FEniCS-preCICE: Coupling FEniCS to other Simulation Software

Ishaan Desai[a], Benjamin Rodenberg[b], Richard Hertrich[b], Alexander Jaust[c], Benjamin Uekermann[a]

[a]Usability and Sustainability of Simulation Software, Institute for Parallel and Distributed Systems, University of Stuttgart

[b]Scientific Computing in Computer Science, Department of Informatics, Technical University of Munich

[c]Simulation of Large Systems, Institute for Parallel and Distributed Systems, University of Stuttgart

preCICE

(a) https://fenicsproject.org/

Contents

preCICE - A Flexible Coupling Library

What is preCICE used for?

Coupling solvers for multi-physics simulations in a partitioned black-box fashion

Working principles of preCICE

Partitioned approach, black-box coupling, massively parallel, highly flexible, library approach

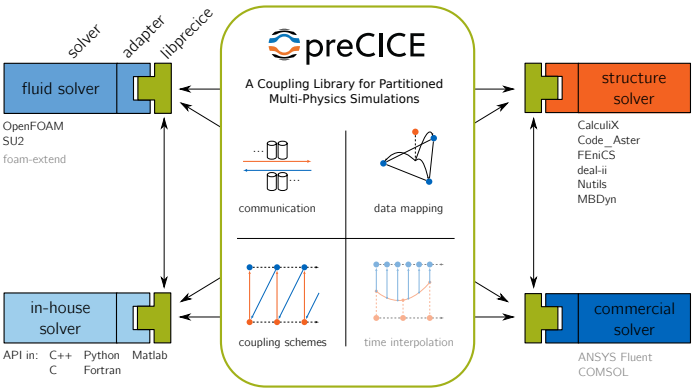


Figure: Overview of preCICE features

Goal of FEniCS-preCICE

To provide a helper package to a FEniCS user to facilitate easy use of preCICE to setup a coupled problem

Design Principles:

- To have a middle layer between high-level FEniCS program and low-level C++ preCICE API
- Use `python-bindings` of preCICE to access C++ API
- To have a highly modular structure which is easy to understand and modify in future
- To handle as many boilerplate tasks as possible inside the adapter

Features of Adapter:

- Adapter supports 2D cases in FEniCS. Users can define boundary conditions using FEniCS `Expression` or `PointSource`
- Adapter needs to be configured with a JSON file

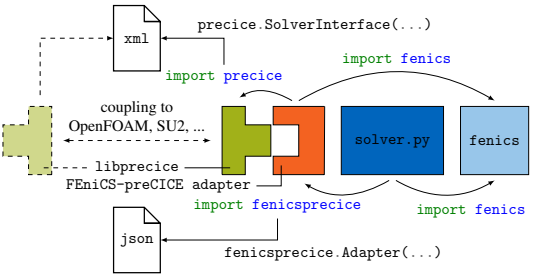


Figure: Functioning of FEniCS-preCICE Adapter

API Functions of FEniCS-preCICE

- Adapter receives configuration information from JSON file
- Adapter retrieves mesh data from user defined FEniCS `FunctionSpace` and the coupling boundary `SubDomain`
- Converting data from FEniCS format to preCICE format and visa-versa is handled internally in the functions `read_data()` and `write_data()`
- Data at coupling boundary can be of the form of a FEniCS `Expression` or FEniCS `PointSource`
- Distributed parallelization in FEniCS is handled out of the box
- Checkpointing functionality for implicit coupling

```
def __init__(self, adapter_config_filename='precice-adapter-config.json'):
    self._interface = precice.Interface(...)

def initialize(self, coupling_subdomain, read_function_space=None,
↪  write_object=None):
    precice_dt = self._interface.initialize()

def read_data(self):
    return data

def write_data(self, write_function):

def create_coupling_expression(self):
    return CouplingExpression(...)

def update_coupling_expression(self, coupling_expression, data):
    coupling_expression.update_boundary_data(nodal_data, x_coordinates,
    ↪  y_coordinates)

def get_point_sources(self, data):
    return x_PointSources, y_PointSources

def store_checkpoint(self, user_u, t, n):
    self._checkpoint = SolverState(user_u.copy(), t, n)
    self._interface.mark_action_fulfilled(
    ↪  precice.action_write_iteration_checkpoint() )

def retrieve_checkpoint(self):
    self._interface.mark_action_fulfilled(
    ↪  precice.action_read_iteration_checkpoint() )
    return self._checkpoint.get_state()
```

Modifying a FEniCS Program to couple using FEniCS-preCICE

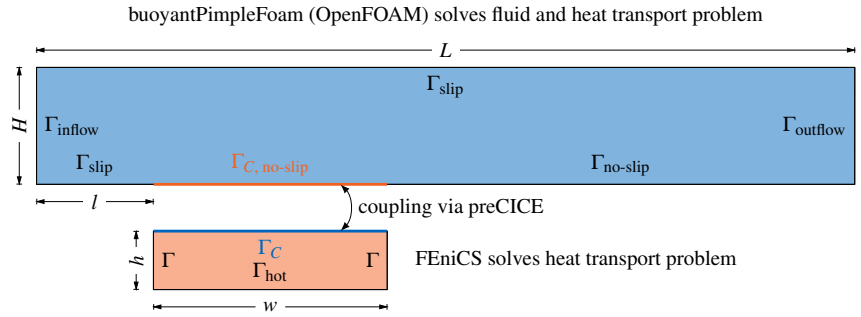
```
from fenics import *

mesh = UnitSquareMesh(10, 10)
class Boundary(SubDomain): ...

V = V_bc = FunctionSpace(mesh, 'P', 2)
u, v = TrialFunction(V), TestFunction(V)
u_D = Expression('...',degree=2)
uncoupled_bc = DirichletBC(V_bc, u_D, Boundary)

# Define initial condition and weak form in FEniCS
...

for t in np.arange(0,T,dt):
        solve(lhs(F) == rhs(F), u, [uncoupled_bc])
```

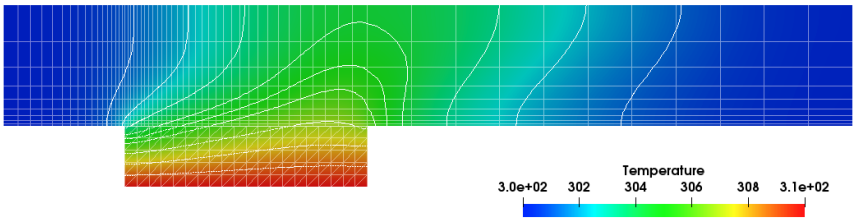
Modifying a FEniCS Program to couple using FEniCS-preCICE

```python
from fenics import *


mesh = UnitSquareMesh(10, 10)
class Boundary(SubDomain): ...


V = V_bc = FunctionSpace(mesh, 'P', 2)
u, v = TrialFunction(V), TestFunction(V)

u_D = Expression('...',degree=2)
uncoupled_bc = DirichletBC(V_bc, u_D, Boundary)




# Define initial condition and weak form in FEniCS
...

for t in np.arange(0,T,dt):



        solve(lhs(F) == rhs(F), u, [uncoupled_bc])
```

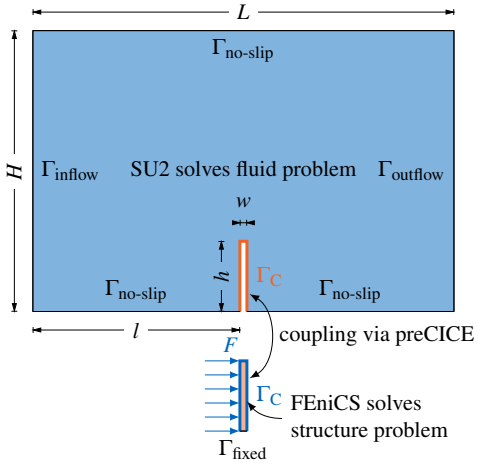
```python
from fenics import *
from fenicsprecice import Adapter

mesh = UnitSquareMesh(10, 10)
class Boundary(SubDomain): ...
class CouplingBoundary(SubDomain): ...

V = V_bc = FunctionSpace(mesh, 'P', 2)
u, v = TrialFunction(V), TestFunction(V)
V_flux = VectorFunctionSpace(mesh, 'P', 1)
u_D = Expression('...',degree=2)
uncoupled_bc = DirichletBC(V_bc, u_D, Boundary)

adapter = Adapter("precice-adapter-config.json")
precice_dt = adapter.initialize(CouplingBoundary, read_function_space=V_bc,
↪  write_object=V_flux)
u_C = adapter.create_coupling_expression()
coupled_bc = DirichletBC(V_bc, u_C, CouplingBoundary)

# Define initial condition and weak form in FEniCS
...

while adapter.is_coupling_ongoing():
  read_data = adapter.read_data()
  adapter.update_coupling_expression(u_C, read_data)
  dt.assign(np.min([fenics_dt, precice_dt]))

  solve(lhs(F) == rhs(F), u_np1, [uncoupled_bc, coupled_bc])
  flux = some_postprocessing(u_np1, V_flux)

  adapter.write_data(flux)
  precice_dt = adapter.advance(dt(0))

  u_n.assign(u_np1)
  t += float(dt)
```

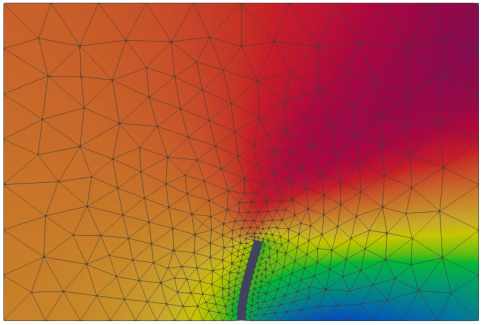
Example Case: Conjugate Heat Transfer Coupling with FEniCS and OpenFOAM





Example Case: Fluid-Structure Interaction Coupling with FEniCS and SU2





Getting FEniCS-preCICE and its Dependencies

Getting the Adapter:

- Maintained under a open-source license here: `https://github.com/precice/fenics-adapter`
- Easy to install: `pip3 install fenicsprecice`
- Latest release can be found here: `https://github.com/precice/fenics-adapter/releases`
- Other dependencies such as Scipy, Numpy, Cython, mpi4py are installed automatically during the adapter installation

Dependencies

- Python (`python3`)
- preCICE (obviously)
- FEniCS
- Python-bindings for preCICE: `pip3 install --user pyprecice`

Summary

- preCICE is a coupling library for partitioned, black-box coupling. Designed for highly flexible and massively parallel use
- FEniCS-preCICE is an adapter to couple FEniCS programs with other software codes using preCICE
- Adapter supports 2D FEniCS cases
- Adapter handles FEniCS data structures and distributed parallelization automatically
- Adapter is modular and flexible to use
- Installation is straightforward using `pip`

Always happy with contributions from the community!

Immediate help required:

- Extending adapter to handle 3D FEniCS cases
- Implementing multiple coupling interfaces handling
- Modify adapter to support FENICS-X and DOLFIN-X

Adding tutorials of coupled problems which use FEniCS
Research collaboration with the preCICE team

Pre-print of reference paper: *Benjamin Rodenberg, Ishaan Desai, Richard Hertrich, Alexander Jaust, and Benjamin Uekermann. FEniCS-preCICE: Coupling FEniCS to other Simulation Software:* `https://arxiv.org/abs/2103.11191`