Functional Mock-Up Interface (FMI) for Tool-Independent Parallel Distributed Simulations

Robert Braun
Division of Fluid and Mechatronic Systems
Linköping University
Hopsan Simulation Tool
Distributed Solvers

Centralized solver

Distributed solvers!
Distributed Solvers

Solver

Equation System

Solver  Solver  Solver  Solver

Component  Component  Component  Component

Equations  Equations  Equations  Equations

TLM  TLM  TLM
Distributed Solvers

Single-threaded:

- FMU 1
- FMU 2
- FMU 3
- FMU 4
- FMU 5

Step Time

Step Time
Experimental Results

![Graph showing simulation time vs. workload for different thread counts]
Functional Mockup Interface (FMI)

- Engine with ECU
- Gearbox with ECU
- Thermal systems
- Automated cargo door
- Chassis components, ECU (e.g. ESP)

Functional mockup interface for dynamic models
Functional Mockup Interface (FMI)

Functional Mockup Unit (zip-file)

- MyModel.fmu
- modelDescription.xml
- MyModel.dll
- MyModel.c
- MyModel.png
- documentation.html
Model Import for Distributed Simulations

Hopsan

Generator

Component 4

C++

OpenModelica

Dymola

Simulink
Model Import for Distributed Simulations

Hopsan

Component 1  Component 2
Component 3  Component 4

Generator
Model Import for Distributed Simulations

<table>
<thead>
<tr>
<th>Processor</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 1</td>
<td>Core 2</td>
</tr>
<tr>
<td>Core 3</td>
<td>Core 4</td>
</tr>
</tbody>
</table>

Done!

Hopsan

Component 1
Component 2
Component 3
Component 4
Importing an FMU

- ModelName.fmu
- ModelName.xml
- modelDescription.xml
- fmuLib.cc
- HopsanCore.dll
- additional FMI files
- MinGW
- ModelName.dll
- 7-Zip
- ModelName.dll
Exporting an FMU

Hopsan

HopsanCore.dll

HopsanFMU.h

HopsanFMU.cpp

ModelName.c

modelDescription.xml

ModelName.hmf

MinGW

HopsanFMU.dll

MinGW

ModelName.dll

7-Zip

modelName.fmu
Exporting an FMU

my_model.fmu

binaries

win32

my_model.dll

HopsanFMU.dll

HopsanCore.dll

resources

my_model.hmf

modelDescription.xml
Transmission Line Elements & FMI

\[ F_{1}(t - \Delta t) \quad v_{1}(t - \Delta t) \quad \text{Transmission Line Element} \quad c_{2}(t) \quad Z_{c} \quad F_{2}(t) = c_{2}(t) + Z_{c}v_{2}(t) \]

\[ \Rightarrow \begin{cases} c \approx \text{force} \\ Z_{c} \approx \text{damping} \end{cases} \]
Transmission Line Elements & FMI

FMU

Transmission Line Element

FMU
Transmission Line Elements & FMI

\[ F_1 \, v_1 \, c_1 \, Z_1 \]

\[
\text{FMU}
\]

\[
\text{<hopsanfmu>}
\text{<tlmport type="mechanic">}
\text{<output>f_1</output>}
\text{<output>v_1</output>}
\text{<input>c_1</input>}
\text{<input>Z_1</input>}
\text{</tlmport>}
\text{</hopsanfmu>}
\]
Work Progress

FMU SDK

With manual modifications

Hopsan

In progress

OpenModelica

Dymola

AMESim

With manual modifications

In progress
Example Model

```model ModelicaSwashPlate
output Real t1(start = 0) "torque";
input Real w1(start = 0) "speed";
input Real angle(start = 0) "plate angle";
output Real F1(start = 0) "force";
output Real x1(start = 0) "position";
output Real v1(start = 0) "velocity";
output Real me1(start = 0) "equivalent mass";
input Real c1(start = 0) "wave variable";
input Real Zc1(start = 0) "characteristic impedance";

parameter Real r = 0.05 "swashplate radius";
parameter Real x_0 = 0 "angular offset";
parameter Real x_0 = 0 "piston offset";
Real np "number of pistons";
Real a1 "shaft angle";
Real dp "angular difference between pistons";
Real smax "maximum piston stroke";

equation
np = 8;
der(a1) = w1;
smax = r * tan(angle);
dp = (2 * 3.141592653589793) / np;
F1 = c1 + Zc1 * v1;
x1 = x_0 + smax * sin(a1 - th_0);
v1 = smax * cos(a1 - th_0) * w1;

algorithm
[...]
end ModelicaSwashPlate;
```

```class MechanicSwashPlate : public ComponentQ {
private:
    Port *mpIn1, *mpIn2, *mpOut1, *mpP1;
    double *mpND_in1, *mpND_in2, *mpND_out1;
    std::vector<double*> mvpND_f1, mvpND_x1, mvpND_v1, mvpND_c1, mvpND_Zc1, mvpND_me1;
    size_t mNumPorts1;
    Integrator mIntegrator;
    double r, offset, startX;

public:
    static Component *Creator() {
        return new MechanicSwashPlate();
    }
    void configure() {
        r = 0.05;
        offset = 0.0;
        registerParameter("r", "Swivel Radius", "[m]", r);
        registerParameter("th_offset", "Angle offset", "[m]", offset);
        mpIn1 = addReadPort("angle", "NodeSignal");
        mpIn2 = addReadPort("movement", "NodeSignal");
        mpOut1 = addWritePort("torque", "NodeSignal");
        mpP1 = addPowerMultiPort("P1", "NodeMechanic");
    }
    void initialize() {
        [...]
    }
    void simulateOneTimestep() {
        [...]
    }
};
```
Example Model

Simulation performance [ms/iteration]

FMU component: 0.014453 ms
C++ component: 0.002513 ms
Remaining Problems

- How to handle relative search paths?
  - DLLs depending on other DLLs
  - Resource files
- Performance improvements
- Intuitive user interface
Thank you!

Questions?