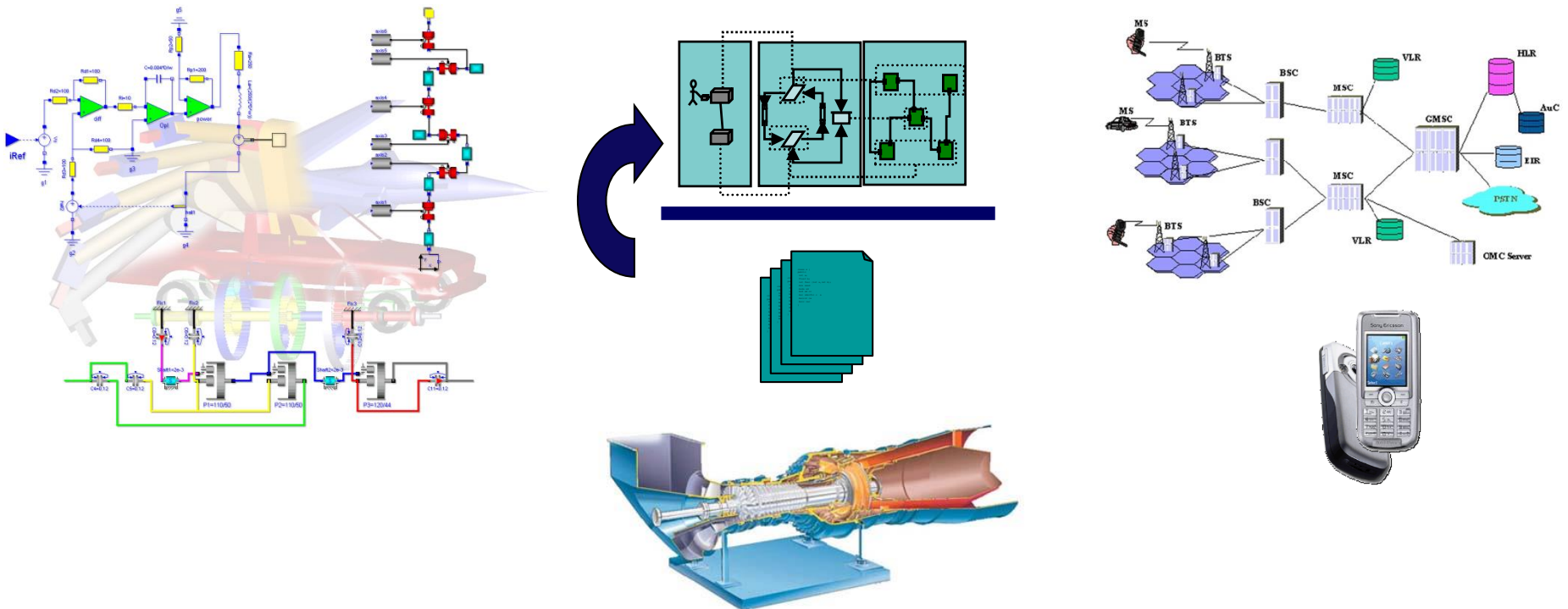


Research in Model-Based Product Development at PELAB in the MODPROD Center

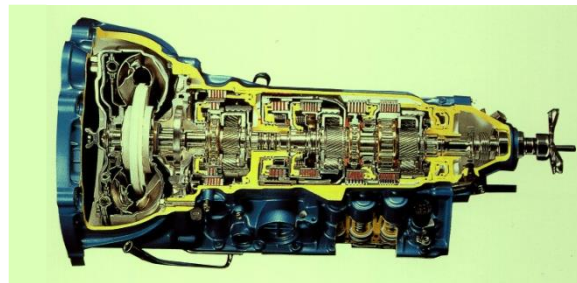
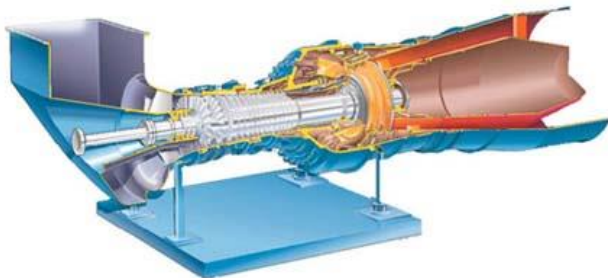
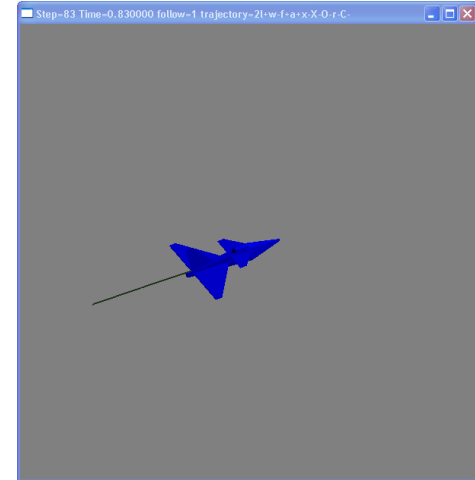
Presentation at MODPROD'2017
Department of Computer and Information Science
Linköping University
2017-02-07

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



Examples of Complex Systems in Engineering

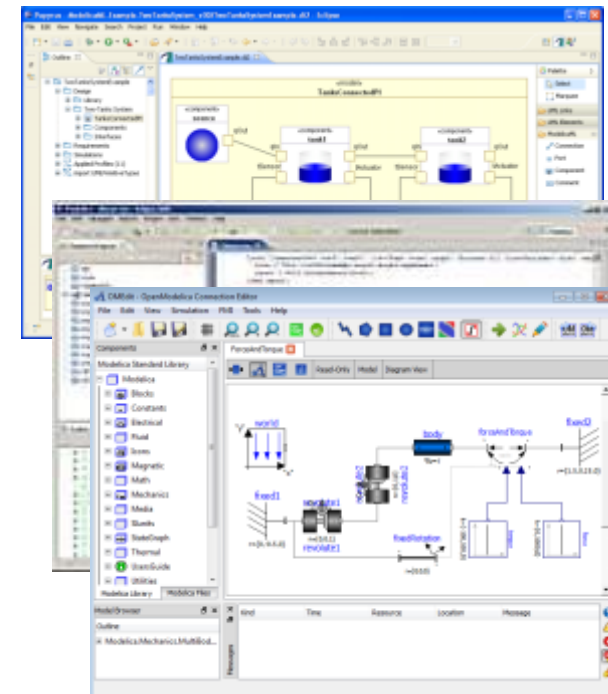
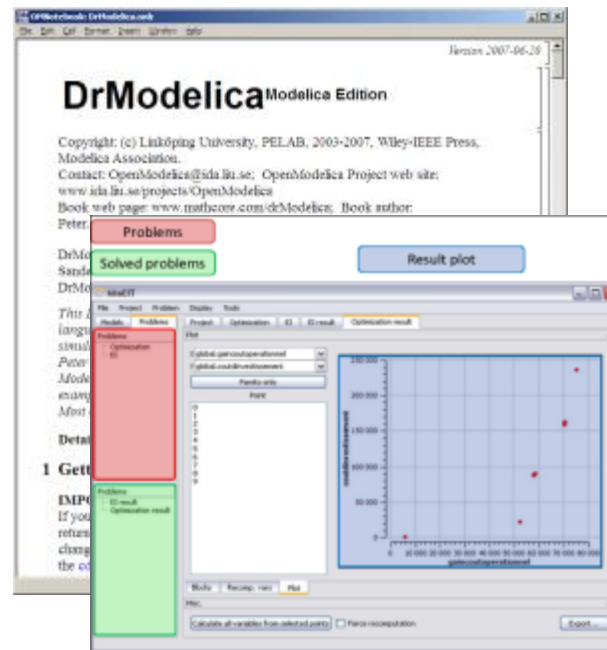
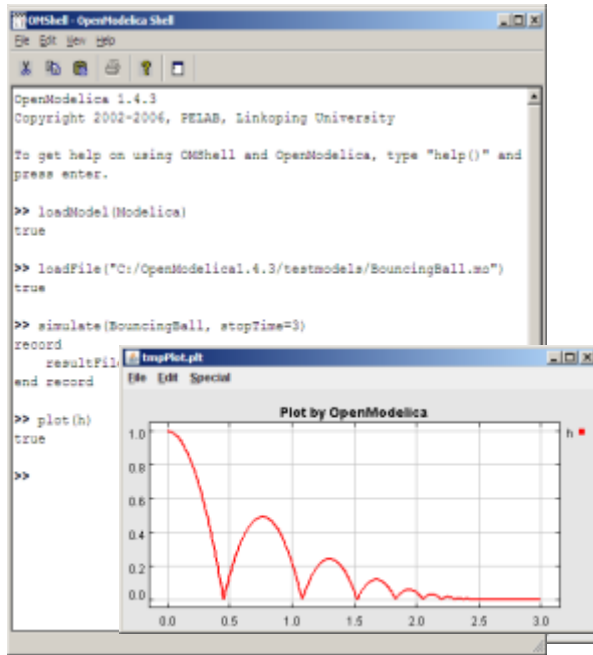
- Robotics
- Automotive
- Aircraft
- Mobile Phone Systems
- Business Software
- Power plants
- Heavy Vehicles
- Process industry



The OpenModelica Open Source Environment

www.openmodelica.org

- Advanced Interactive Modelica compiler (OMC)
 - Supports most of the Modelica Language
 - **Modelica** and **Python** scripting
- Basic environment for creating models
 - **OMShell** – an interactive command handler
 - **OMNotebook** – a literate programming notebook
 - **MDT** – an advanced textual environment in Eclipse
- **OMEdit** graphic Editor
- **OMDebugger** for equations
- **OMOptim** optimization tool
- **OM Dynamic optimizer** collocation
- **ModelicaML** UML Profile
- **MetaModelica** extension
- **ParModelica** extension



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

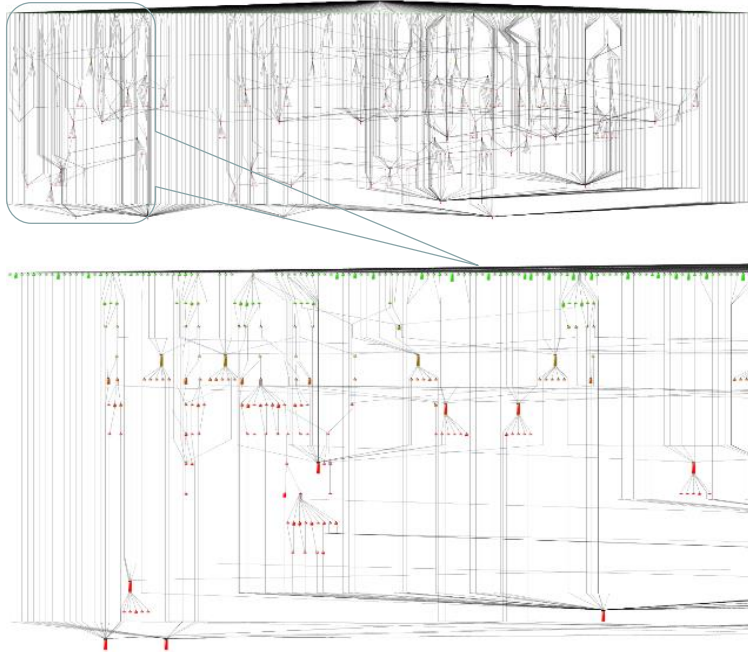
Modeling Support Environments

Parallel Execution Compilation to MultiCore

Mahder Gebremedhin

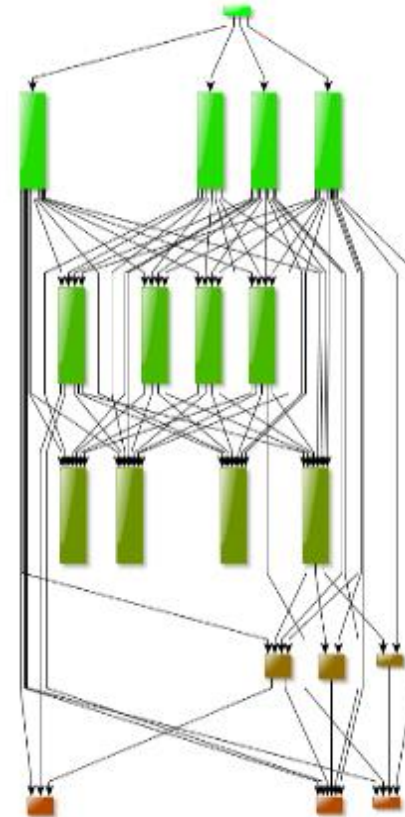
Compiling Models to Efficient Parallel Code (scheduling on multiple cores)

Modelica.Electrical.Spice3.Examples.Spice3BenchmarkFourBitBinaryAdder



Original task system of Four Bit Binary Adder model

1122 Tasks
1360 Edges

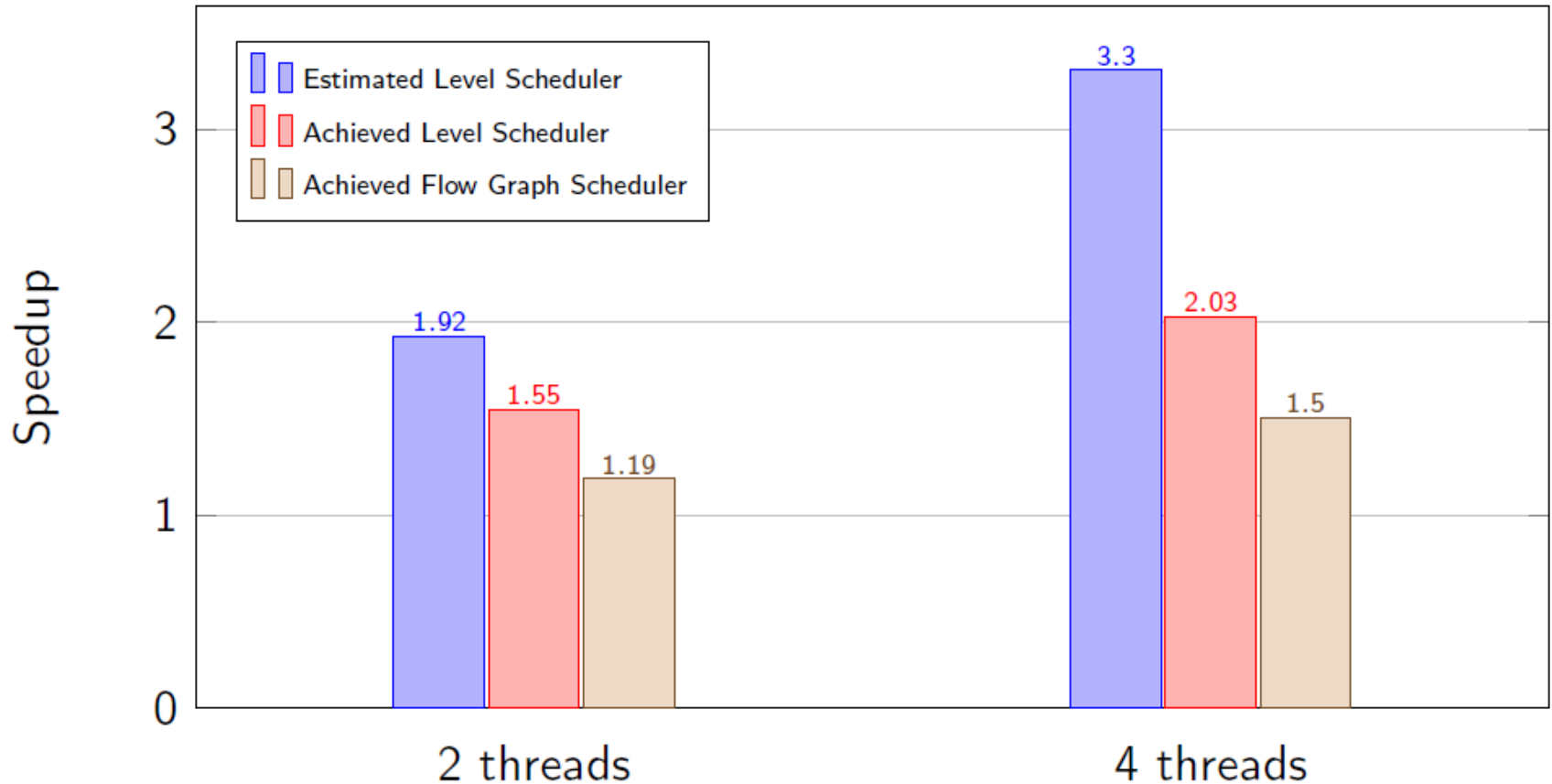


Task system after clustering for level scheduler

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:

The screenshot displays the OMEdit - Transformational Debugger interface, which is divided into three main panels: Variables View, Equations View, and Source View.

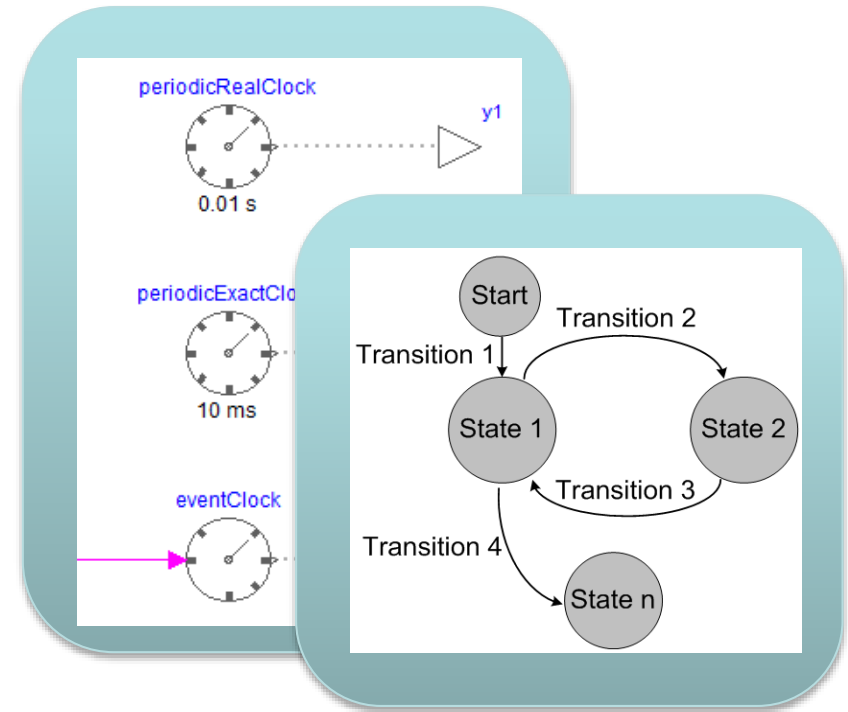
- Variables View:** Located at the top left, it contains a Variables Browser on the left with a tree view showing the model structure (frame, boxBody1, body, frame_a, R, T). To its right are two tables: "Defined In Equations" and "Used In Equations", both with columns for Index, Type, and Equation. Below these tables is a "Variable Operations" section with a list of operations, including "solved: boxBody1.body.frame_a.R.T[1,1] = boxBody1.frame_b.R.T[1,1]" and "substitute: boxBody1.body.frame_a.R.T[1,...xBody1.frameTranslation.frame_a.R.T[1,1]".
- Equations View:** Located at the bottom left, it features an Equations Browser with columns for Index, Type, and Equation, listing equations from -819 to -920. To its right are "Defines" and "Depends" tables, and an "Equation Operations" section showing transformations such as "solve: -world.frame_b.f[2] = (-boxBody1...ame_b.R.T[2,2]) * revolute1.frame_b.f[2]", "scalarize(2): {-world.frame_b.f[1], -worl...rame_b.R.T[2,2]} * revolute1.frame_b.f[2]", "simplify: {-boxBody1.frame_b.R.T[1,1] * ...1.frame_b.f[2], -revolute1.frame_b.f[3]}", "inline: -Modelica.Mechanics.MultiBody.Fr...e_b.f[2] + 1.0 * revolute1.frame_b.f[3]", and "substitute: -Modelica.Mechanics.MultiBo...frame_b.f[2], revolute1.frame_b.f[3]".
- Source View:** Located on the right, it shows the source code from the Source Browser. The code includes comments and function calls like `Frames.planarRotation` and `Frames.absoluteRotation`. A red box highlights a specific section of code, and a black arrow points from the Equations View to this section, illustrating the mapping of dynamic run-time error to source model position.

Mapping dynamic run-time error to source model position

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

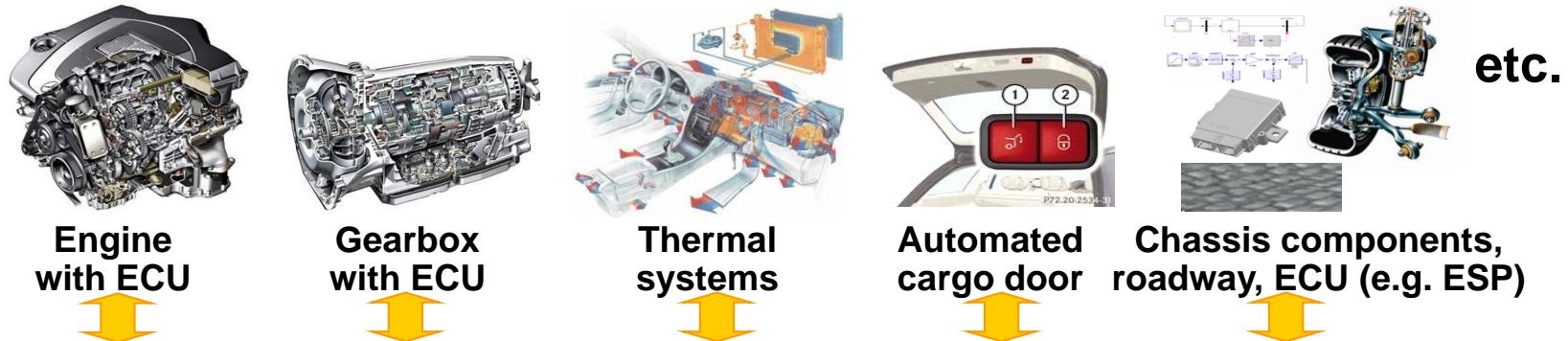
Traceability Support in OpenModelica Using Open Services for Lifecycle Collaboration (OSLC)

*Traceability between requirements, models, and
simulation artifacts*

Alachew Mengist, Adrian Pop,
Adeel Asghar, Peter Fritzson

See separate Talk on wednesday

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



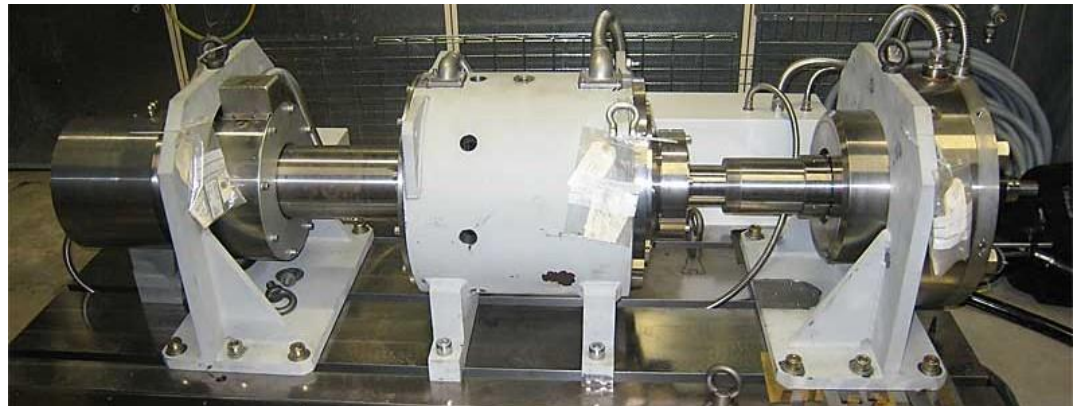
functional mockup interface for model exchange and tool coupling

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)
- **> 60 tools** supporting it (<https://www.fmi-standard.org/tools>)

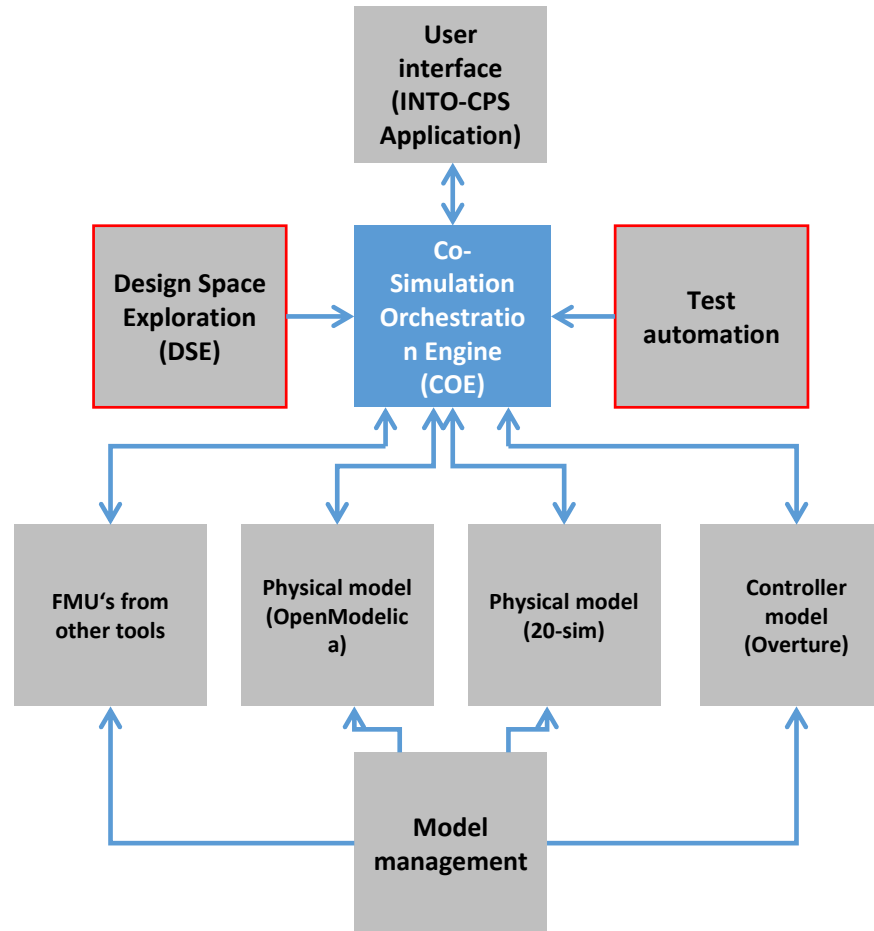
Enhanced FMI Co-simulation, Run-time, and Master Simulation Tool – Work in OPENCPS Project

- 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



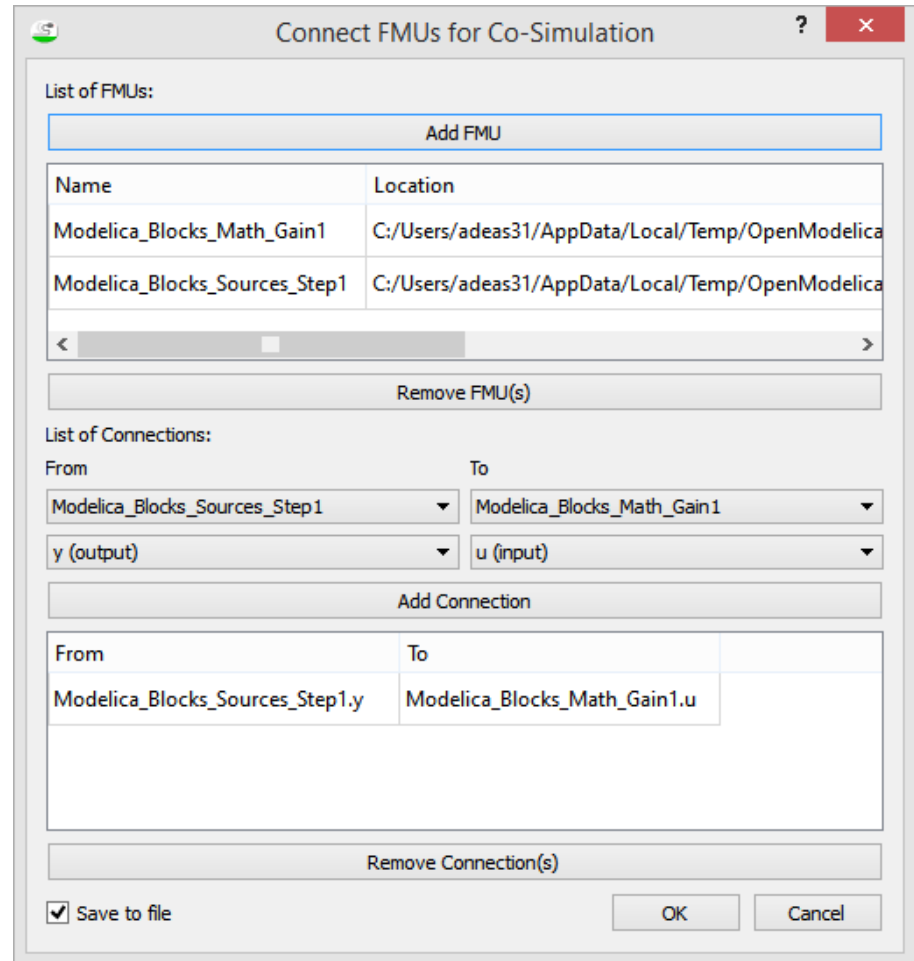


INTO-CPS: Co-Simulation Framework Vision

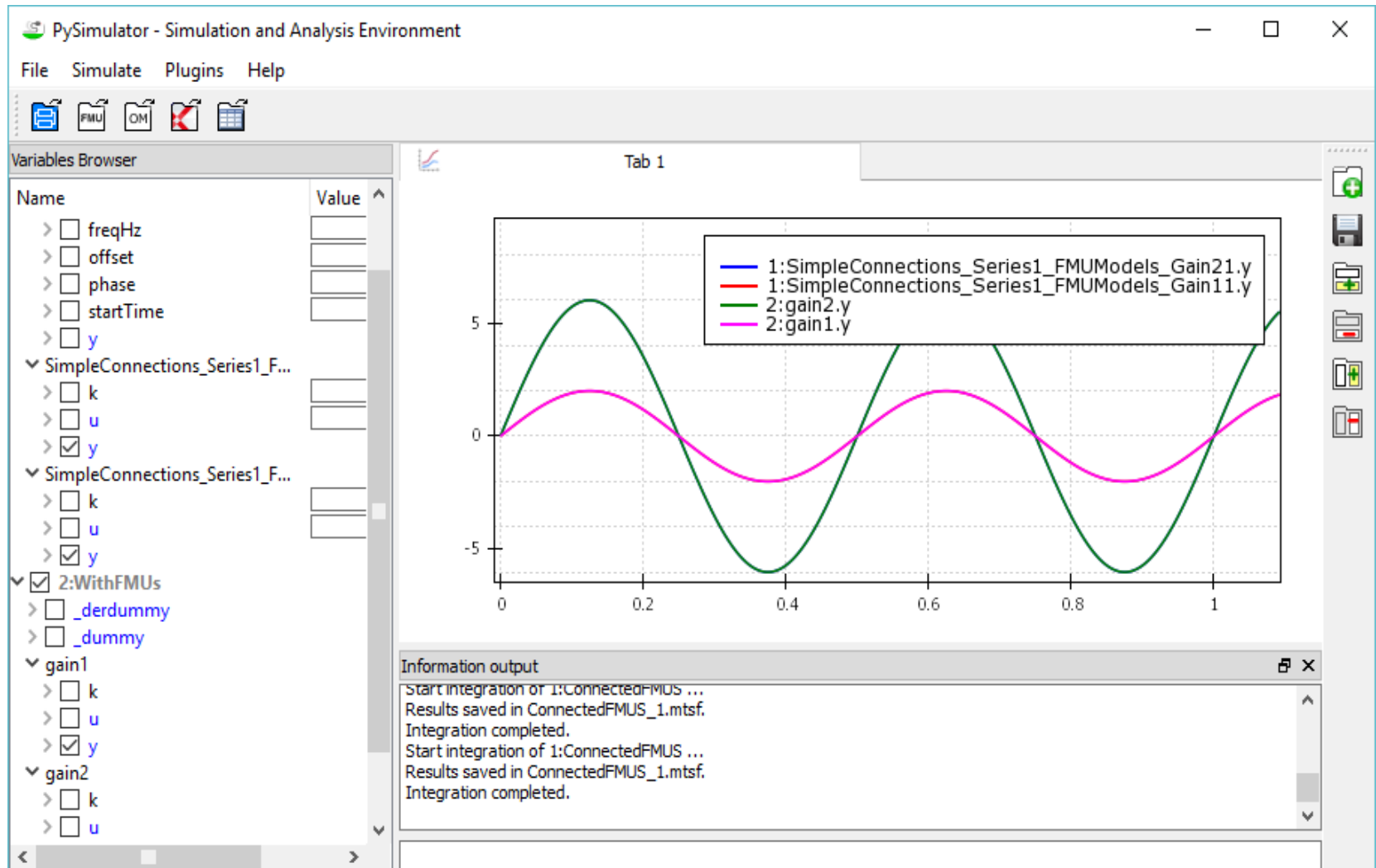


PyModSimA: Simulation of Connected FMUs

- 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.

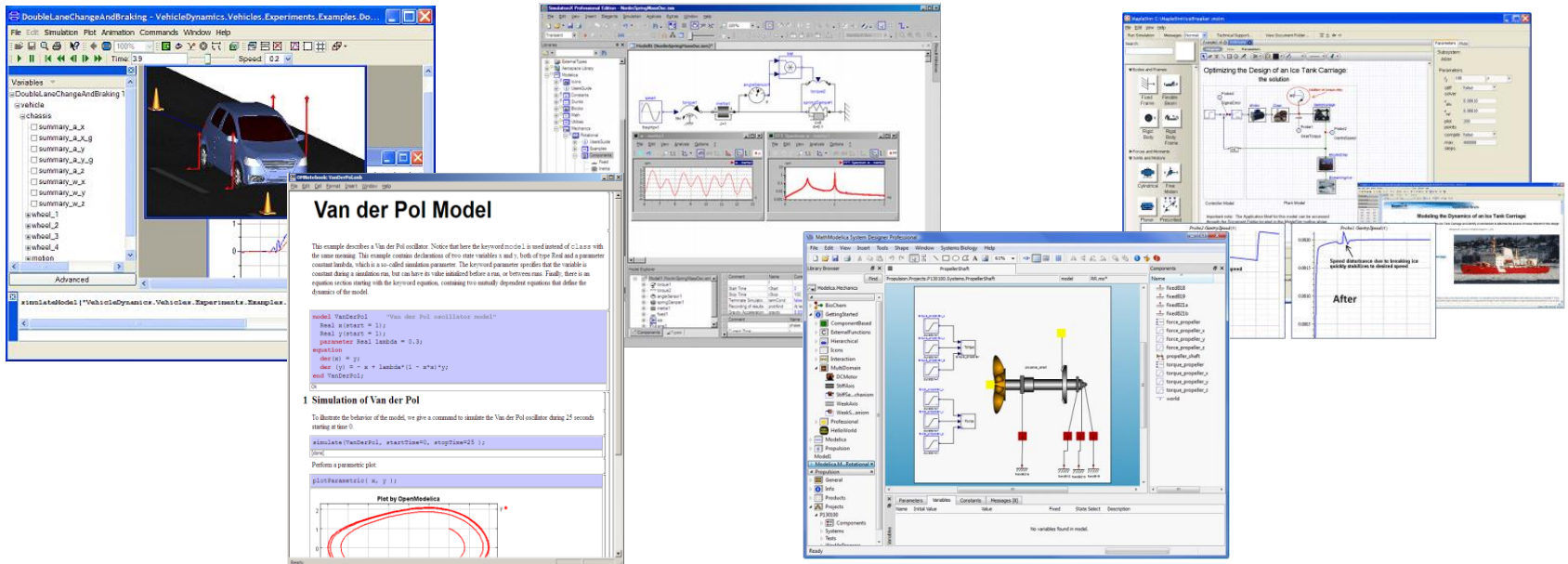


PyModSimA: Co-Simulation of FMUs in Pysimulator

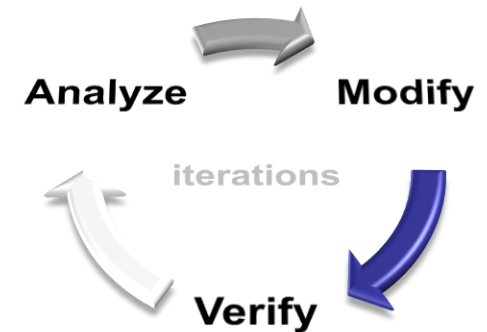
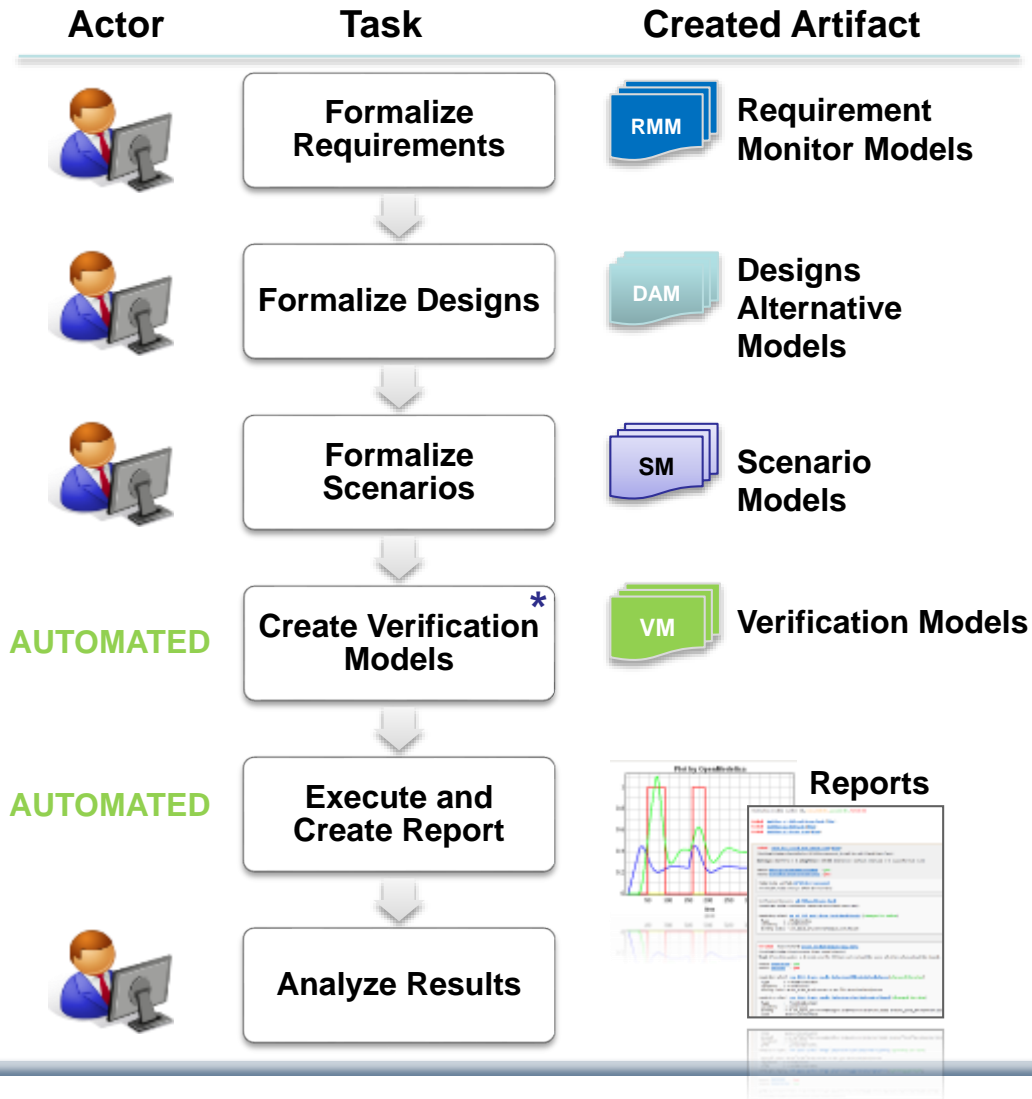


Dynamic Verification/Testing of Requirements vs Usage Scenario Models

Lena Buffoni, Wladimir Schamai, Peter Fritzson and contributions from MODRIO partners



vVDR Method – virtual Verification of Designs vs Requirements



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

Support of vVDR in Modelica within OMEdit in OpenModelica

Libraries

- Mediators
 - operatingPumps
 - cavitating
 - breakMediator
- ToyExample
 - PA
 - PB
 - PumpR
 - SystemModel
 - Scenario1
 - VerifScenario1
 - Scenario2
 - SystemModelBetter
- VDRDefinitions
 - Scenario
 - Requirement
 - Design
- BindingDefinition
 - Client
 - Provider
 - Mediator
 - Preferred

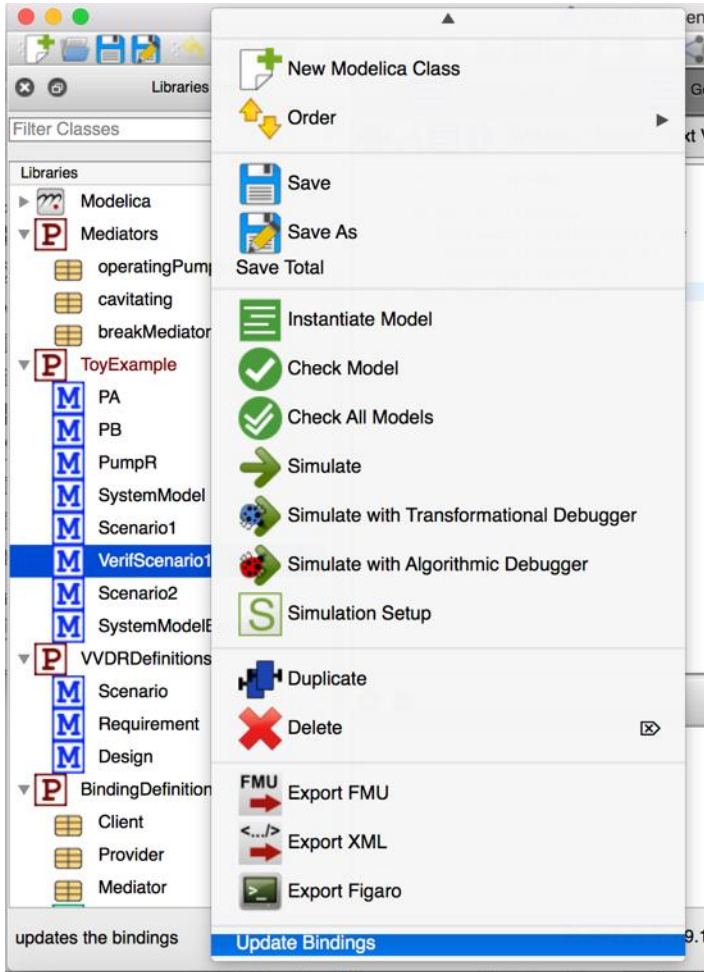
```
5 record operatingPumps
6   extends Mediator(mType = "Boolean",
7     clients = {Client(modelID = "ToyExample.PumpR", component = "inOperation")},
8     providers = {Provider(modelID = "ToyExample.PA", template = "if %getPath.on then 1 else 0"),
9       Provider(modelID = "ToyExample.PB", template = "if (%getPath.volFlowRate) > 0 then 1 else 0")}
10  );
11 end operatingPumps;
12
```

```
1 within ToyExample;
2
3 model VerifScenario1
4   ToyExample.SystemModel md;
5   ToyExample.Scenario1 s1;
6   ToyExample.PumpR r1;
7 end VerifScenario1;
```

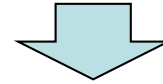
vVDR concepts in **standard Modelica**

- **mediators** mapped to **records**
- **requirements, design, scenarios** mapped to Modelica classes

Single Scenario Generation



```
1 within ToyExample;  
2  
3 model VerifScenario1  
4   ToyExample.SystemModel md;  
5   ToyExample.Scenario1 s1;  
6   ToyExample.PumpR r1;  
7 end VerifScenario1;
```



```
1 within ToyExample;  
2  
3 model VerifScenario1  
4   ToyExample.SystemModel md_autogen_bind_0(timeBreak = s1.timeFailure);  
5   ToyExample.Scenario1 s1;  
6   ToyExample.PumpR r1_autogen_bind_0(cavitate = md_autogen_bind_0.pa.cavitating);  
7   ToyExample.PumpR r1_autogen_bind_1(cavitate = md_autogen_bind_0.pb.cavitating);  
8 end VerifScenario1;
```

Generating correct number of requirement instances

Connecting the design model and the requirement

Batch Scenario Generation

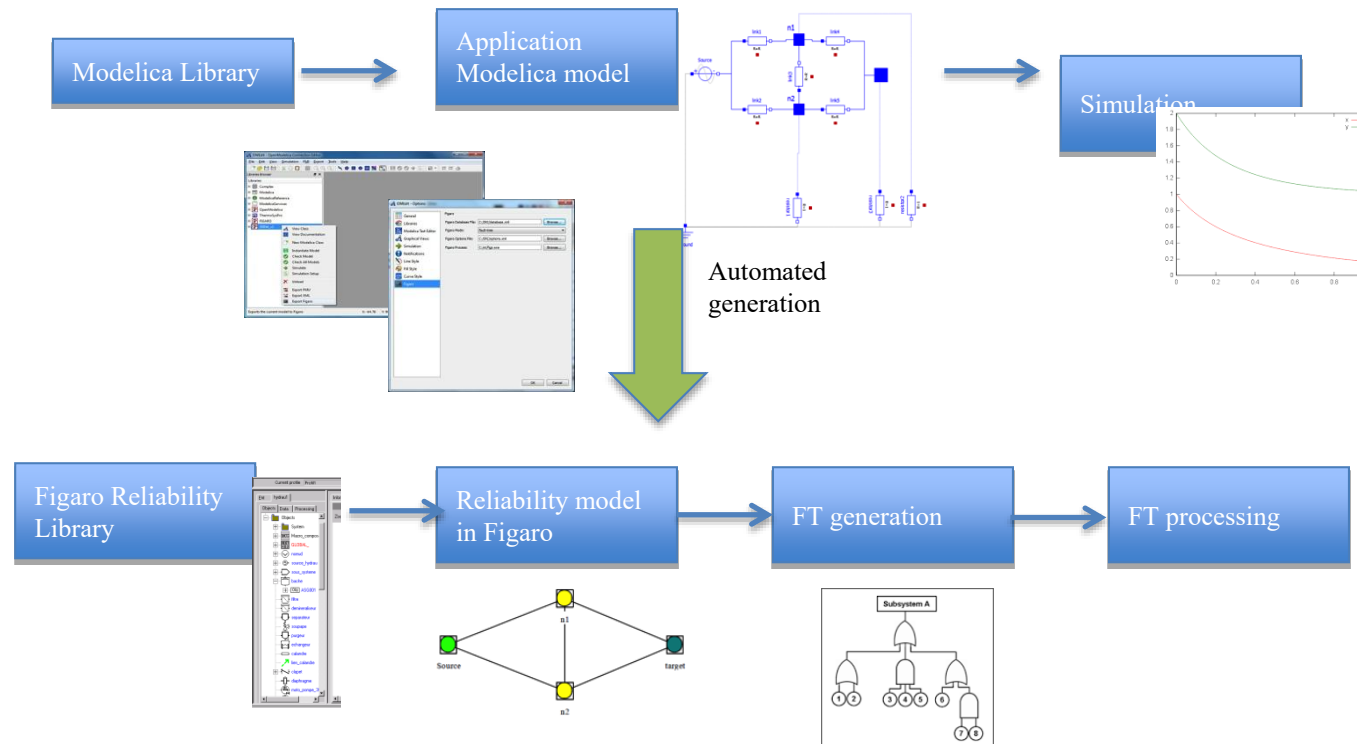
The screenshot displays the Modelica IDE interface. On the left, a context menu is open, showing various actions. The 'Generate Verification Scenarios' option is highlighted at the bottom. The main workspace shows a package named 'GenerateTests' containing three verification models: 'verif...ogen_1', 'verif...ogen_2', and 'verif...ogen_3'. A red box highlights these models, and a curved arrow points from this box to the code editor below. The code editor shows the following code:

```
1 package GenerateTests
2 model verif_model_autogen_1 "Autogenerated verification model"
3   ToyExample.PumpR_agen_PumpR2_autogen_bind_0(cavitate = _agen_SystemModelBetter0.pa.cavitating);
4   ToyExample.PumpR_agen_PumpR2_autogen_bind_1(cavitate = _agen_SystemModelBetter0.pb.cavitating);
5   ToyExample.Scenario1_agen_Scenario11;
6   ToyExample.SystemModelBetter_agen_SystemModelBetter0_autogen_bind_0(timeBreak = _agen_Scenario11.timeFailure);
7 end verif_model_autogen_1;
8 model verif_model_autogen_2 "Autogenerated verification model"
9   ToyExample.PumpR_agen_PumpR2_autogen_bind_0(cavitate = _agen_SystemModel0.pa.cavitating);
10  ToyExample.PumpR_agen_PumpR2_autogen_bind_1(cavitate = _agen_SystemModel0.pb.cavitating);
11  ToyExample.Scenario2_agen_Scenario21;
12  ToyExample.SystemModel_agen_SystemModel0_autogen_bind_0(timeBreak = _agen_Scenario21.timeFailure);
13 end verif_model_autogen_2;
14 model verif_model_autogen_3 "Autogenerated verification model"
15   ToyExample.PumpR_agen_PumpR2_autogen_bind_0(cavitate = _agen_SystemModel0.pa.cavitating);
16   ToyExample.PumpR_agen_PumpR2_autogen_bind_1(cavitate = _agen_SystemModel0.pb.cavitating);
17   ToyExample.Scenario1_agen_Scenario11;
18   ToyExample.SystemModel_agen_SystemModel0_autogen_bind_0(timeBreak = _agen_Scenario11.timeFailure);
19 end verif_model_autogen_3;
20 end GenerateTests;
```


Model-based Failure Mode and Effects Analysis

(Marc Bouissou and Lena Buffoni)

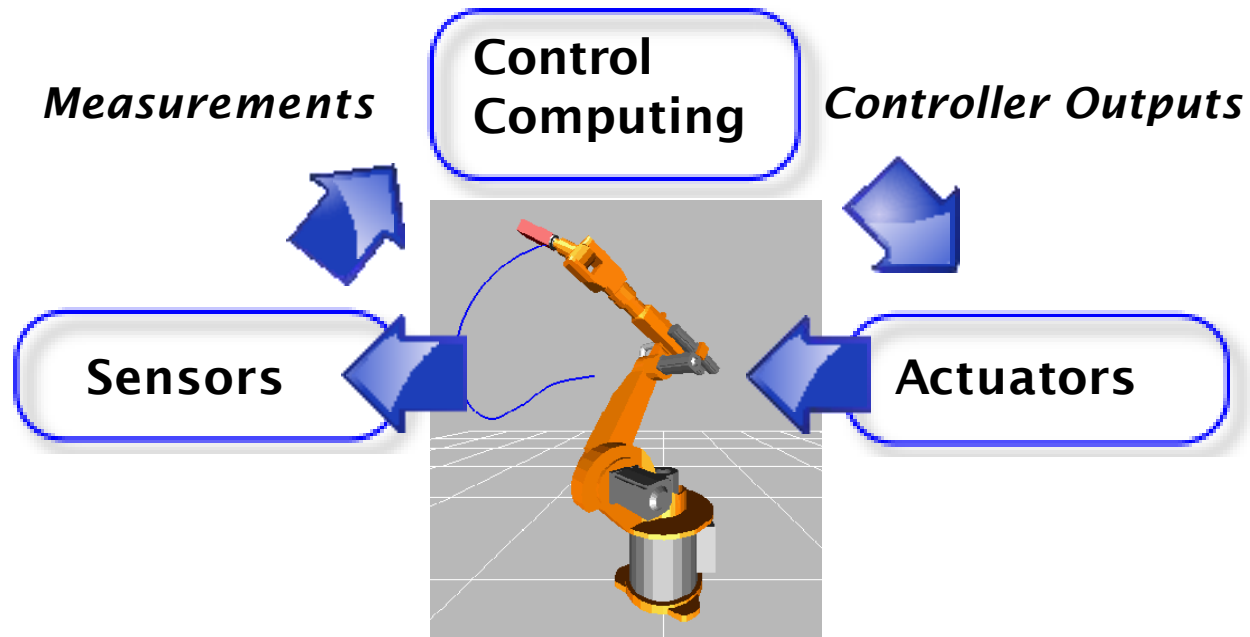
- Modelica models augmented with reliability properties can be used to generate reliability models in Figaro, which in turn can be used for static reliability analysis
- Prototype in OpenModelica integrated with Figaro tool



Real-time Simulation and State-Machine support in Modelica

Bernhard Thiele
Dept Computer and Information Science
Linköping University

Real-Time Control System Applications



Goal: Interactive Real-Time Simulations

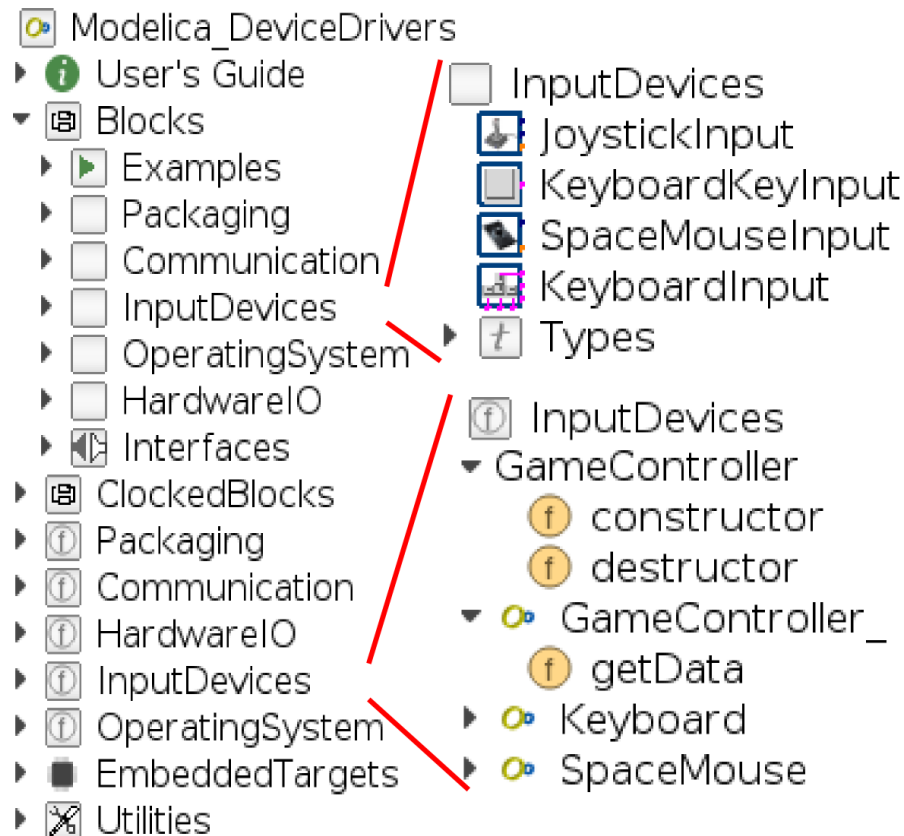
- Human-in-the-Loop (HITL) simulators (including flight, driving, and marine training simulators),
- Hardware-in-the-Loop (HIL) simulators

Needed:

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



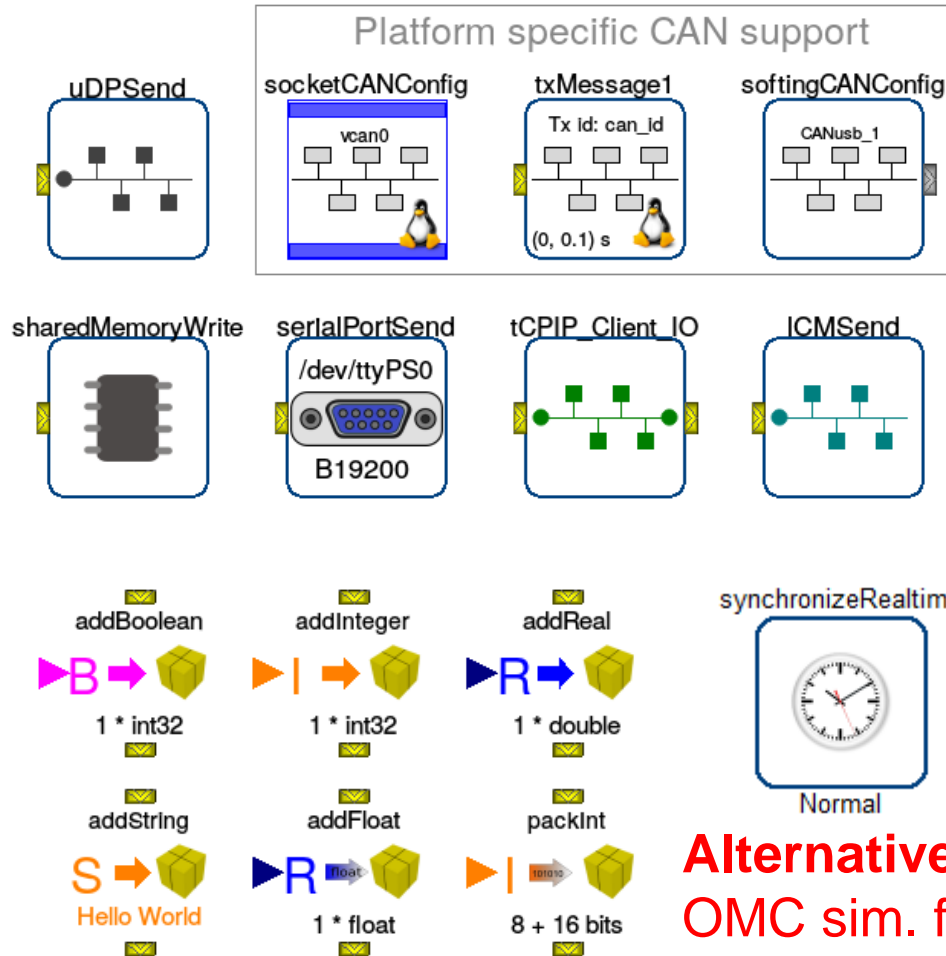
Approach: Modelica_DeviceDrivers Library (MDD)



- 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)

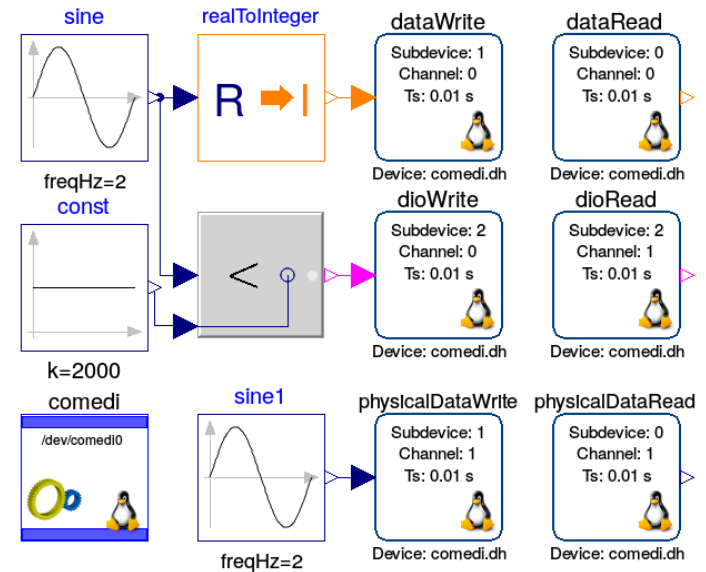
Communication



Input Devices



Hardware I/O (Linux only)



Alternatively:
OMC sim. flag
-rt=1

Challenge of Non-Modelica Standard Constructs

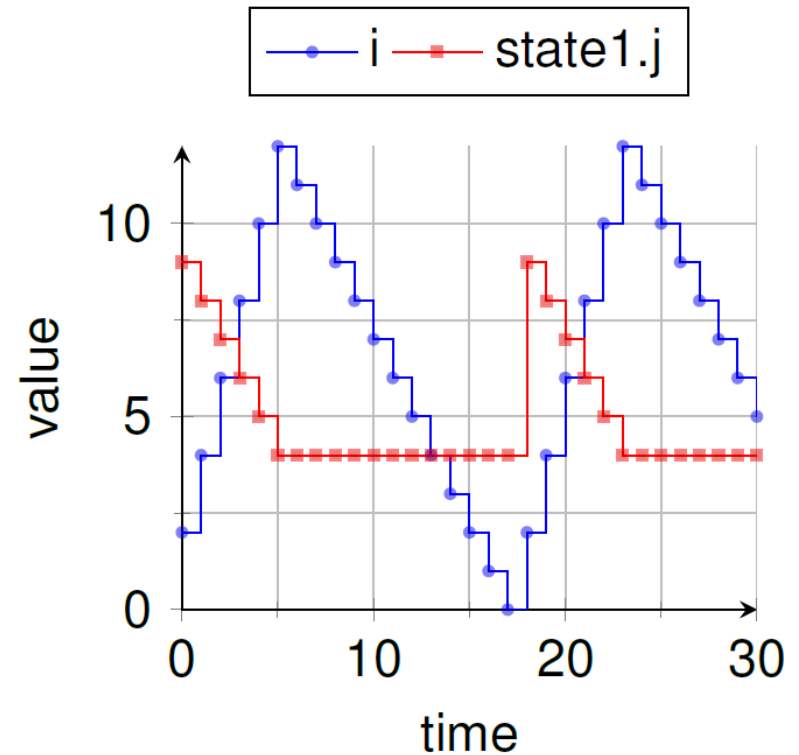
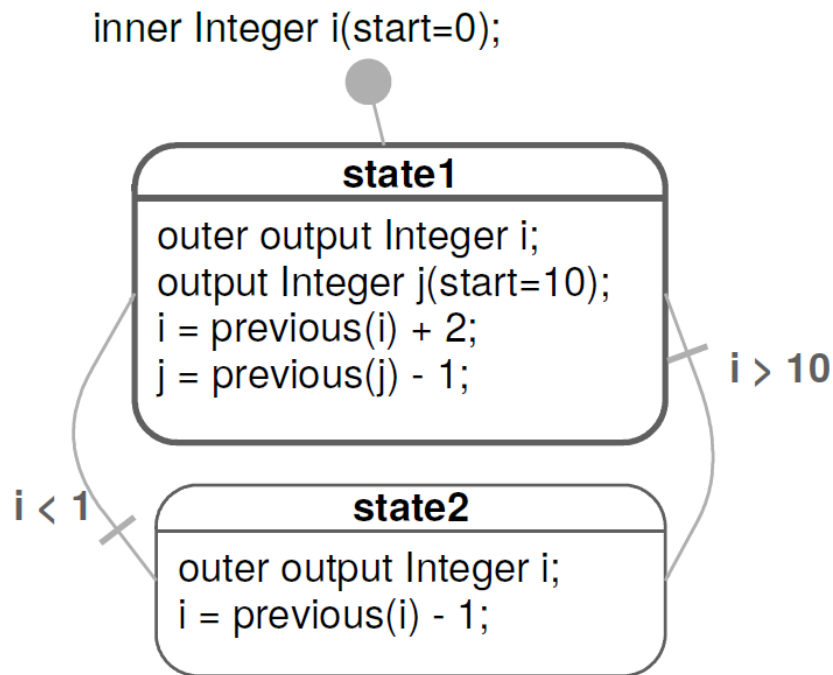
The MDD library was initially developed using the Dymola tool by Bernhard Thiele Tobias Bellmann. Support in OpenModelica poses a challenge:

- MDD is using some **non-Modelica standard** conforming constructs; the MDD code could be partly rewritten to be more conformant, but some **constructs** are **essential**
- Important parts of MDD are now **supported** by **OpenModelica** (thanks to Volker Waurich and others!)
- However, there remain parts which are not yet supported

Plans for the MDD Library

- **Extend** MDD library coverage in OpenModelica
- **Advocate** OM+MDD library as **low-cost and low-effort** solution for **interactive** simulations, particularly in combination with low-latency Linux kernels (e.g. available in Linux distributions like Ubuntu)
- Further library **improvements** and extensions
- Extend the scope of MDD library to **support** restricted **embedded** systems (Martin Sjölund will discuss this later in this talk)

State Machines in Modelica 3.3: Simple Example



Modelica State-Machines in OpenModelica

- Modelica 3.3 introduced language elements for clocked (discrete-time) state machines
- State-Machine textual constructs now supported by OpenModelica (ongoing work to support graphical editing)
- Further plans:
 - Support state-machines in the *new* OMC compiler front-end
 - **Efficient code-generation** suitable for restricted **embedded** targets
 - **Traceability** from **models** to generated **code** fragments (support debugging of state machines, facilitate V&V activities)

Embedded Systems Real-time Control Using OpenModelica

Martin Sjölund
Dept Computer and Information Science
Linköping University

Modelica_DeviceDrivers: Embedded Targets

- Modelica_DeviceDrivers
 - UsersGuide
 - Blocks
 - ClockedBlocks
 - Packaging
 - Communication
 - HardwareIO
 - InputDevices
 - OperatingSystem
 - EmbeddedTargets
 - AVR
 - Blocks
 - Microcontroller
 - ADC
 - DigitalReadBoolean
 - DigitalWriteBoolean
 - PWM
 - SynchronizeRealtime
 - Functions
 - Constants
 - Types
 - Examples
 - Utilities
 - Incubate

- 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.

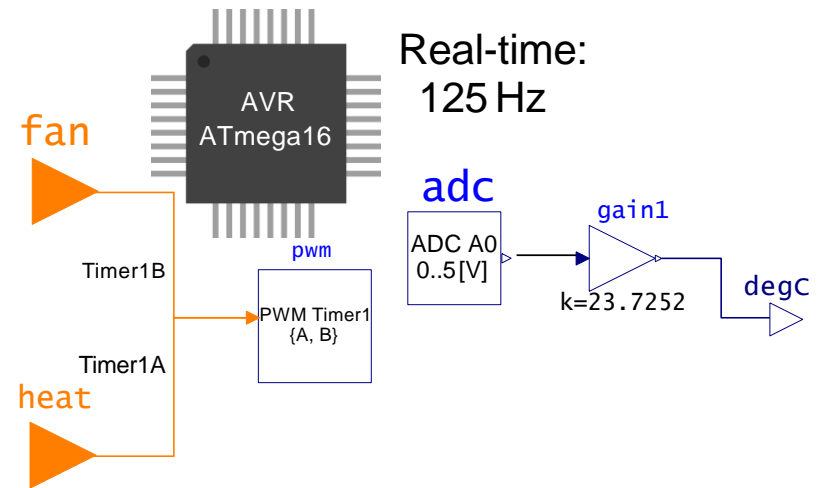
Single Board Heater System (SBHS)

One of the AVR examples included in MDD 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

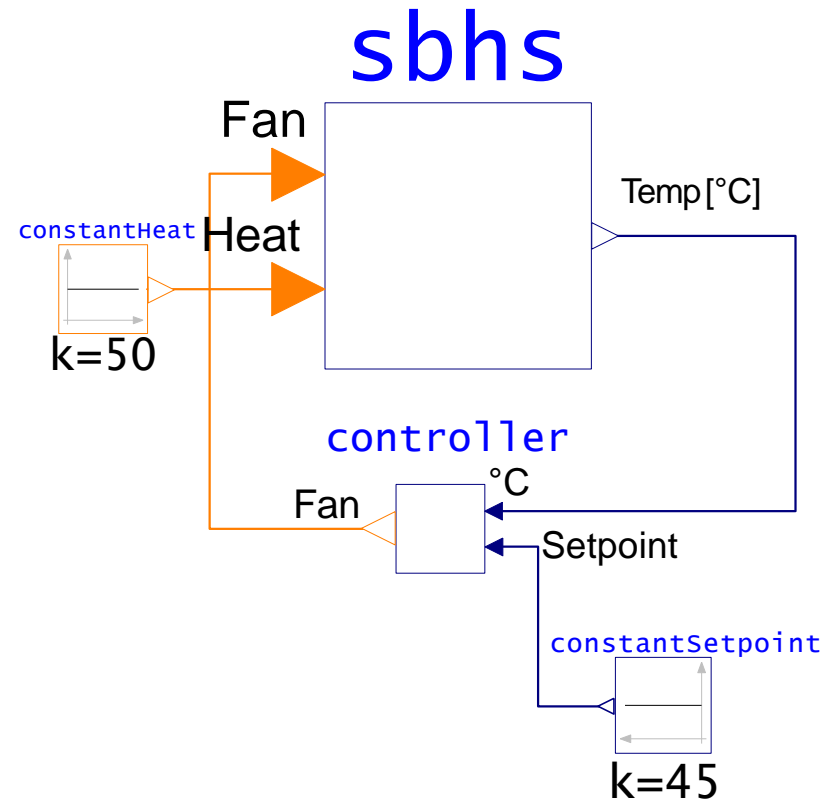
Modeling the SBHS

- ◆ 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.

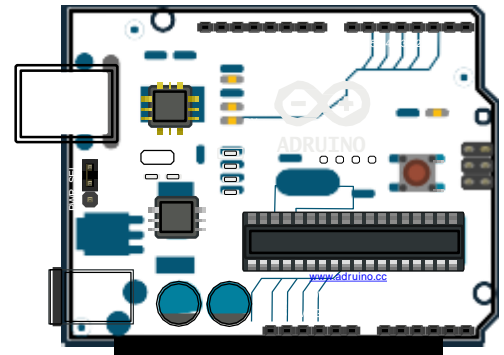


Code Generator

- ◆ Designed to support as many targets as possible.
- ◆ Supports few Modelica constructs.
- ◆ Focuses on generating good code with small footprint.
- ◆ Unsupported constructs such as 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.



SBHS controller using MDD and the new code generator

