FLECS, a Flexible Coupling Shell
Application to Fluid-Structure Interaction *

Margreet Nool[1], Erik Jan Lingen[2], Aukje de Boer[3], and Hester Bijl[3]

[1] CWI, Department of Computing and Control, Amsterdam, The Netherlands
[2] Habanera Software company, Delft, The Netherlands
[3] TU Delft, Faculty of Aerospace Engineering, Delft, The Netherlands

Abstract. Numerical simulations involving multiple, physically different domains can be solved effectively by coupling simulation programs, or *solvers*. The coordination of the different solvers is commonly handled by a *coupling shell*. A coupling shell synchronizes the execution of the solvers and handles the transfer of data from one physical domain to another. In this paper, we introduce FLECS, a flexible coupling shell, designed for implementing and applying an interface for multidisciplinary simulations with superior accuracy. The aim is not to achieve the best possible efficiency or to support a large feature set, but to provide a flexible platform for developing new data transfer algorithms and coupling schemes.

1 Introduction

Fluid-Structure Interaction (FSI) considers coupled fluid-solid problems, characterized by the interaction of fluid forces and structural deformations, which occur in many applications in industry and science. Nowadays, the simulation of FSI becomes more and more important, since future structures become lighter and more flexible and can be applied, e.g., to reduce the load on turbine blades, or, to reduce the noise on cars. Such applications require a real interdisciplinary approach, that can deal with complex physical models and very different scales.

The Faculty of Aerospace Engineering of the Delft University of Technology has started a project to develop a generic, open-source coupling shell, named FLECS [2], that can be used to join two or more arbitrary solvers. FLECS should provide an innovative combination of high order coupling in space and time. Moreover, to improve the accuracy and the efficiency of the computation, multilevel acceleration techniques for the coupling process [6], and fast prototyping and parallelization techniques will be supported.

The majority of coupling shells are embedded subprograms that have been developed for coupling two specific solvers. One exception is the coupling library MPCCI (Mesh based Parallel Code Coupling) [4], which can be used as a separate program. Although MPCCI is relatively easy to use and provides

* Funding for this work was provided by the National Computing Facilities Foundation (NCF), under project numbers NRG-2005.03.

many advanced features, it is less suitable for a scientific research community that is aimed at developing new data transfer algorithms. Numerical acceleration algorithms, like Krylov and multilevel methods - urgently required for efficiency - are not incorporated. Moreover, since MpCCI only provides the binary code, the user can not modify the implementation schedule of MpCCI.

In addition to accurate coupling in space, we want to reduce the partitioning errors in time by using specially designed high-order time integration methods [5]. It is our goal for FLECS to support solvers that run on parallel computers, in order to make FLECS suitable for large applications. In particular, FLECS must be able to deal with data sets that have been distributed over multiple parallel processes. In addition, FLECS should support the implementation of parallel data transfer algorithms. At the present stage of the project, experiments are limited to sequential solvers; each solver, running on its own processing unit, deals with one particular physical domain. The coupling server, running on a third processing unit, takes care of the exchange of data between the solvers. Since FLECS has to deal with separate solver processes, that have been started independently, it is not possible to use `MPI-1` for exchanging data between those processes. For that reason, FLECS is based on `MPI-2` [3].

The remainder of the paper is structured as follows. In Sect. 2, we describe the importance of coupling two solvers properly to solve interdisciplinary problems in an efficient and accurate way. Sect. 3 gives an overview of the design of FLECS. Through the help of a test problem, Sect. 4 illustrates how FLECS can be used. And finally, Sect. 5 contains some conclusions and future plans.

2 Coupling Methods

Interdisciplinary problems can be solved in two ways. In the first way, the so-called *monolithic* approach, a new dedicated solver is developed that solves the whole system at once. Major advantage is that the solver can be optimized for the specific problem. Development of such a complex, entirely new solver, will take an enormous effort, while there already exists many highly efficient and accurate solvers for the separate domains. The other way, called the *partitioned* approach, is to reuse monodisciplinary solvers, that have been developed and tuned for tens of years. In that case, each physical system is solved individually and interaction effects are treated as external conditions. A disadvantage of this approach is that the coupling algorithm is not as straightforward as it looks. Without much care the accuracy of the coupled problem easily reduces to first-order in time, irrespective of the order of the separate solvers.

FLECS provides an efficient coupling interface for partitioned computation of multidisciplinary problems. The design of FLECS allows all kinds of data transfer algorithms to couple different domains in space and time. Numerical acceleration techniques, like multigrid, can be incorporated, too.

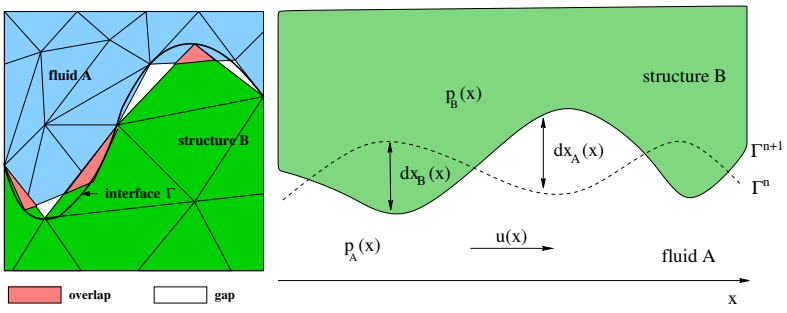

Fig. 1. (a) Non-matching grids in 2D and (b) Configuration of the quasi-1D test problem

2.1 Non-matching Grids

If different grid generators are used for both domains, the mesh interface may not only be non-conforming (nodes at the interfaces do not match, different discretization and/or interpolation order on both discrete interfaces), but also non-matching (cf. Fig. 1(a)) in the way that there are gaps and/or overlaps between the meshes. We remark, that generating matching grids is most of the time not desirable, because, in general, the simulation on one physical domain requires a much finer grid than the simulation on the other one. In the remainder of this paper, we consider FSI rather than some arbitrary physical domain interaction.

De Boer et al. [1] gives a detailed study of data transfer methods, and FSI simulations are performed on non-matching meshes. As coupling method in this paper, we take the radial basis function method (RBF) with large compact support. This method, favored by [1], because of its accuracy and efficiency, does not need orthogonal projection or search algorithms. The coupling between fluid and structure equations arises from the dynamic and kinematic boundary conditions (BC) at the fluid-structure interface. The BC for the displacement on the continuous fluid-structure interface Γ, given by $d\mathbf{x}_A(x) = d\mathbf{x}_B(x)$, where $d\mathbf{x}$ denotes the displacement on either the flow (A) or structure (B) side of the interface. The displacements of the flow points have to be predicted once the displacements of the structure points have been determined. The discrete version of the BC can be formulated as

$$\mathrm{d}\mathbf{x}_A = \mathbf{H}_{AB}\mathrm{d}\mathbf{x}_B,$$

where $\mathbf{H}_{AB} \in \mathbb{R}^{N_A \times N_B}$ is the transformation matrix prescribed by the RBF method. The numbers N_A, the number of flow, and N_B, the number of structure points on the fluid-structure interface, are usually very small compared to the total number of structure and flow points. Analogously, the discrete version of the BC for the pressure forces leads to

$$\mathbf{p}_B = \mathbf{H}_{BA}\mathbf{p}_A,$$

where the transformation matrix $\mathbf{H}_{BA}$ is of size $N_B \times N_A$. The computation of the matrices $\mathbf{H}_{AB}$ and $\mathbf{H}_{BA}$ involves the inversion of a small matrix. The matrices $\mathbf{H_{AB}}$ and $\mathbf{H_{BA}}$ depend on the coordinates of the interface points. If the positions of the interface points have been moved, these matrices are recalculated.

2.2 Coupling Algorithm

We consider, as an example of use, a quasi 1-D channel with a flexible curved wall as shown in Fig. 1(b). The main velocity, u, of the compressible flow is in the x-direction and the structure is modeled as a membrane. The diameter of the tube may vary due to a pressure difference between the pressure in the flow and in the wall. For more details on this test problem, we refer to [1].

The simulation of the compressible flow and the membrane is solved effectively by coupling two solvers. The solvers exchange data to take into account the effects on the other domain. Starting at time t^n, each solver computes the solution at time $t^n + \Delta t^n$ on its own particular domain. In general, Δt^n will be determined by the flow solver. The following steps are carried out to obtain the solution at t^{n+1} from the solution at t^n :

step 1. compute transformation matrices $\mathbf{H}^n_{AB}$ and $\mathbf{H}^n_{BA}$
step 2. obtain the pressure on the structure interface points $\mathbf{p}^n_{B,\gamma} = \mathbf{H}^n_{BA}\mathbf{p}^n_{A,\gamma}$
step 3. calculate the displacements of the structure $\mathrm{d}\mathbf{x}^{n+1}_B$ from the structure equations using the old value of the pressure $\mathbf{p}^n_B$
step 4. use the coupling method to compute the displacements of the tube wall $\mathrm{d}\mathbf{x}^{n+1}_{A,\gamma} = \mathbf{H}^n_{AB}\mathrm{d}\mathbf{x}^{n+1}_{B,\gamma}$
step 5. calculate the new pressure $\mathbf{p}^{n+1}_{A,\gamma}$ from the fluid equations with the new displacements of the tube $\mathrm{d}\mathbf{x}^{n+1}_A$.

The subscript γ denotes that the operations are only performed on data at the discrete interface points. The steps to gather and scatter the data on the interface points have been omitted. The computation of $\mathbf{H}^n_{AB}$ and $\mathbf{H}^n_{BA}$ requires the coordinates on the same time t^n. In Sect. 4, we will return to this example of use.

3 Design Overview

FLECS is decomposed into a *client library* that is to be called from the solver programs, and a *coupling server*, in short *server*, that coordinates the execution of the solvers, takes care of the coupling of the domains and handles the transfer of data between the solvers. Both the client library and the server have been implemented in `C`, so that it is relatively simple to use FLECS in solver programs that have been written in different programming languages like `C++` and `Fortran 90`. In the simplest case the server comprises a single process (as in Fig. 2) that executes the transfer algorithm sequentially.

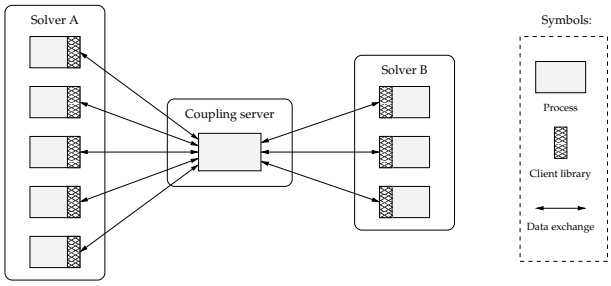

Fig. 2. Schematic representation of FLECS

To limit the complexity of the server, it can only couple two solvers at a time. However, one can couple a solver to two other solver processes by starting a second server.

3.1 The Client Library and its Usage

The client library provides subroutines for establishing a connection with the server; for describing the geometry of the coupling interface; for describing the data that are to be transferred to and from the server; for sending data to the server; and for receiving data back from the server. If a solver program comprises multiple parallel processes, then each process will contain its own copy of the client library (see Fig. 2), and will establish a separate connection with the server.

Each solver program can be started independently, using its standard start-up procedure; there is no need to change the structure of the solver program. In fact, one only needs to extend the solver program with a small number of subroutine calls to the client library. For an overview of the functions exported by the FLECS client library, see Table 1.

To start a coupled simulation, both solvers set up a connection to the server by calling the client function `FLECS_Connect`. Then the solvers inform the server about the data to be transferred between them. Therefore, one or more *point sets* and *data sets* are created at each side of the interface, and, transferred to the server by calling the functions `FLECS_NewPointSet` and `FLECS_NewDataSet`. The solvers exchange data via the server by calling the functions `FLECS_SendDataSet`, `FLECS_RecvDataSet`, `FLECS_Send` and `FLECS_Recv`. The first pair transfers a data set from one solver to the other, and typically invokes a transformation algorithm on the server. The second pair transfers an arbitrary data array between the solvers, and is particularly used to communicate convergence and time stepping information between both solvers. Both pairs of functions require that a send operation *matches* a corresponding receive operation. The description of a single iteration step of the test problem of Sect. 2.2 can be found in Sect. 4.

Table 1. An overview of the functions exported by the FLECS client library

Initialization functions	
FLECS_Init	Initializes MPI, parses the program names, opens the MPI port, publish the name of the server, save the server address
FLECS_Connect	Connects solver and coupling server
FLECS_NewPointSet	Registers point set on coupling interface with coupling server
FLECS_NewCoupling	Defines coupling between point set on this solver to point set on another solver
FLECS_NewElemSet	Defines element set on coupling interface with coupling server
FLECS_NewDataSet	Defines data set associated with point set
Finalization functions	
FLECS_Disconnect	Disconnects solver from coupling server
FLECS_Shutdown	Calls MPI_finalize and cleans up the allocated memory
FLECS_DelPointSet	Deletes registered point set
FLECS_DelCoupling	Deletes created coupling
FLECS_DelElemSet	Deletes created element set
FLECS_DelDataSet	Deletes defined data set
Data exchange functions	
FLECS_SendDataSet	Sends data set to other solver via coupling server
FLECS_RecvDataSet	Receives *transformed* data set from other solver
FLECS_Send	Sends *arbitrary* data array to other solver
FLECS_Recv	Receives *arbitrary* data array from other solver
Miscellaneous functions	
FLECS_SetCoords	Updates coordinates of registered point set
FLECS_ErrorString	Converts error code to human-readable error message
FLECS_UseElemSet	Use element set

3.2 The Coupling Server

The server consists of two parts, as shown in Fig. 3: a *communication and coordination layer*, and a *transfer algorithm*. The communication and coordination layer handles the initialization and finalization of the server; exchanges data between the server and the two solver programs; manages the data structure – including point sets, couplings, and data sets – that have been created by the client library on behalf of the solver programs; and manages the coupling-specific data structures that have been created by a transfer algorithm. Obviously, more than one data set can be associated with a point set. To associate a data set to a particular point set, an integer value `pset`, which uniquely identifies a point set, must be involved in the data set message.

The transfer algorithm handles the conversion of a data set from one point set to another point set. This part of the server is based on a plug-in architecture, that makes it easy to implement new transfer algorithms, see also Sect. 4.1. The transfer algorithm itself can be implemented in any programming language. Since the transfer algorithm is a self-contained module of the coupling server, one can experiment with different types of transfer algorithms without having

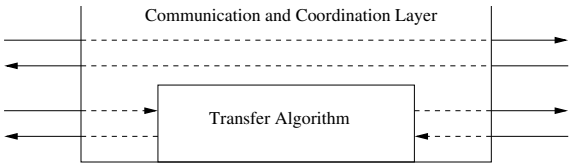

Fig. 3. The coupling server, consisting of a communication and coordination layer and a transfer algorithm. The arrows indicate the flow of data between the coupling server and two solver programs

to worry about non-essential details such as communication between the server and the solver programs.

4 Execution of Test Problem using FLECS Routines

Again, we consider the quasi-1D test problem described in Sect. 2.2. For simplicity, we assume that flow solver $\mathbf{A}$ and structure solver $\mathbf{B}$ have not been parallelized and that each solver process runs on a single processing unit. The server $\mathbf{\Gamma}$ becomes a third process which takes care of the communication and the interpolation of the meshes.

Fig. 4 represents a single FSI iteration outlined in nine (parallel) *stages*. The various steps of the coupling algorithm, as listed in Sect. 2.2, are shown, and, on the solvers $\mathbf{A}$ and $\mathbf{B}$ the calls to the client library of FLECS are inserted. We choose, that the flow solver determines the progress of the integration process, i.e., the time step is calculated by the flow solver. In addition, the flow residual $\mathbf{R}_{\mathbf{A}}^{n+1}$ controls the complete system. When the solver $\mathbf{A}$ has computed the next time step Δt^n, (*stage i*) its value must be sent via the server $\mathbf{\Gamma}$ to the solver $\mathbf{B}$ using the pair (`FLECS_Send`, `FLECS_Recv`, *stage a*).

Figure 4 illustrates that by breaking the computation of the transformation matrices $\mathbf{H}_{AB}^n$ and $\mathbf{H}_{BA}^n$ (**step** 1) into two separate parts, the server can compute these matrices simultaneously with other operations performed by the solvers (*stages c* and *g*). More precisely, no extra wall clock time is needed to compute these matrices. Let us assume that $\mathbf{H}_{BA}^n$ has been calculated in a previous integration step (*stage g*), then the force on the structure can be updated (**step** 2, *stage b*). As a result, we obtain the vector $\mathbf{p}_{B,\gamma}^n$ asked by solver $\mathbf{B}$, by means of a call of `FLECS_RecvDataSet`. Next, solver $\mathbf{B}$ computes the solution at the structure domain (**step** 3, *stage b*) at time $t^n + \Delta t^n$. We observe that the computation of $\mathbf{H}_{AB}^n$ can be postponed, allowing the solver $\mathbf{B}$ to start the computation of **step** 3 earlier. As a consequence, the computation of $\mathbf{H}_{AB}^n$ can be carried out simultaneously with **step** 3 (*stage c*). The vector $\mathrm{d}\mathbf{x}_{B,\gamma}^{n+1}$ is needed to carry out **step** 4, and corresponds to calls of `FLECS_SendDataSet` and `FLECS_RecvDataSet` on the solvers $\mathbf{B}$ and $\mathbf{A}$, respectively (*stage d*).

Again, by transferring $\mathrm{d}\mathbf{x}_{B,\gamma}^{n+1}$ (calling `FLECS_SendDataSet`) first and then the new point set $\mathbf{x}_{B,\gamma}^{n+1}$ (calling `FLECS_SetCoords`, *stage f*) more parallelism

can be obtained. However, as stated above, the transformation must be applied on updated values of $\mathrm{d}\mathbf{x}_{B,\gamma}^{n+1}$. The transformation operation delivers new values $\mathrm{d}\mathbf{x}_{A,\gamma}^{n+1}$ to be transferred to solver $\mathbf{A}$ (calling `FLECS_RecvDataSet`, *stage d*). Next, the flow solver carries out an integration step from t^n till t^{n+1} (**step** 5). Simultaneously, the matrix $\mathbf{H}_{AB}^{n+1}$ for the flow-structure interface can be calculated using the new point sets $\mathbf{x}_{A,\gamma}^{n+1}$ and $\mathbf{x}_{B,\gamma}^{n+1}$ (**step** 1, *stage g*).

Let ε be some given tolerance, and, let t_e be the end time, then if

$$\textbf{finish} = R_A^{n+1} < \varepsilon \mid t_n + \Delta t^n \geq t_e$$

is not true, a new iteration step $n+1$ can be started after calculating the new time step Δt^{n+1}. In case of convergence, the process terminates, and solver $\mathbf{A}$ sends messages to server $\mathbf{\Gamma}$ and solver $\mathbf{B}$. This can be carried out by calls to the routines `FLECS_Send` and `FLECS_Recv` (*stage h*) to notify solver $\mathbf{B}$ to terminate the calculation, followed by calls to `FLECS_Disconnect` and `FLECS_Shutdown` to disconnect the connections and to clean up all `MPI` states.

4.1 The Server Program

The main loop of the server program can look like

```
int main ( int argc, char** argv )
{ flecs_transfer_t  transf;
  int               result;

  FLECS_InitTransfer ( &transf );
  transf.NewCoupling  = NewCoupling;
  transf.DelCoupling  = DelCoupling;
  transf.InitPoints   = InitPoints;
  transf.SetCoords    = SetCoords;
  transf.TransferData = TransferData;

  FLECS_SetTransfer  ( &transf );
  FLECS_SetErrorMode ( FLECS_ERRORS_ABORT );
  FLECS_Init         ( &argc, &argv );
  FLECS_Connect      ( );

  do
  { result = FLECS_MainLoop ( ); }
  while ( ! result );

  FLECS_Shutdown (); return 0;
}
```

The `do while` loop is executed repeatedly until `result` becomes `FALSE`. The subroutine `FLECS_MainLoop` '*listens*' whether there is a message to be received, where after the server copies the data out of the send buffer and will act accordingly. A *message* can be one of the functions listed in Table 1. Assume that the application demands a multigrid approach, and, assume that `FLECS_MainLoop`

receives a message from solver **A** for a new coupling (i.e., solver **A** has called `FLECS_NewCoupling`). Then the server $\mathbf{\Gamma}$ may expect a similar request by solver **B**, followed by new point sets and associated data sets of both solvers. We remark that as a consequence of this simple approach, illustrated by the above program listing, the FLECS' user does not have to implement the server program, but only adds some simple programs `NewCoupling`, `DelCoupling`, `InitPoints`, `SetCoords` and `TransferData`. The latter must include the coupling algorithm. Moreover, the user has to include in his/her solver programs some calls to FLECS routines. Such calls must *correspond*, e.g., `FLECS_Send` and `FLECS_Recv` are appearing in pairs, otherwise an error will be generated.

5 Conclusions and Future Plans

In this paper, we have introduced FLECS, a coupling shell, which can be used as an interface for multidisciplinary simulations, e.g., for fluid-structure interaction computations. A very simple quasi one-dimensional test problem is used to show the usage of an preliminary implementation of FLECS, and an overview of the available routines is given. More investigations are needed to prove its functionality, to experiment with different kind of coupling methods, such as nearest neighbor or Gauss interpolation. We would like to extend the parallel capabilities of FLECS to be able to simulate realistic cases on parallel computer platforms.

References

1. A. de Boer, A.H. van Zuijlen and H. Bijl. *Review of coupling methods for non-matching meshes*, Comput. Methods in Appl. Mech. and Eng., 196 (8), 1515–1525, 2007.
2. E.J. Lingen, M. Nool, A. de Boer and H. Bijl. *Design of Flecs, a flexible coupling shell*, 2006, see http://www.aero.lr.tudelft.nl/FLECS
3. William Gropp, Ewing Lusk and Rajeev Thurs. *Using MPI-2* Advanced Features of the Message Passing Interface The MIT Press, Cambrigde, 1999.
4. MpCCI, The Fraunhofer-Institute for Algorithms and Scientific Computing (SCAI). *Multidisciplinary Simulation through Code Coupling*, see http://www.scai.fraunhofer.de/mpcci.html
5. S. Bosscher A.H. van Zuijlen and H. Bijl. Two level algorithms for partitioned fluid-structure interaction computations. *Comput. Methods Appl. Mech. Eng.*, 196(8):1458–1470, 2007.
6. A.H. van Zuijlen and H. Bijl. Implicit and explicit higher order time integration schemes for structural dynamics and fluid-structure interaction computations. *Comput. Struct.*, 83:93–105, 2005.

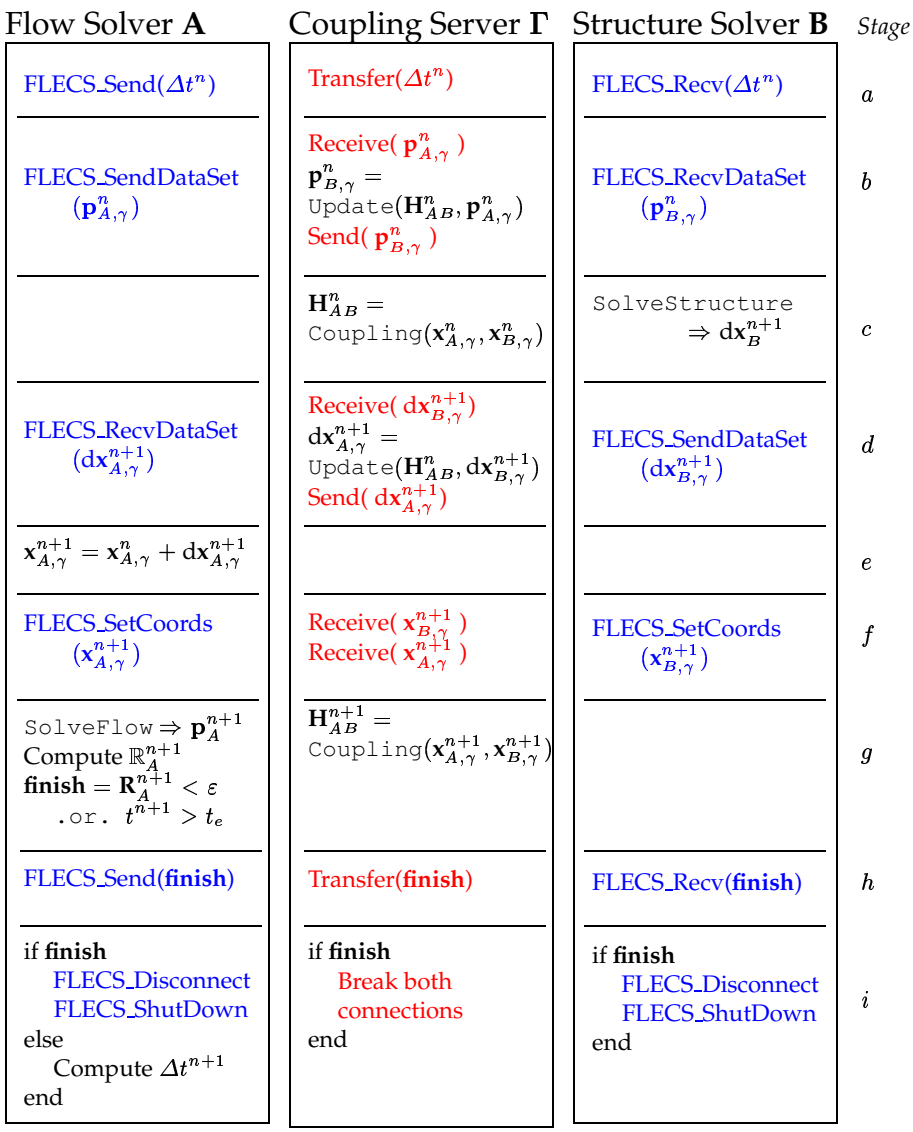

Fig. 4. The $n+1$-th iteration step of the FSI process expressed in FLECS routines (see Table 1). The superscript n stands for the time step, whereas the subscript A or B indicate that the values belong to the domain of the flow solver **A** or the structure solver **B**. A vector refers to interface values in case the underscore γ is present. Here, t^n and Δt^n denote the current time and time step, $\mathbf{x}$ and $\mathrm{d}\mathbf{x}$ the position and displacement of the coordinates, $\mathbb{R}^n_A$ the residual, ε indicates the required accuracy of $\mathbf{x}$ for FSI iteration. `SolveFlow` performs a single iteration with the flow solver, resulting in, among others, the updated value $\mathbf{p}^{n+1}_A$, whereas `SolveStructure` carries out a single iteration with the structure solver, producing, among others, the updated value $\mathrm{d}\mathbf{x}^{n+1}_B$. The interface points of $\mathbf{p}^{n+1}_A$ and $\mathrm{d}\mathbf{x}^{n+1}_B$ are input for the transformation matrices. Finally, the `boolean` **finish** determines whether the program terminates.