www.designcps.com

Model Interpretation for an AUTOSAR compliant Engine Control Function

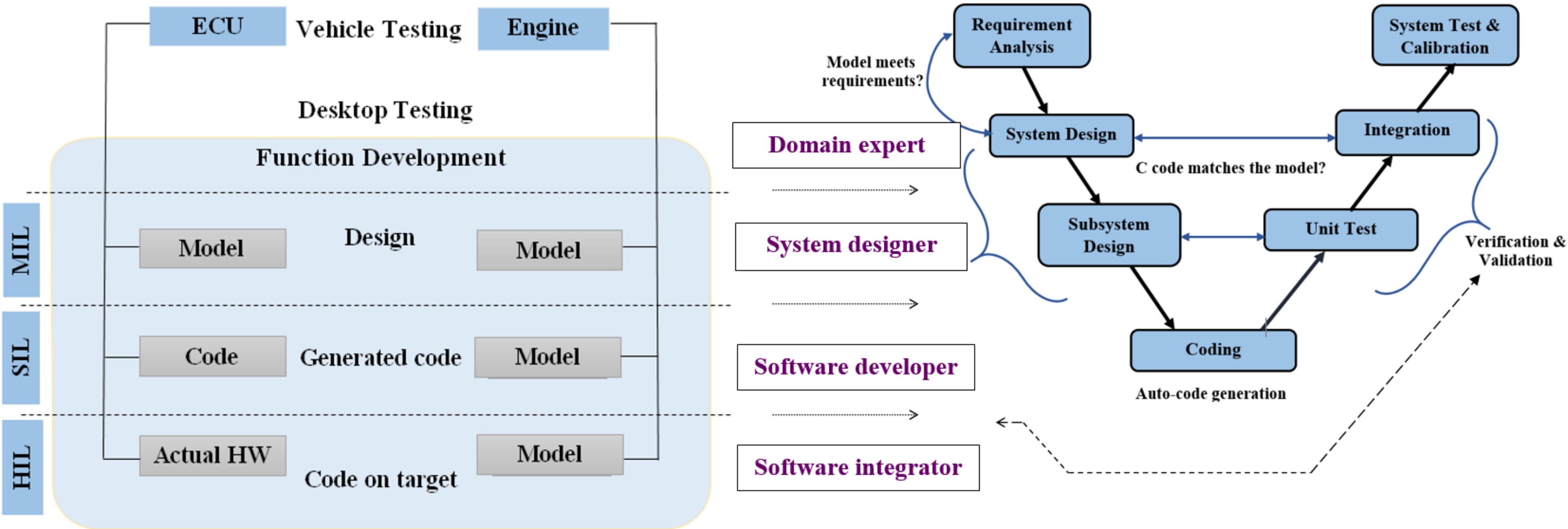
Sakthivel M. SUNDHARAM, Sebastian ALTMEYER, Nicolas NAVET,
University of Luxembourg

Fonds National de la Recherche Luxembourg

(WATERS 2016)
Toulouse, France, July 05, 2016

UNIVERSITÉ DU LUXEMBOURG

State of the art Practice



Model as the main artifact to develop the embedded software

Generative MBD – e.g., MLSL , ASCET-MD etc.

Code, other artifacts automatically generated from model

Code → binary → run on target hardware

Interpreted MBD

- Direct interpretation of design models using *interpretation engine* running on top of target
- No (optional) code, other artifacts generated
- No commercially available interpreted MBDs
- So not practiced in industrial embedded software development life cycle
- But interpretation-based runtime environments are proven (track-record) to be applied

Paper discusses…

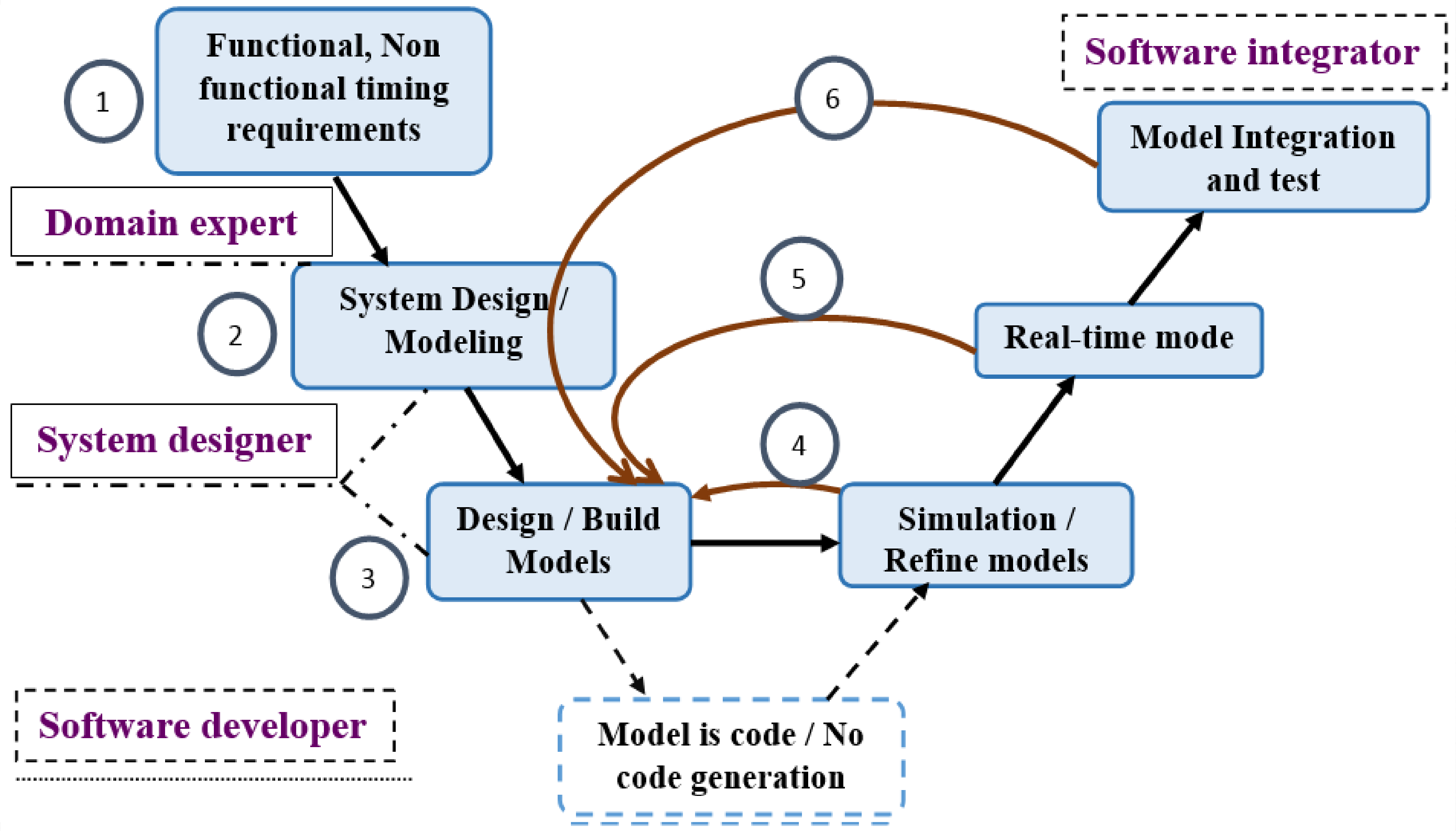
- ***Interpreted MBD*** to an industry case study to investigate it’s applicability to embedded software development cycle

  Interpreted MBD based embedded software development life cycle (proposal)

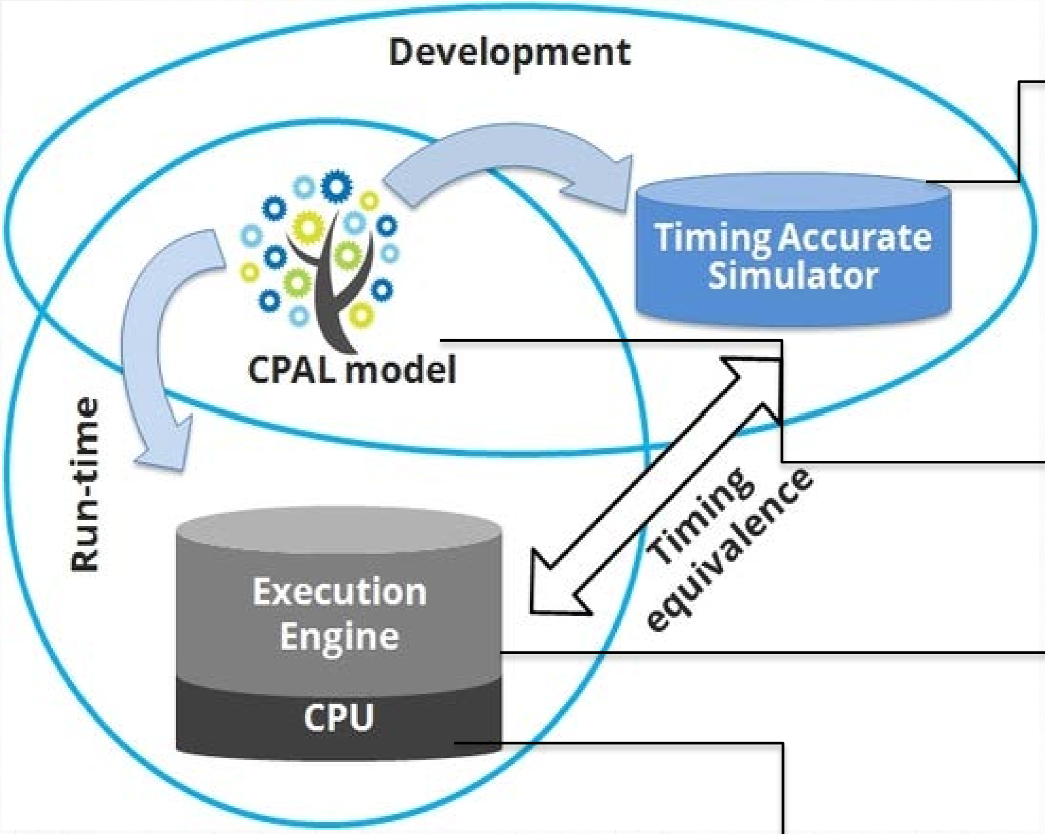
- Exploring the ***theoretical benefits*** of model interpretation with a industrial experiment

  Observations on productivity, simplicity, and performance (discussions)

Lean Development Cycle



CPAL* - an Interpreted MBD



- ✓ execute model in simulation mode or real-time mode
- ✓ timing annotation can be introduced in simulation
- ✓ simulation reflects real-time

- ✓ platform independent
- ✓ model "once"

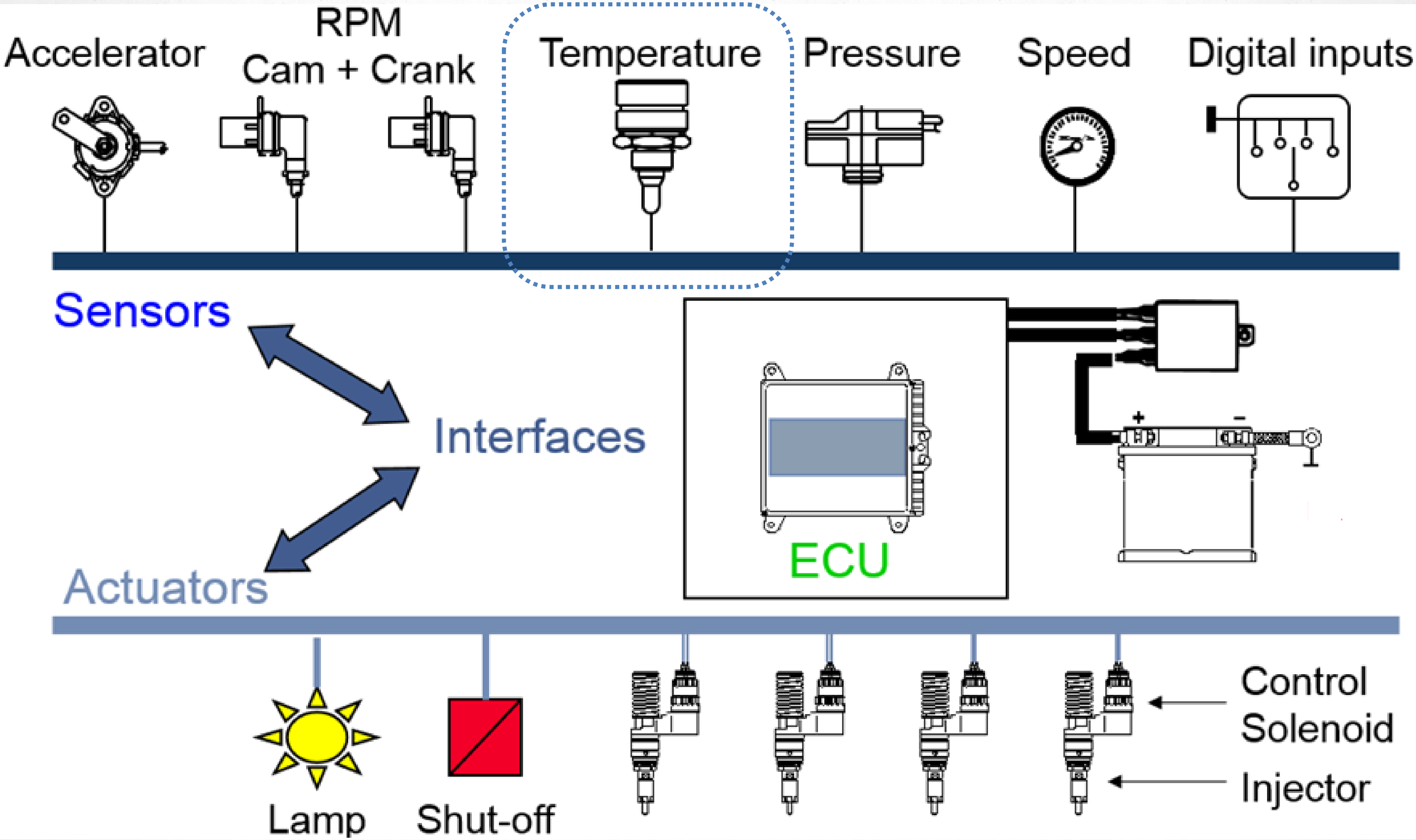
- ✓ platform dependent
- ✓ executes the model on target hardware with or without OS (BMMI)#

- ✓ Raspi, FRDM, ARM Cortex …

*** CPAL – Cyber Physical Action Language**

 # BMMI – Bare Metal Model Interpretation

Engine Control System



Engine Subsystems

Mechanical & ECU

➔ Air System

Air-Filter, Intake Manifold, Turbo-Charger / Super-Charger

Air Mass Sensor, Manifold Pressure/Temperature Sensor, Electronic Throttle

➔ Fuel Injection System

Fuel Tank, Fuel Filter, Fuel Pump, Injector

Fuel Tank Pressure Sensor, Fuel Pump, Electrical Injector, Canister Purge Valve, Fuel Rail Pressure Sensor, Rail Pressure control valve

➔ Cooling System

Coolant (Water) Reservoir, Water Pump, Radiator, Fan

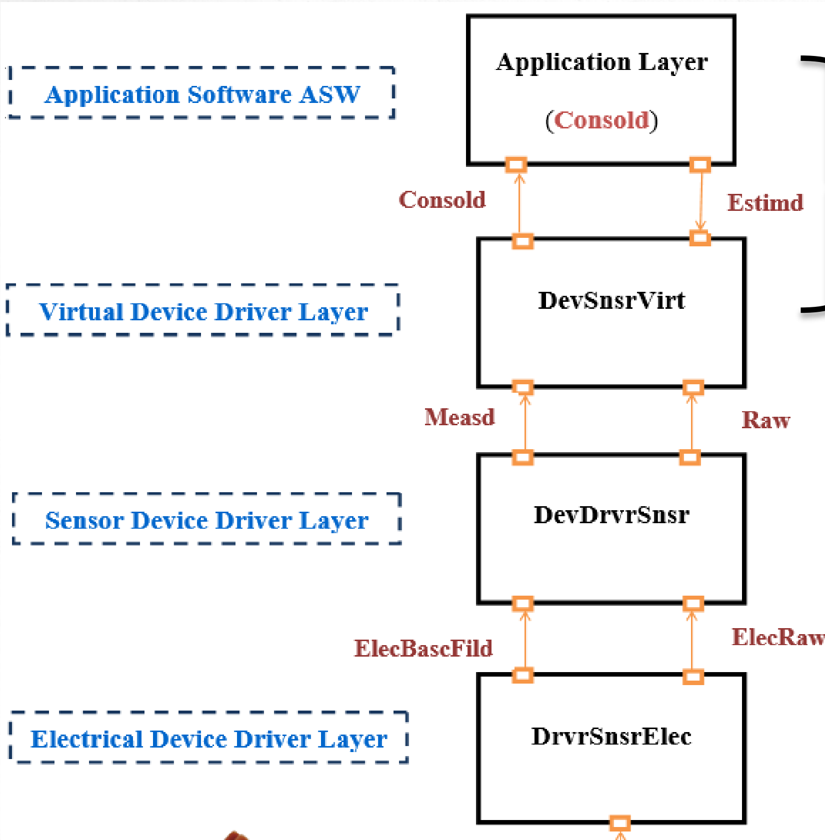
Electrical Water Pump, Electrical Fan, Water Temperature Sensor, Flow Control Valves

➔ Exhaust System

Exhaust Manifold, Exhaust Pipe, Exhaust Muffler, Catalytic Converter

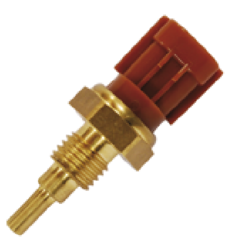
Exhaust Temperature Sensor, Lambda Sensor, NOx Sensor, EGR Valve, Secondary Air Pump, Secondary Air Valve

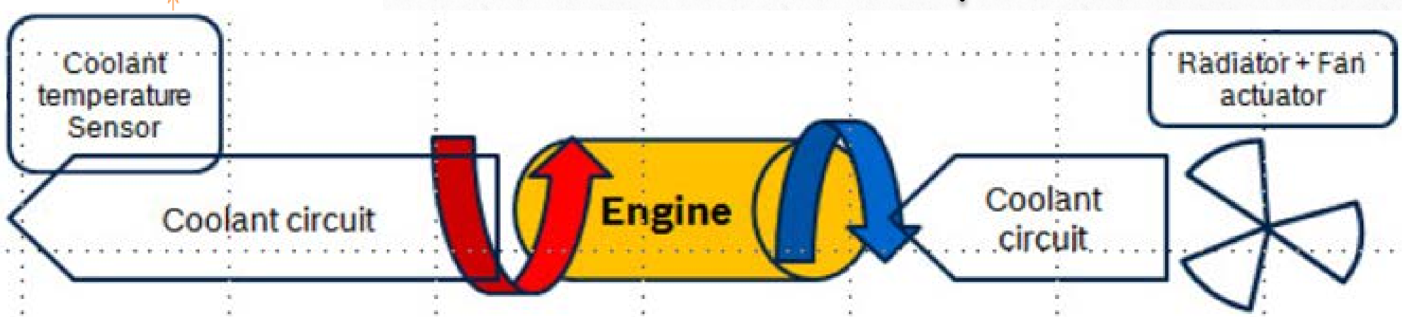
AUTOSAR Case Study



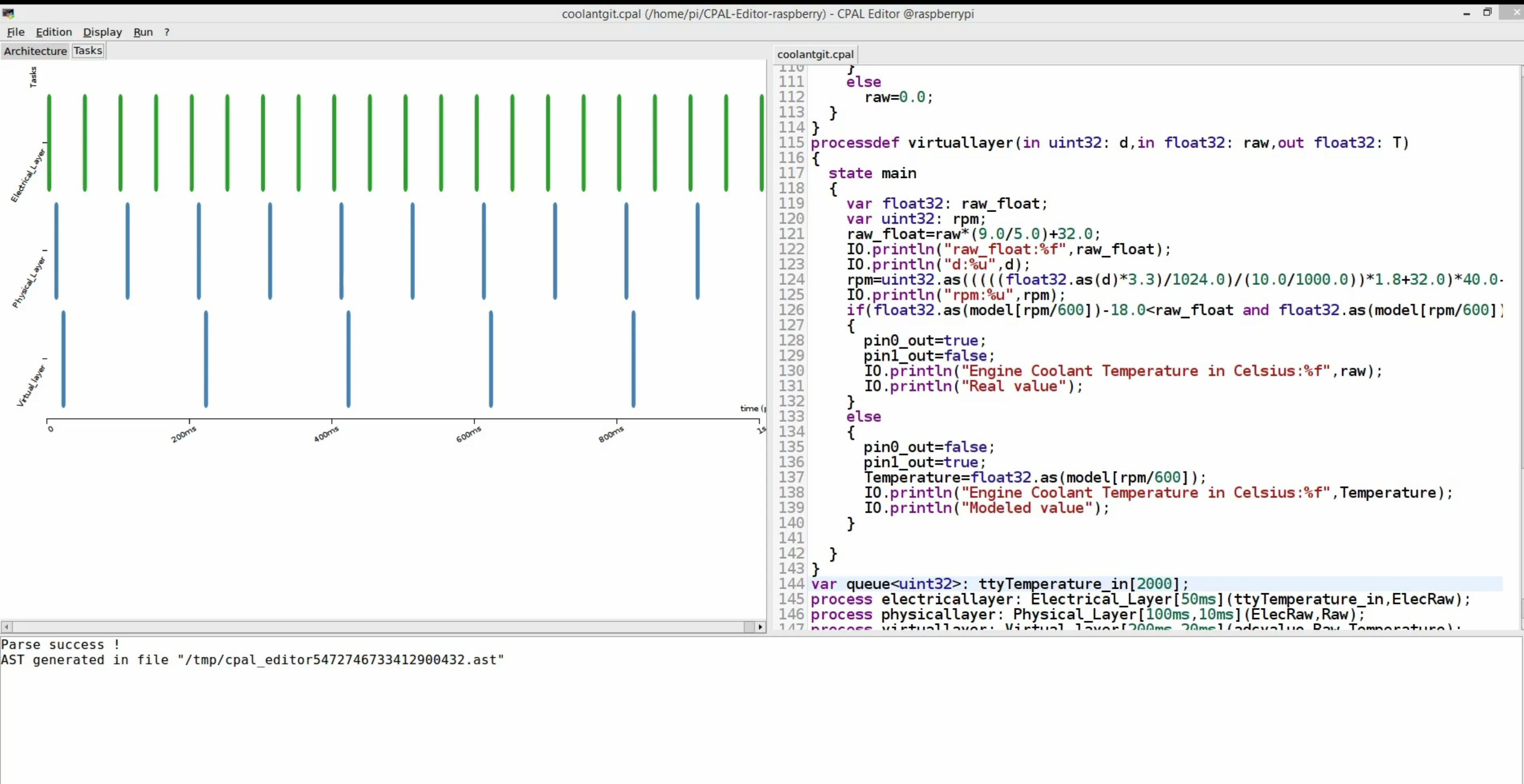
Requirement **R1**: When the difference between sensed value (*Measd*) and estimated value (*Estimd*) from application is 18 deg C, application to consider *Estimd*

Requirement **R2**: When the engine temperature changes, it has to be controlled below *200* deg C *(threshold)* value within t seconds





Let's have a look



Observation # 1 – Early stage execution

Timing accurate simulation
and real-time execution

Finding failure in model is easier (No code)

```
pi@raspberrypi ~/cpal/coolant $ sudo ./cpal_interpreter_raspberry -r git.ast
  => Digital pin found: 0, output
  => Digital pin found: 1, output
  => a serial read_only /dev/ttyTemperature
[0.000000000000:ASSIGN] Assign pin0_out new value: false
[0.000000000000:ASSIGN] Assign pin1_out new value: false
[0.000000000000:ASSIGN] Assign flag new value: 0
[0.000000000000:ASSIGN] Assign adcvalue new value: 0
[0.000000000000:ASSIGN] Assign modelvalue new value: 0
[0.000000000000:ASSIGN] Assign digit new value: 0
[0.000000000000:ASSIGN] Assign mode new value: 0
[0.000000000000:ASSIGN] Assign ElecRaw new value: 0.000000
[0.000000000000:ASSIGN] Assign Raw new value: 0.000000
[0.000000000000:ASSIGN] Assign Temperature new value: 0.000000
[0.000000000000:ASSIGN] Assign model new value: {60,65,70,75,80,85,90,95,100,105
,110,115,120,125,130,135,140,145,150,155,160,165,170,175,180,185,190,195,200,205
,210,215,220,225,230,235}
[0.000000000000:ASSIGN] Assign ttyTemperature_in new value: {}
>> step
[0.000000000000:STATE] process "electricallayer", instance "Electrical_Layer", s
tate "Main"
>> step
[0.010000000000:STATE] process "physicallayer", instance "Physical_Layer", state
 "main"
[0.010000000000:ASSIGN] Assign Raw new value: 0.000000
>> step
[0.020000000000:STATE] process "virtuallayer", instance "Virtual_layer", state "main"
```

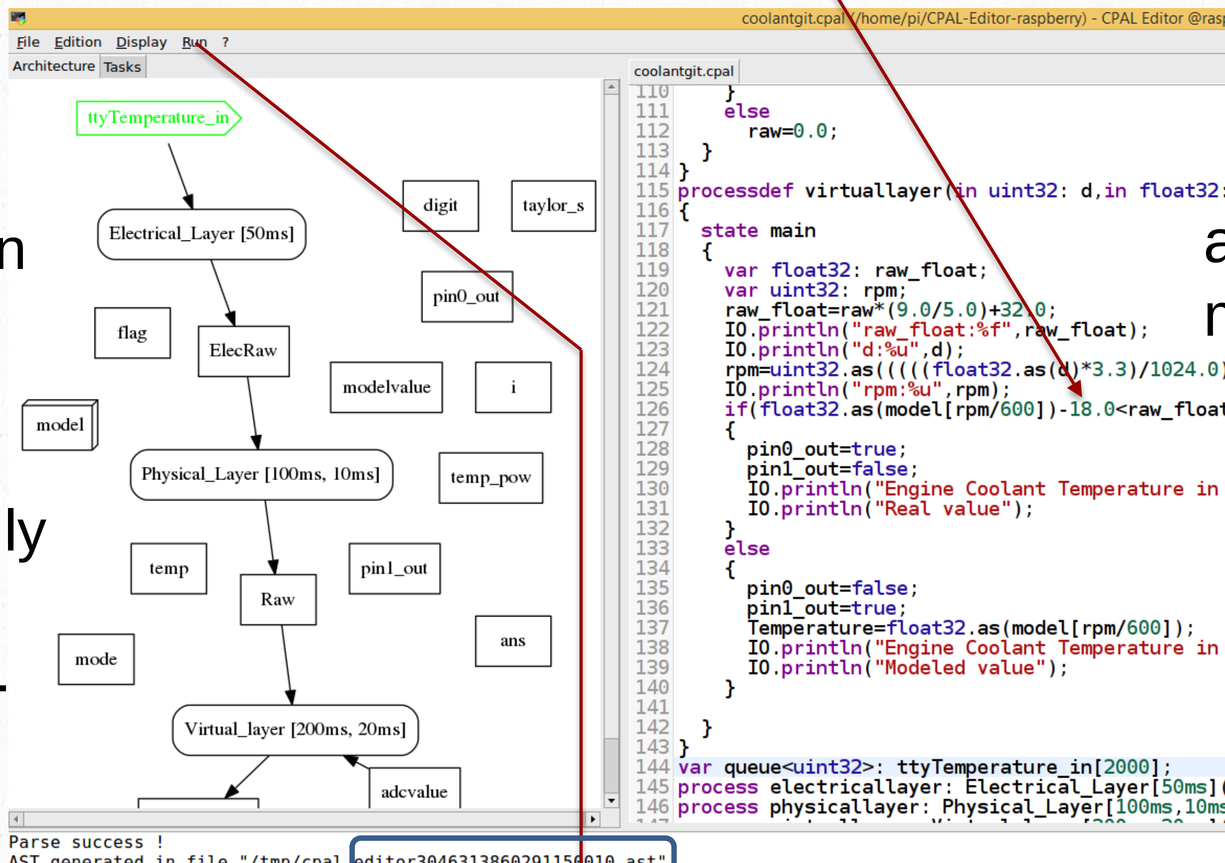
No need of tracing from code to model when failure occurs

Step by step execution – functional verification and model debugging

2 – Requirement change is easier

say **R1** (slide #9) is requested
to be changed ~~18~~ to 12 deg C

No code
No compilation
No linking
No binary
Executable model is readily available with change for **R1**

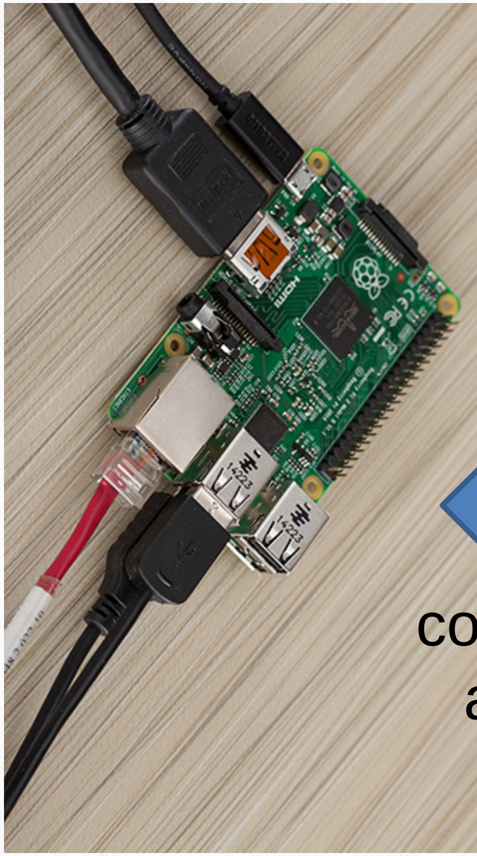


adapt the model to **R1**

Instantaneous execution of change requested

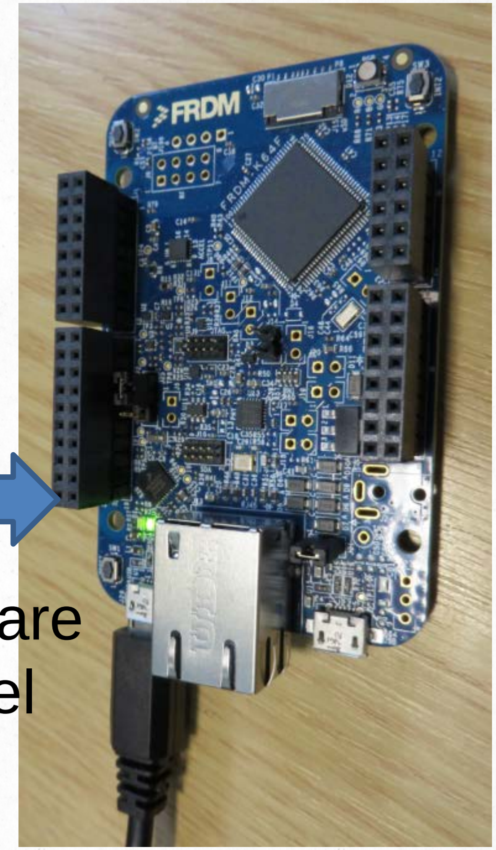
3 – Hardware Independence

CPAL model is readily portable to any hardware



coolantgit.cpal

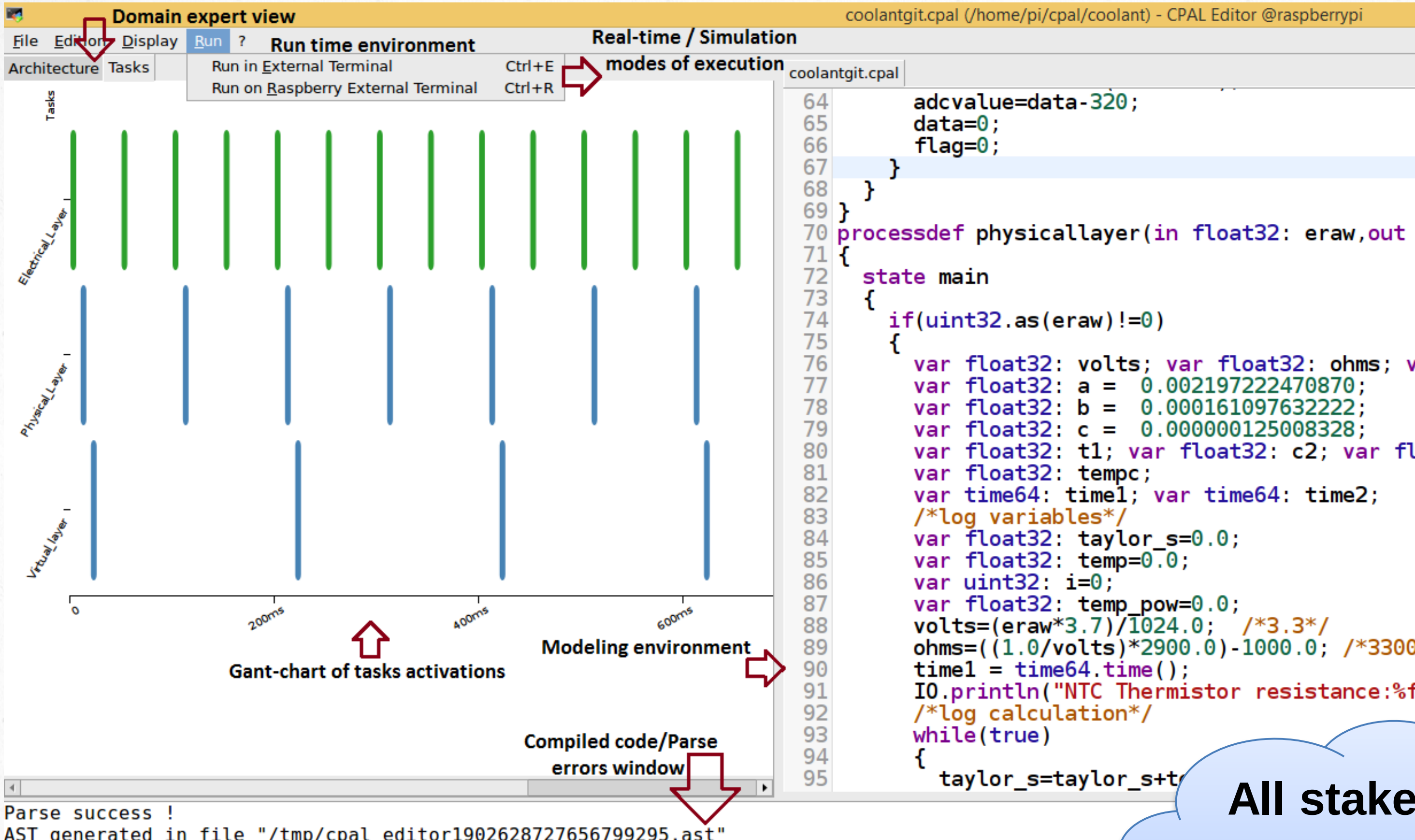
complexity of hardware abstracted in model



Interpretation engine to be adapted to HW - similar to code-generator switch to a new HW

4 – Design exploration

Functional architecture of the system – Domain expert view



Model

Developer view

All stake-holders are connected seamlessly

Scheduling of processes during simulation – timing analysis view

Our thoughts on Low-lights / Next steps

- Code generation is standard practice
- Model interpretation is **slower** than code executed – **Still**…

  calling binary code(computation-intensive portions) from interpreted code

  design phase – model interpretation to benefit productivity / easier verifiability aspects
- Production phase – Code generation to benefit faster execution capability
- Interpretation and code generation are often seen as two alternatives, not as a **continuum**