Design and Operation of Multiscale Modelling Platform

B. Patzák, V. Šmilauer, G. Pacquaut

Czech Technical University in Prague
Faculty of Civil Engineering
Department of Mechanics
Czech Republic

CM3
Computational Multi Physics, Multi Scales and Multi Big Data in Transport Modeling, Simulation and Optimization

Jyväskylä Congress Centre, Finland

May 26, 2015
Outline

- Background and motivation
- Multi-Physics Integration Framework (MuPIF)
  - Design
  - Implementation
  - Application interface
  - Minimal working example
  - Distributed scenario
  - Examples
- Conclusions
Background and motivation

- Learner's dictionary: Platform is a program or set of programs that controls the way a computer works and runs other programs
- FP7 - Multiscale Modelling Platform: Smart design of nano-enabled products in green technologies, 2014-2016
- 8 partners from 5 different countries, 2 industrial end-users

Industrial application for photovoltaic devices and LEDs
- Selenization (CFD+thermodynamics), light conversion (opto+thermal)

- Multi-Physics Integration Framework (MuPIF) to glue various codes
Background and motivation

- The development of a new multiphysics tool for a particular problem would be extremely time and resource consuming.
- A more viable approach lies in combining (reusing) existing single-physics tools to build a customized multiphysics simulation chain for a particular problem.
- Statistics from MMP project partners – 29 responses.

---

**Pie charts showing preferences**
- **Open-source vs. Proprietary**: Open-source is preferred by 64%, Proprietary by 36%.
- **Operating Systems**: Linux dominates with 42%, followed by Windows at 40%.
- **Languages**: C++ is the most popular, followed by Python, Fortran, and other languages.
- **Other**: Post-processing/convertors and visualization are less preferred.
Design of MuPIF framework

- Linux / Windows / Mac operating systems
- Object-oriented, written in Python 2.7
- Installation `pip install mupif`
- ~4000 lines of code
- Uses Pyro4 module and ssh tunnels
- MuPIF allows
  - Data exchange and mesh mapping
  - Integrate/steer individual applications
  - Support for distributed scenarios
## Design of MuPIF framework

<table>
<thead>
<tr>
<th>Application class: defines Application Interface (API). Provides services for data exchange such as <code>getField()</code> and steering such as <code>giveCriticalTimeStep()</code> or <code>solveStep()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Field class: values defined on a mesh - vector, scalar, tensor. Export to VTK</td>
</tr>
<tr>
<td>Property class: values not defined on mesh such as homogenized properties</td>
</tr>
<tr>
<td>Mesh class: topology</td>
</tr>
<tr>
<td>Cell class: representation of a computational cell such as a 2D finite element</td>
</tr>
<tr>
<td>Vertex class: coordinates of a node</td>
</tr>
<tr>
<td>JobManager class: allocation of applications, communications</td>
</tr>
<tr>
<td>PyroUtil class: ssh tunnels, running application server</td>
</tr>
<tr>
<td>Other selected classes: APIerror, EnsightReader2, Function, Octree, PyroFile, Timer, TimeStep, VtkReader2</td>
</tr>
</tbody>
</table>
Application interface - API

- A glue for different simulation tools, **Application** class

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong><strong>init</strong></strong>(self, file)</td>
<td>Constructor. Initializes the application.</td>
</tr>
<tr>
<td>getMesh (self, tstep)</td>
<td>Returns the computational mesh for given solution step.</td>
</tr>
<tr>
<td>getField(self, fieldID, time)</td>
<td>Returns the requested field at given time. Field is identified by fieldID.</td>
</tr>
<tr>
<td>setField(field)</td>
<td>Registers the given (remote) field in application.</td>
</tr>
<tr>
<td>getProperty(self, propID, time, objectID=0)</td>
<td>Returns property identified by its ID evaluated at given time.</td>
</tr>
<tr>
<td>setProperty(self, property, objectID=0)</td>
<td>Register given property in the application</td>
</tr>
<tr>
<td>setFunction(self, func, objectID=0)</td>
<td>Register given function in the application</td>
</tr>
<tr>
<td>solveStep(self, tstep)</td>
<td>Solves the problem for given time step.</td>
</tr>
<tr>
<td>finishStep(self, tstep)</td>
<td>Called after a global convergence within a time step.</td>
</tr>
<tr>
<td>getCriticalTimeStep()</td>
<td>Returns the actual critical time step increment.</td>
</tr>
<tr>
<td>getApplicationSignature()</td>
<td>Returns the application identification</td>
</tr>
<tr>
<td>terminate()</td>
<td>Terminates the application.</td>
</tr>
</tbody>
</table>

B. Patzák, V. Šmilauer, G. Pacquaut, Czech Technical University in Prague
Native and indirect API

- **Native** implementation
  - Direct calls to application's function
  - Python interface (python code, wrapper-Swig, Boost)

- **Indirect** implementation
  - Wrapper around I/O streams or files, more common

![Diagram](image)
API – indirect implementation

Typical workflow

Set → Solve → Get

Remember

produce input file → Execute

Get

Parse output

Typical actions on interface

```
def setProperty(self, property, objectID=0):
    if (property.getPropID() == PropertyID.PID_Concentration):
        # remember the mapped value
        self.values.append(property.getValue())
    else:
        raise APIError.APIError ('Unknown property ID')

def solveStep(self, tstep, stageID=0, runInBackground=False):
    f = open('app3.in', 'w')
    # process list of mapped values and store them into an external file
    for val in self.values:
        f.write(str(val)+'\n')
    f.close()
    # execute external application to compute the average
    os.system("python ./application3.py")

def getPropID(self, propID, tme, objectID=0):
    if (propID == PropertyID.PID_CumulativeConcentration):
        # parse output of application3
        f = open('app3.out', 'r')
        answer = float(f.readline())
        f.close()
        return Property.Property(answer, PropertyID.PID_CumulativeConcentration,
                                  ValueType.Scalar, tme, propID, 0)
    else:
        raise APIError.APIError ('Unknown property ID')
```
Example - Thermo-mechanical coupling

- Staggered solution (one way coupling)

\[ \nabla \cdot \lambda \nabla T + Q(T) = \rho c \frac{\partial T}{\partial t} \]

\[ \nabla \cdot \sigma = 0 \]

\[ d \sigma = C':(d \varepsilon - d \varepsilon'') - d \varepsilon_p - d \varepsilon_f \]
from mupif import *
import application1; import application2

time = 0; timestepnumber=0; targetTime = 100.0 #in hours

app1 = application1.application1(None) # create an instance of application #1
app2 = application2.application2(None) # create an instance of application #2

while (abs(time -targetTime) > 1.e-6): # loop over time steps
    dt2 = app2.getCriticalTimeStep()
    dt = min(app1.getCriticalTimeStep(), dt2)
    time = time+dt
    if (time > targetTime):
        time = targetTime
        timestepnumber = timestepnumber+1
    istep = TimeStep.TimeStep(time, dt, timestepnumber)

    try:
        app1.solveStep(istep) #solve problem 1
        t = app1.getField(FieldID.FID_Temperature, istep) #request temperature field from app1
        app2.setField(t) # register temperature field in app2
        app2.solveStep(istep) # solve second sub-problem
        stress = app2.getField(FieldID.FID_stress, istep)

    except APIError.APIError as e:
        logger.error("Following API error occurred: %s" % e)

app1.terminate(); app2.terminate();
Data transfer between computers

- SSH tunnels with private/public keys
- NAT mapping of remote ports via ssh tunnel
- Bypass firewall, ensure data secure transfer

 Unix: `ssh -L 4000:mech.fsv.cvut.cz:3300 mmp@mech.fsv.cvut.cz`
 Windows: `putty.exe -L 4000:mech.fsv.cvut.cz:3300 mmp@mech.fsv.cvut.cz`
Distributed scenario

- Example06-JobMan

```python
from mupif import *
...
ns = PyroUtil.connectNameServer(nhost=conf.nhost, nsport=conf.nsport, hkey=conf.hkey) #locate nameserver

#establish secure tunnel to JobManager running on (remote) server
try:
    app1Rec = PyroUtil.allocateApplicationWithJobManager (ns, conf.app1JobManRec, conf.jobNatPorts.pop())
    app1 = app1Rec.getApplication()
...
app1.solveStep(istep)
...
```
Examples coming with MuPIF

Example01 – Top level script illustrating simple data exchange between two applications on a single computer
Example02 – Example01 in a distributed version
Example03 – Example01 extended to illustrate indirect implementation of application interface (interaction with application available as external executable)
Example04 – Generation of a field, mapping to another mesh and outputting VTK files.
Example05 – reading Ensight files
Example06 – Illustrates client-server setup, ssh tunnel, simple data transfer
Example06-JobMan – the previous example using a job manager
Example07, Example09 – a vtk reader and unit representation
PingTest – Example06 extended for several server locations and testing connectivity time
PingTest

- Check running MuPIF servers across MMP partners
Conclusions

- MuPIF platform can glue various codes
- Demonstrated on simple local/distributed scenarios
- Acknowledgement to FP7 project “Multiscale Modelling Platform: Smart design of nano-enabled products in green technologies” and the partners

Thank you