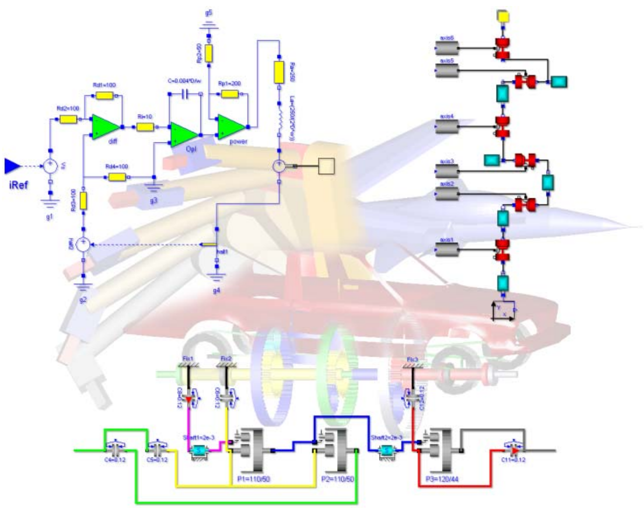
Research in Model-Based Product Development at PELAB in the MODPROD Center

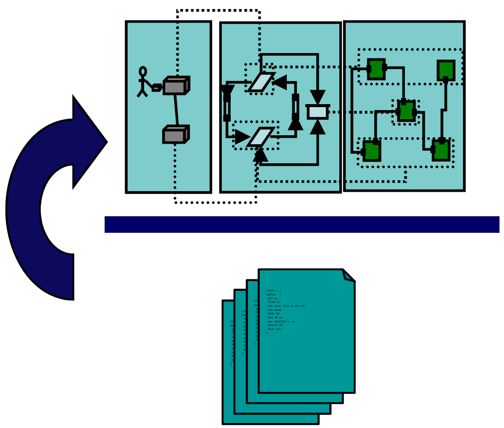
Presentation at MODPROD'2018

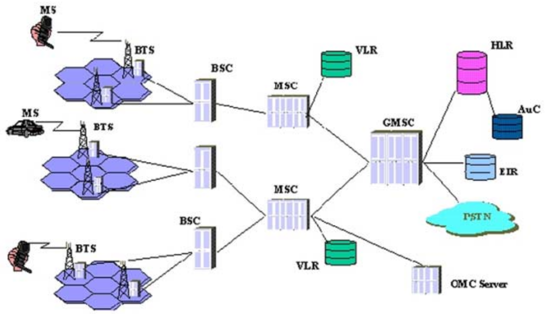
Department of Computer and Information Science
Linköping University

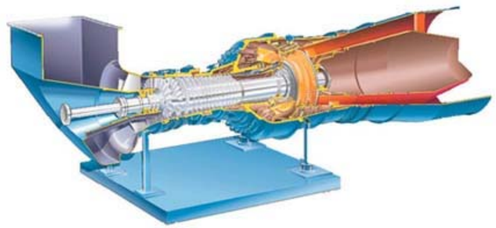
2018-02-06

Peter Fritzson, Lennart Ochel, Adrian Pop,
Lena Buffoni, Bernhard Thiele, Martin Sjölund



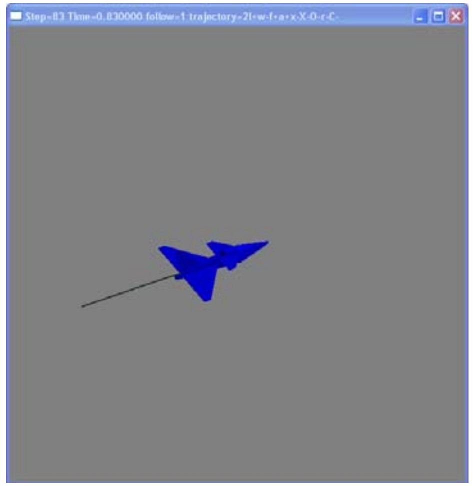


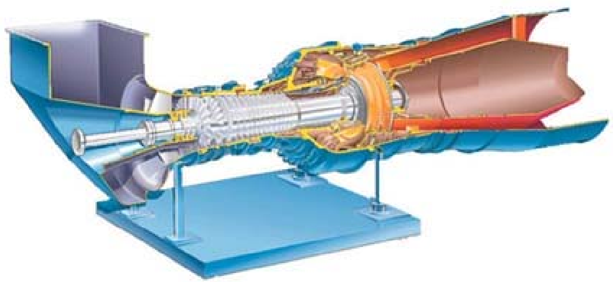


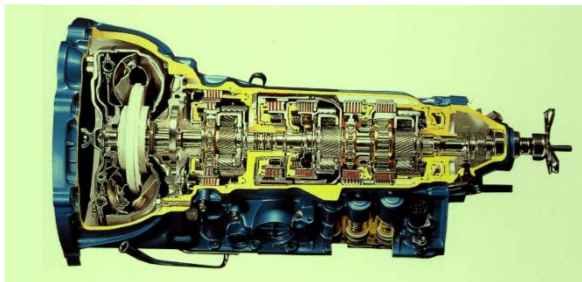


Examples of Complex Systems in Engineering

- Robotics
- Automotive
- Aircraft
- Mobile Phone Systems
- Business Software
- Power plants
- Heavy Vehicles
- Process industry







Research

Modeling-Language Design

Model-Based Co-simulation with FMI and TLM

Model Debugging

Model-Based Fault Analysis

Multi-Core based Simulation

Embedded System Real-Time Modeling

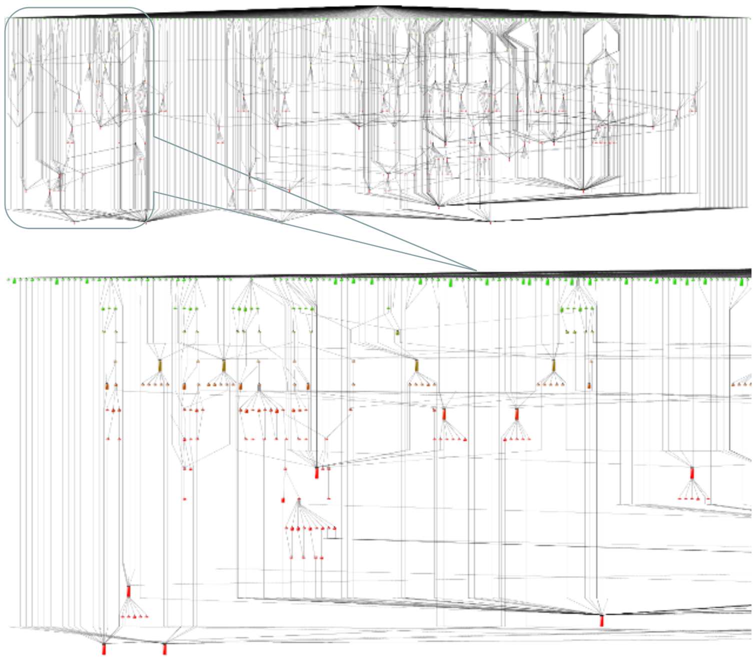
Modeling Support Environments

Parallel Execution
Compilation to MultiCore

Mahder Gebremedhin, Peter Fritzson

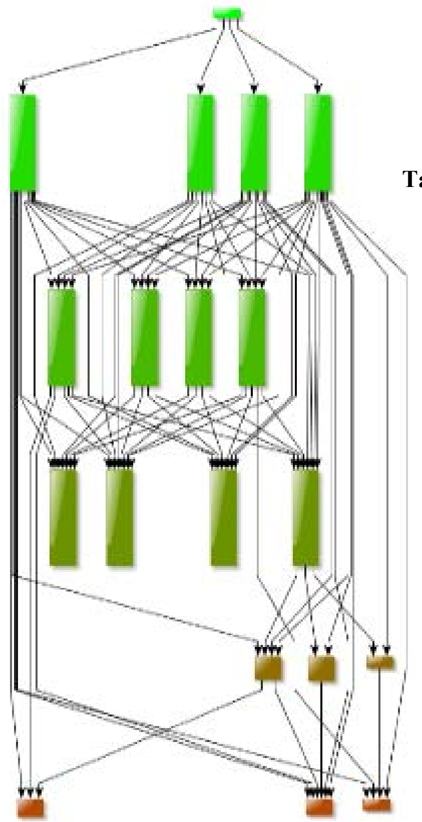
Compiling Models to Efficient Parallel Code
(scheduling on multiple cores)

Modelica.Electrical.Spice3.Examples.Spice3BenchmarkFourBitBinaryAdde



Original task system of Four Bit Binary Adder model

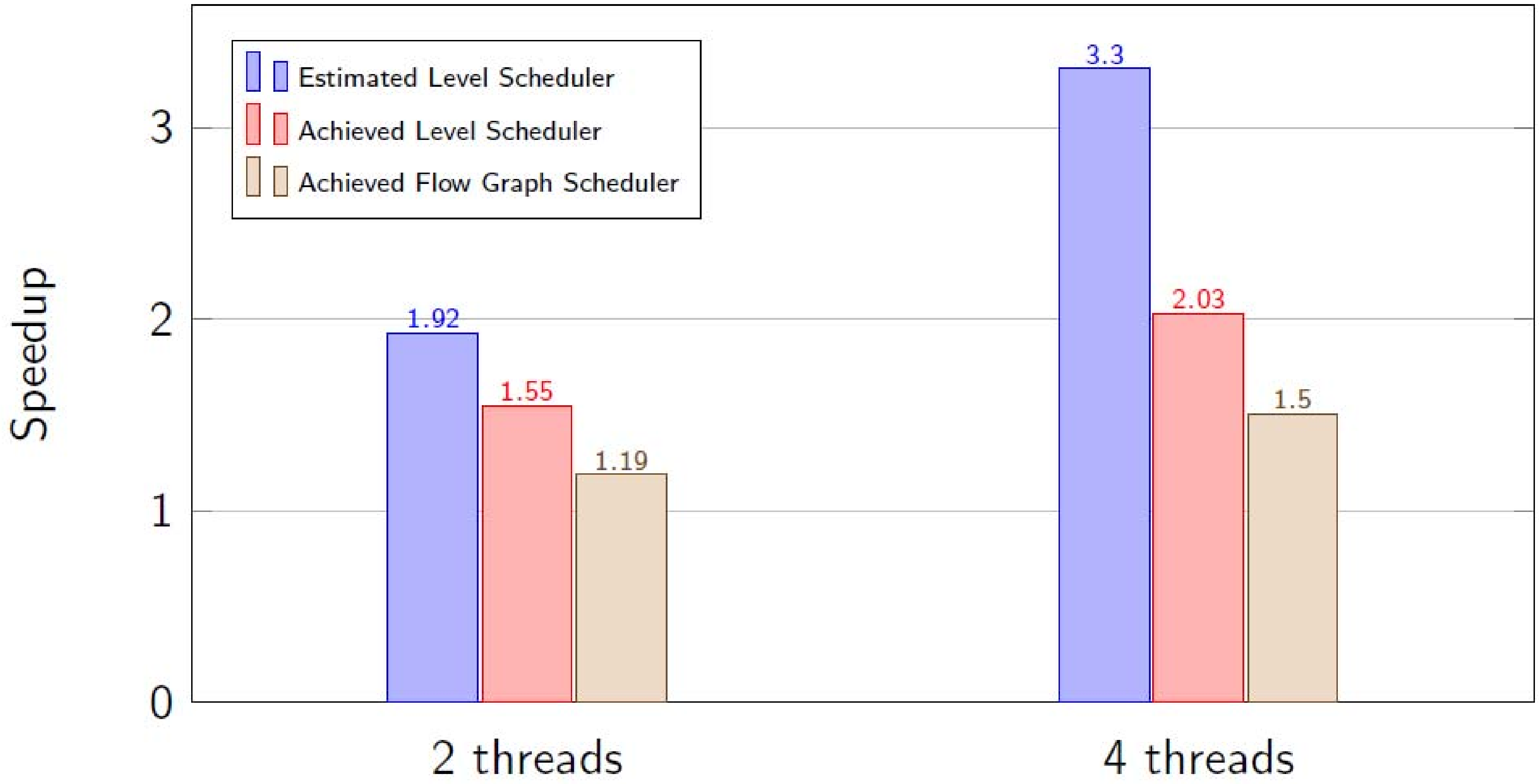
1122 Tasks
1360 Edges



Task system after clustering for level scheduler

18 Tasks
72 Edges

Speedup using Dynamic Scheduling on 4-core laptop Modelica Model CaurLowPassSC



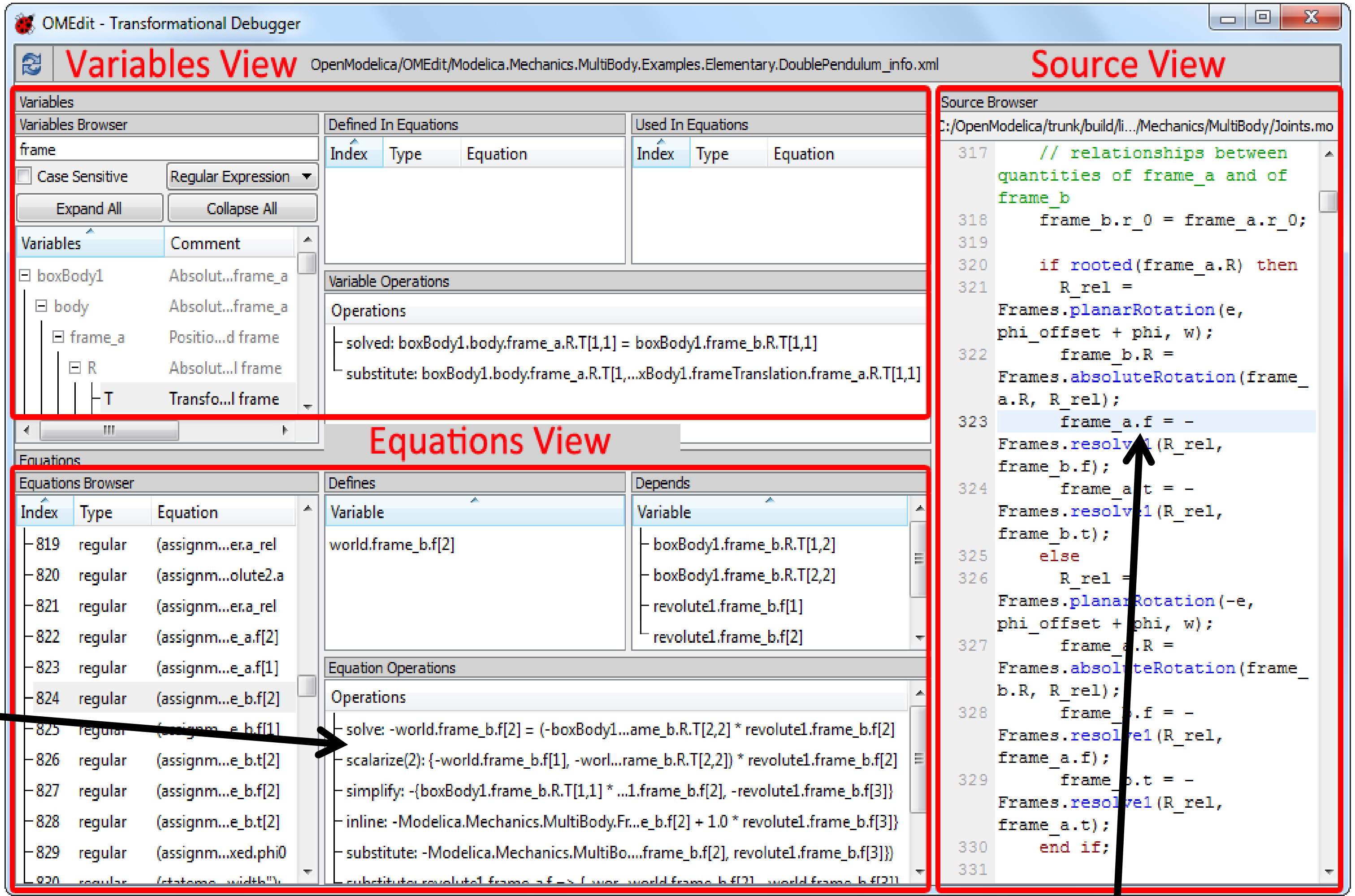
Model Debugging and Performance Analysis

Martin Sjölund,
Adeel Asghar, Adrian Pop
Dept Computer and Information Science
Linköping University

Integrated Static-Dynamic OpenModelica Equation Model Debugger

Efficient handling of **Large** Equation Systems

Showing equation transformations of a model:



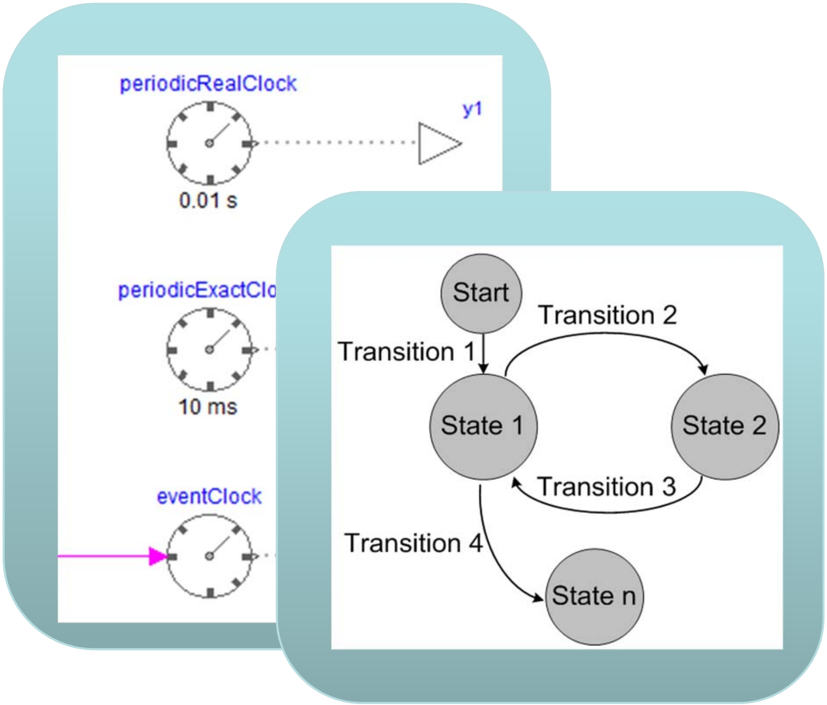
Mapping dynamic run-time error to source model position

MODELICA pelab

Research on Debugging in OPENCPS Project

Debugging of new features

- clocked synchronous models
- real-time debugging and event tracing
- graphic support for state machine debugging

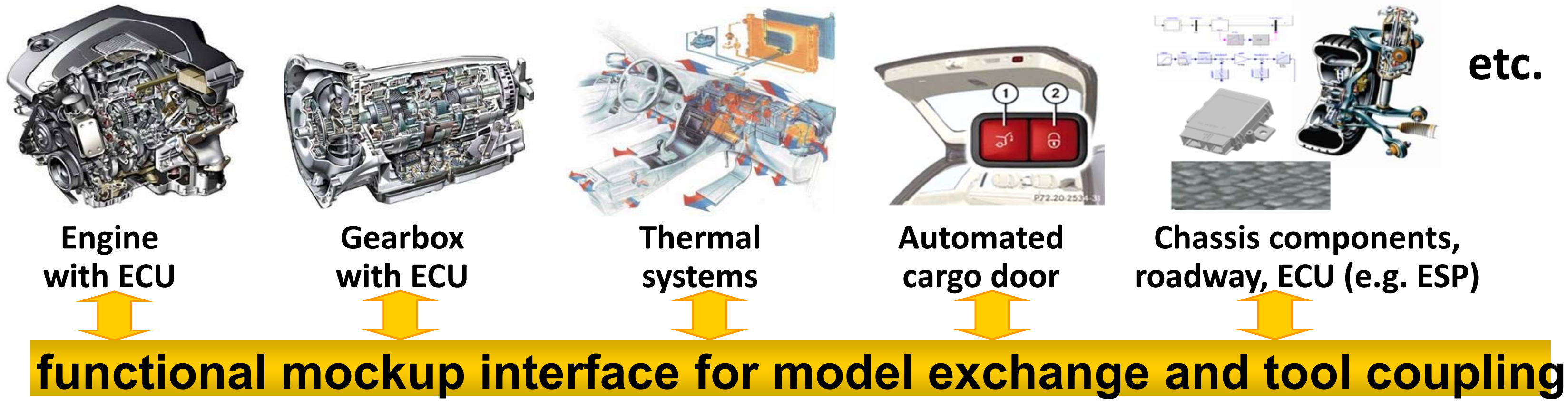


Critical for efficient usability by industrial partners!

Co-simulation, FMI, Modeling Traceability, etc.

Adrian Pop, Alachew Mengist, Lennart Ochel, Robert Braun, Adeel Asghar, Arunkumar Palanisamy, Peter Fritzson

General Tool Interoperability & Model Exchange Functional Mock-up Interface (FMI)



courtesy Daimler

- FMI development was started by ITEA2 MODELISAR project. FMI is a Modelica Association Project now
- **Version 1.0**
- FMI for Model Exchange (released Jan 26,2010)
- FMI for Co-Simulation (released Oct 12,2010)
- **Version 2.0**
- FMI for Model Exchange and Co-Simulation (released July 25,2014)
- **> 100 tools** supporting it (https://www.fmi-standard.org/tools)

Enhanced FMI Co-simulation, Run-time, and Master Simulation Tool – Work in OPENCPS Project

- Further **extensions** to the FMI standard to support TLM-based co-simulation including support for SKF mechanical bearing models
- **Enhanced run-time** for efficient co-simulation of FMUs, including FMUs from OpenModelica and Papyrus
- General **Master** simulation tool support for FMI

MODELICA pelab

OMSimulator

Lennart Ochel

12th MODPROD Workshop, February 6-7, 2018

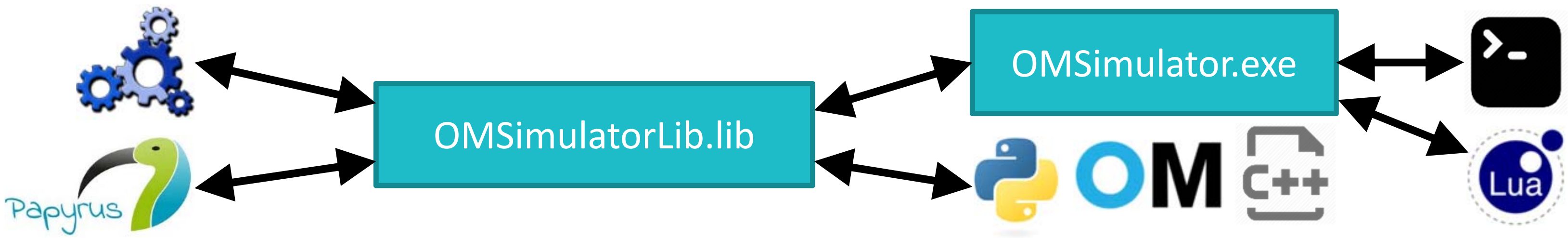
LINKÖPING
UNIVERSITY

OMSimulator

- Co-simulation environment primarily based on FMUs
- Combining TLM and FMI approaches for co-simulation

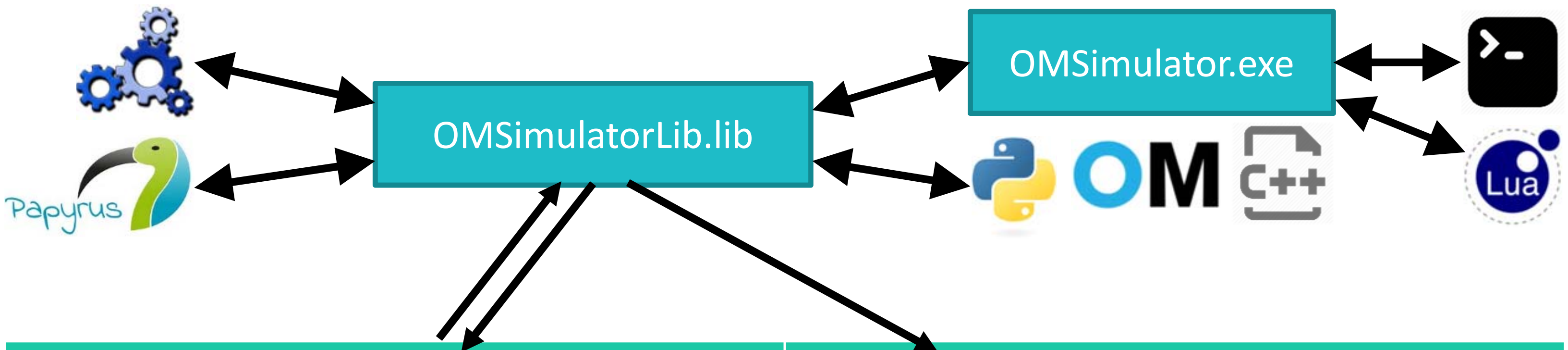
TLM	FMI
• Physical connections	• Signal connections
• Delayed connections	• Non-delayed connections
• Distributed processes	• Single process

OMSimulator



OMTLMSimulator	OMFMISimulator
• Physical connections • Delayed connections • Distributed processes	• Signal connections • Non-delayed connections • Single process

OMSimulator



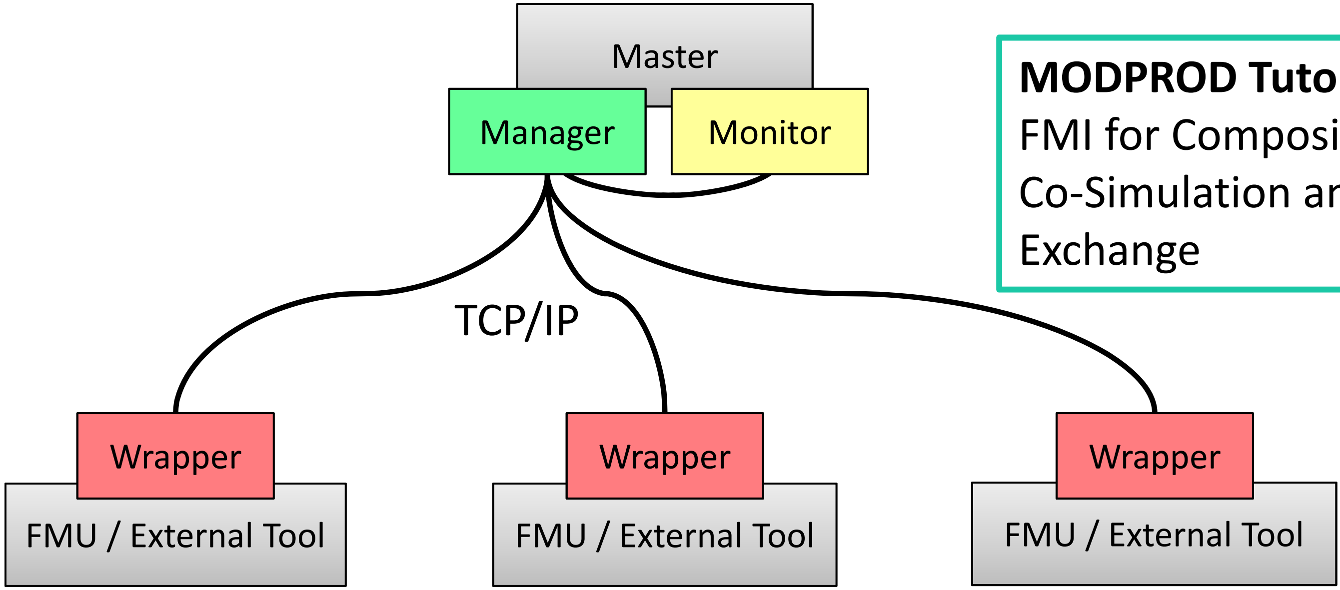
OMTLMSimulator	OMFMISimulator
• Physical connections • Delayed connections • Distributed processes	• Signal connections • Non-delayed connections • Single process

OMTLMSimulator

- Dedicated talk at MODPROD
- Bottom-up approach

MODPROD Parallel Session 2a:
TLM-based co-simulation using FMI and direct tool connections

MODPROD Tutorial 2:
FMI for Composite Modelling, Co-Simulation and Model Exchange



OMTLMSimulator

- Dedicated talk at MODPROD
- Bottom-up approach
- Only TLM connections
- Distributed simulation
- External tool integration (e.g. Simulink, Adams, BEAST)

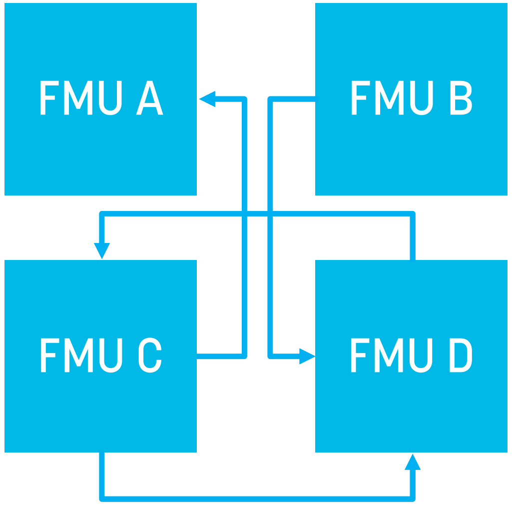
MODPROD Parallel Session 2a:
TLM-based co-simulation using FMI and direct tool connections

MODPROD Tutorial 2:
FMI for Composite Modelling, Co-Simulation and Model Exchange

OMFMISimulator

- Simulator for connected FMUs
- Only signal connections
- Top-down approach
- Single process
- Scripting interface

MODPROD Tutorial 2:
FMI for Composite Modelling, Co-Simulation and Model Exchange

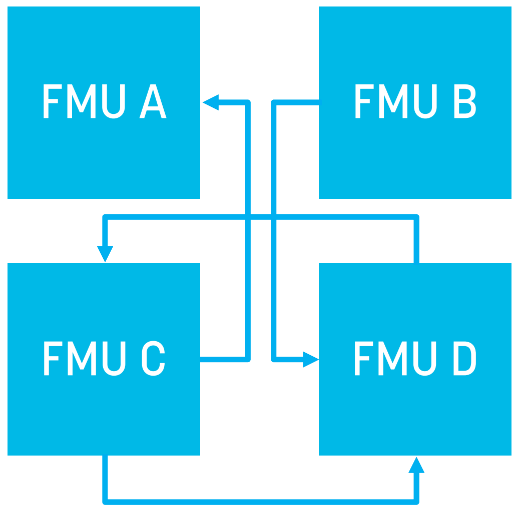


OMFMISimulator

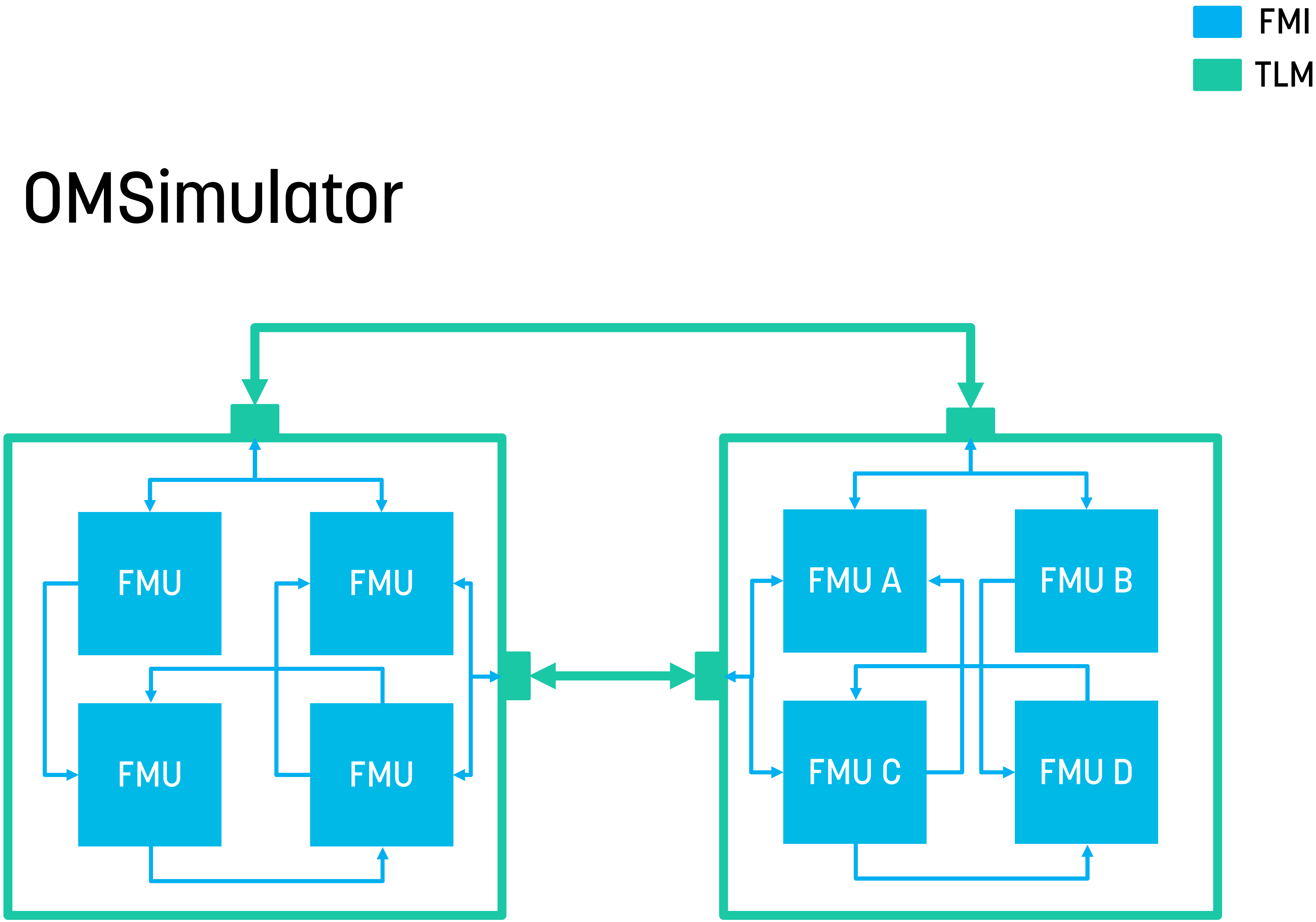
Python interface:

```
m = oms.newModel()
# instantiate FMUs
oms.instantiateFMU(m, "A.fmu", "A")
oms.instantiateFMU(m, "B.fmu", "B")
oms.instantiateFMU(m, "C.fmu", "C")
oms.instantiateFMU(m, "D.fmu", "D")
# add connections
oms.addConnection(m, "A.u",  "C.y1")
oms.addConnection(m, "B.y",  "D.u1")
oms.addConnection(m, "C.y2", "D.u2")
oms.addConnection(m, "C.u",  "D.y")
```

MODPROD Tutorial 2:
FMI for Composite Modelling, Co-Simulation and Model Exchange



OMSimulator



adeas31 | adrpo | alash325 | arun3688 | bernhard-thiele
hyumo | lenaRB | lochel | mahge | robbr48 | sjoelund

www.liu.se

LINKÖPING UNIVERSITY

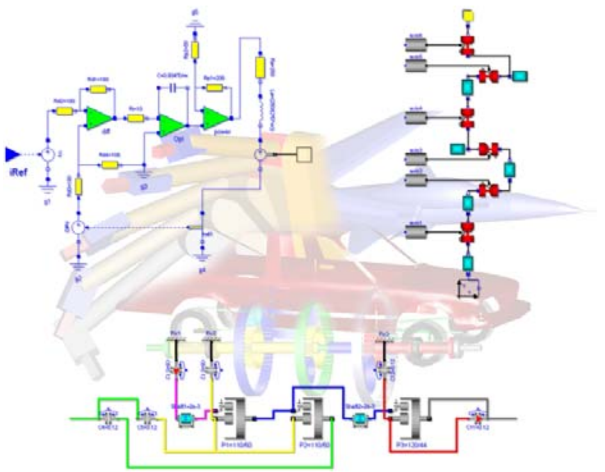
Ongoing Projects and Research Interests

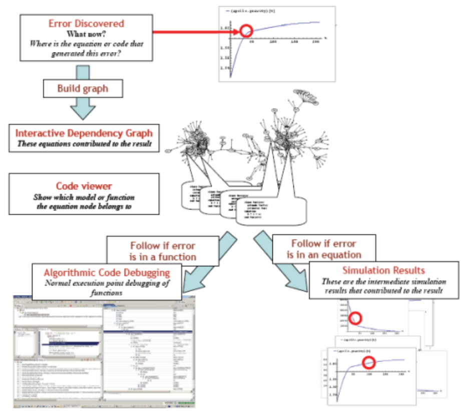
Adrian.Pop@liu.se

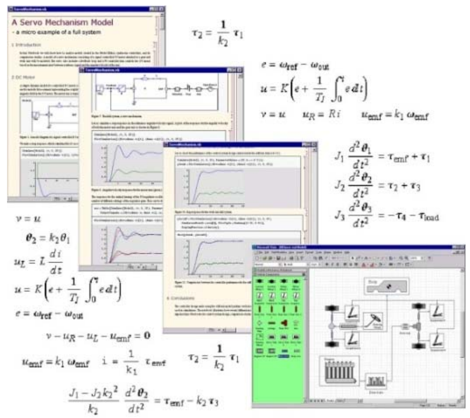
2018-02-06

Open Source Modelica Consortium
Programming Environment Laboratory
Department of Computer and Information Science
Linköping University

LINKÖPING UNIVERSITY







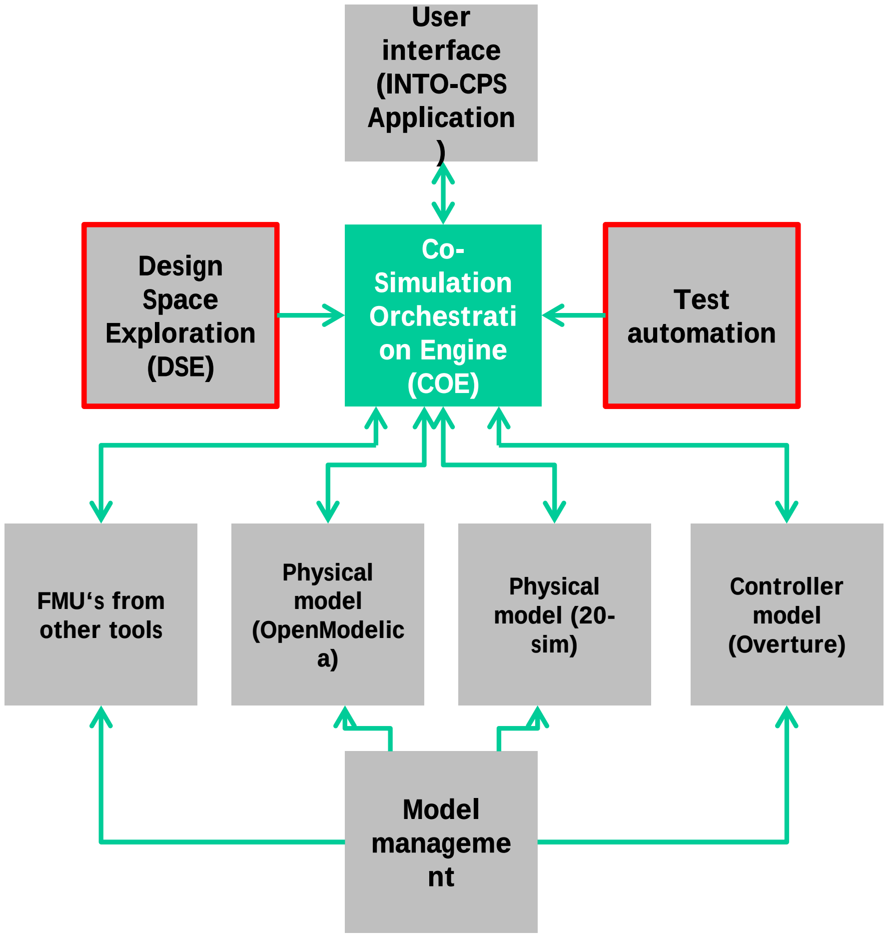
OPEN SOURCE MODELICA CONSORTIUM   OpenModelica   MODELICA   pelab   LINKÖPINGS UNIVERSITET

Research Interests

- Languages
  - Design, Implementation, Debugging, Profiling, Integrated Environments

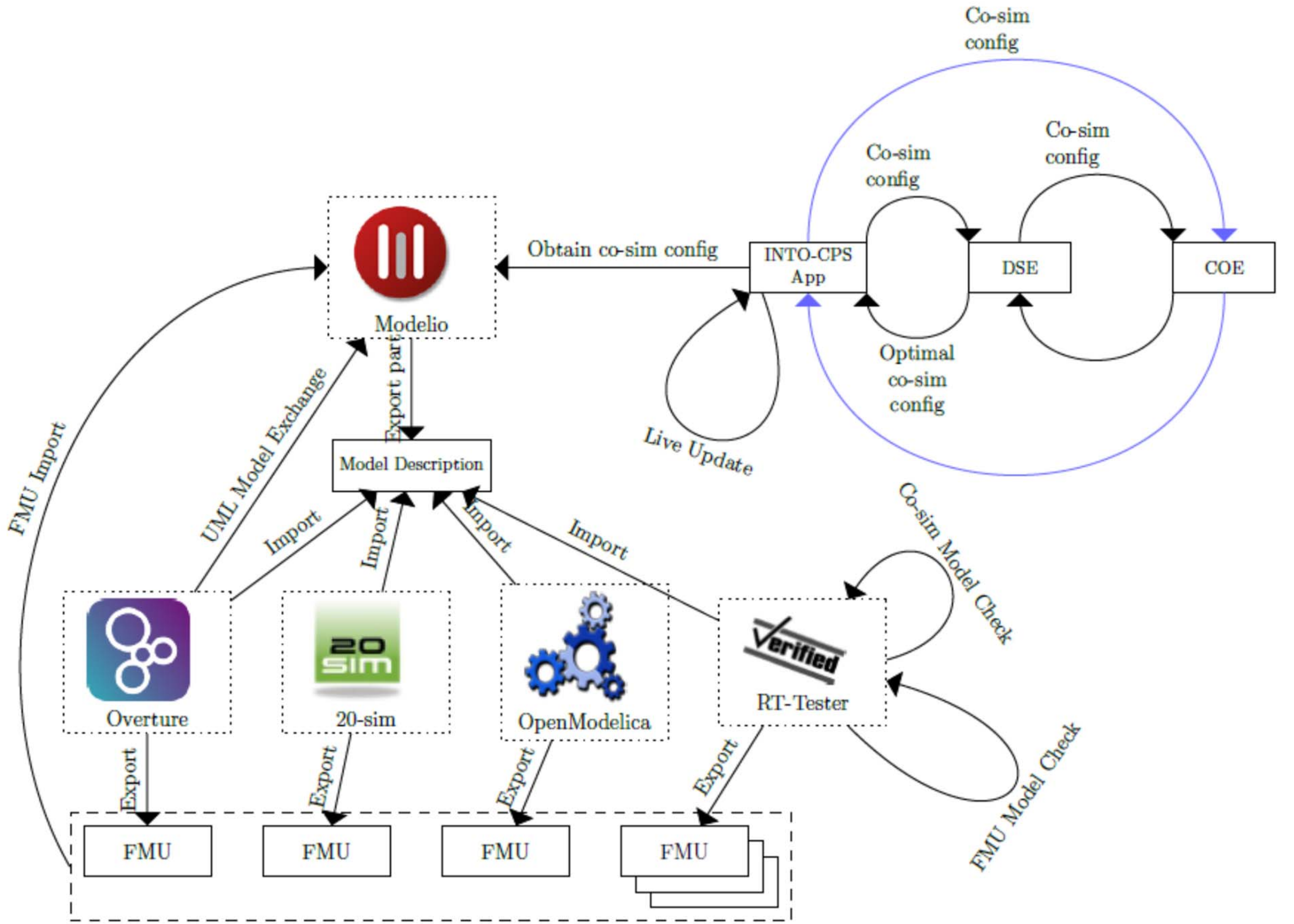
- Modeling and Simulation
  - Equation-based object-oriented languages
  - Standards extensions for Modelica and FMI
  - Requirements Engineering and Traceability
  - Efficient simulation (different approaches to system solving, parallelization, separate code generation, separate compilation)
  - Multi-mode – variable system of equations
  - Code generation for both large system models and embedded systems

INTO-CPS: Co-Simulation Framework Vision



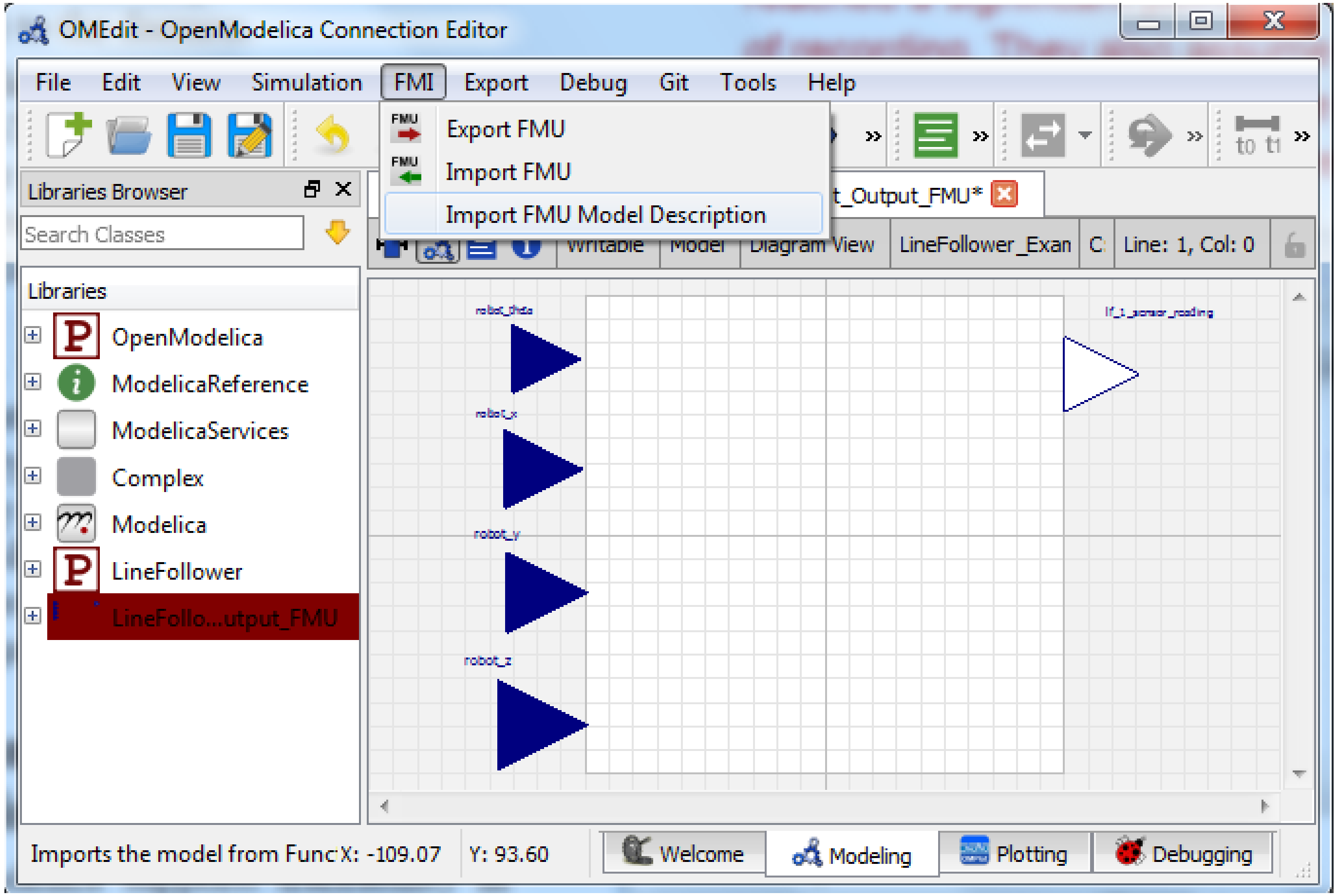
Horizon 2020 Programme

INTO-CPS - toolchain

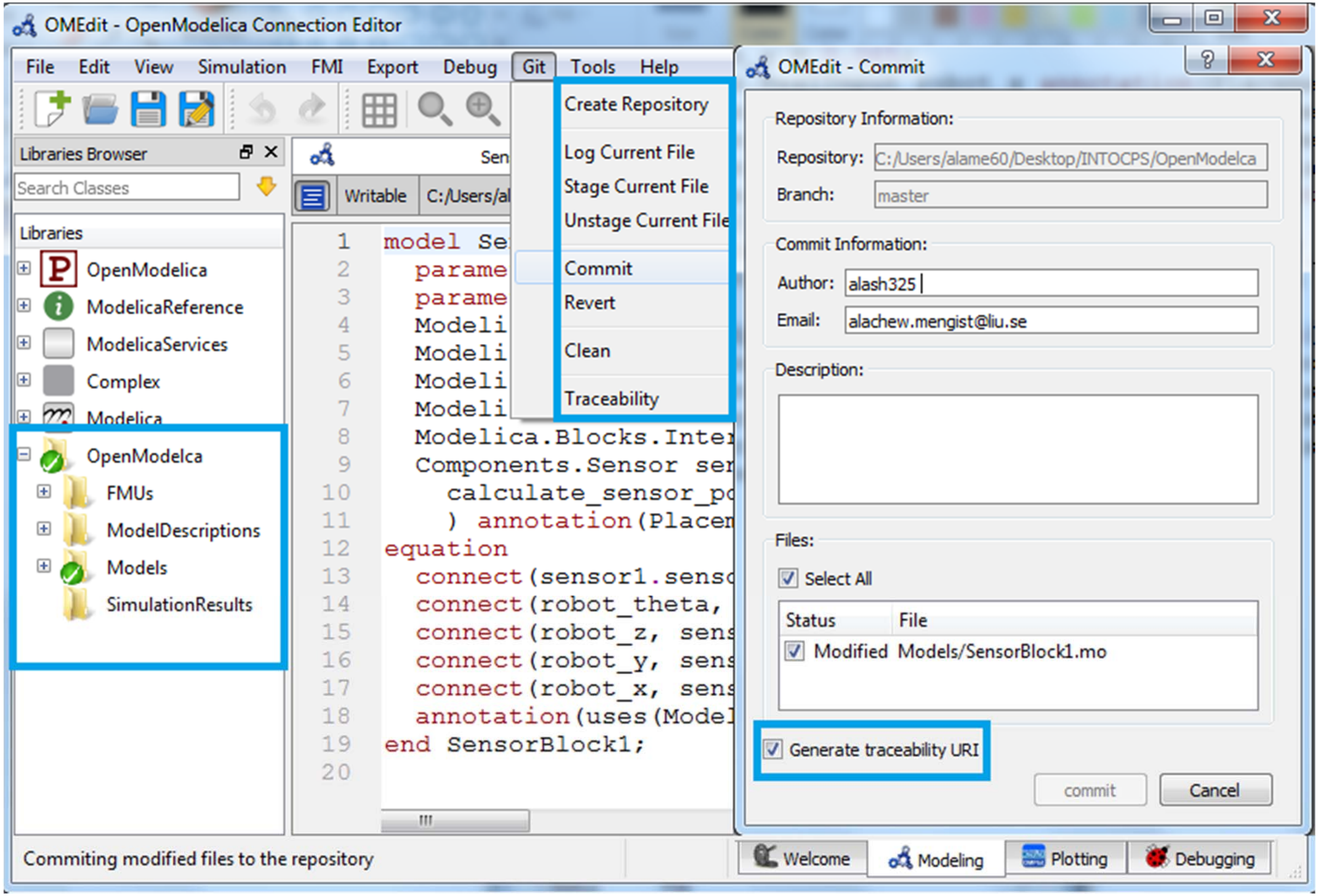


Import Model Description XML File

- Import model description XML interface files (linked with requirements)
- Create Modelica model stub containing the inputs and outputs specified in *modelDescription.xml*

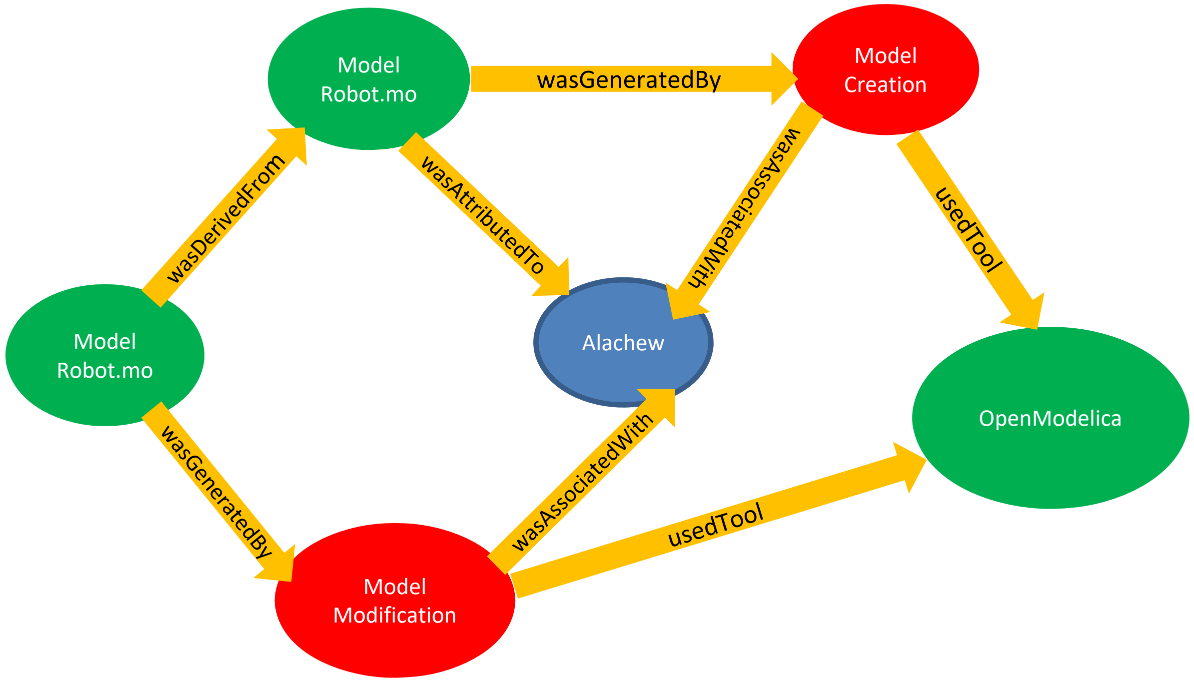


Model Management with Git Integration



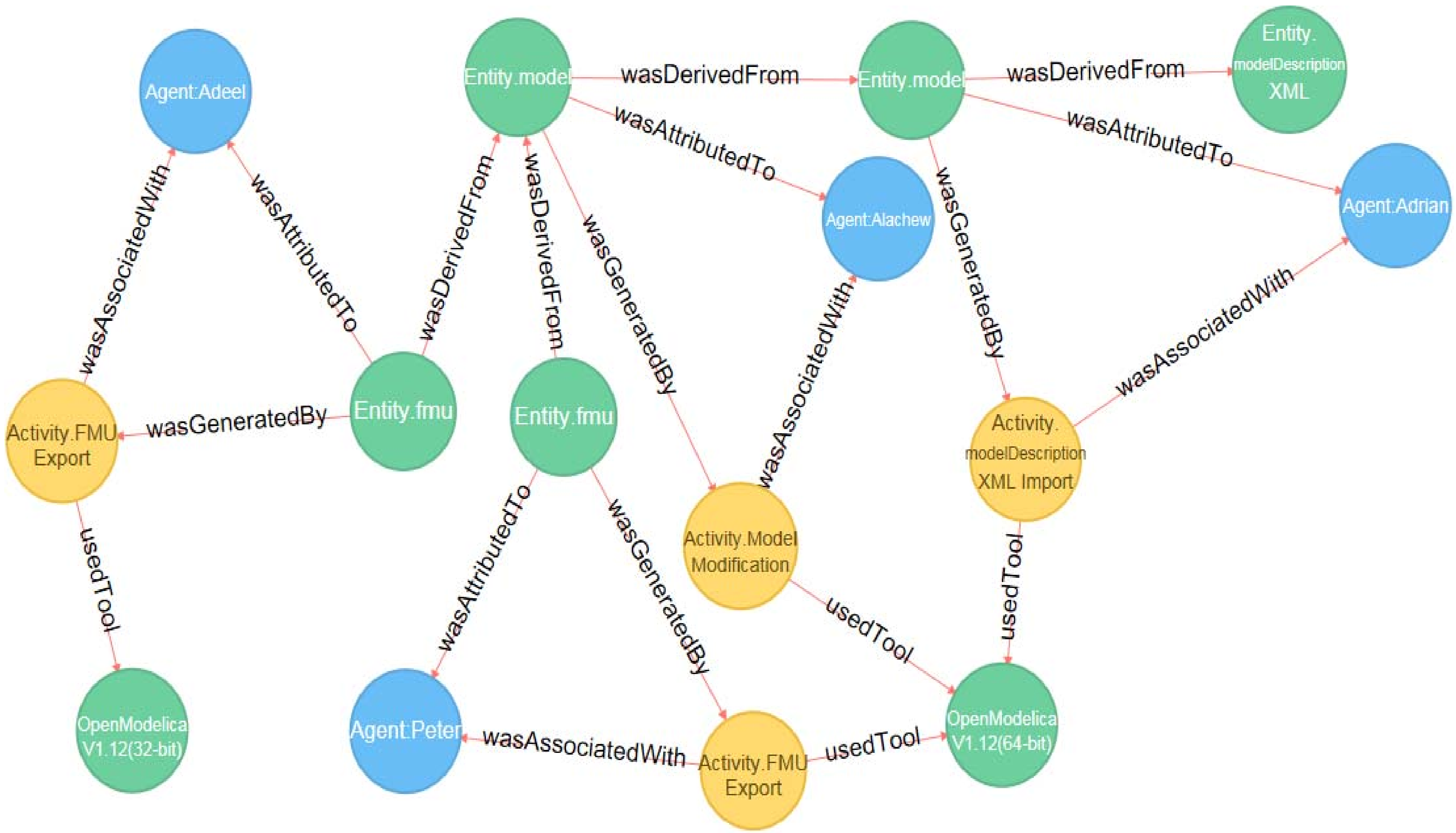
Traceability Support in OpenModelica

- Entities (e.g. Modelica files, FMUs) are shown in green
- Activities (e.g. Model creation, Model modification, FMU export) are shown in red
- Agents (e.g. a user with the name "Alachew") are shown in blue
- Their relationships (e.g. wasGeneratedBy, wasDerivedFrom, usedTool, …) are shown in orange.



An example of traceability information sent from OpenModelica to the daemon and visualized in the Neo4j database

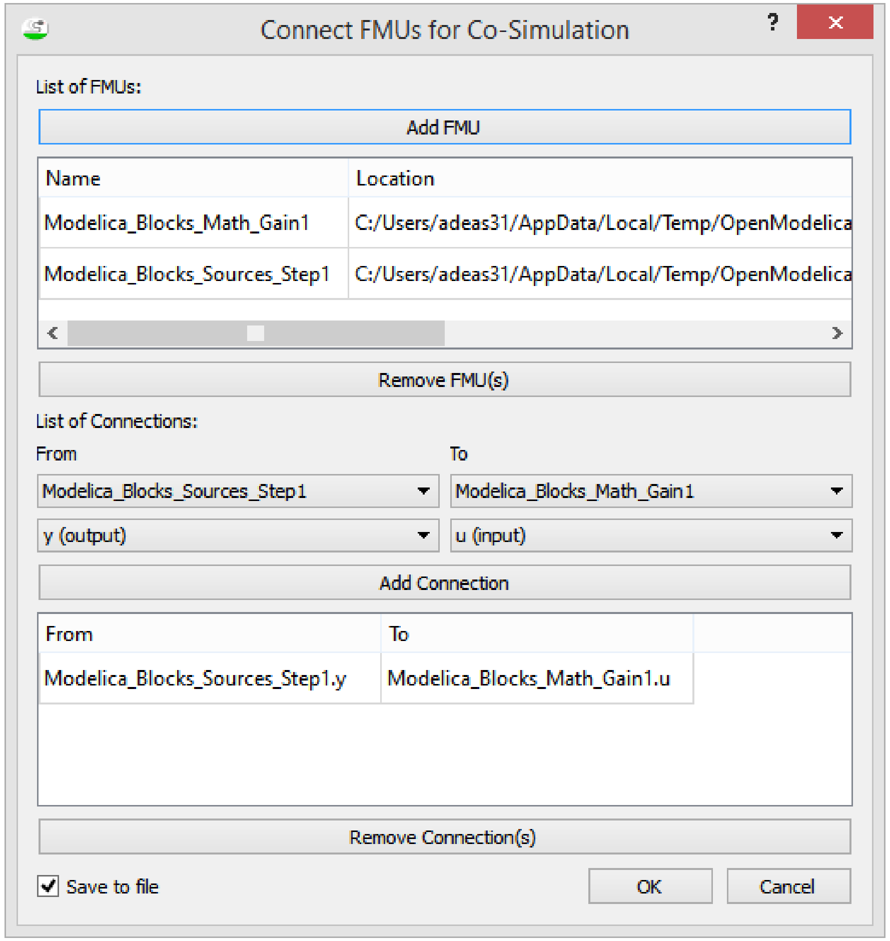
Traceability Information collected by OpenModelica



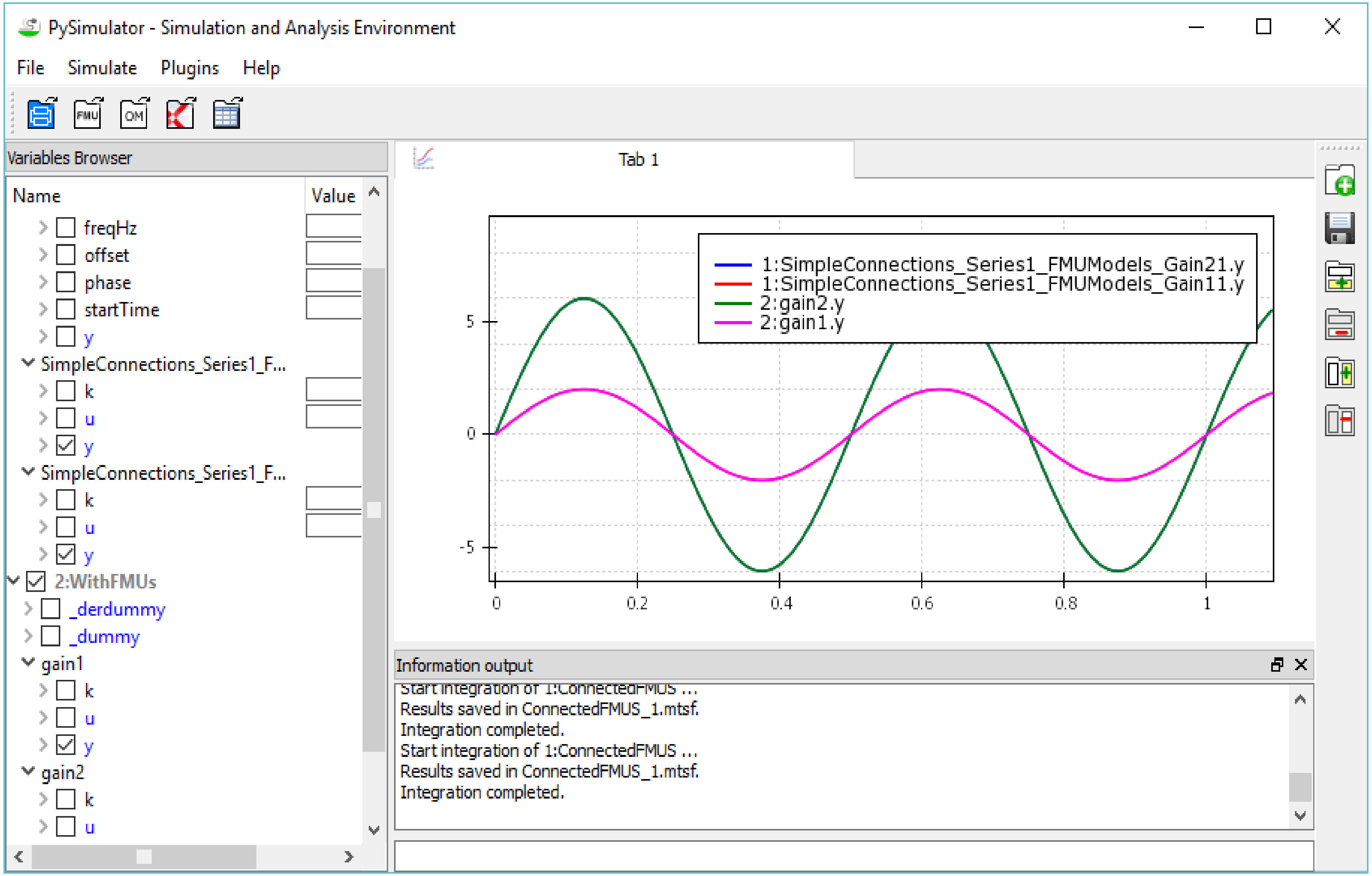
Using Open Services for Lifecycle Collaboration (OSLC)

PyModSimA: Simulation of Connected FMUs

- Create a model containing several FMUs connected to each other.
- XML format is used to define connections between FMUs.
- Connect FMUs using the graphical user interface.



PyModSimA: Co-Simulation of FMUs in Pysimulator



PyModSimA: Parallel regression testing

Regression Report

Given Error Tolerance:	1.0e-3
Disk space of all used result files:	34.4 MB
Total number of compared files:	12 against 4 baseline files
Total number of compared variables:	17,376 (17,095 passed, 281 failed)
Disk space of full report directory:	12.3 MB
Generated:	10:57AM on April 14, 2015 by PySimulator
Time Taken:	00h:00m:58s

No	Description
1	Number of total passed files / Number of total failed files
2	Percentage of total passed files
3	{Number of passed files / Number of failed files} for this name, e.g. ...Rectifier.
4	{Number of passed files / Number of failed files} in the corresponding directory, e.g. FMU1.0
5	Number of variables contained in the file
6	Number of variables that are compared
7	Number of variables greater than given tolerance with link to the list of these variables
8	Maximum error of all compared variables

Legend

Result file name	Status	Dymola	FMU1.0	FMU2.0	OpenModelica
4 result file names	7 / 5		4 passed / 0 failed	1 passed / 3 failed	2 passed / 2 failed
	67%		100% passed	25% passed	50% passed
		Baseline			
Modelica.Electrical.Analog.Examples.Rectifier	1 / 2	206	216 / 191 [7.9e-6]	233 / 180 / 14 [2.0e-2]	207 / 191 / 102 [3.5e-1]
Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.fullRobot	2 / 1	6,345	5,700 / 5,660 [2.1e-4]	5,730 / 5,588 / 148 [1.1e-2]	4,763 / 4,742 [3.6e-4]
Modelica.Mechanics.Rotational.Examples.CoupledClutches	3 / 0	170	194 / 166 [2.4e-4]	173 / 166 [2.7e-5]	153 / 145 [4.6 e-5]
Modelica.Mechanics.Rotational.Examples.SimpleGearShift	1 / 2	130	139 / 119 [8.4e-5]	156 / 123 / 16 [1.1e-2]	134 / 105 / 1 [1.5e-3]

5 6 7 8

Legend Coloring:

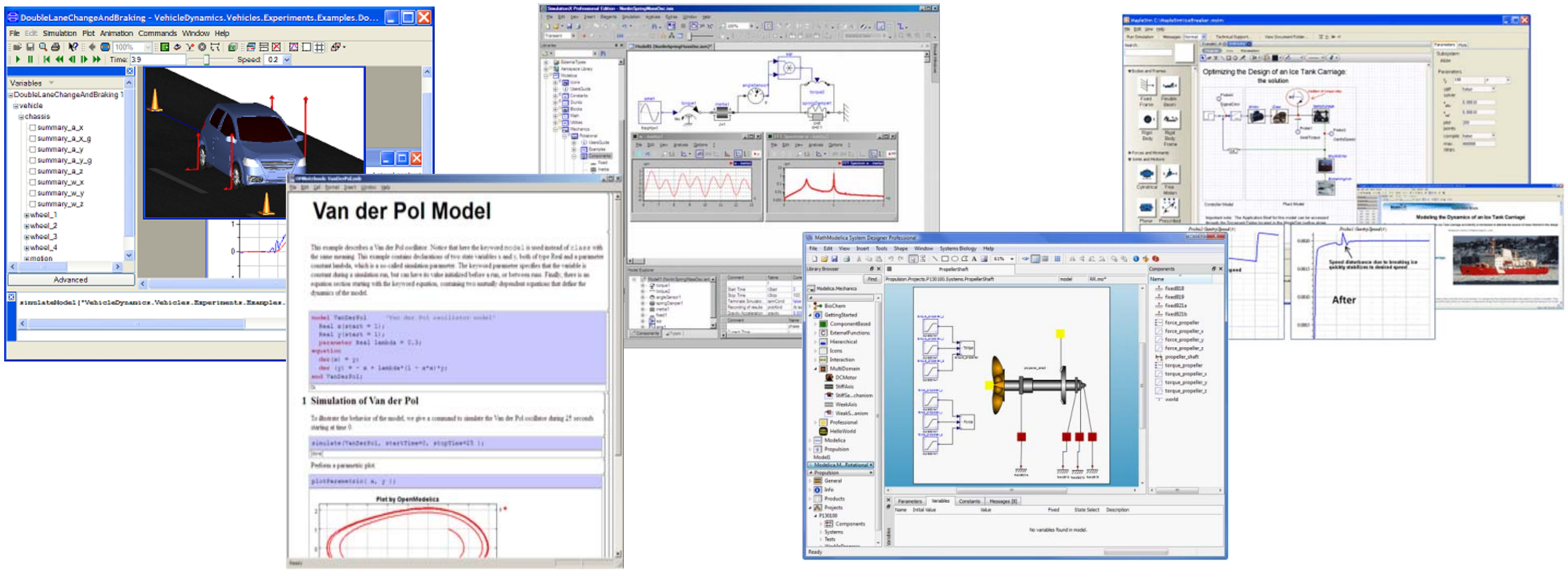
Red:
* Per file: Comparison failed, i.e. at least one variable with large error
* Per column or row: Only 0-50% of the corresponding files passed the test

Orange:
* Per column or row: >50% and < 100% of the corresponding files passed the test
* Total: >50% and < 100% of all files passed the test

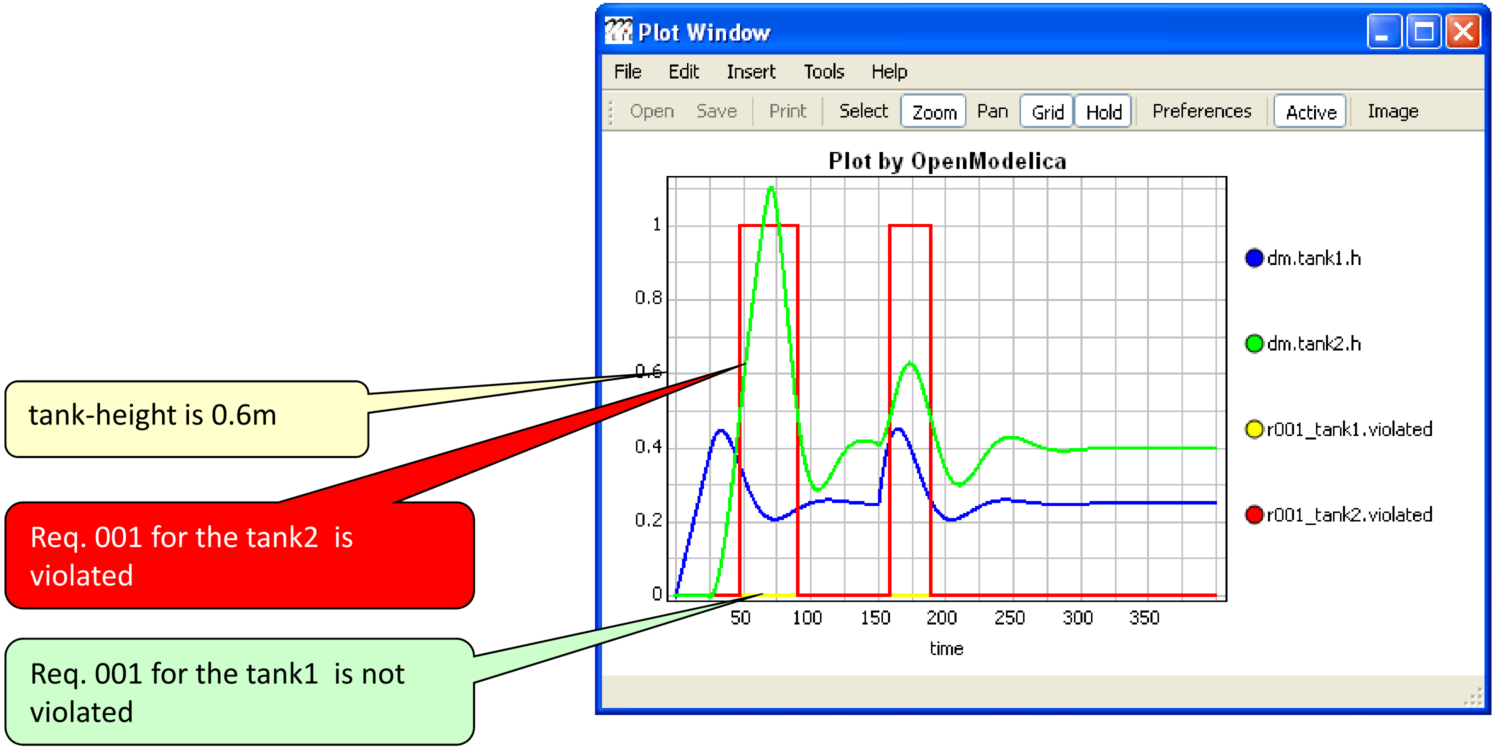
Green:
* Per file: Comparison passed, i.e. all compared variables passed the test
* Per column or row: 100% of the corresponding files passed the test

Dynamic Verification/Testing of Requirements vs Usage Scenario Models

Lena Buffoni et al

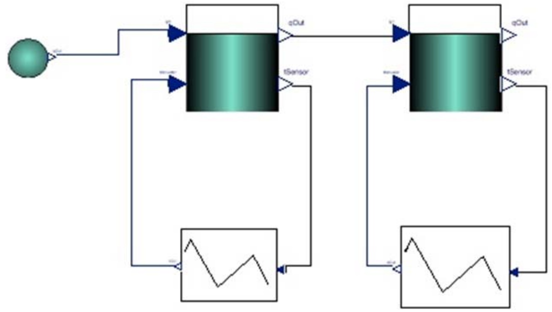


Dynamic Requirement Evaluation



Testing a single verification model in Modelica

- Req. 001: The volume of each tank shall be at least 2 m3.
- Req. 002: The level of liquid in a tank shall never exceed 80% of the tank height.
- Req. 003: After each change of the tank input flow, the controller shall, within 20 seconds, ensure that the level of liquid in each tank is equal to the reference level with a tolerance of ± 0.05 m.
- ...



Start with constant flow and increase at t=150

Design alternative: two tank model

Design alternative: two tank model

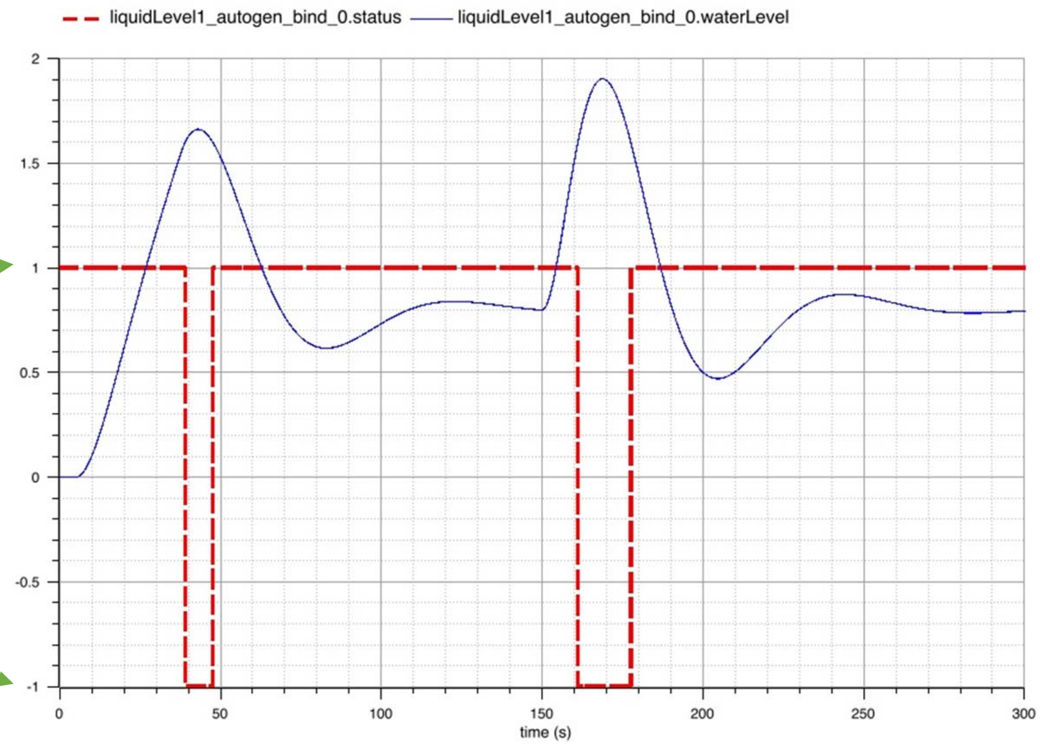
One possible test scenario

Analyzing a single requirement status

Req. 002: The level of liquid in a tank shall never exceed 80% of the tank height.

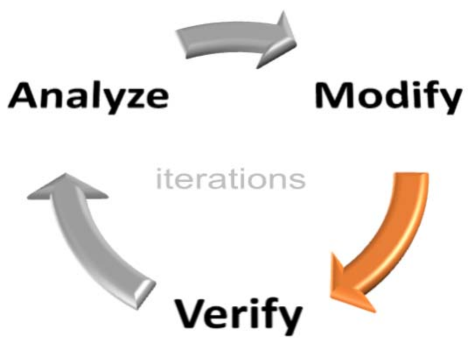
Requirement not violated

Requirement violated



vVDR Method –
virtual Verification of Designs vs Requirements

Actor	Task	Created Artifact
Formalize Requirements		RMM Requirement Monitor Models
Formalize Designs		DAM Designs Alternative Models
Formalize Scenarios		SM Scenario Models
Create Verification Models	AUTOMATED	VM Verification Models
Execute and Create Report	AUTOMATED	Reports
Results		

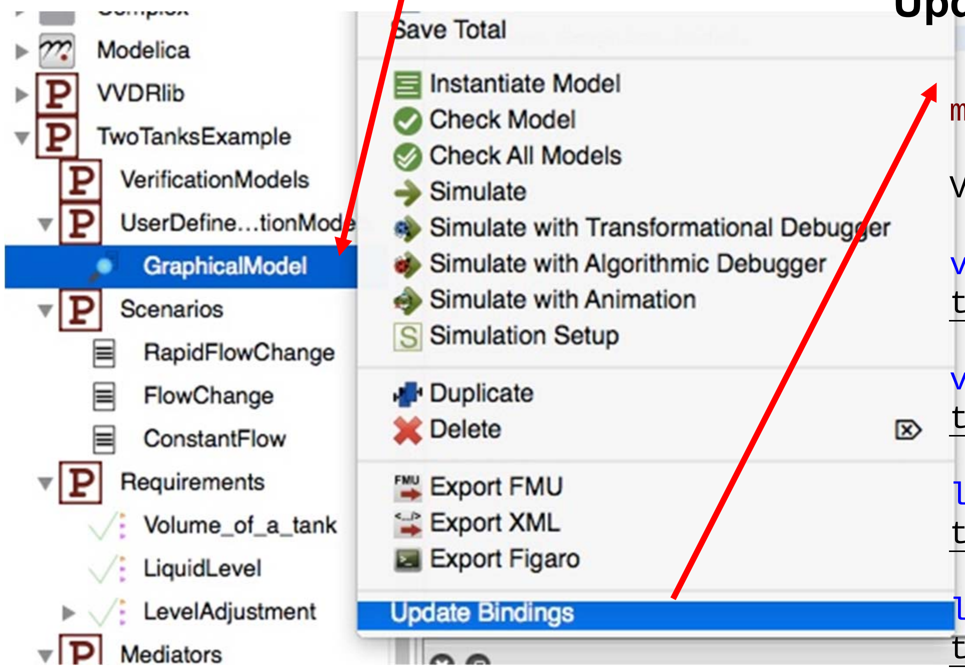


Goal: Enable on-demand verification of designs against requirements using automated model composition at any time during development.

Automatic model composition

Initial AnalysisModel:

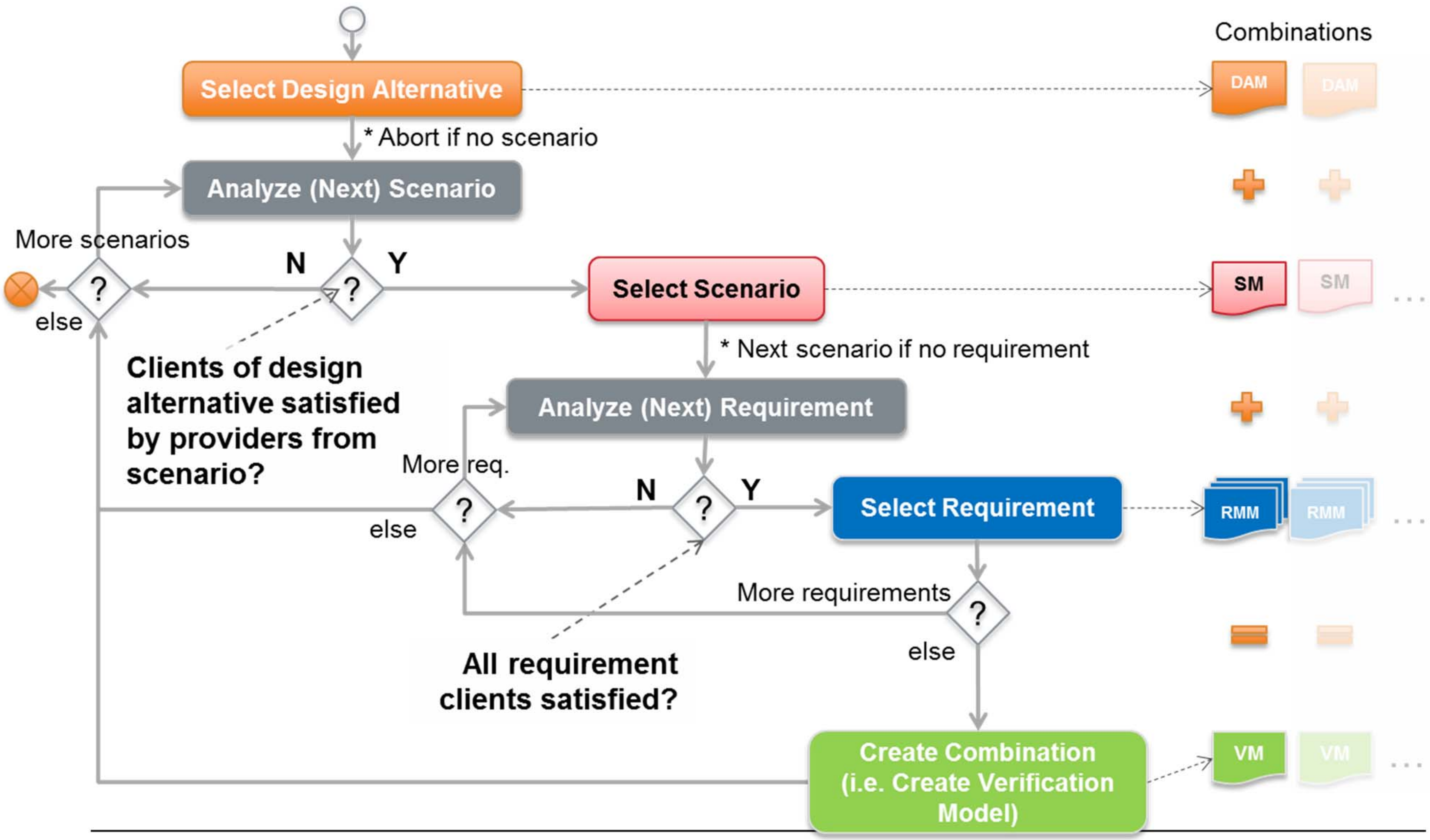
```
model GraphicalModel
  extends VVDRlib.Verification.VerificationModel;
  Requirements.Volume_of_a_tank volume_of_a_tank1#
  Requirements.LiquidLevel liquidLevel1#
  Design.TwoTanksDesign#
end GraphicalModel;
```



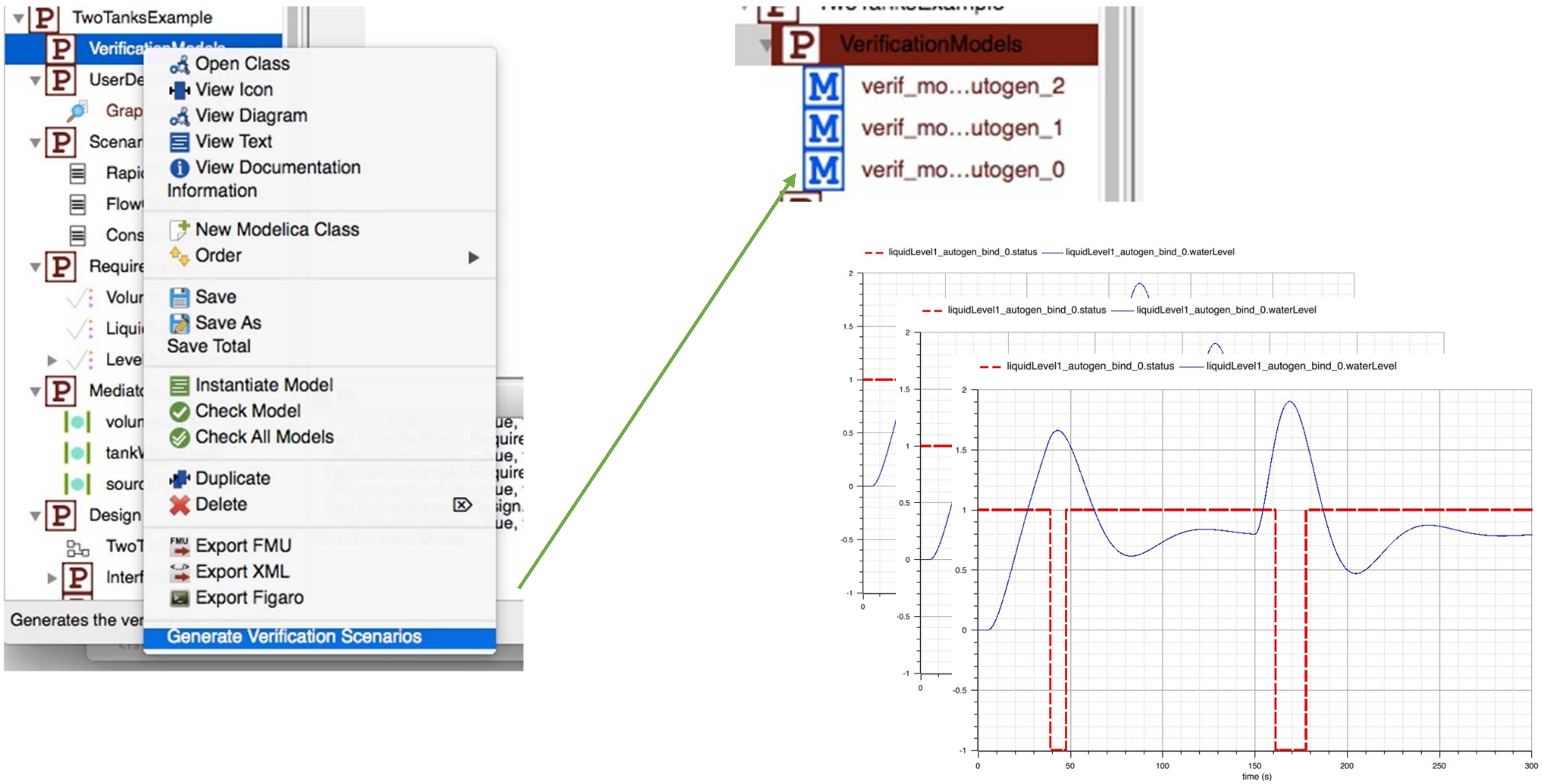
Updated AnalysisModel model with a binding:

```
model GraphicalModel
  extends
VVDRlib.Verification.VerificationModel;
  TwoTanksExample.Requirements.Volume_of_a_tank
volume_of_a_tank1_autogen_bind_0(tankVolume =
twoTanksDesign1.tank1.volume)#
  TwoTanksExample.Requirements.Volume_of_a_tank
volume_of_a_tank1_autogen_bind_1(tankVolume =
twoTanksDesign1.tank2.volume)#
  TwoTanksExample.Requirements.LiquidLevel
liquidLevel1_autogen_bind_0(waterLevel =
twoTanksDesign1.tank1.levelOfLiquid)#
  TwoTanksExample.Requirements.LiquidLevel
liquidLevel1_autogen_bind_1(waterLevel =
twoTanksDesign1.tank2.levelOfLiquid)#
  TwoTanksExample.Design.TwoTanksDesign
twoTanksDesign1#
end GraphicalModel;
```

Generating/Composing Verification Models



Automatic verification scenario generation



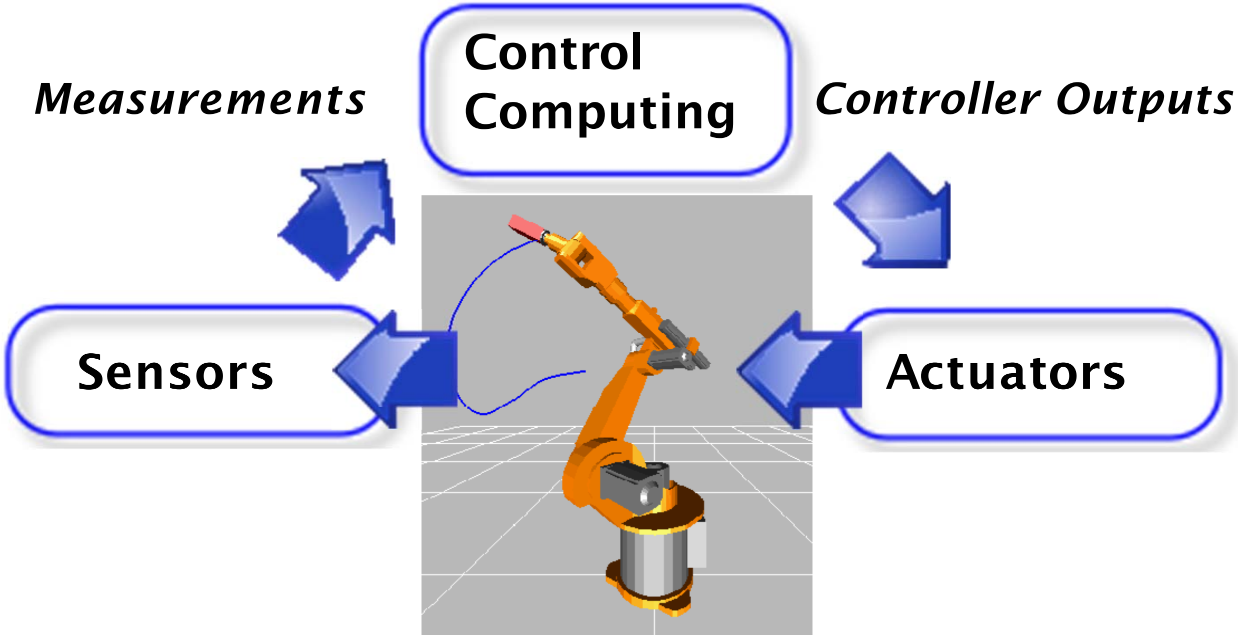
Real-time Simulation in Modelica

New ITEA3 project EMPHYSIS

EMbedded systems with PHYSIcal models In production code Software

Bernhard Thiele
Dept Computer and Information Science
Linköping University

Real-Time Control System Applications



Interactive Real-Time Simulations

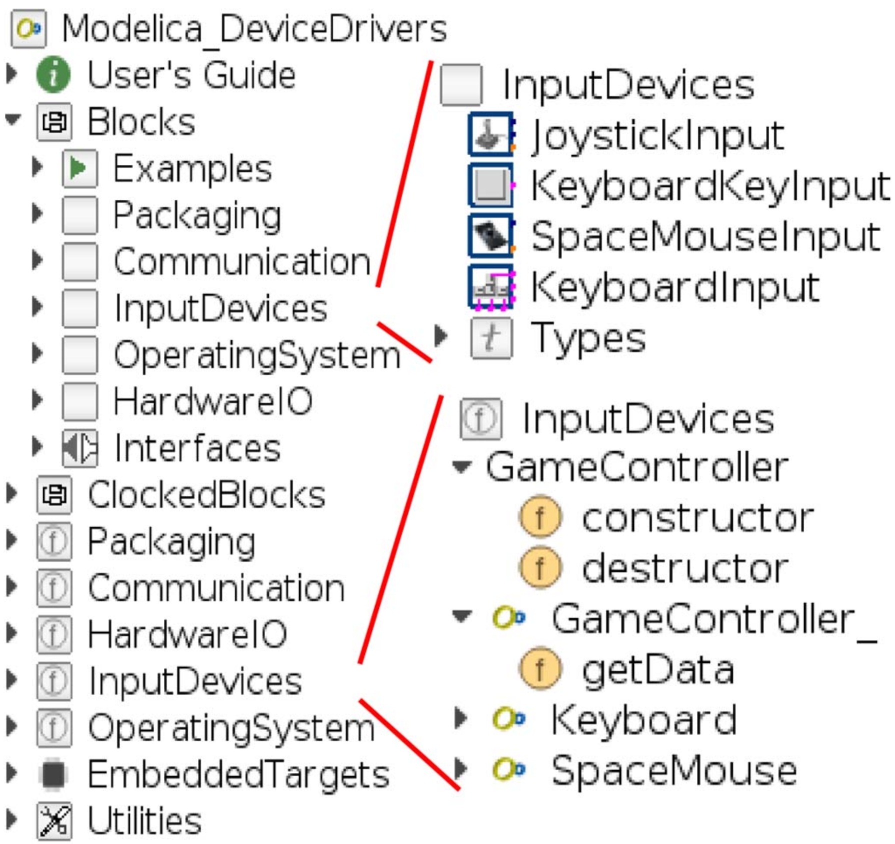
- Human-in-the-Loop (HITL) simulators (including flight, driving, and marine training simulators),
- Hardware-in-the-Loop (HIL) simulators

Needed:

- Synchronize simulation with "wall clock" time
- Access hardware devices

MODELICA pelab

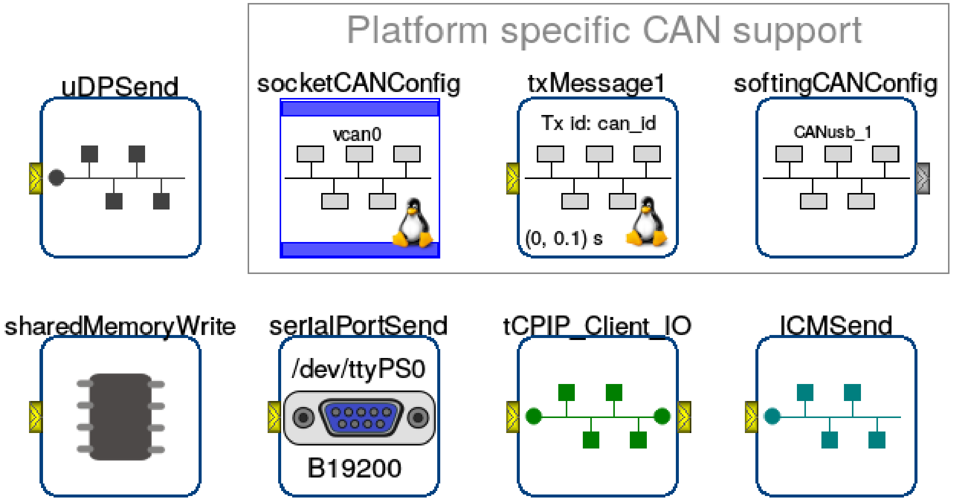
Approach: Modelica_DeviceDrivers Library (MDD)

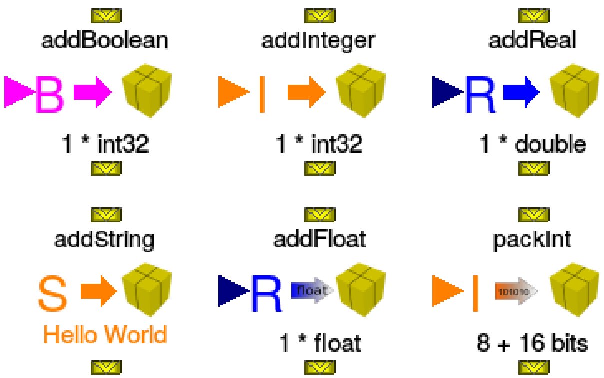


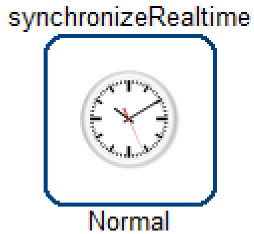
- Free library for interfacing hardware drivers
  https://github.com/modelica/Modelica_DeviceDrivers
- Layered Design:
  - Block Layer: Drag & drop graphical interface
  - Function Layer: Modelica (external C) functions
    C-Code Layer: OS specific C code

Featured MDD Blocks (Mostly Cross-Platform)

Communication

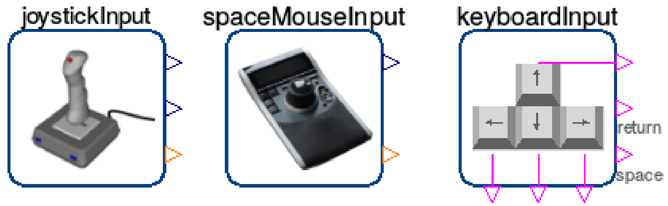




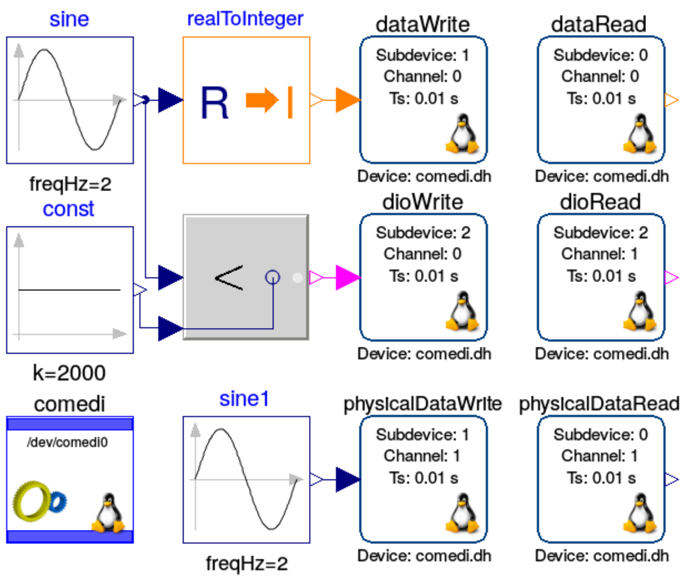


Alternatively:
OMC sim. flag
-rt=1

Input Devices



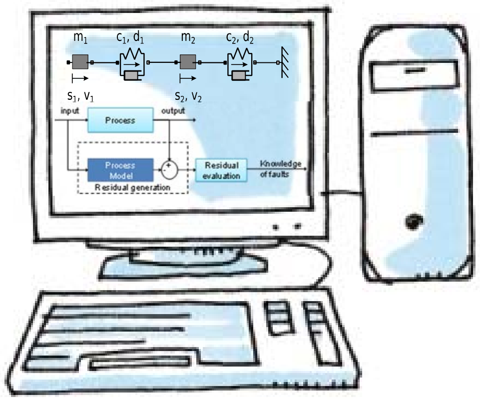
Hardware I/O (Linux only)

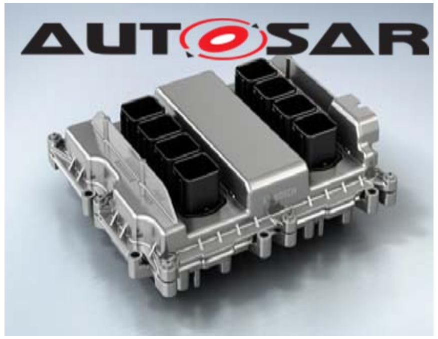


Project Overview and Swedish Consortium

2018-02-06

National Swedish Coordinators:
Dan Henriksson, Dassault Systèmes
Bernhard Thiele, Linköping University





Overall Project Coordinators:
Oliver Lenord, Christian Bertsch, Robert Bosch GmbH
Prof. Dr. Martin Otter, DLR

ITEA3

EMPHYSIS - EMbedded systems with PHYSIcal models In the production code Software (2018-2021)

Addressed market:

- Software for automotive embedded systems
- Software generation and modeling & simulation tools

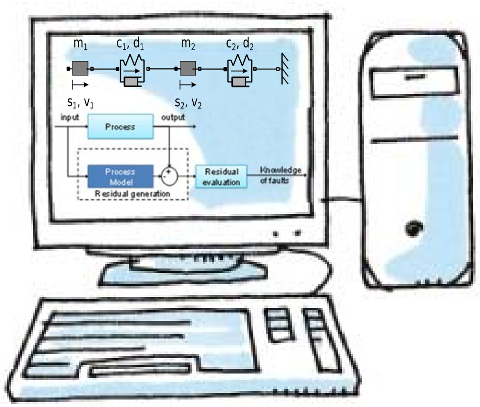
Involved Partners
(Budget: 14152 k€):

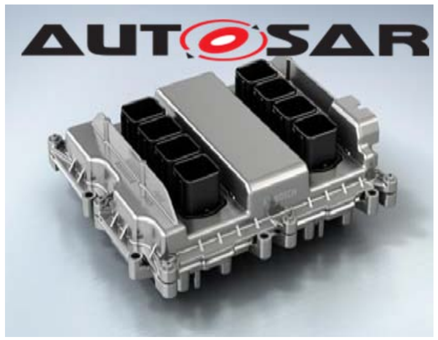
Swedish Consortium (Budget: 1711 k€, partially funded by VINNOVA):

- Dassault Systèmes (national coordination)
- Volvo Personvagnar AB
- Autoliv Electronics AB
- Modelon AB
- Linköping University (adm national coordination, 2PY)
- RISE SICS East (3.2 PY)
- Volvo Lastvagnar AB (*may join Swedish consortium during 2018*)

Project Goals

- Bridging the gap between modelling and simulation tools and embedded systems through a new interface definition (eFMI)
- Enabling the efficient implementation of advanced control and diagnosis functions with physical models
- Seamless and easy re-use of physical models both for offline simulation and on the ECU
- Collaborative development of advanced ECU-software

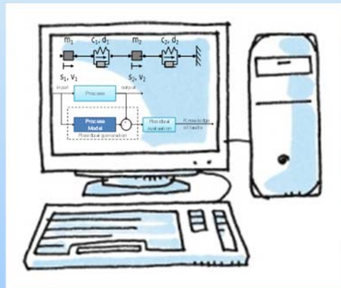




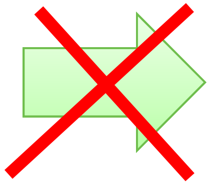
Technology Gap between Modeling and Simulation Tools and Embedded Software

ITEA 3

Physical Modelling Tools:
High-level modeling,
Model libraries
symbolic manipulation
solvers, advanced numerics



SIMULATION X
Powered by ITI
Dymola
AMESim
MapleSim
Advanced System-Level Modeling
OpenModelica
etc.



No automation,
Models
re-implemented
(hand-coded)

ECU code generation tools.
(Simulink, with special extensions (target link), ASCET)

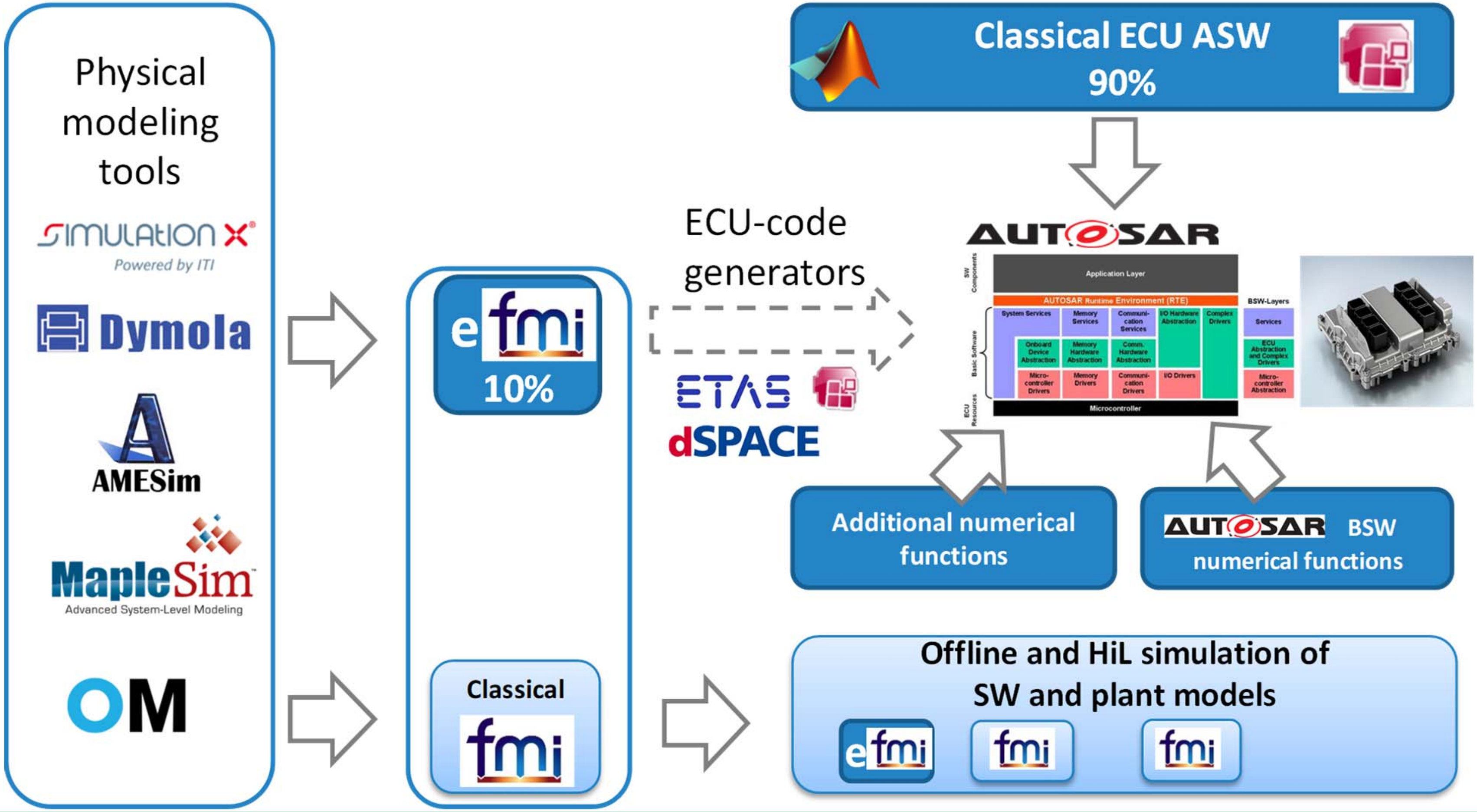
Signal-flow oriented,
with strong restrictions
(e.g., no continuous states)

THE C PROGRAMMING LANGUAGE
ASCET

Currently the design flow for physical models in ECU software is **interrupted**

Bridging the gap between modelling and simulation tools and embedded systems through a new interface definition (eFMI)

ITEA 3



Seamless model-based design of ECU-Software based on physical models.

EMPHYSIS - EMbedded systems with PHYSIcal models In the production code Software

Innovation:

- Seamless engineering of ECU applications from physical models to production code.
- New model exchange standard (eFMI) for ECU targets.
- Advanced physics based control and diagnosis functions for increasingly complex applications.

Business impact:

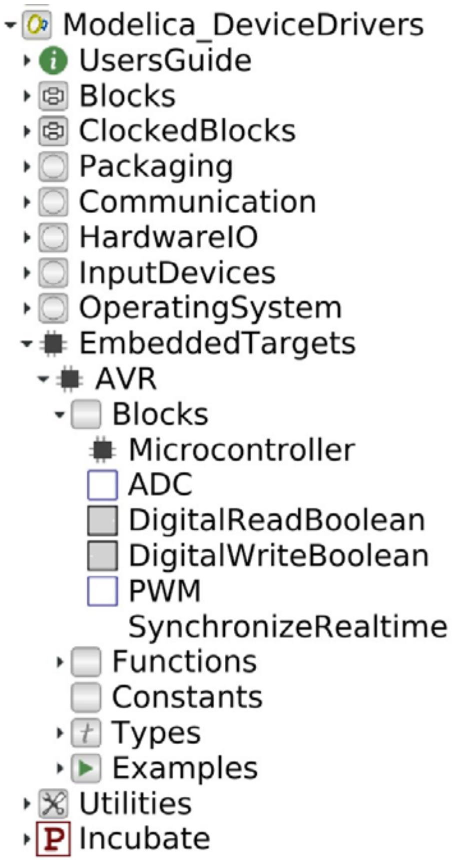
- Increased productivity of ECU software development coping with an increasing complexity.
- Better performance and efficiency of vehicles.
- Enabling new innovations in the field of mobility solutions and control engineering.

SPONSORED BY THE
Federal Ministry of Education and Research

Embedded Systems Real-time Control Using OpenModelica

Martin Sjölund
Bernhard Thiele
Dept Computer and Information Science
Linköping University

Modelica_DeviceDrivers: Embedded Targets



- Explicitly model the hardware available in the microcontroller.
- The library includes external objects that deal with the microcontroller constants and flags.
- The AVR package handles Atmel's ATmega microcontrollers and includes analog and digital I/O as well as real-time synchronization.

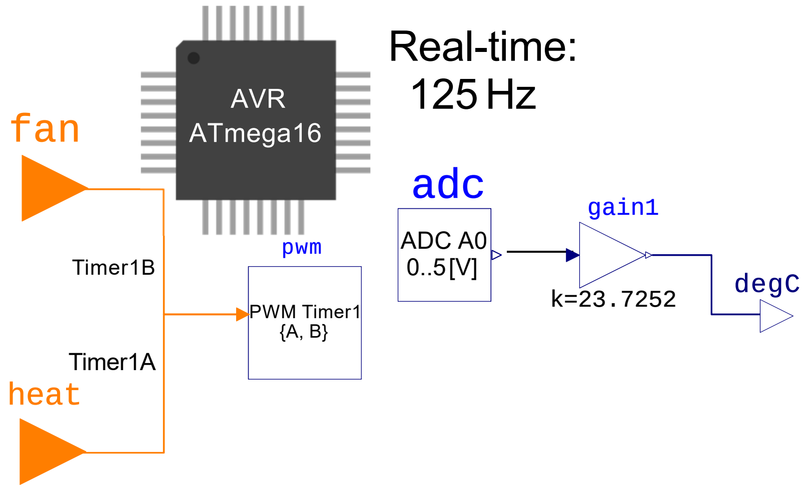
Single Board Heater System (SBHS)

One of the AVR examples included in Modelica Device Drivers library is the *Single Board Heater System* (SBHS, http://sbhs.fossee.in/), which was developed by IIT Bombay and is used for teaching and learning control systems. It consists of:

- Heater assembly
- Fan
- Temperature sensor
- AVR ATmega16 microcontroller
- Associated circuitry

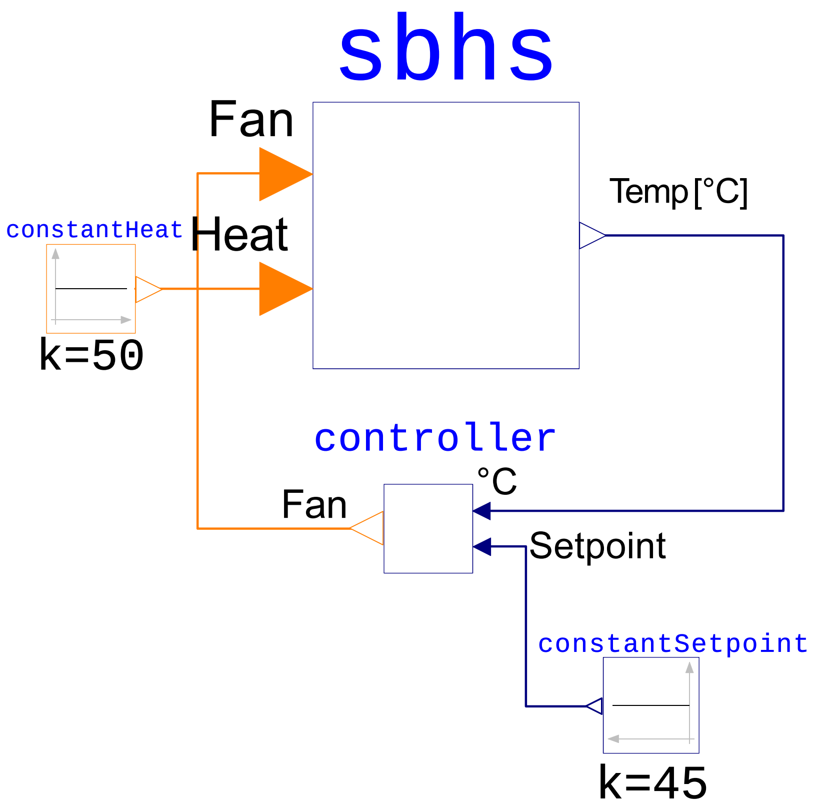
Modeling the SBHS

- Uses a real-time controller (here set @125 Hz).
- Uses pulse width modulation (PWM) to control the heater and fan.
- Uses an analog-to-digital converter (ADC) block to read the temperature (0V=0C, linear gain; the SBHS does the rest in hardware).
- Includes code for the LCD (not shown in the diagrams).



Controlling temperature using the fan

- The example feeds the heat assembly a constant (PWM) voltage.
- It then includes a PID controller with a fixed setpoint, trying to keep the temperature at a constant 45 °C by sending a PWM signal to the fan.

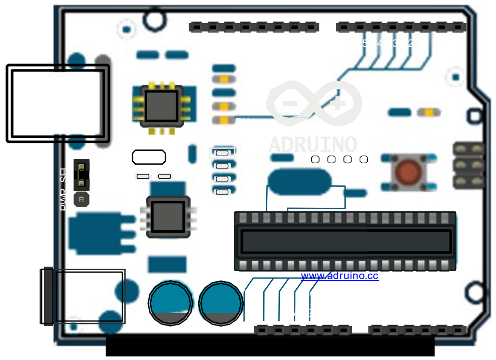


Code Generator

- Designed to support as many targets as possible.
- Supports few Modelica constructs.
- Focuses on generating good code with small footprint.
- Unsupported constructs such a linear systems are rejected.
- Reasonably predictable execution times.
- FMU-like interface (statically linked).

Target Agnostic

- No support for Atmel AVR or Arduino in the compiler.
- Compiler generates simple C code without use of OS or C library.
- Not a single malloc call, even during initialization.
- All hardware I/O and clocks is handled by the Modelica_DeviceDrivers library.



Using the Code Generator

Listing 1: Command sequence to use the code generator

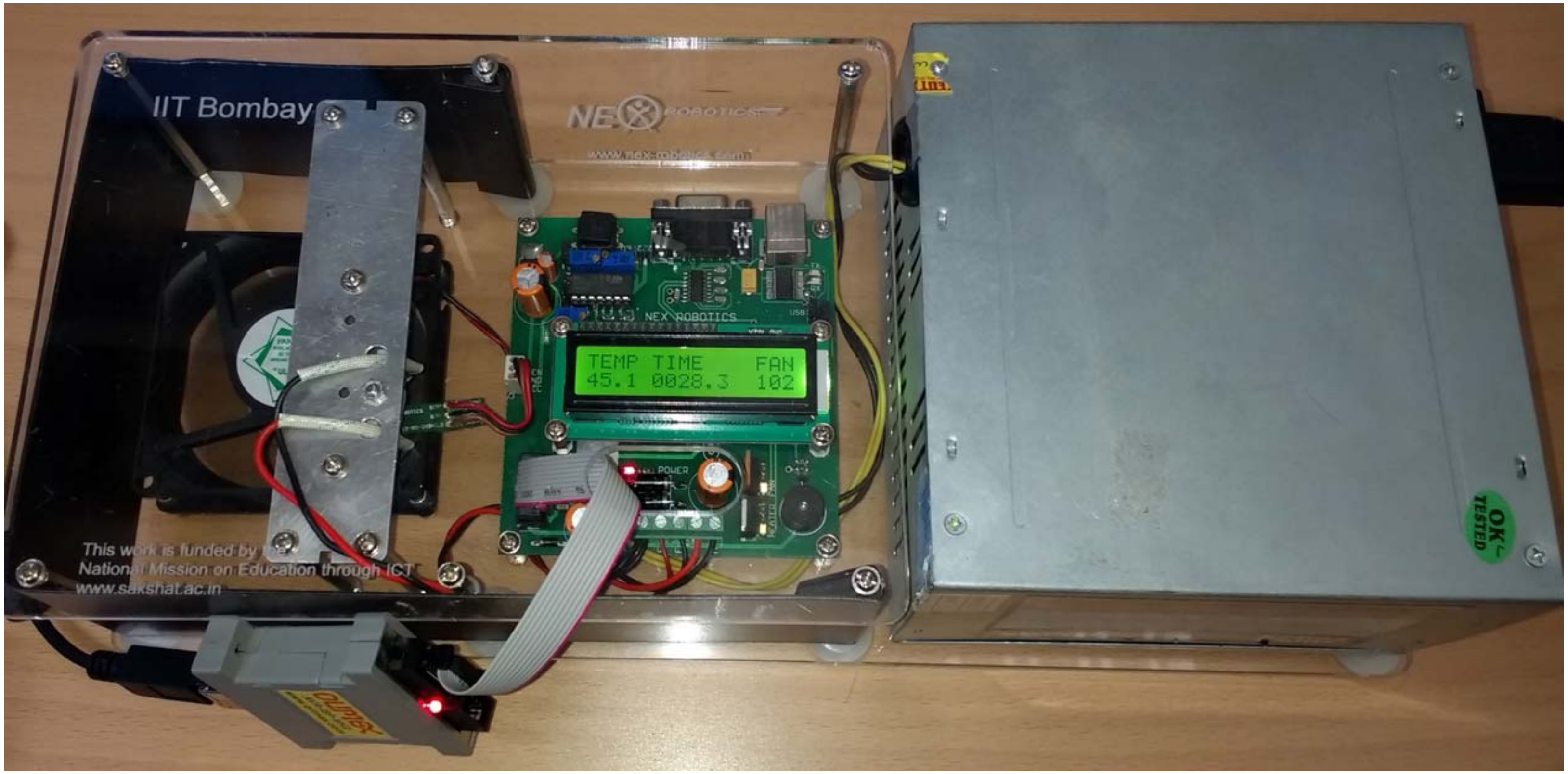
```
# Generate a generic C-file
omc --preOptModules+= evaluateParameters --
    evaluateFinalParameters --evaluateProtectedParameters --
    replaceEvaluatedParameters -s --simCodeTarget=
    ExperimentalEmbeddedC M.mo
# Compile the C-code, targetting an ATmega328P clocked at @16
    MHz
avr-gcc -Os -std=c11 -ffunction -sections -fdata -sections -
    mmcu=atmega328p -DF_CPU =16000000 UL -I ~/ OpenModelica/
    build/include/omc/c -Wl,--gc-sections M_main.c -o M_avr - I
    ~/dev/Modelica_DeviceDrivers/Modelica_DeviceDrivers/
    Resources/Include /home/marsj/dev/SBHS/ModelicaLibs/
    libModelicaExternalC.a
# Create a hex-file used by avrdude
avr-objcopy -O ihex -R .eeprom M_avr M.hex
# Upload the hex-file corresponding to the controller using the
    Arduino USB protocol. Assume the processor is ATmega328P
avrdude -F -V -c arduino -p ATMEGA328P -P /dev/ttyACM0 -b
    115200 -U flash:w:M.hex
avr-size M_avr
```

Code Generator Comparison, Full vs Simple

	Old code generator: Full source-code FMU rgetita g	Simple code generator targeting 8-bit AVR
8-bit AVR		
Hello World (0 equations)	43 kB flash memory 23 kB variables (RAM)	130 B flash memory 0 B variables (RAM)
SBHS Board (real-time PID, LCD, etc)	68 kB flash memory 25 kB variables (RAM)	4096 B flash memory 151 B variables (RAM)

Table: The full code generator has a high overhead due to large strings, etc. being embedded in the executable whereas the simple code generator only contains code that is necessary to simulate the model. It also consumes a lot more program memory when more equations are added to the system. The largest 8-bit AVR processor MCUs (Micro Controller Units) have 16 kB SRAM. The common ATmega328p (Arduino Uno) has 2 kB SRAM. The ATmega16 we target has 1 kB SRAM available (stack, heap, and global variables).

SBHS controller using MDD and the new code generator



www.liu.se