){
  if(precice.isActionRequired(cowic)){
    saveOldState(); // save checkpoint
    precice.markActionFulfilled(cowic);
  }
  precice.readBlockVectorData(displID, vertexSize, vertexIDs, displacements)
      ;
  setDisplacements(displacements);
  dt = beginTimeStep(); // e.g. compute adaptive dt
  dt = min(precice_dt, dt);
  computeTimeStep(dt);
  computeForces(forces);
  precice.writeBlockVectorData(forceID, vertexSize, vertexIDs, forces);
  precice_dt = precice.advance(dt);
  if(precice.isActionRequired(coric)){ // timestep not converged
    reloadOldState(); // set variables back to checkpoint
    precice.markActionFulfilled(coric);
  }
  else{ // timestep converged
    endTimeStep(); // e.g. update variables, increment time
```

```
46   }
47 }
48 precice.finalize(); // frees data structures and closes communication
       channels
49 delete[] vertexIDs, forces, displacements;
50 turnOffSolver();
```

Listing C.2. preCICE adapter structure

Appendix D

MBDyn input/output file example

D.1 MBDyn input file

Main File

```
1  # $Header: /var/cvs/mbdyn/mbdyn/mbdyn-1.0/tests/forces/strext/socket/
       simplerotor/mapping/simplerotor2_mapping,v 1.7 2017/01/12 15:02:57
       masarati Exp $
2  #
3  # MBDyn (C) is a multibody analysis code.
4  # http://www.mbdyn.org
5  #
6  # Copyright (C) 1996-2017
7  #
8  # Pierangelo Masarati <masarati@aero.polimi.it>
9  # Paolo Mantegazza <mantegazza@aero.polimi.it>
10 #
11 # Dipartimento di Ingegneria Aerospaziale - Politecnico di Milano
12 # via La Masa, 34 - 20156 Milano, Italy
13 # http://www.aero.polimi.it
14 #
15 # Changing this copyright notice is forbidden.
16 #
17 # This program is free software; you can redistribute it and/or modify
18 # it under the terms of the GNU General Public License as published by
19 # the Free Software Foundation (version 2 of the License).
20 #
21 #
22 # This program is distributed in the hope that it will be useful,
23 # but WITHOUT ANY WARRANTY; without even the implied warranty of
24 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
25 # GNU General Public License for more details.
26 #
27 # You should have received a copy of the GNU General Public License
28 # along with this program; if not, write to the Free Software
29 # Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
30 #
```

```
31 # Author: Giuseppe Quaranta <quaranta@aero.polimi.it>
32
33
34 begin: data;
35   problem: initial value;
36 end: data;
37
38 set: real DT = 5e-4;
39 set: real INITIAL_TIME = 0.0;
40 set: real FINAL_TIME = 10.0;
41
42
43 begin: initial value;
44
45   initial time: INITIAL_TIME;
46   final time: FINAL_TIME;
47   time step: DT;
48 # set: const integer TIMESTEP = 1234;
49 # strategy: change, postponed, TIMESTEP; # changes the time step as
       specified by the drive caller labeled TIMESTEP, whose definition is
       postponed (this is simply a placeholder)
50
51   method: ms, .6;
52   nonlinear solver: newton raphson, modified, 5;
53   linear solver: umfpack, colamd, mt, 1;
54 # linear solver: naive, colamd;
55 # linear solver: umfpack;
56
57   tolerance: 1e-6;
58   max iterations: 1000;
59
60   derivatives coefficient: 1e-9;
61   derivatives tolerance: 1e-6;
62   derivatives max iterations: 100;
63
64   output: iterations;
65   output: residual;
66 end: initial value;
67
68 # number of beam3 elements
69 set: integer N = 10;
70
71 begin: control data;
72   structural nodes:
73     +1 # clamped node
74     +2*N # other nodes
75   ;
76   rigid bodies:
77     +2*N # mass of other nodes
78   ;
```

```
79    joints:
80      +1 # clamp
81      +2*N # other total joints on nodes to force 2D
82    ;
83    beams:
84      +N # the whole beam
85    ;
86    forces:
87      +1 # loads the beam
88      +1 # load on last node
89    ;
90  # file drivers:
91  # +1
92  # ;
93  end: control data;
94
95  # root reference
96  set: const integer ROOT = 1;
97  set: const real Xroot = 0.0;
98  set: const real Yroot = 0.0;
99  set: const real Zroot = 0.0;
100 reference: ROOT, Xroot, Yroot, Zroot, eye, null, null;
101
102
103 # material density [kg/m^3]
104 set: real rho = 100.;
105 # beam width (z direction)
106 set: real w = 4.e-3;
107 # beam height (y direction)
108 set: real h = 6.e-4;
109 # beam lenght (x direction)
110 set: real L = 0.04;
111 # lenght of half beam3 element
112 set: real dL = L/(2*N);
113
114 # mass of single chunck
115 set: real m = rho*w*h*dL;
116
117 # elastic properties
118 set: real E = 2.5e5;
119 set: real nu = 0.35;
120 set: real G = E/(2*(1+nu));
121 set: real A = w*h;
122 set: real Jz = 1./12.*w*h^3;
123 set: real Jy = 1./12.*h*w^3;
124 set: real Jp = Jy + Jz;
125 set: real damp = .01;
126
127 set: integer curr_node;
128
```

```
129 begin: nodes;
130   structural: ROOT, static,
131     reference, ROOT, null,  # position
132     reference, ROOT, eye,  # orientation
133     reference, ROOT, null,  # initial velocity
134     reference, ROOT, null;  # angular velocity
135
136   set: curr_node = 2;
137   include: "beam.nod";
138   set: curr_node = 4;
139   include: "beam.nod";
140   set: curr_node = 6;
141   include: "beam.nod";
142   set: curr_node = 8;
143   include: "beam.nod";
144   set: curr_node = 10;
145   include: "beam.nod";
146   set: curr_node = 12;
147   include: "beam.nod";
148   set: curr_node = 14;
149   include: "beam.nod";
150   set: curr_node = 16;
151   include: "beam.nod";
152   set: curr_node = 18;
153   include: "beam.nod";
154   set: curr_node = 20;
155   include: "beam.nod";
156 end: nodes;
157
158 #begin: drivers;
159 # set: const integer INPUT = 200;
160 # file: INPUT, stream,
161 #  stream drive name, "TS_DRV", # irrelevant but needed
162 #  create, yes,
163 #  path, "/tmp/mbdyn.ts.sock", # or use port
164 #  1; # one channel: the time step
165 # drive caller: TIMESTEP, file, INPUT, 1; # replace placeholder with file
      driver
166 #end: drivers;
167
168
169 set: const integer BEAM_NNODES = 2*N+1;
170 set: real da = .005;
171 set: integer CURR_BEAM = 1;
172
173
174 begin: elements;
175   joint: 500+ROOT, clamp, ROOT, node, node;
176
177   set: curr_node = 2;
```

```
178   include: "beam.elm";
179   include: "joint.elm";
180   set: curr_node = 4;
181   include: "beam.elm";
182   include: "joint.elm";
183   set: curr_node = 6;
184   include: "beam.elm";
185   include: "joint.elm";
186   set: curr_node = 8;
187   include: "beam.elm";
188   include: "joint.elm";
189   set: curr_node = 10;
190   include: "beam.elm";
191   include: "joint.elm";
192   set: curr_node = 12;
193   include: "beam.elm";
194   include: "joint.elm";
195   set: curr_node = 14;
196   include: "beam.elm";
197   include: "joint.elm";
198   set: curr_node = 16;
199   include: "beam.elm";
200   include: "joint.elm";
201   set: curr_node = 18;
202   include: "beam.elm";
203   include: "joint.elm";
204   set: curr_node = 20;
205   include: "beam.elm";
206   include: "joint.elm";
207
208     force: CURR_BEAM, external structural mapping,
209     socket,
210     create, yes,
211         path, "/tmp/mbdyn.node.sock",
212         no signal,
213         coupling,
214           # loose,
215           tight,
216     #reference node, 1,
217     #orientation, euler 123,
218     #use reference node forces, yes,
219     points number, 3* BEAM_NNODES,
220       CURR_BEAM,
221         offset, null,
222         offset, 0,da, 0.,
223         offset, 0., 0., da,
224       CURR_BEAM + 1,
225         offset, null,
226         offset, 0,da, 0.,
227         offset, 0., 0., da,
```

```
228       CURR_BEAM + 2,
229         offset, null,
230         offset, 0,da, 0.,
231         offset, 0., 0., da,
232       CURR_BEAM + 3,
233         offset, null,
234         offset, 0,da, 0.,
235         offset, 0., 0., da,
236       CURR_BEAM + 4,
237         offset, null,
238         offset, 0,da, 0.,
239         offset, 0., 0., da,
240       CURR_BEAM + 5,
241         offset, null,
242         offset, 0,da, 0.,
243         offset, 0., 0., da,
244       CURR_BEAM + 6,
245         offset, null,
246         offset, 0,da, 0.,
247         offset, 0., 0., da,
248       CURR_BEAM + 7,
249         offset, null,
250         offset, 0,da, 0.,
251         offset, 0., 0., da,
252       CURR_BEAM + 8,
253         offset, null,
254         offset, 0,da, 0.,
255         offset, 0., 0., da,
256       CURR_BEAM + 9,
257         offset, null,
258         offset, 0,da, 0.,
259         offset, 0., 0., da,
260       CURR_BEAM + 10,
261         offset, null,
262         offset, 0,da, 0.,
263         offset, 0., 0., da,
264       CURR_BEAM + 11,
265         offset, null,
266         offset, 0,da, 0.,
267         offset, 0., 0., da,
268       CURR_BEAM + 12,
269         offset, null,
270         offset, 0,da, 0.,
271         offset, 0., 0., da,
272       CURR_BEAM + 13,
273         offset, null,
274         offset, 0,da, 0.,
275         offset, 0., 0., da,
276       CURR_BEAM + 14,
277         offset, null,
```

```
278         offset, 0,da, 0.,
279         offset, 0., 0., da,
280       CURR_BEAM + 15,
281         offset, null,
282         offset, 0,da, 0.,
283         offset, 0., 0., da,
284       CURR_BEAM + 16,
285         offset, null,
286         offset, 0,da, 0.,
287         offset, 0., 0., da,
288       CURR_BEAM + 17,
289         offset, null,
290         offset, 0,da, 0.,
291         offset, 0., 0., da,
292       CURR_BEAM + 18,
293         offset, null,
294         offset, 0,da, 0.,
295         offset, 0., 0., da,
296       CURR_BEAM + 19,
297         offset, null,
298         offset, 0,da, 0.,
299         offset, 0., 0., da,
300       CURR_BEAM + 20,
301         offset, null,
302         offset, 0,da, 0.,
303         offset, 0., 0., da,
304     # echo, "flap_points.dat", surface, "flap.dat", output, "flap_H.dat",
          order, 2, basenode, 12, weight, 2, stop;
305     mapped points number, 204,
306     sparse mapping file, "flap_H.dat";
307
308 # constant absolute force in node 11
309   force: 2,absolute,
310     2*N + 1,
311     position, null,
312     0., 1., 0.,
313     # slope, initial time, final time / forever, initial value
314     ramp, .0, 0., 1., 0.;
315 end: elements;
316
317
318 # vim:ft=mbd
```

Listing D.1. MBDyn input file example

Beam nodes

```
1 # $Header: /var/cvs/mbdyn/mbdyn/mbdyn-web/documentation/examples/beam.nod,v
    1.2 2008/11/05 20:47:14 masarati Exp $
```

```
2
3  structural: curr_node, dynamic,
4    reference, ROOT, (curr_node - 1) * dL, 0.0, 0.0,
5    reference, ROOT, eye,
6    reference, ROOT, null,
7    reference, ROOT, null;
8
9  structural: curr_node + 1,dynamic,
10   reference, ROOT, curr_node * dL, 0.0, 0.0,
11   reference, ROOT, eye,
12   reference, ROOT, null,
13   reference, ROOT, null;
```

Listing D.2. MBDyn beam nodes

Beam elements and Bodies

```
1  # $Header: /var/cvs/mbdyn/mbdyn/mbdyn-web/documentation/examples/beam.elm,v
       1.2 2008/11/05 20:47:14 masarati Exp $
2
3  # body: BODY_LABEL, NODE_LABEL,
4  # mass
5  # reference node offset
6  # inertia tensor
7
8  body: 1000+ curr_node, curr_node,
9    m,
10   null,
11   diag, 1./12.*(h^2+w^2)*m, 1./12.*(dL^2+w^2)*m, 1./12.*(dL^2+h^2)*m;
12
13 body: 1000+ curr_node + 1,curr_node,
14   m,
15   null,
16   diag, 1./12.*(h^2+w^2)*m, 1./12.*(dL^2+w^2)*m, 1./12.*(dL^2+h^2)*m;
17
18
19 # beam 3 nodes
20 # NODE1_LABEL, offset
21 # NODE2_LABEL, offset
22 # NODE3_LABEL, offset
23 # orientation from node
24 # CONSTUTIVE LAW: EA, GAy, GAz, GJ, EJy, EJz
25 #linear elastic generic, diag,
26 #EA, GAy, GAz, GJ, EJy, EJz,
27
28
29 beam3: 100+curr_node,
30   curr_node - 1,null,
31   curr_node, null,
```

```
32   curr_node + 1,null,
33   eye,
34   linear viscoelastic generic,
35     diag, E*A, G*A*5./6., G*A*5./6., G*Jp, E*Jy, E*Jz,
36         diag, damp*E*A, damp*G*A*5./6., damp*G*A*5./6., damp*G*Jp, damp*E*Jy,
               damp*E*Jz,
37 #  proportional, damp,
38   same,
39   same;
```

Listing D.3. MBDyn beam elements

Joints

```
1  joint: 500+curr_node, total joint,
2    ROOT,
3      position, null,
4      position orientation, eye,
5      rotation orientation, eye,
6    curr_node,
7      position, null,
8      position orientation, eye,
9      rotation orientation, eye,
10
11   position constraint, 0,0,1,null,
12
13   orientation constraint, 1,1,0,null;
14
15 joint: 500+curr_node+1, total joint,
16   ROOT,
17     position, null,
18     position orientation, eye,
19     rotation orientation, eye,
20   curr_node+1,
21     position, null,
22     position orientation, eye,
23     rotation orientation, eye,
24
25   position constraint, 0,0,1,null,
26
27   orientation constraint, 1,1,0,null;
```

Listing D.4. MBDyn joints

D.2 MBDyn output file structure

`.mov` file

For each time step the file contains one row for each node whose output is required. The rows contain the following columns:

- 1: the label of the node
- 2–4: the three components of the position of the node
- 5–7: the three Euler angles that define the orientation of the node
- 8–10: the three components of the velocity of the node
- 11–13: the three components of the angular velocity of the node
- 14–16: the three components of the linear acceleration of the dynamic and modal nodes (optional)
- 17–19: the three components of the angular acceleration of the dynamic and modal nodes (optional)

All the quantities are expressed in the global frame, except for the relative frame type of dummy node, whose quantities are, by definition, in the relative frame.

`.ine` file

This output file refers only to dynamic nodes, and contains their inertia. For each time step, it contains information about the inertia of all the nodes whose output is required. Notice that more than one inertia body can be attached to one node; the information in this file refers to the sum of all the inertia related to the node. The rows contain the following columns:

- 1: the label of the node
- 2–4: item the three components of the momentum in the absolute reference frame
- 5–7: item the three components of the momenta moment in the absolute reference frame, with respect to the coordinates of the node, thus to a moving frame
- 8–10: the three components of the derivative of the momentum
- 11–13: the three components of the derivative of the momentum moment

`.frc` file

An external structural element writes one line for each connected node at each time step in this file. Each line contains the following columns:

- 1: the label of the element and that of the corresponding node; the format of this field is `element_label@node_label`
- 2–4: the three components of the force
- 5–7: the three components of the moment

If a reference node is defined, a special line is output for the reference node, containing the following columns:

- 1: the label of the element and that of the corresponding node; the format of this field is `element_label#node_label`

- 2–4: the three components of the force applied to the reference node, as received from the peer
- 5–7: the three components of the moment applied to the reference node, as received from the peer
- 8–10: the three components of the force that are actually applied to the reference node, in the global reference frame
- 11–13: the three components of the moment with respect to the reference node that are actually applied to the reference node, in the global reference frame
- 14–16: the three components of the force resulting from the combination of all nodal forces, in the global reference frame
- 17–19: the three components of the moment with respect to the reference node resulting from the combination of all nodal forces and moments, in the global reference frame

`.act` file

This type of file contains the output related to beam elements. The internal forces and couples are computed from the interpolated strains along the beam by means of the constitutive law, at the two evaluation points. For each time step and for each element, the format of the columns is:

- 1: the label of the beam
- 2–4: the three components of the force at the first evaluation point
- 5–7: the three components of the couple at the first evaluation point
- 8–10: the three components of the force at the second evaluation point
- 11–13: the three components of the couple at the second evaluation point

`.jnt` file

The output concerning joint elements is generally made of a standard part, plus some extra information depending on the type of joint, which, when available, is described along with the joint description. Here the standard part is described:

- 1: the label of the joint
- 2–4: the three components of the reaction force in a local reference
- 5–7: the three components of the reaction couple in a local frame
- 8–10: the three components of the reaction force in the global frame
- 11–13: the three components of the reaction couple, rotated into the global frame

for total joints:

- 14–16: the three components of the relative displacement in the reference frame of the element

- 17–19: the three components of the relative rotation vector in the reference frame of the element
- 20–22: the three components of the relative velocity in the reference frame of the element
- 23–25: the three components of the relative angular velocity in the reference frame of the element

Acronyms

FSI	Fluid-Structure Interaction
MBDyn	MultiBody Dynamics analysis software
preCICE	precise Code Interaction Coupling Environment
ALE	arbitrary Lagrangian-Eulerian
CFD	Computational Fluid Dynamics
CSM	Computational Solid Mechanics
NSE	Navier Stokes Equations
PDE	Partial Differential Equations
VWP	Virtual Work Principle
FEM	Finite Element Method
AME	added mass effect
FPI	fixed point iteration
IQN-ILS	interface quasi Newton with inverse Jacobian from a least squares model
RBF	Radial Basis Function
API	application programming interface
XML	extensible markup language
MPI	message passing interface
TCP/IP	Transmission Control Protocol/Internet Protocol

Bibliography

[1] G. L. Ghiringhelli, P. Masarati, M. Morandini, and D. Muffo, "Integrated aeroservoelastic analysis of induced strain rotor blades," *Mechanics of Advanced Materials and Structures*, vol. 15, no. 3-4, pp. 291–306, 2008.

[2] R. C. Batra, *Elements of continuum mechanics*. Aiaa, 2006.

[3] J. Cheng, G. Zhao, Z. Jia, Y. Chen, S. Wang, and W. Wen, "Sliding free lagrangian-eulerian finite element method," in *International Conference on Computational Science*, 2006.

[4] J. Donea, A. Huerta, J.-P. Ponthot, and A. Rodríguez-Ferran, "Arbitrary l agrangian–e ulerian methods," *Encyclopedia of Computational Mechanics Second Edition*, pp. 1–23, 2017.

[5] J. T. Xing, "Chapter 3 - fundamentals of continuum mechanics," in *Fluid-Solid Interaction Dynamics*, J. T. Xing, Ed. Academic Press, 2019, pp. 57 – 101. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128193525000033

[6] S. Lipton, J. A. Evans, Y. Bazilevs, T. Elguedj, and T. J. Hughes, "Robustness of isogeometric structural discretizations under severe mesh distortion," *Computer Methods in Applied Mechanics and Engineering*, vol. 199, no. 5-8, pp. 357–373, 2010.

[7] A. Bertram, *Elasticity and plasticity of large deformations*. Springer, 2012.

[8] A. De Boer, M. Van der Schoot, and H. Bijl, "Mesh deformation based on radial basis function interpolation," *Computers & structures*, vol. 85, no. 11-14, pp. 784–795, 2007.

[9] E. Ramm and W. Wall, "Fluid-structure interaction based upon a stabilized (ale) finite element method," in *4th World Congress on Computational Mechanics: New Trends and Applications, CIMNE, Barcelona*, 1998, pp. 1–20.

[10] L. Quartapelle and F. Auteri, *Fluidodinamica incomprimibile*. Casa editrice ambrosiana, 2013.

[11] ——, *Fluidodinamica comprimibile*. Casa editrice ambrosiana, 2013.

[12] S. B. Pope, "Turbulent flows," 2001.

[13] G. Galdi, *An introduction to the mathematical theory of the Navier-Stokes equations: Steady-state problems*. Springer Science & Business Media, 2011.

[14] R. W. Ogden, *Non-linear elastic deformations*. Courier Corporation, 1997.

[15] K. D. Hjelmstad, *Fundamentals of structural mechanics*. Springer Science & Business Media, 2007.

[16] G. L. Ghiringhelli, P. Masarati, and P. Mantegazza, "Multibody implementation of finite volume c beams," *AIAA journal*, vol. 38, no. 1, pp. 131–138, 2000.

[17] G. Hou, J. Wang, and A. Layton, "Numerical methods for fluid-structure interaction—a review," *Communications in Computational Physics*, vol. 12, no. 2, pp. 337–377, 2012.

[18] H. Hanche-Olsen, "Buckingham's pi-theorem," *NTNU: http://www. math. ntnu. no/˜ hanche/notes/buckingham/buckingham-a4. pdf*, 2004.

[19] J.-D. Hardtke, "On buckingham's π-theorem," *arXiv preprint arXiv:1912.08744*, 2019.

[20] R. W. Fox, A. T. McDonald, and J. W. Mitchell, *Fox and McDonald's introduction to fluid mechanics*. John Wiley & Sons, 2011.

[21] S. Longo, *Analisi Dimensionale e Modellistica Fisica: Principi e applicazioni alle scienze ingegneristiche*. Springer Science & Business Media, 2011.

[22] E. De Langre, *Fluides et solides*. Editions Ecole Polytechnique, 2001.

[23] B. Hübner, E. Walhorn, and D. Dinkler, "A monolithic approach to fluid–structure interaction using space–time finite elements," *Computer methods in applied mechanics and engineering*, vol. 193, no. 23-26, pp. 2087–2104, 2004.

[24] P. Ryzhakov, R. Rossi, S. Idelsohn, and E. Oñate, "A monolithic lagrangian approach for fluid–structure interaction problems," *Computational mechanics*, vol. 46, no. 6, pp. 883–899, 2010.

[25] T. Richter, *Fluid-structure interactions: models, analysis and finite elements*. Springer, 2017, vol. 118.

[26] J. Degroote, K.-J. Bathe, and J. Vierendeels, "Performance of a new partitioned procedure versus a monolithic procedure in fluid–structure interaction," *Computers & Structures*, vol. 87, no. 11-12, pp. 793–801, 2009.

[27] C. Farhat, K. G. Van der Zee, and P. Geuzaine, "Provably second-order time-accurate loosely-coupled solution algorithms for transient nonlinear computational aeroelasticity," *Computer methods in applied mechanics and engineering*, vol. 195, no. 17-18, pp. 1973–2001, 2006.

[28] M. Mehl, B. Uekermann, H. Bijl, D. Blom, B. Gatzhammer, and A. Van Zuijlen, "Parallel coupling numerics for partitioned fluid–structure interaction simulations," *Computers & Mathematics with Applications*, vol. 71, no. 4, pp. 869–891, 2016.

[29] E. H. van Brummelen, "Added mass effects of compressible and incompressible flows in fluid-structure interaction," *Journal of Applied mechanics*, vol. 76, no. 2, 2009.

[30] U. Küttler and W. A. Wall, "Fixed-point fluid–structure interaction solvers with dynamic relaxation," *Computational mechanics*, vol. 43, no. 1, pp. 61–72, 2008.

[31] B. M. Irons and R. C. Tuck, "A version of the aitken accelerator for computer iteration," *International Journal for Numerical Methods in Engineering*, vol. 1, no. 3, pp. 275–277, 1969.

[32] B. W. Uekermann, "Partitioned fluid-structure interaction on massively parallel systems," Ph.D. dissertation, Technische Universität München, 2016.

[33] R. Haelterman, J. Degroote, D. Van Heule, and J. Vierendeels, "The quasi-newton least squares method: a new and fast secant method analyzed for linear systems," *SIAM Journal on numerical analysis*, vol. 47, no. 3, pp. 2347–2368, 2009.

[34] B. Uekermann, H.-J. Bungartz, B. Gatzhammer, and M. Mehl, "A parallel, black-box coupling algorithm for fluid-structure interaction," in *Proceedings of 5th International Conference on Computational Methods for Coupled Problems in Science and Engineering*, 2013, pp. 1–12.

[35] R. Haelterman, A. E. Bogaers, K. Scheufele, B. Uekermann, and M. Mehl, "Improving the performance of the partitioned qn-ils procedure for fluid–structure interaction problems: Filtering," *Computers & Structures*, vol. 171, pp. 9–17, 2016.

[36] D. Blom, F. Lindner, M. Mehl, K. Scheufele, B. Uekermann, and A. van Zuijlen, "A review on fast quasi-newton and accelerated fixed-point iterations for partitioned fluid–structure interaction simulation," in *Advances in Computational Fluid-Structure Interaction and Flow Simulation*. Springer, 2016, pp. 257–269.

[37] F. Lindner, M. Mehl, K. Scheufele, and B. Uekermann, "A comparison of various quasi-newton schemes for partitioned fluid-structure interaction," in *Proceedings of 6th International Conference on Computational Methods for Coupled Problems in Science and Engineering, Venice*, 2015, pp. 1–12.

[38] T. Kajishima and K. Taira, "Immersed boundary methods," in *Computational Fluid Dynamics*. Springer, 2017, pp. 179–205.

[39] R. van Loon, P. D. Anderson, F. N. van de Vosse, and S. J. Sherwin, "Comparison of various fluid–structure interaction methods for deformable bodies," *Computers & structures*, vol. 85, no. 11-14, pp. 833–843, 2007.

[40] C. Degand and C. Farhat, "A three-dimensional torsional spring analogy method for unstructured dynamic meshes," *Computers & structures*, vol. 80, no. 3-4, pp. 305–316, 2002.

[41] A. O. González, A. Vallier, and H. Nilsson, "Mesh motion alternatives in openfoam," *PhD course in CFD with OpenSource software*, 2009.

[42] H.-J. Bungartz, F. Lindner, B. Gatzhammer, M. Mehl, K. Scheufele, A. Shukaev, and B. Uekermann, "precice–a fully parallel library for multi-physics surface coupling," *Computers & Fluids*, vol. 141, pp. 250–258, 2016.

[43] F. Lindner, M. Mehl, and B. Uekermann, "Radial basis function interpolation for black-box multi-physics simulations," 2017.

[44] S. Chen, M. t. Wambsganss, and J. Jendrzejczyk, "Added mass and damping of a vibrating rod in confined viscous fluids," in *asme*, 1976.

[45] C. Conca, A. Osses, and J. Planchard, "Added mass and damping in fluid-structure interaction," *Computer methods in applied mechanics and engineering*, vol. 146, no. 3-4, pp. 387–405, 1997.

[46] J. Gauthier, A. Giroux, S. Etienne, and F. Gosselin, "A numerical method for the determination of flow-induced damping in hydroelectric turbines," *Journal of Fluids and Structures*, vol. 69, pp. 341–354, 2017.

[47] G. Ricciardi and E. Boccaccio, "Modelling of the flow induced stiffness of a pwr fuel assembly," *Nuclear Engineering and Design*, vol. 282, pp. 8–14, 2015.

[48] P. Causin, J.-F. Gerbeau, and F. Nobile, "Added-mass effect in the design of partitioned algorithms for fluid–structure problems," *Computer methods in applied mechanics and engineering*, vol. 194, no. 42-44, pp. 4506–4527, 2005.

[49] J. Degroote, P. Bruggeman, R. Haelterman, and J. Vierendeels, "Stability of a coupling technique for partitioned solvers in fsi applications," *Computers & Structures*, vol. 86, no. 23-24, pp. 2224–2234, 2008.

[50] T. Bodnár, G. P. Galdi, and Š. Nečasová, *Fluid-structure interaction and biomedical applications*. Springer, 2014, ch. 13.

[51] C. Förster, W. A. Wall, and E. Ramm, "The artificial added mass effect in sequential staggered fluid-structure interaction algorithms," in *ECCOMAS CFD 2006: Proceedings of the European Conference on Computational Fluid Dynamics, Egmond aan Zee, The Netherlands, September 5-8, 2006*. Delft University of Technology; European Community on Computational Methods . . . , 2006.

[52] B. Gatzhammer, "Efficient and flexible partitioned simulation of fluid-structure interactions," Ph.D. dissertation, Technische Universität München, 2014.

[53] A. K. Shukaev, "A fully parallel process-to-process intercommunication technique for precice." Master's thesis, Technische Universitt München, Jun. 2015.

[54] B. Uekermann, H.-J. Bungartz, L. Cheung Yau, G. Chourdakis, and A. Rusch, "Official precice adapters for standard open-source solvers," in *Proceedings of the 7th GACM Colloquium on Computational Mechanics for Young Scientists from Academia*, 2017.

[55] P. Masarati, M. Morandini, and P. Mantegazza, "An efficient formulation for general-purpose multibody/multiphysics analysis," *Journal of Computational and Nonlinear Dynamics*, vol. 9, no. 4, 2014.

[56] V. Giavotto, M. Borri, P. Mantegazza, G. Ghiringhelli, V. Carmaschi, G. Maffioli, and F. Mussi, "Anisotropic beam theory and applications," *Computers & Structures*, vol. 16, no. 1-4, pp. 403–413, 1983.

[57] D. H. Hodges, "Review of composite rotor blade modeling," *AIAA journal*, vol. 28, no. 3, pp. 561–565, 1990.

[58] T. Kim, A. M. Hansen, and K. Branner, "Development of an anisotropic beam finite element for composite wind turbine blades in multibody system," *Renewable Energy*, vol. 59, pp. 172–183, 2013.

[59] P. Masarati, "A formulation of kinematic constraints imposed by kinematic pairs using relative pose in vector form," *Multibody System Dynamics*, vol. 29, no. 2, pp. 119–137, 2013.