Seamless Coupling of PDE-based Simulations with the Coupling Library preCICE

SimulTech 2022

Benjamin Uekermann et al.

University of Stuttgart
Keynote Presentations from Previous Years

- **2021, Peter Fritzson**: *The OpenModelica Environment and Its Use for Development of Sustainable Cyber-physical Systems and Digital Twins*
- **2019, Hans Vangheluwe**: *Co-Simulation – A Research Agenda*
- More buzzwords: coupling, co-simulation, interoperability, . . .
Keynote Presentations from Previous Years

- **2021, Peter Fritzson**: *The OpenModelica Environment and Its Use for Development of Sustainable Cyber-physical Systems and Digital Twins*
- **2019, Hans Vangheluwe**: *Co-Simulation – A Research Agenda*
- More buzzwords: coupling, co-simulation, interoperability, …

In my community:

Simulation = continuous-time (numerical) simulation of physical systems
1. Introduction: FMI vs. Coupling Libraries
2. Coupling Library preCICE
1. Introduction: FMI vs. Coupling Libraries
import pyfmi

model1 = load_fmu('Model1.fmu')
model2 = load_fmu('Model2.fmu')

# my own simple master algorithm
while (not done())
    model1.set('u', u1)
    model1.do_step()
    y1 = model1.get('y')
    u2 = y1
    model2.set('u', u2)
    model2.do_step()
    y2 = model2.get('y')
    u1 = accelerate(y2)
import pyfmi
model1 = load_fmu('Model1.fmu')
model2 = load_fmu('Model2.fmu')

# my own simple master algorithm
while (not done()):
    model1.set('u', u1)
    model1.do_step()
    y1 = model1.get('y')
    u2 = y1
    model2.set('u', u2)
    model2.do_step()
    y2 = model2.get('y')
    u1 = accelerate(y2)

- FMUs regarded as black boxes, Goals: flexibility + protect IP
FMUs regarded as black boxes, Goals: flexibility + protect IP

Framework approach: coupling calls models
import pyfmi
model1 = load_fmu('Model1.fmu')
model2 = load_fmu('Model2.fmu')

# my own simple master algorithm
while (not done()):
    model1.set('u', u1)
    model1.do_step()
    y1 = model1.get('y')
    u2 = y1
    model2.set('u', u2)
    model2.do_step()
    y2 = model2.get('y')
    u1 = accelerate(y2)

- FMUs regarded as black boxes, Goals: flexibility + protect IP
- Framework approach: coupling calls models
- Units are (for example) ODEs, typically relatively simple components
Functional Mock-up Interface (… as I see it)

1 import pyfmi
2 model1 = load_fmu('Model1.fmu')
3 model2 = load_fmu('Model2.fmu')

4 # my own simple master algorithm
5 while (not done()):
6    model1.set('u', u1)
7    model1.do_step()
8    y1 = model1.get('y')
9    u2 = y1
10   model2.set('u', u2)
11   model2.do_step()
12   y2 = model2.get('y')
13   u1 = accelerate(y2)

- FMUs regarded as black boxes, Goals: flexibility + protect IP
- Framework approach: coupling calls models
- Units are (for example) ODEs, typically relatively simple components
- Typically, communicated data not too large ⇝ HPC not sooo important
PDE Simulation Units

Blood flow through human aorta.
Picture: Totounferoush et al. 2021

Space resolution and more accuracy needed \( \Rightarrow \) PDEs
PDE Simulation Units

Blood flow through human aorta.
Picture: Totounferoush et al. 2021

- Space resolution and more accuracy needed $\leadsto$ PDEs
- Coupling data space-dependent $\leadsto$ data mapping required

Fluid Model $\leftrightarrow$ Solid Model
PDE Simulation Units

Blood flow through human aorta.

Picture: Totounferoush et al. 2021

- Space resolution and more accuracy needed \(\leadsto\) PDEs
- Coupling data space-dependent \(\leadsto\) data mapping required
- Models (aka. solvers): sophisticated software
PDE Simulation Units

Blood flow through human aorta.
Picture: Totounferoush et al. 2021

- Space resolution and more accuracy needed $\rightsquigarrow$ PDEs
- Coupling data space-dependent $\rightsquigarrow$ data mapping required
- Models (aka. solvers): sophisticated software
- Regard models as black boxes, Goals: flexibility + to be minimally invasive
PDE Simulation Units

- Space resolution and more accuracy needed \(\rightsquigarrow\) PDEs
- Coupling data space-dependent \(\rightsquigarrow\) data mapping required
- Models (aka. solvers): sophisticated software
- Regard models as black boxes,
  Goals: flexibility + to be minimally invasive
- Costly simulations, large amounts of distributed coupling data \(\rightsquigarrow\) HPC a must

Blood flow through human aorta.
Picture: Totounferoush et al. 2021
Library approach: the models call the coupling
- Minimally invasive
- Start models separately
- Coupling logic more complicated
Seamless Coupling of PDE Units: Peer-to-Peer Approach

- No central server-like entity
- Peer-to-peer communication between ranks whose coupling interfaces overlap
  - No scaling issues
  - Coupling numerics (data mapping, acceleration, . . .) need to run on distributed data
Software Solutions

- **OpenPALM / CWIPI** (Coupling With Interpolation Parallel Interface) from CERFACS + ONERA (F)
- **MUI** (Multiscale Universal Interface) from UKRI-STFC (UK) + Brown + Berkeley + IBM (US)
- **DTK** (Data Transfer Kit) from Sandia + ORNL (US)
- **preCICE** from U Stuttgart + TUM (D)
- (MpCCI from Fraunhofer SCAI (D), commercial, limited HPC capabilities)
- ...many more
Selling Points of preCICE

- Free software (LGPL3)
- Extensive user documentation (when converted to PDF, more than 250 pages)
- Vivid and quickly growing community
- Industry-ready support program
- Ready-to-use adapters for OpenFOAM, FEniCS, deal.II, CalculiX, Nutils, SU2, ...
- API in C++, C, Fortran, Python, Matlab, Julia
- Scalability up to complete supercomputers
- xSDK member, pre-installed on more and more supercomputers
- Robust quasi-Newton coupling (and soon waveform iteration)
- Coupling of arbitrary many components (arbitrary many = more than two)
- CI with tests on all levels (unit, integration, bindings, adapters, system)
2. Coupling Library preCICE
Overview

CFD solver -> FEM solver
Overview

- CFD solver
- in-house solver
- FEM solver
- Particle solver
- libprecice

Diagram showing connections between solvers, adapters, and libraries.
Overview

CFD solver
OpenFOAM
SU2
FASTEST
ExaDG

in-house
solver

API in:
C++
Python
Matlab
C
Fortran
Julia

libprecice

FEM solver
deal.II
FEniCS(-X)
Nutils
CalculiX
code_aster
MBDyn
DUNE-FEM
DuMuX
Elmer FEM

Particle
solver

MercuryDPM
XDEM

adapter

API in: C++
C
Python
Fortran
Matlab
Julia

MercuryDPM
XDEM
A Coupling Library for Partitioned Multi-Physics Simulations

CFD solver
OpenFOAM
SU2
FASTEST
ExaDG

in-house solver
API in:
C++
Python
Matlab
Fortran
Julia

solver
adapter
libprecice

coupling schemes

data mapping

communication

time interpolation

coupling schemes

FEM solver
deal.II
FEniCS(-X)
Nutils
CalculiX
code_aster
MBDyn
DUNE-FEM
DuMuX
Elmer FEM

Particle solver
MercuryDPM
XDEM
import precice

interface = precice.Interface('FluidSolver', 'precice-config.xml', rank, size)

Some details omitted. Follow preCICE Course to learn the API step-by-step.
import precice

interface = precice.Interface('FluidSolver', 'precice-config.xml', rank, size)

positions = ...

#define interface mesh, 2D array with shape (n, dim)

vertex_ids = interface.set_mesh_vertices('Fluid-Mesh', positions)

Some details omitted. Follow preCICE Course to learn the API step-by-step.
import precice

interface = precice.Interface('FluidSolver', 'precice-config.xml', rank, size)

positions = ...
#define interface mesh, 2D array with shape (n, dim)
vertex_ids = interface.set_mesh_vertices('Fluid-Mesh', positions)

interface.initialize()

Some details omitted. Follow preCICE Course to learn the API step-by-step.
```python
import precice

interface = precice.Interface('FluidSolver', 'precice-config.xml', rank, size)

positions = ...  # define interface mesh, 2D array with shape (n, dim)
vertex_ids = interface.set_mesh_vertices('Fluid-Mesh', positions)

interface.initialize()

while interface.is_coupling_ongoing():  # main time loop

Some details omitted. Follow preCICE Course to learn the API step-by-step.
```
import precice

interface = precice.Interface('FluidSolver', 'precice-config.xml', rank, size)

positions = ...  #define interface mesh, 2D array with shape (n, dim)
vertex_ids = interface.set_mesh_vertices('Fluid-Mesh', positions)

interface.initialize()

while interface.is_coupling_ongoing():  # main time loop

    u = solve_time_step()  # returns new solution

Some details omitted. Follow preCICE Course to learn the API step-by-step.
import precice

interface = precice.Interface('FluidSolver', 'precice-config.xml', rank, size)

positions = ...  # define interface mesh, 2D array with shape (n, dim)
vertex_ids = interface.set_mesh_vertices('Fluid-Mesh', positions)

interface.initialize()

while interface.is_coupling_ongoing():  # main time loop

    displacements = interface.read_block_vector_data('Displacement', vertex_ids)
    u = solve_time_step(displacements)  # returns new solution
    forces = compute_forces(u)  # returns 2D array with shape (n, dim)

    interface.write_block_vector_data('Force', vertex_ids, forces)

Some details omitted. Follow preCICE Course to learn the API step-by-step.
import precice
interface = precice.Interface('FluidSolver', 'precice-config.xml', rank, size)

positions = ... # define interface mesh, 2D array with shape (n, dim)
vertex_ids = interface.set_mesh_vertices('Fluid-Mesh', positions)

interface.initialize()

while interface.is_coupling_ongoing():  # main time loop
	displacements = interface.read_block_vector_data('Displacement', vertex_ids)
	n = solve_time_step(displacements)  # returns new solution

forces = compute_forces(u)  # returns 2D array with shape (n, dim)

interface.write_block_vector_data('Force', vertex_ids, forces)

interface.advance()

Some details omitted. Follow preCICE Course to learn the API step-by-step.
import precice

interface = precice.Interface('FluidSolver', 'precice-config.xml', rank, size)

positions = ... #define interface mesh, 2D array with shape (n, dim)
vertex_ids = interface.set_mesh_vertices('Fluid-Mesh', positions)

precice_dt = interface.initialize()

while interface.is_coupling_ongoing(): # main time loop
    dt = compute_adaptive_dt()
    dt = min(precice_dt, dt)

    displacements = interface.read_block_vector_data('Displacement', vertex_ids)
    u = solve_time_step(dt, displacements) # returns new solution
    forces = compute_forces(u) # returns 2D array with shape (n, dim)
    interface.write_block_vector_data('Force', vertex_ids, forces)

    precice_dt = interface.advance(dt)

Some details omitted. Follow preCICE Course to learn the API step-by-step.
Demo: Conjugate Heat Transfer with OpenFOAM and Nutils

Conjugate Heat Transfer

- Coupled heat transfer between a fluid (convection and conduction) and a solid (only conduction)

OpenFOAM\textsuperscript{1}, ESI/OpenCFD

- Open-source C++ finite-volume framework
- Widely used for CFD (industry and academia)
- We use buoyantPimpleFoam, transient compressible NSE

Nutils

- Open-source Python programming library for Finite Element Method computations
- Developed by Evalf Computing, spin-off of TU Eindhoven
- We use it here for the simple heat equation (conduction in the solid)

\textsuperscript{1}OPENFOAM is a registered trade mark of OpenCFD Limited, producer and distributor of the OpenFOAM software via www.openfoam.com.
Demo: Conjugate Heat Transfer with OpenFOAM and Nutils

Boundary conditions and geometry

\[ \Gamma_D : T_{\text{Solid}} = T_{\text{Fluid}} \]
\[ \Gamma_N : q_{\text{Fluid}} = q_{\text{Solid}} \]

\[ u_\infty = 0.1 \]
\[ T_\infty = 300 \]

\[ q = 0 \]
\[ \text{slip} \]
\[ \text{no-slip} \]

Additional parameters

- \( Pr = 0.01 \)
- \( Re = \rho u_\infty d/\mu = 500 \) (use characteristic length \( d = \) plate length)
- \( k_s = 100 \cdot k_f \) (thermal conductivities)
Demo: Conjugate Heat Transfer with OpenFOAM and Nutils

```bash
unlockinglapus301.. ~/flow-over-heated-plate/solid-nutils [git] [ develop ] % /run.sh
opened log at file: /home/uekernb/public_html/solid.py/log-35.html
vets v6.3 "parak-gusuk"
start Mon Jul 11 17:57:25 2022
Running nutils
```

```
optimize > solve > solving 21 dof system to machine precision using direct solver
optimize > solve > solver returned with residual 0e+00
optimize > constrained 21/126 dofs
optimize > optimum value -1.46e-11
```
Demo: Conjugate Heat Transfer with OpenFOAM and Nutils
Scalability: Time per Timestep

- Travelling density pulse (Euler equations) through artificial coupling interface
- DG solver Ateles (U Siegen), $5.7 \cdot 10^7$ dofs
- Nearest neighbor mapping and communication
- Uekermann. *FSI on Massively Parallel Systems* (2016)
- SuperMUC Thin Nodes
Users

Oden Institute
USA
CIRA
Italy
Universität Siegen
Fluid Mechanics
Germany
Universität Stuttgart
Applied Mechanics
Germany
DHCAE Tools
Germany
EuroCFD
France
Global Research for Safety
Germany
MPH/IPP
Germany

Technische Universität Darmstadt
Germany
University of the Free State
South Africa
Singapore
Singapore
NRG
Netherlands
Strathclyde
CFO & FSI
United Kingdom
Sirrun
Italy
Austria

TUM
Germany
FNB
Germany
Technische Universität Darmstadt
Germany
Universität Stuttgart
HWS
Germany
TU Delft
Wind Energy
Netherlands
University of Split
Croatia

KIT
Germany
KU Leuven
Belgium
Manchester
The University of Manchester
Aerodynamics
United Kingdom
MTU Aero Engines
Germany
Lawrence Livermore
National Laboratory
USA
Corvid Technologies
USA

Max Planck Institute
Germany
University of Siegen
Aerodynamics
Netherlands
Heat and Mass Transfer
TC
Spain
IFL
Germany
ATA Engineering
USA
Korea Atomic Energy Research Institute
South Korea

Exeter
Energy Technology
Netherlands
Computational Fluid Dynamics
United Kingdom
What Our Users Do

- Coupling mechanics and electrophysiology in skeletal muscles
- Benjamin Maier, IPVS, U Stuttgart
- More information on www.precice.org/testimonials
What Our Users Do

- Optimization of thermal groundwater heat pump usage in Munich
- GEO.KW project
- More information on www.precice.org/testimonials
What Our Users Do

- Hybrid simulation methods for wind modelling in urban areas
- Aerodynamics research group, MACE, The University of Manchester
- More information on www.precice.org/testimonials
What Our Users Do

- Heat simulation on the moon
- Chair of Astronautics, Technical University of Munich
- More information on www.precice.org/testimonials
Couple FMUs via preCICE?

```python
# preCICE-FMI script
import precice, pyfmi
interface = precice.Interface('Model1', 'precice-config.xml', rank, size)
model = load_fmu('Model1.fmu')

vertex_ids = interface.set_mesh_vertices('Model1_Mesh', positions)
interface.initialize()

while interface.is_coupling_ongoing():
    u = interface.read_data('u', vertex_ids)
    model.set('u', u)
    model.do_step()
    y = model.get('y')
    interface.write_data('y', vertex_ids, y)
    interface.advance(dt)
```
Summary

- Sometimes, you need PDE-based models ⇝ sophisticated, legacy, stand-alone software packages
- Easier (minimally invasive) if we turn things around: the models call the coupling
- For performance reasons: peer-to-peer coupling
- Coupling libraries can help, e.g. preCICE
- Why preCICE? Community, many adapters, documentation, HPC, . . .
- You could still couple FMUs with preCICE
Resources

- ✉️ benjamin.uekermann@ipvs.uni-stuttgart.de
- 🌐 precice.org
- 🔄 github.com/precice/
- 🍛 precice.discourse.group
- 🌐 @preCICE_org
- 📖 Chourdakis G, Davis K, Rodenberg B et al.
  
  *preCICE v2: A sustainable and user-friendly coupling library*
  [version 1; peer review: 2 approved]
  Open Res Europe 2022, 2:51 doi:10.12688/openreseurope.14445.1

- 🏹️ Cluster of Excellence EXC 2075 – 390740016 – *Data-Integrated Simulation Science*  
  University of Stuttgart