Wrapper for Functional Mock-up Interface in TLM-based Asynchronous Co-Simulation

Robert Braun, SICS East Swedish ICT

2016-10-19

1. Introduction

Functional Mock-up Interface (FMI) is a tool-independent standard for connecting simulation tools [2]. One tool can export a model as a Functional Mockup Unit (FMU), a ZIP package with the file extension FMU. This file is in turn loaded by the master simulation tool, which can connect and simulate the model. An FMU file contains a model description XML file called `modelDescription.xml`, binaries for different platforms and other optional content. It is important that the FMU contains a binary file for the platform where the master simulation tool is executed.

There are two versions of the FMI standard: FMI for Co-Simulation and FMI for Model Exchange. The main difference is that FMUs for Co-Simulation contain their own built-in solvers, and only exchange data at predefined communication points. FMUs for Model Exchange require a solver in the master simulation tool.

Including FMUs in the TLM framework requires a wrapper. `FMIWrapper` is a generic wrapper for connecting functional mockup units (FMUs) to the TLM framework. It uses the FMI Library from Modelon [1] to load an FMU, and the TLMPlugin for socket communication with the framework, see fig. 1.

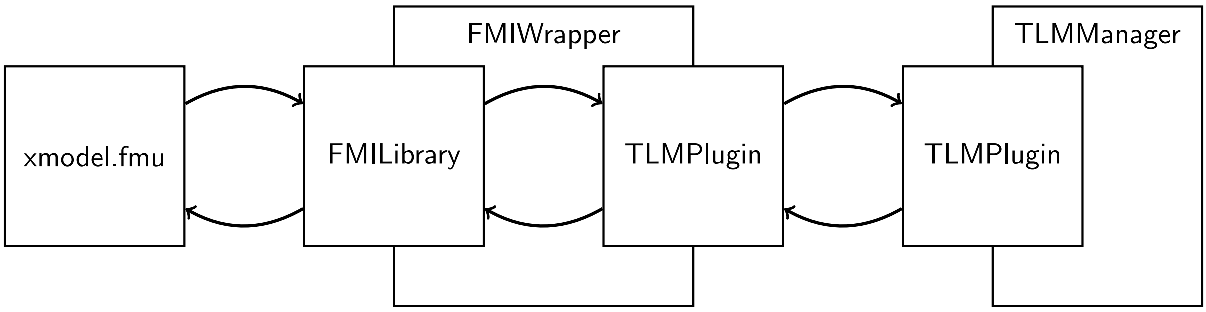

Figure 1: Blabla...

Both FMI for co-simulation and FMI for model exchange are supported. Model exchange requires a solver in the wrapper executable. For this reason, two solvers from the Sundials package are included.

2. Setting up a simulation

The FMI wrapper is started by the StartTLMFMIWrapper startup script. This script generates the `tlm.config` file and calls the FMIWrapper executable. The executable takes the following arguments:

```
FMIWrapper <path> <fmufile> <solver> <debug>
```

Example:

```
FMIWrapper C:\temp\folder mymodel.fmu solver=CVODE -d
```

The last to arguments are optional. Available solvers are `Euler`, `RungeKutta`, `CVODE` and `IDA`. Section 4 contains more details about the different solvers. These can currently only be changed by modifying the startup script, i.e. not from the graphical interface.

An FMU keep track of its variables by integer numbers called `value references`. However, it does not provide any information about the mapping between its variables and the TLM interface. Hence, this information must be provided by the user. A configuration file called `fmi.config` is used for this purpose, see listing 1.

Listing 1: A configuration file maps value references to TLM variables

```
substeps,100
name,tlm1
position,6,7,8
orientation,136,139,142,137,140,143,138,141,144
speed,9,10,11
ang_speed,145,146,147
force,133,134,135,167,168,169
name,tlm2
position,12,13,14
orientation,145,146,147,148,149,150,151,152,153
speed,15,16,17
ang_speed,154,155,156
force,136,137,138,170,171,172
```

Listing 2: Value references for variables are obtained from `modelDescription.xml`

```
<ScalarVariable
  name="fMITLMInterface3D1.t[1]"
  valueReference="167"
  variability="continuous"
  causality="local"
  >
  <Real/>
</ScalarVariable>
```

As can be seen, data is stored in a comma-separated format. The first line specifies the number of substeps used for FMI for Co-Simulation (see section 3). After this comes the port information. Each port is specified by name, position, orientation, speed, angular speed and force, according to the TLM interfaces in the TLM framework. The numbers after each keyword are the value references. These can be obtained by analyzing the `modelDescription.xml` file. Listing 2 shows an example. The variable first torque component has the value reference 167. Hence, this number should be inserted as number four on the ”force” line in `fmi.config`.

3. FMI for Co-Simulation

With FMI for Co-Simulation the solver is embedded within the FMU. Variables can only be exchanged at predefined communication points. Hence, it is not possible for the solver to obtain interpolated force variables during internal iteration steps. Keeping the force constant during the entire communication interval may, however, have a negative effect on numerical stability. For this reason it is possible to divide each communication interval into a fixed number of *sub steps*, as defined in the `fmi.config`. In this way the forces can at least be updated in the FMU at more fine-grained intervals. Pseudo-code for the simulation loop in the wrapper is shown in listing 3. The real code also contains error handling, but it has been excluded here to enhance readability.

Listing 3: Pseudo code for the simulation loop with FMI for co-simulation

```
while (tcur < tend) {
    double hsub = hmax/nSubSteps;
    for(size_t i=0; i<nSubSteps; ++i) {
        x = fmu.get_real(x_vr,3);
        T = fmu.get_real(T_vr,9);
        v = fmu.get_real(v_vr,3);
        w = fmu.get_real(w_vr,3);
        f = TLMPlugin.GetForce(tcur,x,T,v,w);
        fmu.set_real(f_vr[j],6,f);

        TLMPlugin.SetMotion(tcur,x,T,v,w);

        fmu.do_step(tcur,hsub);
        tcur+=hsub;

        x = fmu.get_real(x_vr,3);
        T = fmu.get_real(T_vr,9);
        v = fmu.get_real(v_vr,3);
        w = fmu.get_real(w_vr,3);
    }
}
```

Note that it is necessary to read the motion variables from the FMU before obtaining the force from the TLMPlugin. At the end of each major step it is also necessary to call `GetForce()` before calling `SetMotion()`. The reason for this is that `SetMotion()` requires updated input variables which are retreived by `GetForce()`.

4. FMI for model exchange

With model exchange, the wrapper must provide a solver for the FMU. Three solvers are available: explicit Euler, 4th order Runge-Kutta, and the CVODE and IDA solvers from Sundials [3]. Listing 4 shows pseudo code for one major step (i.e. one communication interval) with the IDA solver. Note that the solver is used with one step mode. This means that it takes one step at a time, until its internal time exceeds the next communication interval. The CVODE solver requires a callback function for obtaining derivatives of state variables (i.e. "right-hand side"). The IDA solver requires a similar callback for obtaining the residuals.

Listing 4: Pseudo code for the simulation loop with FMI for model exchange

```
double position[3],orientation[9],speed[3],ang_speed[3],force[6];

x = fmu.get_real(x_vr,3);
T = fmu.get_real(T_vr,9);
v = fmu.get_real(v_vr,3);
w = fmu.get_real(w_vr,3);
f = TLMPlugin.GetForce(tcur,x,T,v,w);
fmu.set_real(f_vr[j],6,f);

y = fmu.get_continuous_states();
dy = fmu.get_derivatives();

tcur += h;

while(tc < tcur){
    IDASolve(mem, tcur, &tc, y, dy, IDA_ONE_STEP);
}

fmu.set_continuous_states(y);
```

5. Example: Double Pendulum with Dymola and OpenModelica

This section will explain how to build a meta model of a double pendulum using FMUs exported from Dymola and OpenModelica. Figure 2 shows a sketch of the model. The resulting meta model is found in the `/MetaModels/FmuFmuPendulum` folder. One FMU for Co-simulation and one for Model Exchange will be used. Both models use a custom Modelica library called FMILIB for the TLM interfaces. This library is shipped together with the FMI wrapper.

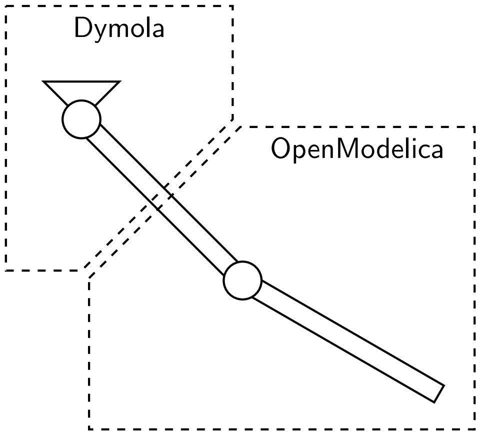

Figure 2: A double pendulum is simulated using FMUs exported from Dymola and OpenModelica

5.1. Preparing Dymola FMU of first part pendulum

The Dymola model consists of half the first pendulum and the fixed attachment to the inertial system, see figure fig. 3. It is normally advisable to decouple a model at its weakest point, which in this case would be at the joint between the two pendulum arms. In this case the model is decoupled in the middle of the first pendulum only to demonstrate the possibilities. Force and torque variables must be specified as input variables at the top level of the model, in order to get the correct number of equations and variables. Hence, we need to add the equations shown in listing 5.

Listing 5: Input variables must be specified on top level in the Modelica models

```
  [...]
  //Define FMI interface model
  FMITLM.FMITLM_Interface_3D.FMITLMInterface3D fMITLMInterface3D;

  //Define input force and input torque
  input Real f[3](start = zeros(3));
  input Real t[3](start = zeros(3));
equation
  //Assign force and torque in interface model with input variables
  fMITLMInterface3D.f = f;
  fMITLMInterface3D3D.t = t;
  [...]
```

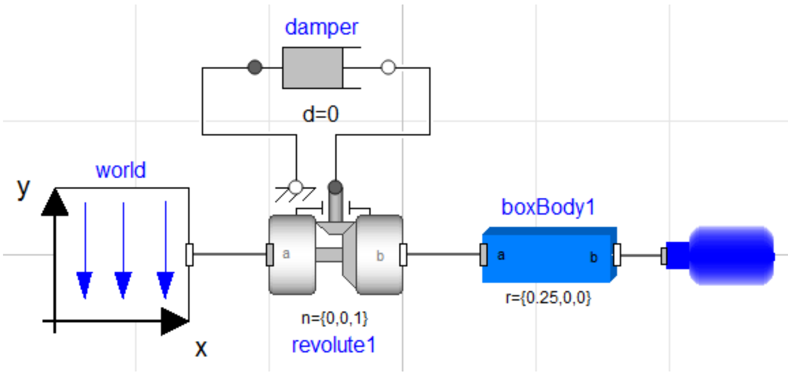

Figure 3:

Now it is time to export the model so it can be used with the framework.

1. **Transform the model to an FMU for co-simulation.**
   Use a suitable solver, for example CVODE.
2. **Put the `.fmu` file in a new subfolder.**
3. **Create an empty text file called `fmi.config` in the same folder.**
4. **Specify number of substeps on the first line.**
   For example: "substeps,10"
5. **Open `modelDescription.xml`.**
   By opening the `.fmu` as a zip package.
6. **Locate the TLM variables and write the value references in `fmi.config`.**
   The desired variables are position, orientation, velocity, angular velocity, force and torque:
   `fMITLMInterface3D.r[x]`, `fMITLMInterface3D.A[x,y]`,
   `fMITLMInterface3D.v[x]`, `fMITLMInterface3D.w[x]`,
   `fMITLMInterface3D.f[x]`, `fMITLMInterface3D.t[x]`
7. **Enter the value reference for each variable in `fmi.config`.**
   See section section 2 for more information.

5.2. Preparing OpenModelica FMU of second part of pendulum

The OpenModelica model as shown in fig. 4 consists of the second half of the first beam and the second beam. The export process is exaclty the same as for Dymola, except that the FMU exported should be for model exchange.

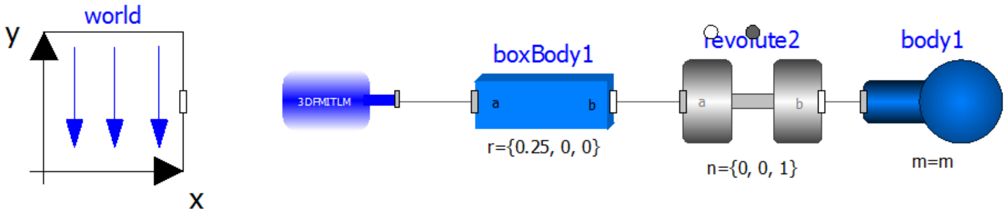

Figure 4: The second half of the first beam and the second beam are modelled in OpenModelica

5.3. Building co-simulation system model

When both FMUs have been generated together with one `fmi.config` file each, it is time to build the meta model.

1. **Create a new meta model.**
   Either by hand or by using the graphical user interface.
2. **Add the FMUs as submodels.**
   Use `StartTLMFmiWrapper` as start command.
3. **Fetch TLM interfaces.**
4. **Add a connection between the two models.**
   Default TLM parameters should work fine.
5. **Make sure interfaces are aligned.**
6. **Choose solver for model exchange.**
   By editing `StartTLMFmiWrapper.bat`.
   Use for example `solver=CVODE` or `solver=IDA`.

When done, the meta model XML file should look similar to listing 6. In the graphical interface, the meta model should look like fig. 5. The model is now ready to be simulated!

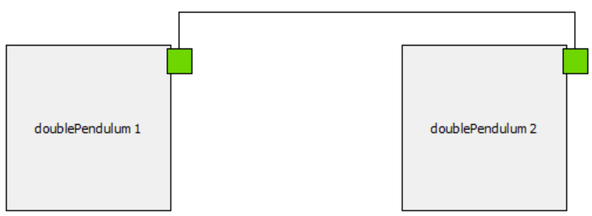

Figure 5: Graphical view of the complete meta model

Listing 6: XML description of the complete meta model

```xml
<?xml version='1.0' encoding='ISO-8859-1'?>
<Model Name="doublePendulum">
  <SubModels>
    <SubModel Name="doublePendulum1"
              ModelFile="doublePendulum1.fmu"
              StartCommand="StartTLMFmiWrapper"
              Position="0,0,0"
              Angle321="0,0,0">
      <InterfacePoint Name="tlm"
                      Position="0.25,0,0"
                      Angle321="0,0,0"/>
    </SubModel>
    <SubModel Name="doublePendulum2"
              ModelFile="doublePendulum2.fmu"
              StartCommand="StartTLMFmiWrapper"
              Position="0.25,0,0"
              Angle321="0,0,0">
      <InterfacePoint Name="tlm"
                      Position="0,0,0"
                      Angle321="0,0,0"/>
    </SubModel>
  </SubModels>
  <Connections>
    <Connection From="doublePendulum1.tlm"
                To="doublePendulum2.tlm"
                Delay="1e-4"
                Zf="10000"
                Zfr="100"
                alpha="0.2">
    </Connection>
  </Connections>
  <SimulationParams ManagerPort="11113" StopTime="3" StartTime="0"/>
</Model>
```

Bibliography

[1] Modelon AB. Fmi library: part of jmodelica.org, 2014.

[2] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf. The Functional Mockup Interface for tool independent exchange of simulation models. In *8th International Modelica Conference 2011*, Como, Italy, September 2009.

[3] Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.