V International Conference on Computational Methods in Marine Engineering
MARINE 2013
B. Brinkmann and P. Wriggers (Eds)

CONCEPT AND REALIZATION OF THE COUPLING SOFTWARE EMPIRE IN MULTIPHYSICS CO-SIMULATION

T. WANG, S. SICKLINGER, R. WÜCHNER AND K.-U. BLETZINGER

Lehrstuhl für Statik
Technische Universität München
Arcisstr. 21, D-80333 München, Germany
e-mail: tianyang.wang@tum.de, stefan.sicklinger@tum.de, wuechner@tum.de, kub@tum.de,
web page: http://www.st.bv.tum.de

Key words: multiphysics, co-simulation, partitioned analysis, fluid-structure interaction

Abstract. The purpose of the software EMPIRE is to perform n-code co-simulation for solving multiphysics problems. EMPIRE provides a flexible way for constructing various co-simulation environments by introducing the concepts of *connection* and *filter*. It also provides data operations useful for general co-simulation including mapping between non-matching grids, extrapolation in time and relaxation in iterative coupling. The concepts and ingredients of EMPIRE are presented in this paper. Finally, the software is demonstrated by two FSI simulations.

1 INTRODUCTION

Multiphysics problems can be formulated by coupled systems of partial differential equations and ordinary differential equations. Two classical strategies for solving coupled systems are the monolithic strategy and the partitioned strategy. In the monolithic strategy, the global equation of the coupled systems is formulated and solved, whereas in the partitioned strategy, the single systems are solved separately and coupled together by exchanging information at the interfaces. With the partitioned strategy, different simulation codes can work together in a co-simulation to solve multiphysics problems, see Figure 1. From the software development point of view, partitioned strategy has a big advantage of reusing existing and well-tested simulation codes for single field problems.

One example of multiphysics is the simulation of wind turbines to study the fluid-structure-interaction (FSI) between the wind load and the turbine blades, see Figure 2. Usually, computational structural mechanics (CSM) and computational fluid dynamics (CFD) are coupled together to simulate the wind turbine with a fixed rotational speed. If the real time rotational speed of the wind turbine is needed, the model of the power

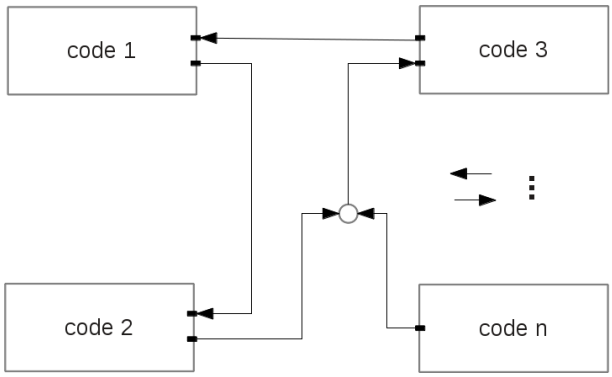

Figure 1: Co-simulation with n codes

generator should be added to the co-simulation. Additional simulation codes can be added to the co-simulation to simulate more complex scenarios.

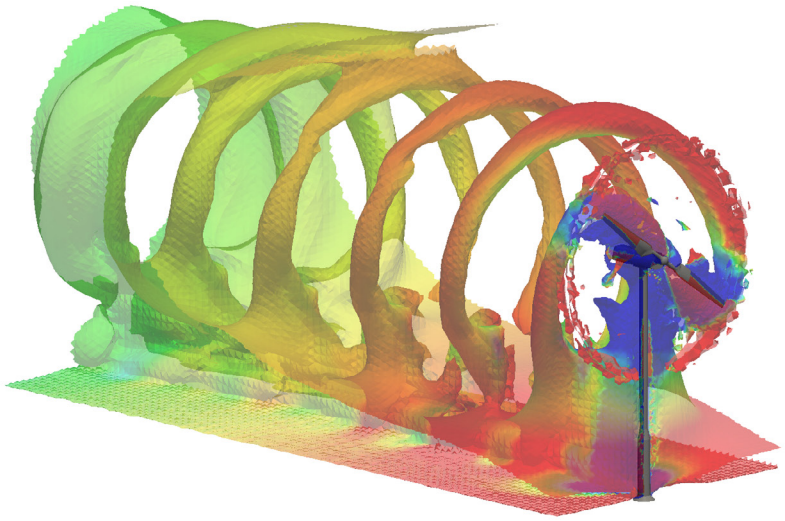

Figure 2: FSI simulation of a wind turbine

The newly designed software EMPIRE (Enhanced MultiPhysics Interface Research Engine) enables co-simulation with n-codes, to solve general multiphysics problems. The concepts of EMPIRE will be introduced in the next section.

2 CONCEPTS OF EMPIRE

Figure 3 shows co-simulation in the EMPIRE environment. The software EMPIRE has two components, the coupling code Emperor and the library EMPIRE_API for the simulation codes to communicate with Emperor. The data communication instance between codes is called *connection*. Inside a connection, the data can be manipulated by

operators called *filters*.

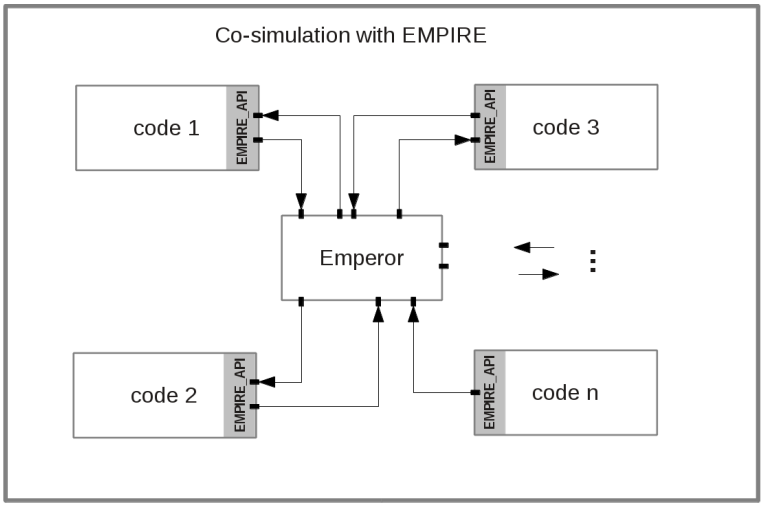

Figure 3: Co-simulation with EMPIRE

2.1 Coupling code Emperor

The coupling code Emperor couples all simulation codes together to perform co-simulation. Having a separate coupling code brings advantages including:

- the simulation codes only communicate with Emperor, without having the knowledge of the other codes in the co-simulation environment;
- unified format for data communication is defined by Emperor, which is followed by all simulation codes;
- the co-simulation environment can be defined in a single file that is read by Emperor;
- the data operations during communication are contained within Emperor which eliminates the need to modify existing simulation codes.

The server-client model is adopted with Emperor as the server and the simulation codes as the clients. The server opens a port at the beginning of the co-simulation to which an arbitrary number of clients can connect. The communication between the server and the client is performed using MPI-2.2.

2.2 Communication Library EMPIRE_API

The simulation codes communicate with Emperor using EMPIRE_API. It is a library containing functions such as connecting to and disconnecting from Emperor, communicating data of a certain type (e.g. mesh, degree of freedom, signal, etc) with Emperor.

EMPIRE_API is written in C++, but interfaced by C functions. As an example, the function of sending a finite element mesh to Emperor is quoted below:

```
void EMPIRE_API_sendMesh(int numNodes, int numElems, double *nodes, int
*nodeIDs, int *numNodesPerElem, int *elems);
```

where the input arguments consist of the number of nodes, the number of elements, the coordinates of all nodes, the IDs of all nodes, the number of nodes of all elements and the connectivity table of all elements.

EMPIRE_API can be compiled and linked together with simulation codes written in languages compatible with C, e.g. C, C++, Fortran, Python, Java, MATLAB, etc.

2.3 Definition of the Co-simulation Scenario by Connections

A connection is the data communication between different codes. It can have multiple inputs and outputs, which makes it more flexible for different scenarios. The inputs are the data sent from simulation codes and received by Emperor, and the outputs are the data sent from Emperor and received by simulation codes, see Figure 4. The co-simulation scenario can be defined by sequences and iterations of connections.

Figure 4: Connection in Emperor

To show how the co-simulation scenario is defined, the co-simulation of a wind turbine with three codes is taken as an example. The three codes are the CFD code, the CSM code and the generator code respectively. The CFD code can be expressed as

$$f_F = F(u_F), \tag{1}$$

where u_F is the displacements of the turbine, and f_F is the wind forces acting on the turbine. The code of the generator can be expressed as

$$T_R = G(T_A), \tag{2}$$

where T_A is the acting torque on the turbine shaft (computed from the wind forces on the turbine blades) and T_R is the reacting torque caused by the generator. The CSM code can be expressed as

$$u_S = S(f_S, T_{RS}), \tag{3}$$

where f_S is the fluid forces on the turbine, and T_{RS} is the reacting torque. The connections between the codes are defined in the left diagram in Figure 5 (where Emperor is omitted but the data must go through it). During the co-simulation, the CFD code computes the wind forces on the turbine and sends them to the generator code (connection 1). Then the generator computes the reacting torque according to the wind forces and sends it to the CSM code (connection 2). Then the CFD code also sends the wind forces to the CSM code (connection 3). Finally the CSM code can compute the displacements of the turbine according to both the wind forces and the reacting torque on the shaft and then send the displacements to the CFD code (connection 4). This completes one iteration. In the flowchart in Figure 5, the sequences and iterations of the connections are shown. The connections are put inside two nested iterations. The outer iteration is the time stepping, and the inner iteration is the iterative coupling which means the connections are run iteratively until the data are converged.

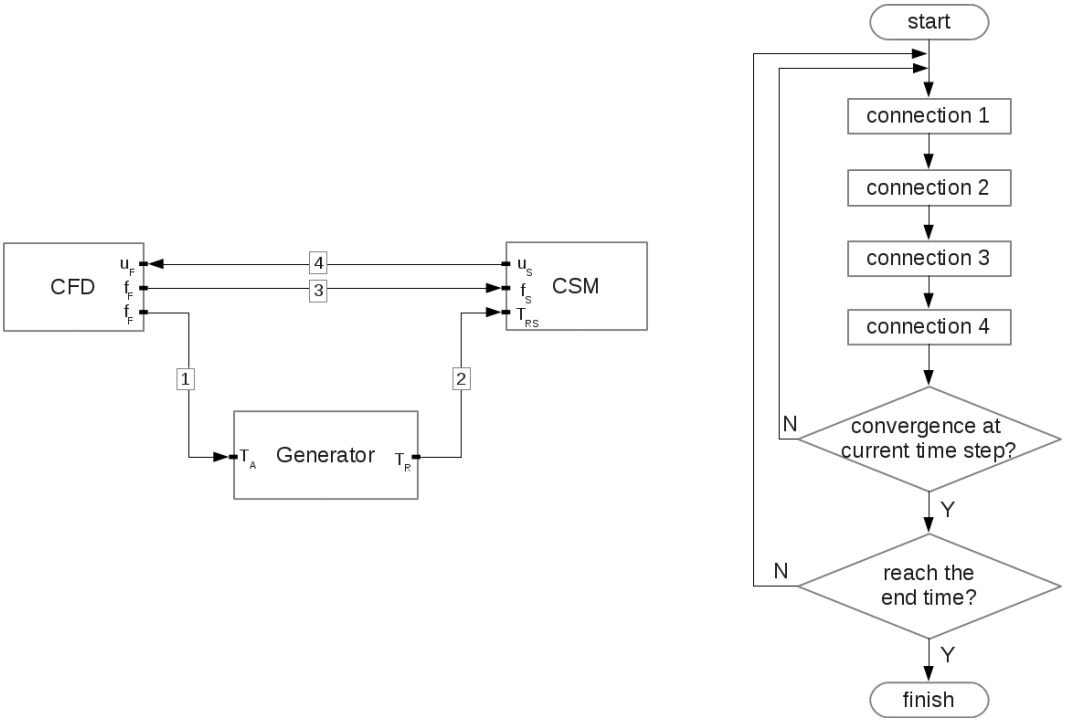

Figure 5: Co-simulation scenario of a wind turbine simulation

The connections are defined in the input XML file of Emperor, as well as the sequences and iterations of the connections. Emperor performs data communication accordingly, i.e. send or receive data at the expected time. However, because the simulation codes are running without being controlled by Emperor, the communication would fail if some simulation code does not send or receive data at the expected time. To assist a user in setting up a co-simulation, Emperor can output pseudo code for each simulation code

specifying how to do communication. The communication will work as expected if each simulation code follows the communication pattern in the pseudo code.

2.4 Filters in Connection

The inputs and outputs of a connection are generally different. Therefore, a sequence of operators may be applied which are named filters in EMPIRE. A filter can have multiple inputs and outputs. The filters inside a connection are shown in Figure 6.

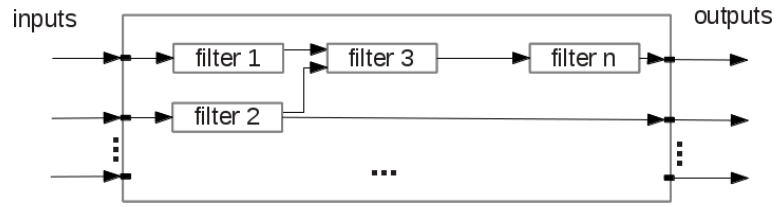

Figure 6: Filters inside a connection

Filters that are usually used in a co-simulation are introduced below.

2.4.1 Mapping Filter

At the interface between two surface coupled domains, the grids from both sides are usually non-matching, so the data cannot be assigned between the grids directly. In Emperor, the mortar method is implemented to map data between non-matching grids. The condition for mapping a data field from domain A to domain B is

$$u_B(x) = u_A(x), \tag{4}$$

where x is the coordinates, and $u_A(x)$ and $u_B(x)$ are the data field on A and B, respectively. By discretizing the fields with finite element method one has

$$\begin{aligned} u_A(x) &= N_A^T \cdot u_A, \\ u_B(x) &= N_B^T \cdot u_B, \end{aligned} \tag{5}$$

where u_A and u_B are values of $u_A(x)$ and $u_B(x)$ on grid points, and N_A and N_B are the shape function vectors of A and B, respectively. Mortar method applies weighted residual approach for (4) using N_B as the test function vector as

$$\int_B N_B(u_B(x) - u_A(x))\, dB = 0. \tag{6}$$

Apply (5) in (6) one has

$$\int_B N_B \cdot N_B^T\, dB \cdot u_B = \int_B N_B \cdot N_A^T\, dB \cdot u_A, \tag{7}$$

which can be reorganized in matrix-vector form as

$$C_{BB} \cdot u_B = C_{BA} \cdot u_A. \tag{8}$$

Since C_{BB} and C_{BA} are both sparse matrices, solving (8) takes much less computational effort than the case of full matrices. In (7), if the test functions are replaced by the dual shape functions, C_{BB} becomes a diagonal matrix [1, 2] so that (8) can be solved by computing the inverse matrix of C_{BB}. It is shown in [4] that using mortar method in FSI with a certain condition, both displacements and pressures can be mapped consistently while the energy being conserved. This is the advantage of mortar method compared with interpolation-like methods.

In Figure 7, it is shown that mortar method works well in mapping prototyped data field on a curved surface.

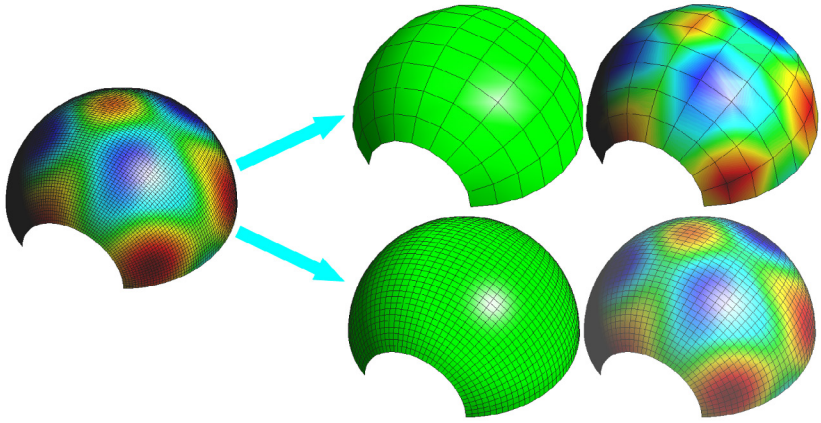

Figure 7: Mapping data field on a spherical surface with mortar method

2.4.2 Extrapolation Filter

At the beginning of a new time step, a prediction of the data at the coupled interface may have to be made. For example in Figure 5, at the beginning of time step $n+1$, the CFD code does not have the turbine displacements u_F^{n+1}. Therefore a prediction of it ($\hat{u}_F^{n+1}$) is computed and used in (1) as

$$f_F^{n+1} = F(\hat{u}_F^{n+1}). \tag{9}$$

The prediction is usually computed by extrapolation of values of previous time steps. The simplest extrapolation method is to use the value from the last time step as $\hat{u}_F^{n+1} = u_F^n$. More complicated extrapolation methods can be found in [5, 6].

2.4.3 Relaxation Filter

In iterative coupling, relaxation can be used to stabilize the simulation. Assume the input and output of the relaxation filter are x_{in} and x_{out} respectively, then at the $k+1$ iteration, the new output is computed by

$$\begin{aligned} x_{out}^{k+1} &= x_{out}^{k} + \omega(x_{in}^{k+1} - x_{out}^{k}) \\ &= (1-\omega)x_{out}^{k} + \omega x_{in}^{k+1}, \end{aligned} \tag{10}$$

where ω is the relaxation factor. It can be constant for all iterations, or can be computed dynamically in each iteration by Aitken's Δ^2 method to accelerate the convergence [7].

3 NUMERICAL EXAMPLES OF FSI

The example driven cavity with flexible bottom introduced in [9] is simulated, see Figure 8. The fluid domain is a 2D square with a periodic velocity imposed at the top. An inlet and an outlet are shown at the top of the left and right wall respectively. The bottom is modelled by a membrane structure. Figure 9 shows the result at the time $t = 4.25$(s).

Another example is the FSI benchmark by Turek [8], where the incompressible fluid flows around a cylinder and an elastic bar behind it, see Figure 10. The test case FSI3 in [8] is simulated, and the result at the time $t = 3.9$(s) is shown in Figure 11.

4 CONCLUSIONS

EMPIRE is designed and implemented for solving general multiphysics problems with n-code co-simulation. It has the following characteristics:

- Flexibility: co-simulation in a general scenario is available, since an arbitrary number of simulation codes and arbitrary connections among them is allowed.
- Modularity: partitioned strategy is used to apply existing simulation codes; the simulation codes are connected by Emperor; communication interface is provided in the library EMPIRE_API; object oriented programming with C++ is used to implement EMPIRE.
- Efficiency: communication is realized by using MPI; efficient mortar method is implemented for mapping data between non-matching grids; methods are implemented to accelerate co-simulation, e.g. extrapolation and relaxation.
- Usability: co-simulation is set up in a single file (input XML file of Emperor); simulation codes in all C-compatible languages can link to the library EMPIRE_API; various filters are provided for data operations including mapping, extrapolation and relaxation.

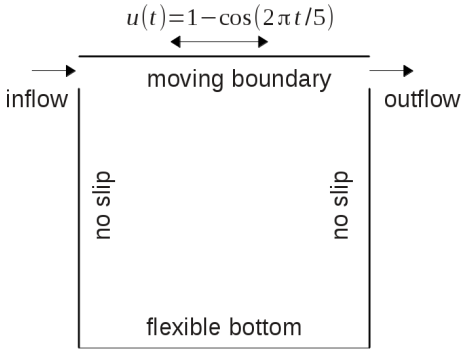

Figure 8: Driven cavity with flexible bottom

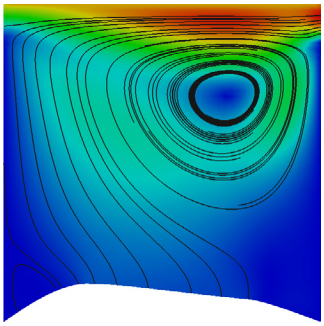

Figure 9: Result – velocity field and streamline at $t = 4.25$(s)

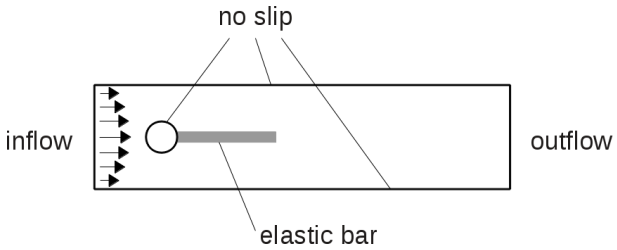

Figure 10: Flow around cylinder and an elastic bar behind it

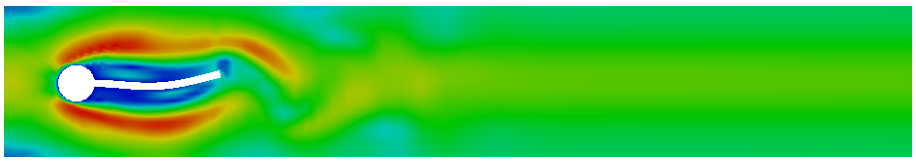

Figure 11: Result – velocity field at $t = 3.9$(s)

The next steps of EMPIRE include new coupling algorithms for n-code coupling where traditional coupling algorithms may have problems, and allowing adjustable time step length for simulation codes. The software will be further verified by examples of large scale n-code co-simulation.

REFERENCES

[1] Hartmann, S., Brunssen, S., Ramm, E. and Wohlmuth, B. Unilateral non-linear dynamic contact of thin-walled structures using a primal-dual active set strategy. *Int. J. Numer. Meth. Engng* (2007) **70**:883–912.

[2] Klöppel, T., Popp, A., Küttler, U. and Wall, W.A. Fluid-structure interaction for non-conforming interfaces based on a dual mortar formulation. *Comput. Methods Appl. Mech. Engrg* (2011) **200**:3111–3126.

[3] Unger, R., Haupt, M.C. and Horst, P. Application of Lagrange multipliers for coupled problems in fluid and structural interactions. *Computers & Structures.* (2007) **85**:796–809.

[4] Boer, de A., Zuijlen, van A.H. and Bijl, H. Comparison of conservative and consistent approaches for the coupling of non-matching meshes. *Comput. Methods Appl. Mech. Engrg* (2008) **197**:4284–4297.

[5] Felippa, C.A., Park, K.C. and Farhat, C. Partitioned analysis of coupled mechanical systems. *Comput. Methods Appl. Mech. Engrg* (2001) **190**:3247–3270.

[6] Farhat, C. and Piperno, S. Partitioned procedures for the transient solution of coupled aeroelastic problems – Part II: energy transfer analysis and three dimensional applications. *Comput. Methods Appl. Mech. Engrg* (2001) **190**:3147–3170.

[7] Küttler, U. and Wall, W.A. Fixed-point fluid-structure interaction solvers with dynamic relaxation. *Comput. Mech.* (2008) **43**:61–72.

[8] Turek, S. and Hron, J. Proposal for numerical benchmarking for fluid-structure interaction between an elastic object and laminar incompressible flow. In: Bungartz, H.J. and Schäfer, M. (Eds), *Fluid-structure interaction: modelling, simulation, optimization*, Springer, (2006) 371–385.

[9] Mok, D.P. and Wall, W.A. Partitioned analysis schemes for the transient interaction of incompressible flows and nonlinear flexible structures. In: Wall, W.A., Bletzinger, K.-U. and Schweitzerhof, K. (Eds), *Proceedings of trends in computational structural mechanics*, CIMNE, (2001) 689–698.