Model Interpretation for an AUTOSAR compliant Engine Control Function

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

(WATTERS 2016)
Toulouse, France, July 05, 2016
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 $\rightarrow$ binary $\rightarrow$ 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 its applicability to embedded software development cycle

  Interpreted MBD based embedded software development life cycle (proposal)

• Exploring the **theoretical benefits** of model interpretation with an industrial experiment

  Observations on productivity, simplicity, and performance (discussions)
Lean Development Cycle

1. Functional, Non-functional timing requirements
   - Domain expert

2. System Design/Modeling
   - System designer

3. Design/Build Models
   - Software developer

4. Simulation/Refine models
   - Model is code/No code generation

5. Real-time mode

6. Software integrator
   - Model Integration and test
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

Sensors

Accelerator
RPM Cam + Crank
Temperature
Pressure
Speed
Digital inputs

Interfaces

Actuators

ECU

Lamp
Shut-off

Control Solenoid
Injector
Engine Subsystems

➔ **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
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

```plaintext
else
  raw=0.0;
}

processdef virtualLayer(in uint32: d, in float32: raw, out float32: T)
{
  state main
  {
    var float32: raw_float;
    var uint32: rpm;
    raw_float=raw*(9.0/5.0)+32.0;
    IO.println("raw_float=%f",raw_float);
    IO.println("d=%d",d);
    rpm=uint32.as(((float32.as(d)*3.3)/1024.0)/(10.8/1000.0))*.18+32.0*40.0;
    IO.println("rpm=%f",rpm);
    if(float32.as(model[rpm/600])<18.0<raw_float and float32.as(model[rpm/600])<18.0<raw_float and float32.as(model[rpm/600])<18.0<raw_float and float32.as(model[rpm/600])<18.0<raw_float and float32.as(model[rpm/600])<18.0<raw_float)
    {
      pin0_out=true;
      pin1_out=false;
      IO.println("Engine Coolant Temperature in Celsius:%f",raw);
      IO.println("Real value");
    }
    else
    {
      pin0_out=false;
      pin1_out=true;
      Temperature=float32.as(model[rpm/600]);
      IO.println("Engine Coolant Temperature in Celsius:%f",Temperature);
      IO.println("Modeled value");
    }
  }
}

var queue<uint32>: ttyTemperature_in[2000];
process electricalLayer: Electrical_Layer[50ms](ttyTemperature_in,ElecRaw);
process physicalLayer: Physical_Layer[100ms,10ms](ElecRaw,Raw);
process virtualLayer: Virtual_Layer[30ms,70ms](idealRaw,RawTemperature);
```

Parse success!
AST generated in file "/tmp/cpal_editor5472746733412980432.ast"
Observation # 1 – Early stage execution

Timing accurate simulation and real-time execution

Finding failure in model is easier (No code)

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 48 to 12 deg C

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

Instantaneous execution of change requested
CPAL model is readily portable to any hardware

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

complexity of hardware abstracted in model
# 4 – Design exploration

Functional architecture of the system – Domain expert view

Model

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