REALIZING COSIMULATION IN AND WITH A MULTIPHYSICS FRAMEWORK

Philipp Bucher\textsuperscript{a}, Aditya Ghantasala\textsuperscript{a}, Pooyan Dadvand\textsuperscript{b}, Roland Wüchner\textsuperscript{a}, Kai-Uwe Bletzinger\textsuperscript{a}

\textsuperscript{a}Chair of Structural Analysis
Technische Universität München (TUM)
Arcisstr. 21, D-80333 München, Germany
e-mail: philipp.bucher@tum.de, ga29wun@mytum.de, wuechner@tum.de, kub@tum.de

\textsuperscript{b}Altair Engineering
Avenida Diagonal, 682, 08034 Barcelona, Spain;
Universitat Politècnica de Catalunya (UPC)
Edifici C1, CIMNE - Plate 2, C. Jordi Girona, 1-3, 08034 Barcelona, Spain
e-mail: pooyan@altair.com, pooyan.dadvand@upc.edu

Key words: Coupled Problems, Multiphysics Problems, Software Design, Applications, High Performance Computing, MPI, FSI, Mapping, Partitioned Coupled Simulation

Abstract. Simulating coupled problems using a multiphysics framework is different from the classical approach using dedicated coupling tools. It can have several advantages such as reduced memory footprint or more efficient communication between the involved solvers. The realization of coupled simulations with a multiphysics framework is presented together with important details of the software design such as data management, data communication, mapping, and distributed computing. Several examples from different physical disciplines with coupling internal and external solvers are shown.

1 INTRODUCTION

Coupled problems are often conducted with the partitioned approach by using black-box solvers/codes. The coupling between the solvers is mostly done with dedicated coupling tools such as EMPIRE \cite{1}, comana \cite{2} or preCICE \cite{3}. This approach inherently has disadvantages such as duplication of data as well as data communication between the solvers/codes and the coupling tool.

Performing coupled simulations within a multiphysics framework can mitigate or solve those issues as presented in this work. It builds on and extends \cite{4}.

After the introduction in the first chapter, chapter two discusses the software design for realizing CoSimulation in the multiphysics framework Kratos Multiphysics \cite{5,6} (Kratos).
The third chapter focuses on mapping as a crucial component of CoSimulation. Chapter four addresses the important aspects of CoSimulation in distributed environments. In chapter five the details of realizing CoSimulation with Kratos solvers are presented, followed by chapter six which shows the coupling to external solvers. The last chapters give a brief overview and conclusion of this work as well as an outlook. Finally, the acknowledgements finish this work.

2 SOFTWARE DESIGN

The implementation of CoSimulation features in a multiphysics framework is crucial for an efficient, versatile, and flexible simulation of coupled problems. In Kratos this is realized with the CoSimulationApplication. This chapter extends and continues the work of [4] and focuses on some important aspects of the software design and implementation.

First, the main components of the CoSimulationApplication are briefly explained below, their interaction is shown in figure 1.

- **SolverWrapper**: Baseclass and CoSimulationApplication-interface for all solvers/codes participating in the coupled simulation, each solver/code has its own specific version.

- **CoupledSolver**: Implements coupling schemes such as weak/strong coupling with Gauss-Seidel/Jacobi pattern. It derives from SolverWrapper such that it can be used in nested coupled simulations.

- **IO**: Responsible for communicating and data exchange with external solvers/codes

- **DataTransferOperator**: Transfers data from one discretization to another, e.g. by use of mapping techniques

- **CouplingOperation**: Tool for customizing coupled simulations

- **ConvergenceAccelerator**: Accelerating the solution in strongly coupled simulations by use of relaxation techniques

- **ConvergenceCriteria**: Checks if convergence is achieved in a strongly coupled simulation.

- **Predictor**: Improves the convergence by using a prediction as initial guess for the coupled solution

Kratos uses Python as scripting language and C++ as backend for performance critical tasks. This combination has proven to be very useful in practice as it combines the flexibility of Python with the performance of C++. The CoSimulationApplication makes use of these features which results in a flexible and performant framework. The flexibility is very important for coupled simulations as many of them require some special treatments.

Having the interfaces available in Python also means that integration of functionalities from other libraries (e.g. mapping) can be achieved easily.
2.1 Data management

Data plays a very important role in coupled simulations. Hence smart and efficient solutions are required to avoid potential overhead, both in terms of computation as well as memory.

The CoSimulationApplication uses the Kratos native data structure ModelPart. It is an elaborate data structure that provides all the necessary functionalities such as mesh, geometry, nodes, elements, and different databases.

The ModelPart is also used by the Kratos solvers, hence their data can be accessed directly without any upfront manipulation, no extra computation/memory overhead is necessary when accessing the data of Kratos solvers. This is a large advantage of performing coupled simulations with a multiphysics framework.

2.2 DataTransferOperator: Transferring data between ModelParts

The ModelPart is used as central data storage, each solver has at least one of them. CoSimulation requires the transfer of data between solvers, which means that data has to be transferred from one ModelPart to another. How the data is transferred strongly depends on its format. Possible formats are:

- Data not related to any field or geometry
- Field data which belongs to a mesh (or geometry)

An example of the first type is a single degree of freedom (SDOF) solver, which has only one degree of freedom that is not related to a mesh or geometry. Another example is input from a sensor. If combined with a model that has a geometry, then of course they can be associated in specific geometrical locations, but be themselves they don’t have it. In those cases it is sufficient to copy the values, no mapping techniques are required.
A prominent example of the second type is fluid-structure interaction (FSI), where typically the displacements computed by the structural solver have to be transferred to the fluid solver, and the loads computed by the fluid solver have to be transferred to the structural solver. The loads and displacements are associated with the interface meshes of the respective solvers. Usually, those meshes are not matching due to the different technical requirements of the solvers, hence mapping techniques have to be used for transferring the data.

In the CoSimulationApplication the DataTransferOperator fulfills this task. Different operators are available, for mapping, copying of values, and others. This separation of concepts helps to keep the framework flexible as well as to split the responsibilities.

2.3 Customizing CoSimulation with the CouplingOperation

Coupled simulations can be done for many different applications, and in different ways. Hence the requirements for the coupling framework differ vastly for different cases. To fulfill the requirements and to give the necessary flexibility, mechanisms for customization have to be given.

This is the task of the CouplingOperation in the CoSimulationApplication. It is an object that can be used optionally in different places to perform different tasks related to the coupling. Examples of those tasks are writing output, computing auxiliary quantities such as normals or scaling values.

Adding a custom task is simple, it only requires adding a Python script that implements the desired functionality.

3 MAPPING

The transfer of data between non-matching discretizations requires the use of mapping techniques. As the solvers/codes participating in a coupled simulation often use meshes for solving their respective physics such as the finite element method (FEM), mapping is a crucial component of CoSimulation.

However as already explained in 2.2, mapping is not always needed and hence is treated as a separate, optional component. In Kratos, the mapping is implemented in the MappingApplication (see [7] and [8]). It is integrated into the the CoSimulationApplication as a DataTransferOperator.

As the mapping is implemented in Kratos, it can directly access the internal data structure, the ModelPart (see 2.1). Therefore any memory overhead due to data duplication is avoided, which is very important especially for mapping between volume meshes.

The MappingApplication provides different mapping techniques such as nearest neighbor or nearest element. Furthermore, the mapping can be done in 1D (line to line), 2D (surface to surface), or 3D (volume to volume). It fully supports message passing interface (MPI) in distributed environments. More details can be found in [8].
4 COSIMULATION IN DISTRIBUTED ENVIRONMENTS

Large coupled problems are often conducted using high-performance computing (HPC) systems such as clusters or supercomputers. These systems employ special programming techniques for distributed computing such as MPI. Kratos supports distributed environments, see [9]. The CoSimulationApplication supports it too, also it implements functionalities to facilitate the solvers working distributed. Different scenarios for CoSimulation in distributed environments exist, as shown in figure 2.

![Figure 2: Different scenarios for running CoSimulation in distributed environments (4 MPI-processors)](image)

Efficient algorithms and methods for data exchange are required for distributed computing to avoid communication between the processors as it is overhead. This means that gathering and scattering of data should be avoided as much as possible and replaced with more efficient versions such as peer-to-peer communication. The CoSimulationApplication and especially the MappingApplication make use of peer-to-peer communication whenever possible.

5 COUPLING INSIDE KRATOS

As a multiphysics framework Kratos provides solution techniques for different physics such as structural analysis (within the StructuralMechanicsApplication), fluid dynamics (within the FluidDynamicsApplication), or discrete elements (within the DEMApplication). As explained in 2.1, the solvers use the ModelPart as common data structure.

5.1 Common interface of Kratos solvers

Besides using the same data structure, the Kratos solvers also have a common user interface, the AnalysisStage. This interface accommodates the different solvers and also
is used throughout the \textit{CoSimulationApplication}. The main functions are:

- **Initialize**: This function is called once at the beginning of the simulation, e.g. reads the input files and prepares the internal data structures.

- **RunSolutionLoop**: Iterates through the time steps. Is split up into the following six functions:
  - **AdvanceInTime**: Advancing in time and preparing the data structure for the next time step.
  - **InitializeSolutionStep**: Applying boundary conditions.
  - **Predict**: Predicting the solution of this time step to accelerate the solution.
  - **SolveSolutionStep**: Solving the problem for this time step. This is the only function that can be called multiple times in an iterative solution procedure.
  - **FinalizeSolutionStep**: Updating internals after solving this time step.
  - **OutputSolutionStep**: Writing output at the end of a time step.

- **Finalize**: Finalizing and cleaning up after the simulation.

The interface of the \textit{SolverWrapper} (see \cite{2}) follows the same interface, which makes the integration of \textit{Kratos} solvers into the \textit{CoSimulationApplication} straightforward.

5.2 Examples

This section shows several examples of coupled problems, solved with \textit{Kratos} solvers.

5.2.1 Simulation of rock fall protection net

Rockfall protection nets are often used in mountainous areas to protect cars and streets from falling rocks. \cite{10} and \cite{11} simulate and investigate this coupled problem using \textit{Kratos}.

The coupling is done in a partitioned way with the \textit{CoSimulationApplication}. The rocks are simulated with the discrete element method (DEM) by using the \textit{DEMAplication}. The net is simulated using the FEM with the \textit{StructuralMechanicsApplication}. The setup of the coupling is shown in figure \ref{fig:co_simulation} (left). The impact loads exerted by the rocks are mapped to the structural model, and the displacements and velocities computed by the structural solver are mapped back to the DEM for updating the boundary. Figure \ref{fig:co_simulation} (right) shows an application.
5.2.2 FSI with SDOF solver

This example shows the coupling of a computational fluid dynamics (CFD) solver (using the FluidDynamicsApplication of Kratos) with an SDOF solver as already mentioned in 2.2. The CFD models the real geometry of the obstacle in the flow, whereas the SDOF solver only models the stiffness and damping of the structural system. The forces computed by the CFD are summed up and applied to the SDOF as an external load. After solving, the displacements are then transferred to all the CFD nodes on the boundary.

This concept is illustrated in figure 4 on the left, with a square cross-section. The results of the vertical movement of the square are shown on the right. An example of such a simulation would be the flow over a flexible bridge.

5.2.3 Munich Olympic stadium roof

Munich was the host of the Olympic games in 1972, for which the Olympic stadium was built. The main construction of the stadium roof is a hanging cable net covered with acrylic plates, it is held up and supported by masts and pylons, see figure 5.

The behavior of the roof under wind loading is investigated. For this, a CFD model (using the FluidDynamicsApplication) and a FEM model (using the StructuralMechanics-
Application) were created, see figure 6. The FSI coupling is done with the CoSimulation-Application. Figure 7 shows the displacements of the roof in strong wind.

Figure 5: Olympic stadium in Munich

Figure 6: Structural FEM model of the Olympic stadium roof

6 COUPLING TO EXTERNAL SOLVERS

Coupling to external solvers requires an additional step compared to coupling internal (Kratos) solvers: The data exchange between the solvers/codes. As explained in 2.1, the data of internal solvers can be accessed directly. The internal data structure of other solvers/codes can usually not be directly accessed from outside, hence the interface data of external solvers has to be imported to/exported from the CoSimulationApplication. This way they can be treated like internal solvers, which simplifies the implementation. Unfortunately, it also leads to memory overhead as the data on the interface is duplicated, which can play an important role depending on the application case. A remedy for this memory overhead could be to directly communicate data between solvers, but then no coupling functionalities of the CoSimulationApplication such as mapping or convergence acceleration can be used.

The IO (see 2) now plays an important role as it is responsible for the data exchange which is done via interprocess communication (IPC). Different communication methods exist, most often used are file- or socket-based communication.
The detached-interface approach (see [4]) is used to minimize the dependencies of external solvers on the CoSimulationApplication. The idea is to have an independent interface for data exchange that simplifies the integration into external solvers by avoiding any dependencies on Kratos and the CoSimulationApplication.

The CoSimIO [12] is a small tool that implements the detached-interface approach. It is integrated into the CoSimulationApplication and can be used to perform coupled simulations, see [13].

6.1 Example: Wind Turbine

This example shows the application of the presented work to a full wind turbine FSI simulation, which is done within the research project WINSENT ([14]). The CFD solver FLOWer [15] is coupled with Kratos which uses the StructuralMechanicsApplication for the structural simulation of the turbine and the CoSimulationApplication for the coupling. The CFD model and the adaptions in FLOWer to make the coupling of the entire turbine possible were done by Giorgia Guma from the Institute of Aerodynamics and Gas Dynamics at the University of Stuttgart.

FLOWer implements an interface to the coupling library EMPIRE. To minimize the changes necessary in FLOWer for the coupling with the CoSimulationApplication, a simplified version of the EMPIRE interface using file-based communication was developed following the detached-interface approach. This interface is a single file header only C++ file with no dependencies, which made the integration is FLOWer straightforward.

Two structural models were created, using beam and using shell elements. The displacements of the blades can be seen in [8].
Figure 8: Displacements for turbine models under operating conditions, for beam model (left) and shell model (right)

7 CONCLUSIONS AND OUTLOOK

This work presented the realization of CoSimulation by using the multiphysics framework Kratos and the CoSimulationApplication. Different aspects of software design and implementation are covered. Mapping was discussed as a crucial component of CoSimulation, as well as conducting coupled simulations in distributed HPC environments. Several real-world application examples show the capabilities of the CoSimulationApplication.

Future extensions of this work can be the application to volume coupled problems or coupling to other particle methods such as the material point method (MPM). Furthermore the features for CoSimulation in distributed environments can be improved. This includes the detached interface CoSimIO to support MPI. Also, different methods for data exchange with external solvers using IPC such as sockets or pipes can be implemented to improve the performance and efficiency of the data exchange.

8 ACKNOWLEDGEMENTS

The authors want to acknowledge the support of the German Federal Ministry for Economic Affairs and Energy (BMWi) within the WINSENT project (FKZ 0324129F). Furthermore, the support of the Leibniz Supercomputing Centre (www.lrz.de) is acknowledged for providing computing time on the GCS Supercomputer SuperMUC-NG in the projects pr94va and pn56ba.
The support of the colleagues from the Chair of Structural Analysis (Technical University of Munich) and the Institute of Aerodynamics and Gas Dynamics (University of Stuttgart) is acknowledged, especially Giorgia, Máté and Klaus, for providing and helping with examples.

REFERENCES


