Functional Mock-up Interface

Michael Wetter
Simulation Research Group

October 12, 2016

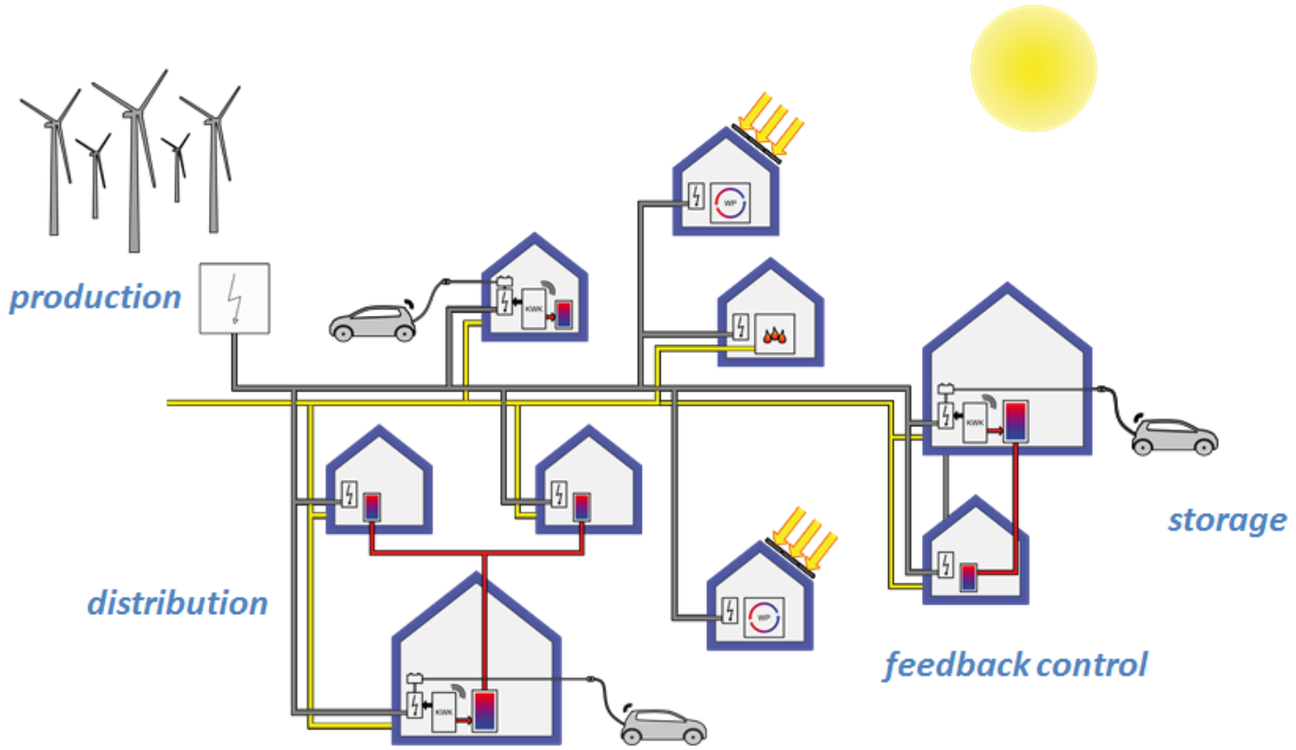
Lawrence Berkeley National Laboratory

Overview

The purpose is to

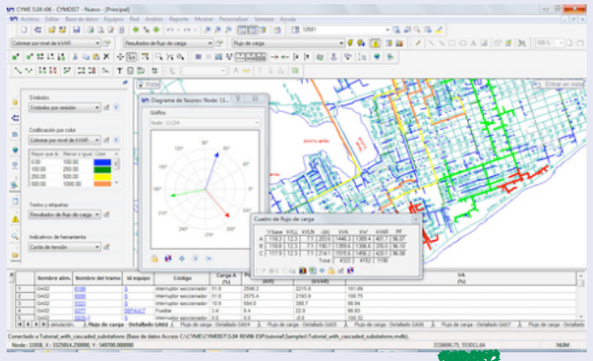
1. get a basic understanding of the Functional Mock-up Interface (FMI) standard,
2. understand the different Functional Mock-up Units (FMU), and
3. learn how to create and simulate a Functional Mock-up Unit.

Use cases

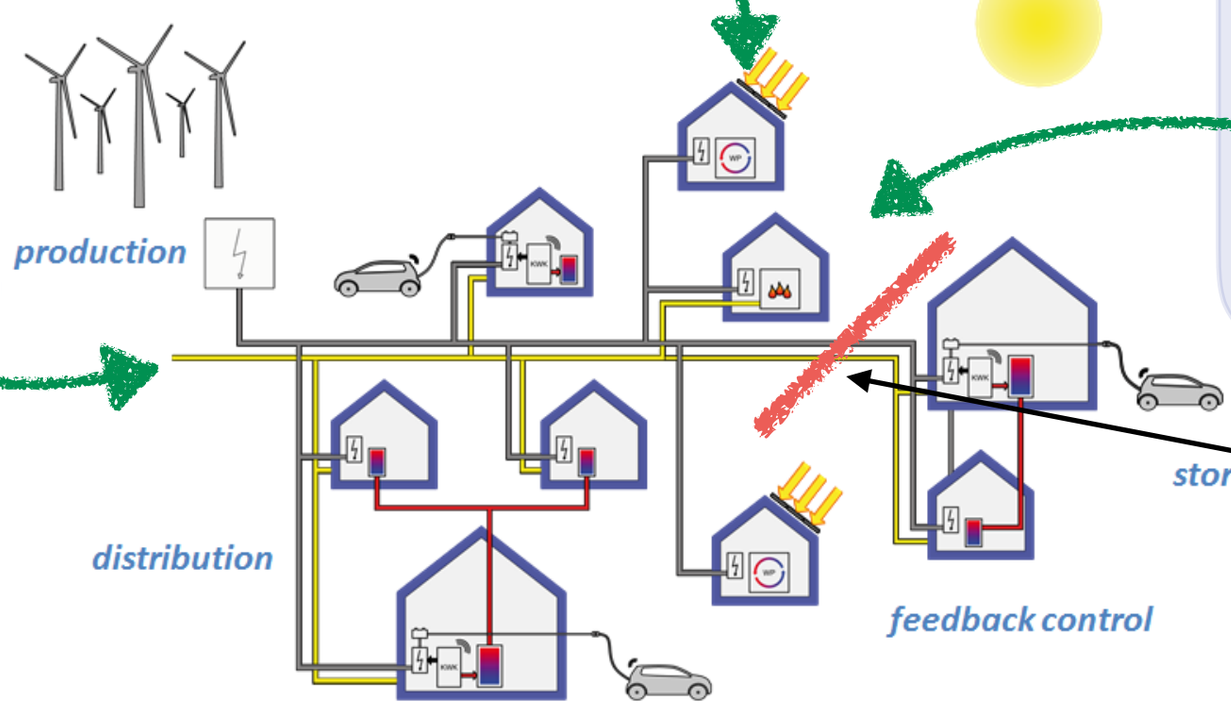


Use cases

Link an electrical grid simulator



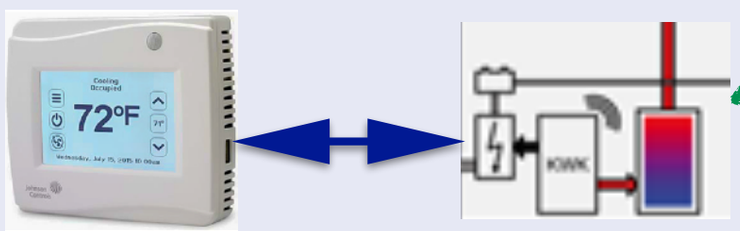
Share models in manufacturer catalog



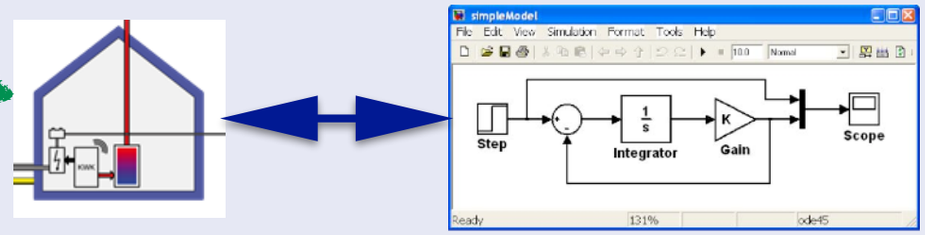
Interface model with hardware for HIL

Partition large models for computing efficiency

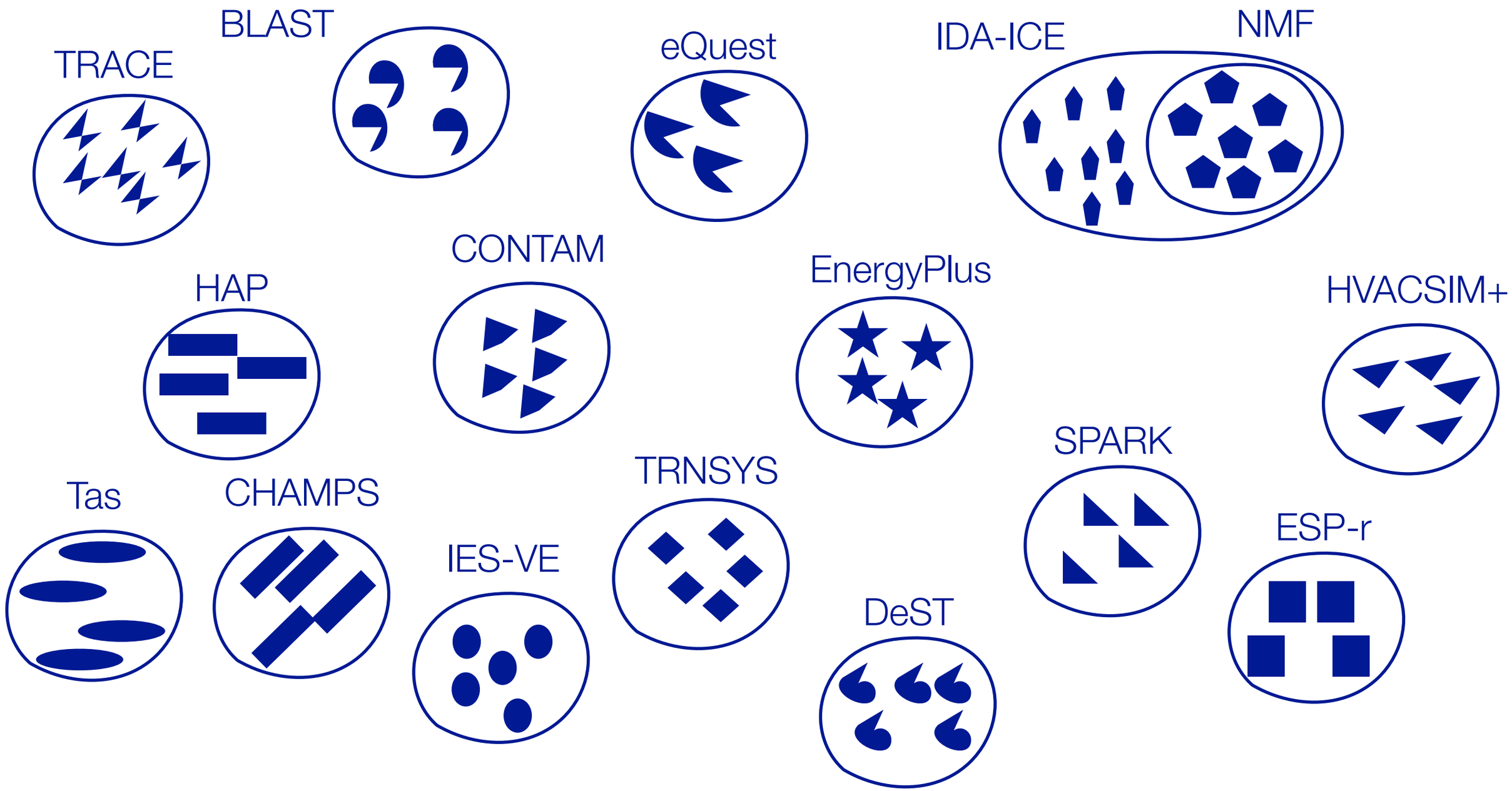
Export model for use in a control system



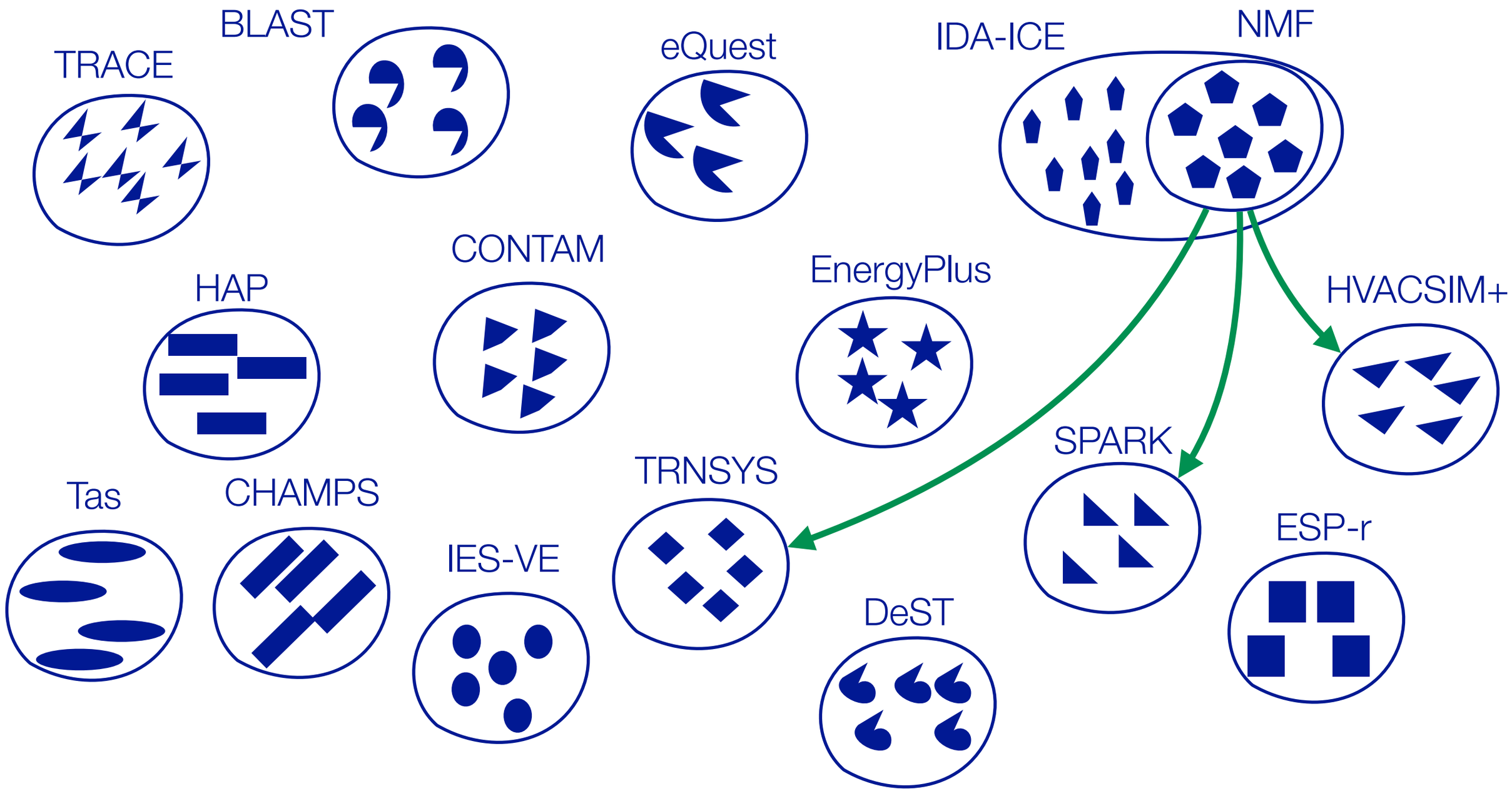
Share a model with a controls engineer



Let us develop building simulation programs, **but** each with a mutually incompatible model format, different semantics and incompatible software architecture…



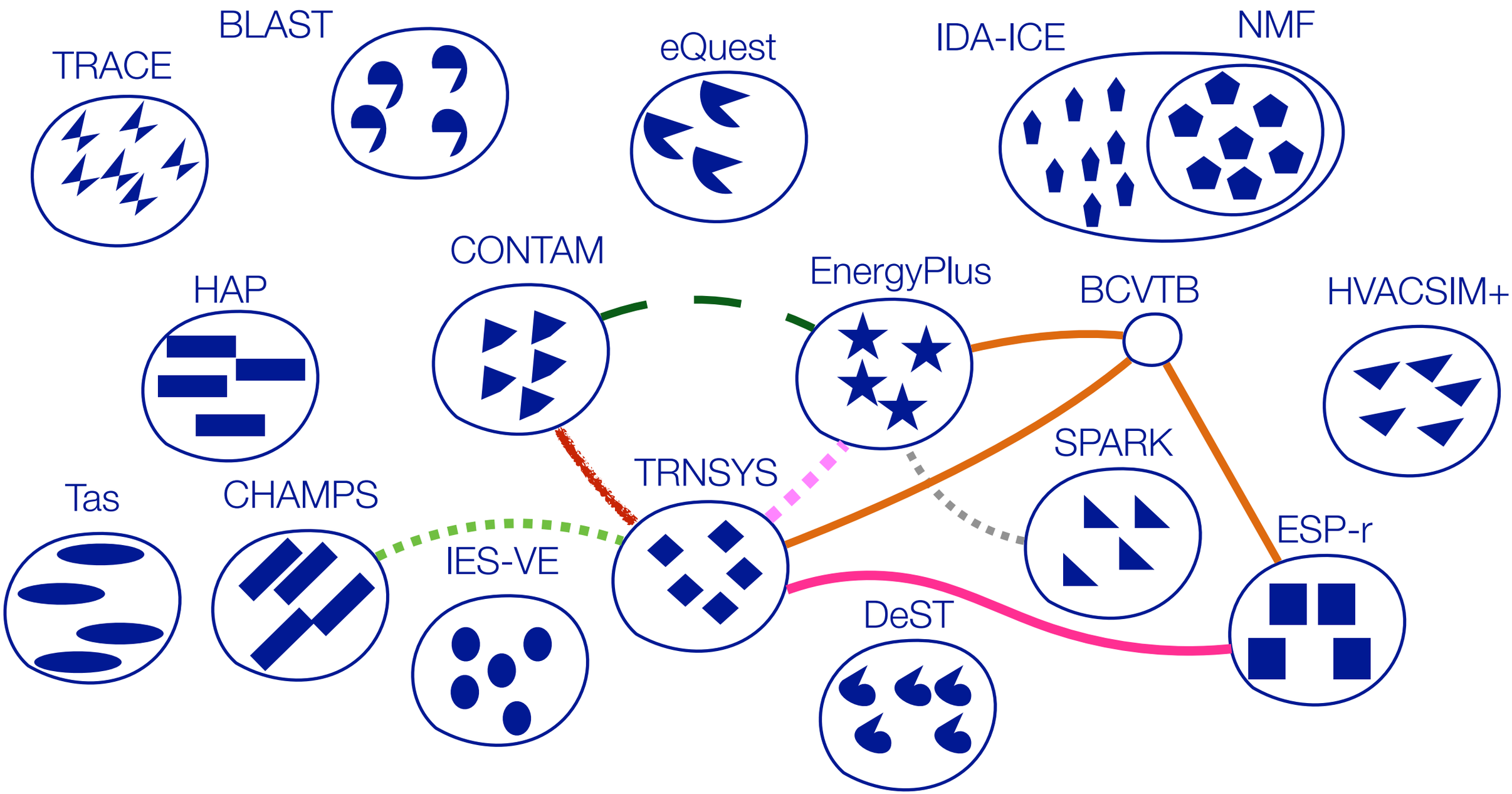
20 years ago, there was a brilliant recognition that models can be developed **once**, stored in a repository and exported to simulators



It probably was ahead of its time, but stopped by ASHRAE TC 4.7.

Per Sahlin and Pavel Grozman. Symbolic Processing and Code Generation of Equation Based HVAC&R Simulation Models. ASHRAE Technical Paper 839, 1996.

In absence of being able to share models, let's co-simulate, but each with a different API and different — if any — semantics….



Looks like a nice idea, but very difficult to realize.
And lacks any standard and rigor until some tools started using FMI.

FUNCTIONAL MOCK·UP INTERFACE

The Functional Mockup Interface has been developed to exchange dynamic models and simulators.

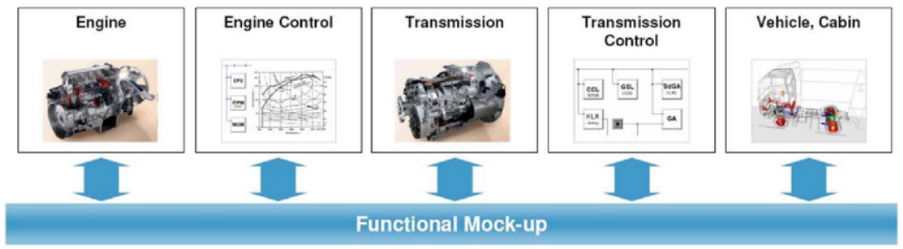
Developed within MODELISAR, an ITEA2 project to improve significantly the design of systems and of embedded software in vehicles.

ITEA project, 28 partners, 178 person years, 26 Mill. € budget, July 2008 - June 2011.

First version published in 2010. Now supported by >80 tools.

Functional Mockup Interface Standard

- defines an open interface, to be implemented by a Functional Mock-up Unit (FMU)
- FMI functions are called by a simulator that imports an FMU
- Coupling may be done locally or across the internet.
- FMU may be
  - self-integrating (co-simulation) or
  - require the master tool to perform the integration (model-exchange)



Cosimulation of the behavioral models and the embedded controller software

The Functional Mockup Interface has been developed to exchange dynamic models and simulators.

FMI separates

- description of interface (xml) from
- functionality (C API).

FMI standardizes

a) a set of C-functions, to be implemented by a model/simulator,

b) an XML-model description file to be provided by a model/simulator, and

c) the distribution file format to be used by a model/simulator.

A model/simulator which implements FMI is called a Functional Mock-up Unit (FMU).

FMUs for co-simulation and model exchange.

Model to be simulated

$$\dot{x}(t) = f\big(x(t), t\big)$$
$$x(0) = x_0$$

Model Exchange

For $(x(t), t)$

returns $f\big(x(t), t\big)$

The master needs to provide a differential equation solver, and synchronize variables.

Co-Simulation

For $(x(t_k), t_k)$

returns $(x(t_{k+1}), t_{k+1})$

that satisfies the differential equation.

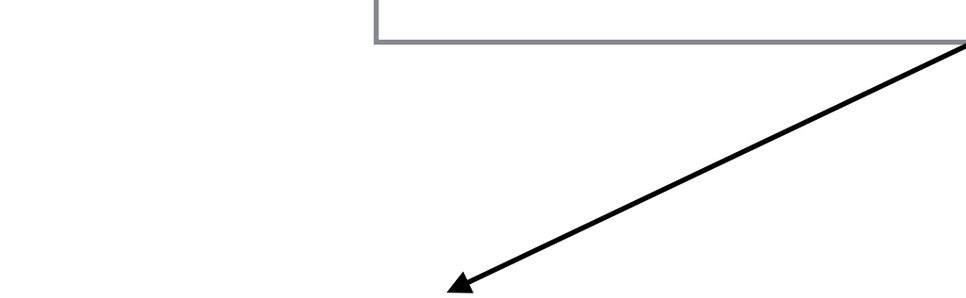
The master only needs to synchronize variables.

Notes: For simplification, discrete variables and algebraic constraints were omitted in slide.

FMI for Co-Simulation and for Model Exchange

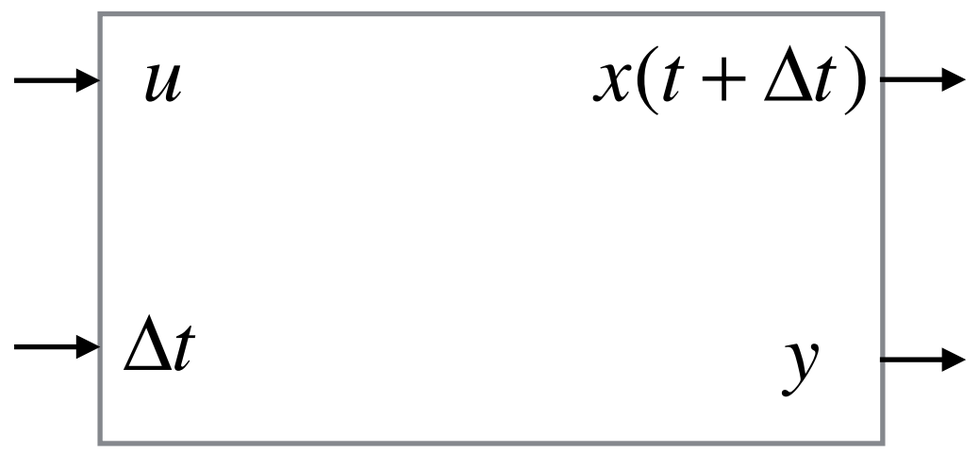
$$\frac{dx}{dt} = u - x,\ x(0) = 1.0$$

$$y = -x$$



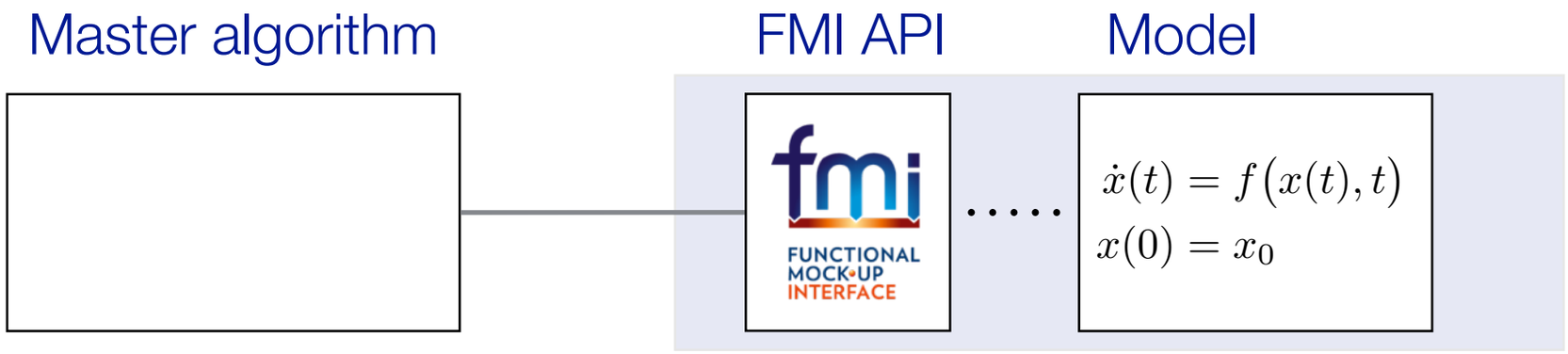
FMI for Model Exchange

FMI for Co-Simulation

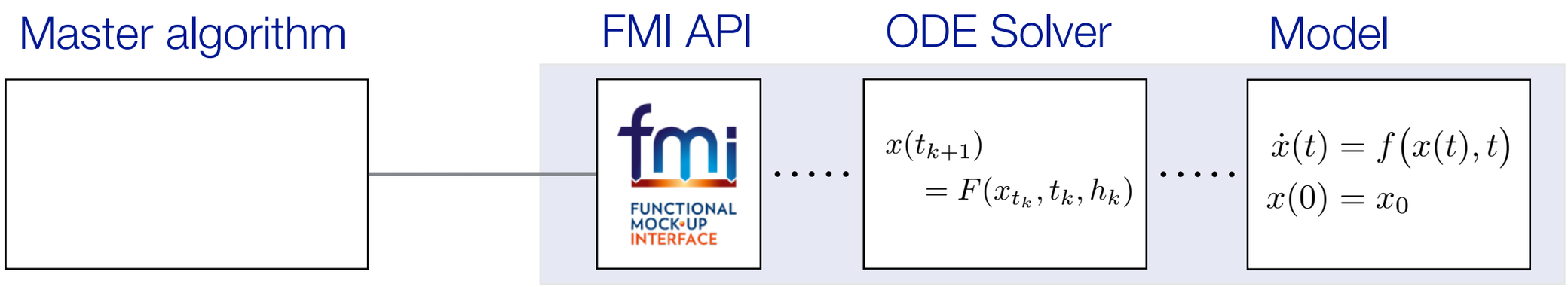


FMI architecture

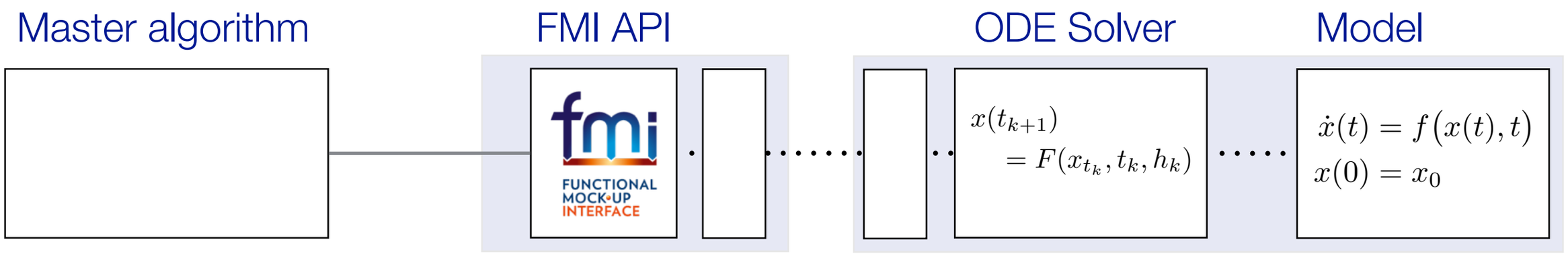
Model exchange



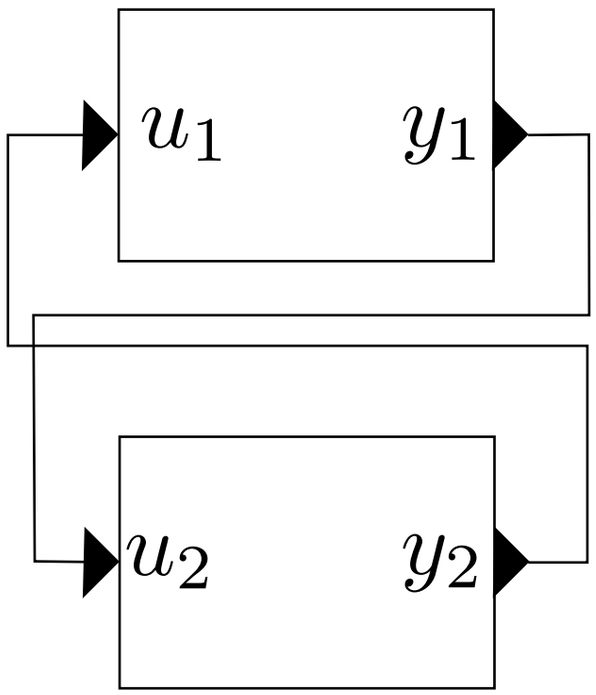
Co-Simulation



Co-Simulation with tool coupling



How do you evaluate compositions of FMUs?



Case 1

$$y_1 = \int_0^t u_1(s)\,ds$$

$$y_2 = (u_2 + 1)^3$$

Case 2

All FMUs have direct feedthrough

$$y_1 = u_1$$

$$y_2 = (u_2 + 1)^3$$

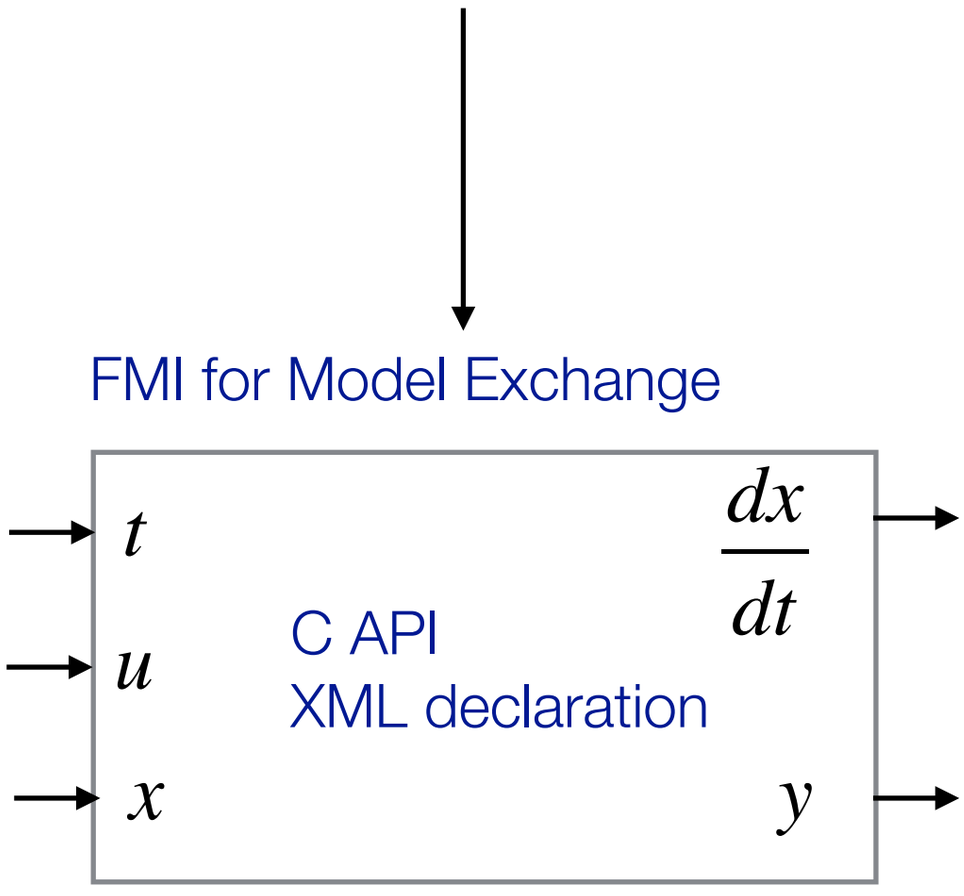
Need to know the model structure, which is optionally declared in the modelDescription.xml file.

Note: This is why in EnergyPlus, the ExternalInterface imposes a one time step delay.

From a model to an FMU

$$\frac{dx}{dt} = u - x,\ x(0) = 1.0$$

$$y = -x$$



XML*- file contains

a) name of variables
b) value reference of variables
c) causality of variables
d) variable dependencies

C API* contains functions to

a) initialize the model
b) set continuous states
c) set inputs
d) get derivatives
e) get outputs
f) terminate the model

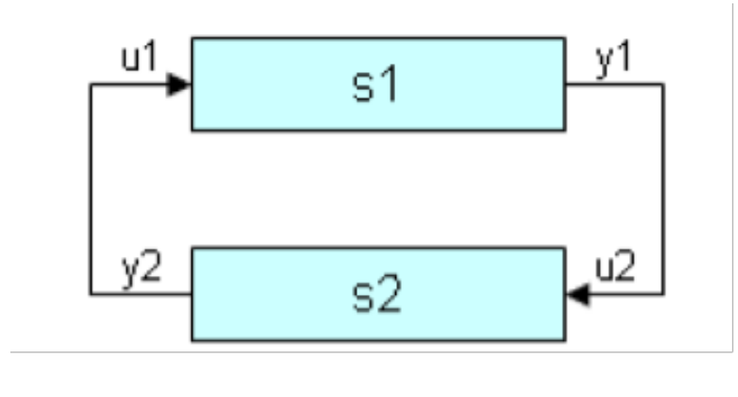
*XML and C API contain additional information which are not listed for simplicity.

An FMU is a zip file with model description, documentation, and binaries or C source code

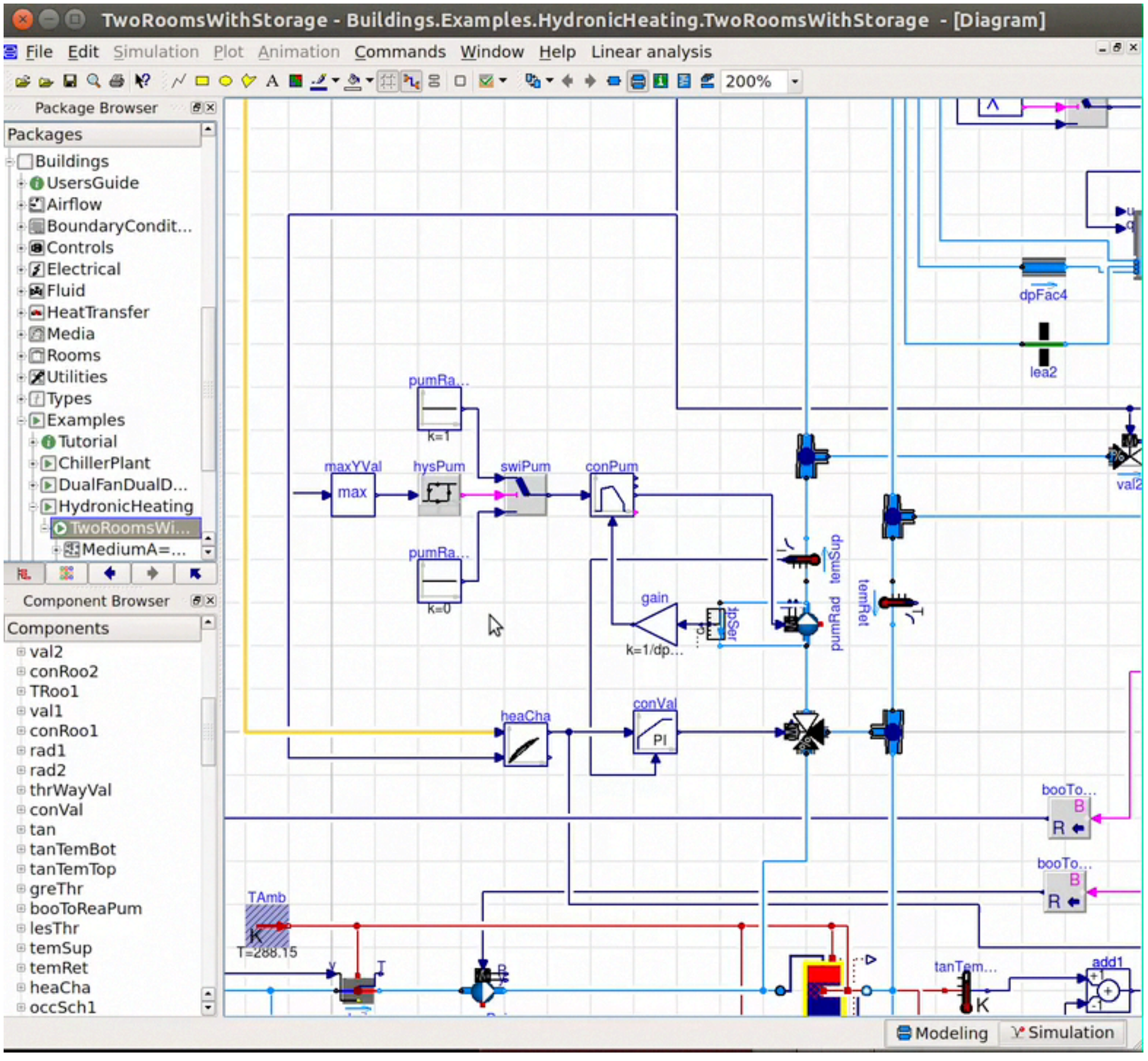
```
// Structure of zip-file of an FMU
modelDescription.xml  // Description of model (required file)
model.png             // Optional image file of model icon
documentation         // Optional directory containing the model documentation
_main.html            // Entry point of the documentation
  <other documentation files>
sources
// Optional directory containing all C-sources
// all needed C-sources and C-header files to compile and link the model
// with exception of: fmiPlatformTypes.h and fmiFunctions.h
binaries           // Optional directory containing the binaries
  win32            // Optional binaries for 32-bit Windows
  <modelIdentifier>.dll  // DLL of the model interface implementation
  // Optional object Libraries for a particular compiler
    VisualStudio8          // Binaries for 32-bit Windows generated with
                           // Microsoft Visual Studio 8 (2005)
    <modelIdentifier>.lib  // Binary libraries
  gcc3.1                   // Binaries for gcc 3.1
   ...
  win64                    // Optional binaries for 64-bit Windows
    ...
  linux32                  // Optional binaries for 32-bit Linux
    ...
  linux64                  // Optional binaries for 64-bit Linux
    ...
resources                  // Optional resources needed by the model
< data in model specific files which will be read during initialization >
```

Integrating a simulator is independent of the tool that generated the simulator

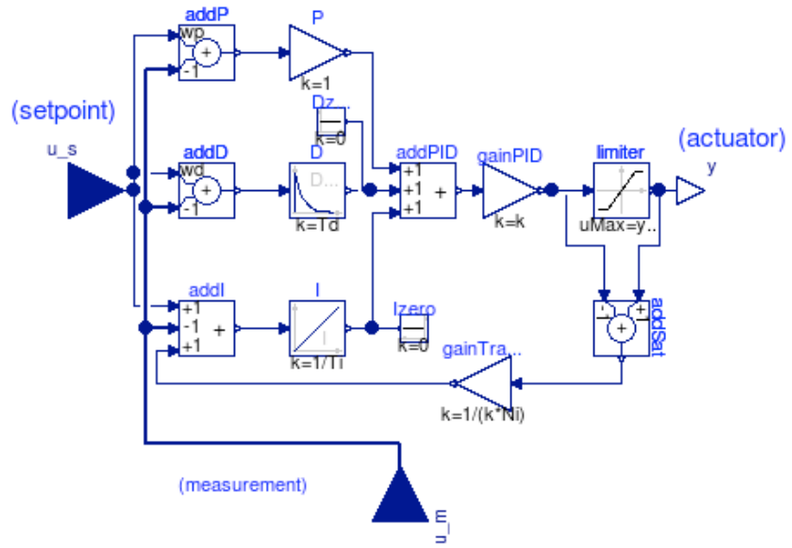
```
//Instantiate models
fmi2Component s1 = s1_mi2Instantiate(“Tool1", "", "Model1", "", ...);
fmi2Component s2 = s2_fmi2Instantiate("Tool2", "", "Model2", "", ...);
tStart = 0;  tStop = 10;  h = 0.01;

//Initialize models
s1_fmi2SetupExperiment(s1, …, startTime, …, stopTime);  …
s1_fmi2EnterInitializationMode(s1); … s2_fmi2ExitInitializationMode(s2);

//Simulation sub-phase
tc = tStart;
while((tc < tStop) && (status == fmi2OK))
    s1_fmi2GetReal(s1, ..., 1, &y1); //retrieve outputs
    s2_fmi2GetReal(s2, ..., 1, &y2);

    s1_fmi2SetReal(s1, ..., 1, &y2); //set inputs
    s2_fmi2SetReal(s2, ..., 1, &y1);

    status1 = s1_fmiDoStep(s1, tc, h, fmi2True);  //call slaves
    status2 = s2_fmiDoStep(s2, tc, h, fmi2True);

    tc+=communicationStepSize;  //increment master time
}

//Shutdown sub-phase
if (status == fmi2OK)
{  s1_fmi2Terminate(s1);  s1_fmi2FreeInstance(s1); …}
```



Exporting control sequences from Dymola



Exporting control model from JModelica



Export controller as an FMU

```
from pymodelica import compile_fmu

fmu_name = compile_fmu("Annex60.Controls.Continuous.LimPID")
```

Load and simulate an FMU

```
from pyfmi import load_fmu

m = load_fmu(fmu_name)
res = m.simulate()
```

Exercise

Creating an FMU from Dymola

a) Start Dymola and switch to the modeling tab

b) Implement following first-order model

$$\frac{dx}{dt} = u - x,\ x(0) = 1.0$$

$$y = -x$$

The Modelica code for the model is

```
model MyFirstFMU
  "This model simulates the exponential decay curve."
  Real x(start = 1.0) "state variable";
  Modelica.Blocks.Interfaces.RealInput u "input variable";
  Modelica.Blocks.Interfaces.RealOutput y "output variable";
equation
    der(x) = u-x;
    y = -x;
end MyFirstFMU;
```

c) Export the model as an FMU for Model Exchange 2.0

d) Unzip the FMU and look at the model description file

e) Import it and simulate it in Dymola, Ptolemy II, OpenModelica, JModelica or an other tool

Questions