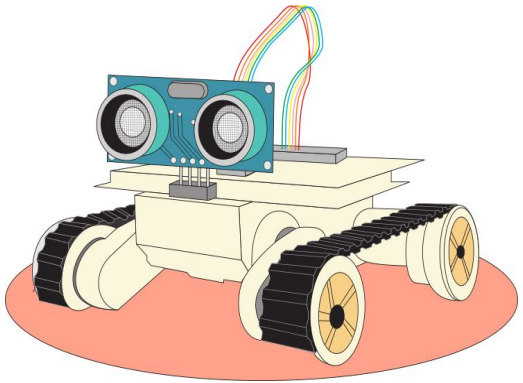
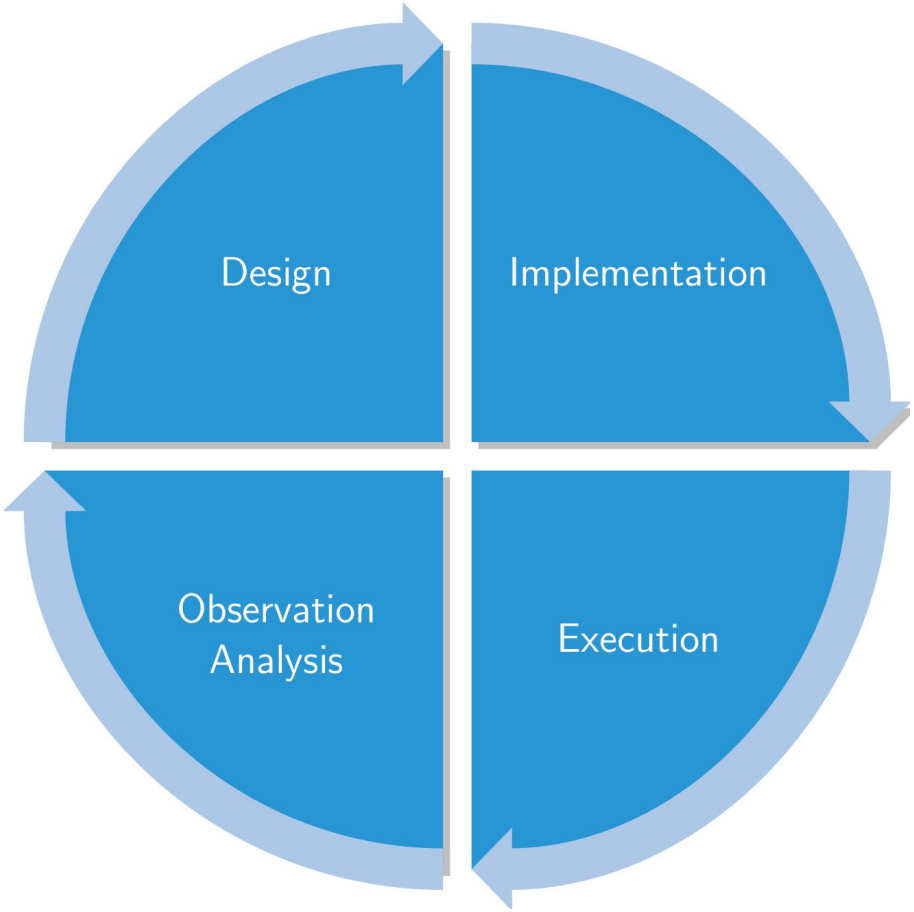
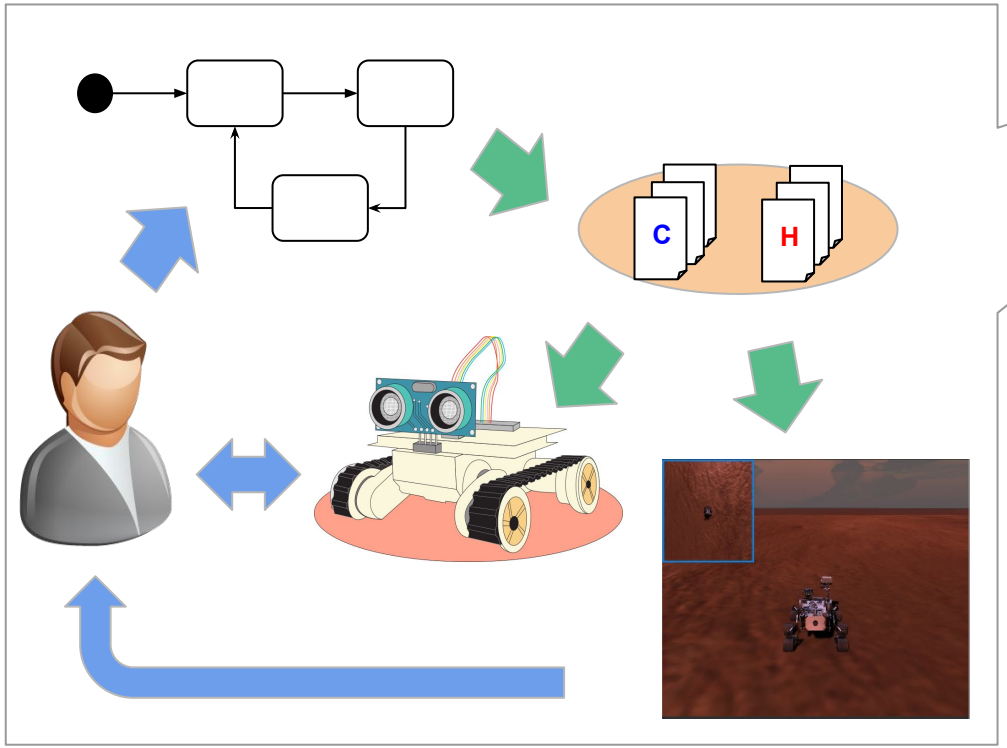
Using UML-RT and Papyrus-RT for the Design, Execution, and Observation of a Rover System

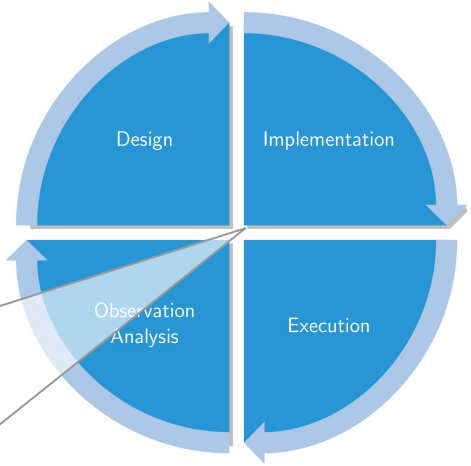


Overview

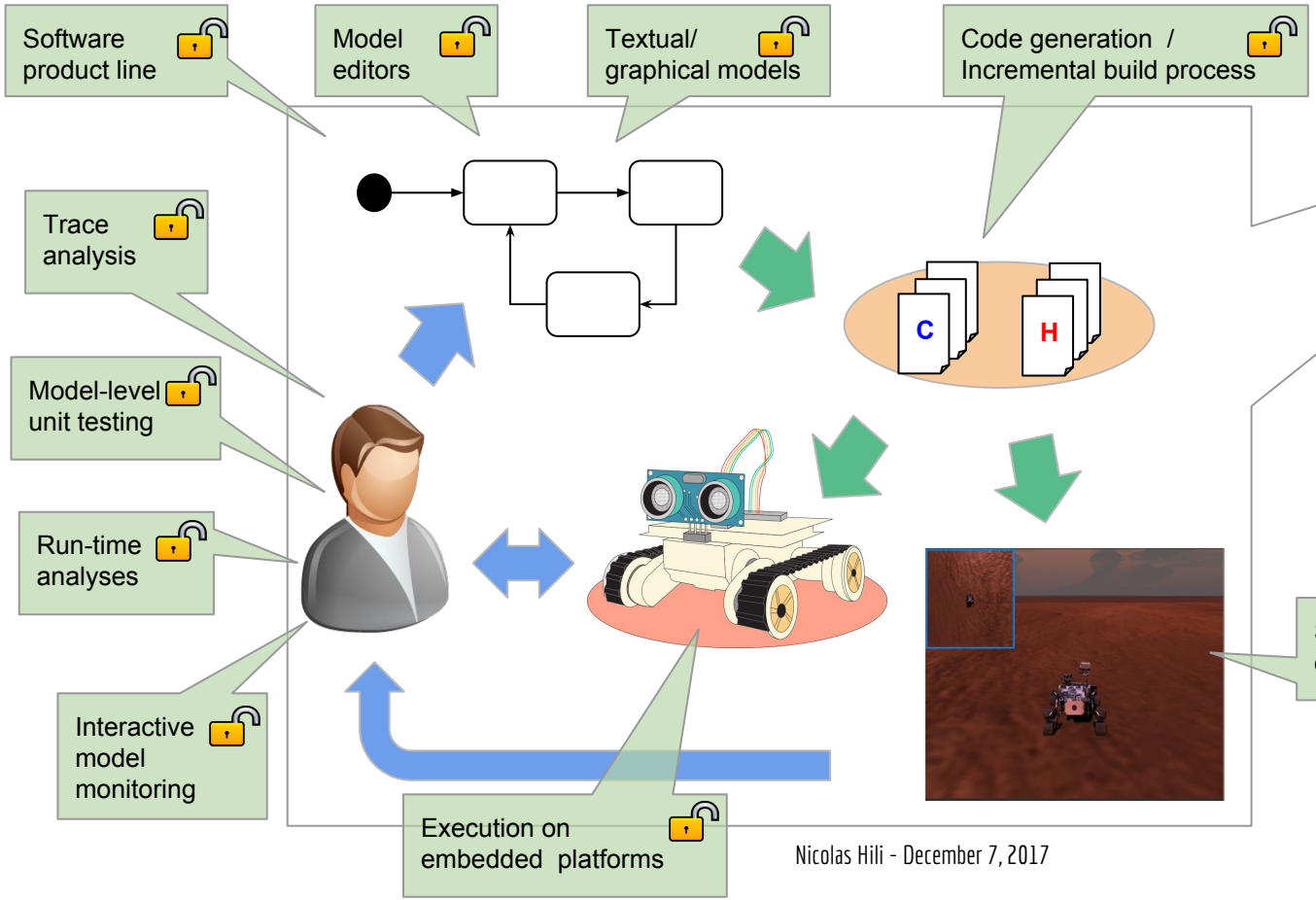


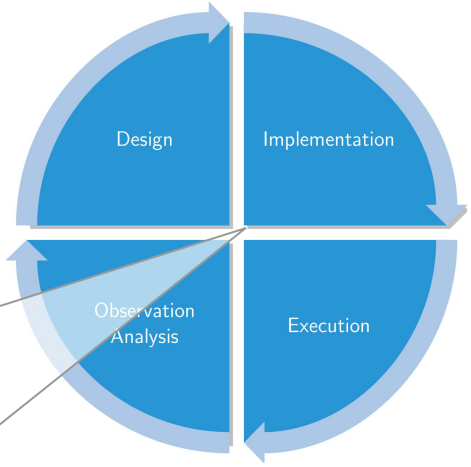
Design - Implementation - Execution - Analysis

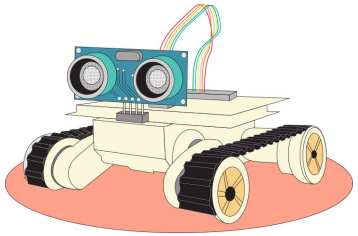




Design - Implementation - Execution - Analysis

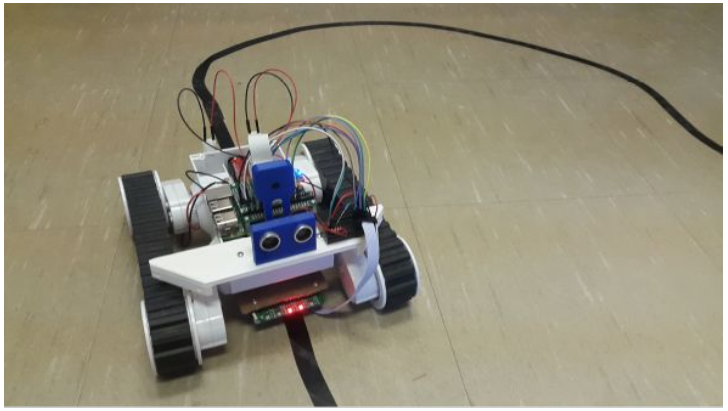


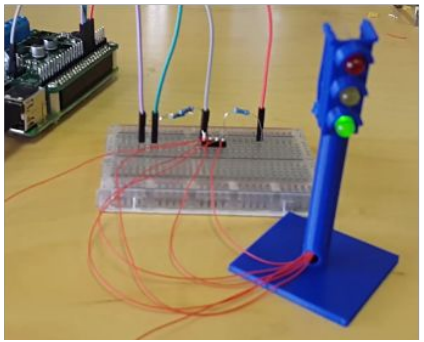




PolarSys Rover Development

- PolarSys Rover
  - Pololu Dagu Rover 5 Tracked Chassis
  - Auto-calibrating line sensor LSS05
  - Ultrasonic detection sensor SR04
  - Raspicam
  - 3D printed extensions
- Traffic Light
  - Raspberry-powered
  - 3D printed model of the traffic light





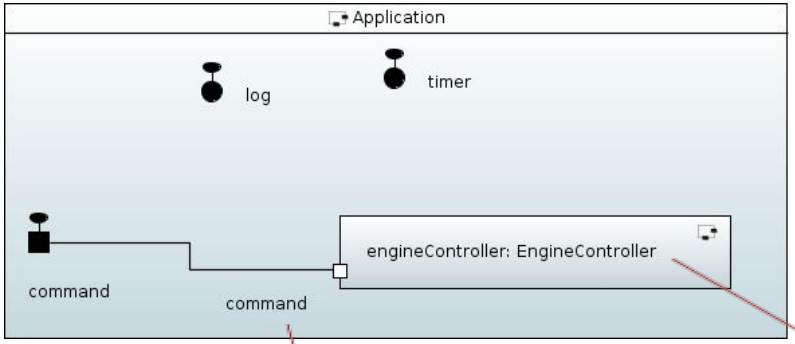
Papyrus-RT Overview

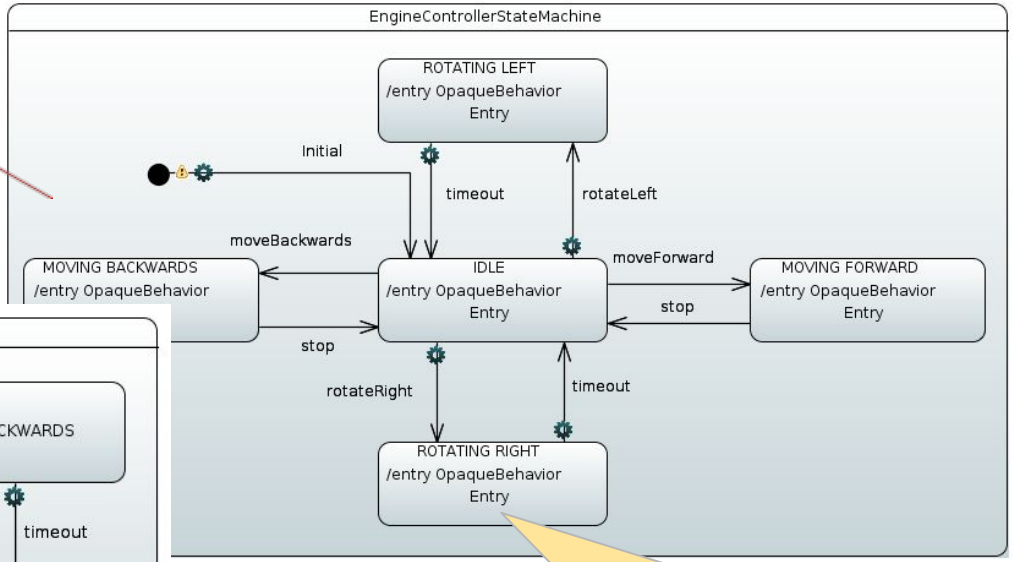
- Papyrus for Real-Time industrial-grade, complete modeling environment for the development of complex, software intensive, real-time, embedded, cyber-physical systems.
- Part of PolarSys
  - Eclipse Working Group
  - Open source for embedded systems
- Building on
  - Eclipse Modeling Framework (EMF), Xtext, Papyrus
- History
  - 2015: V0.7.0
  - March 2017: v0.9
  - Fall 2017: v1.0

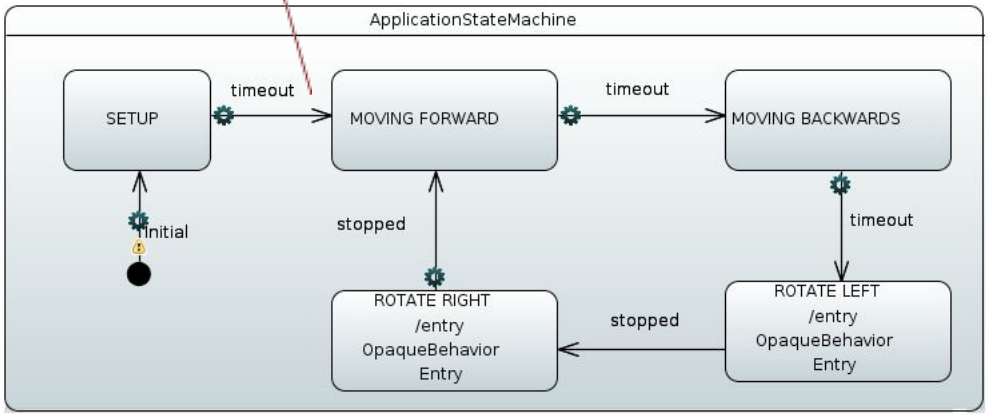
[https://wiki.eclipse.org/Papyrus-RT]

PolarSys Rover Models:

Engine Controller



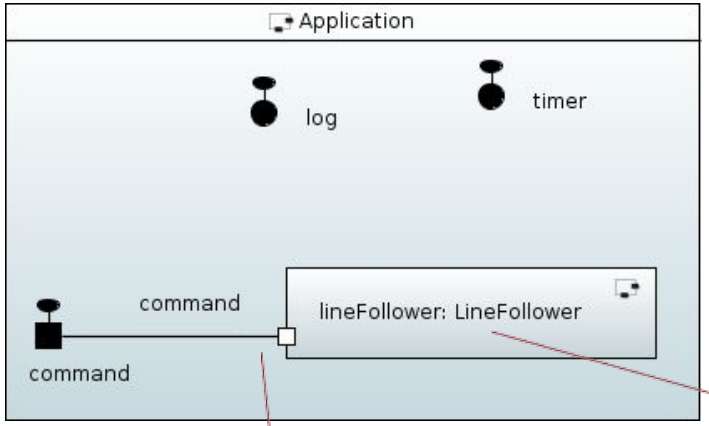


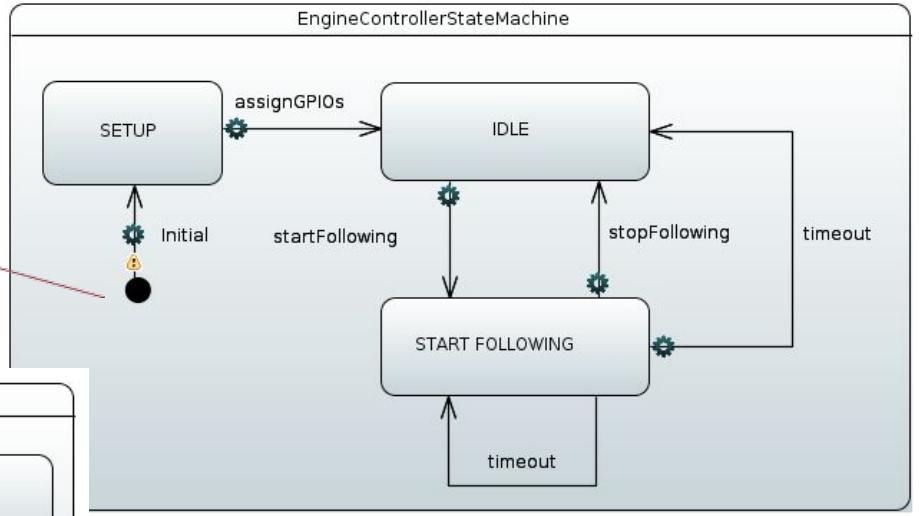


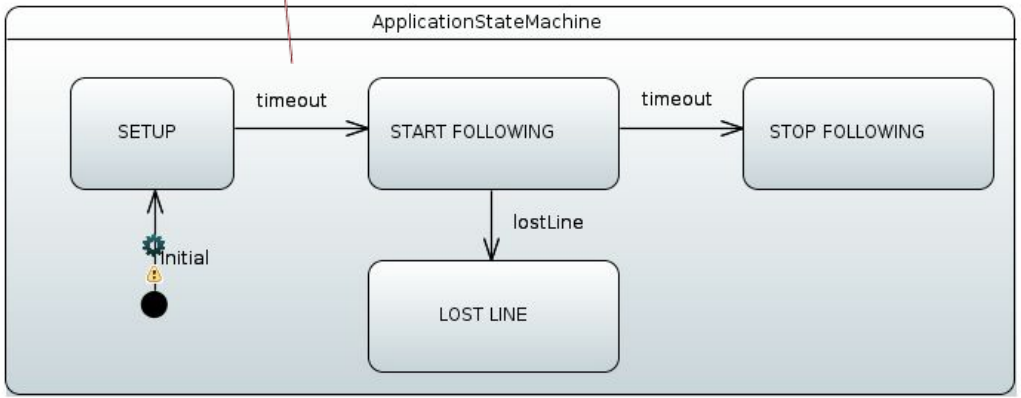

```
int s = angle/360*100;
long long m= …;
this.timerId  = timer.informIn(UMLRTTimespec(
        s,1000000*m);
...
```

PolarSys Rover Models:

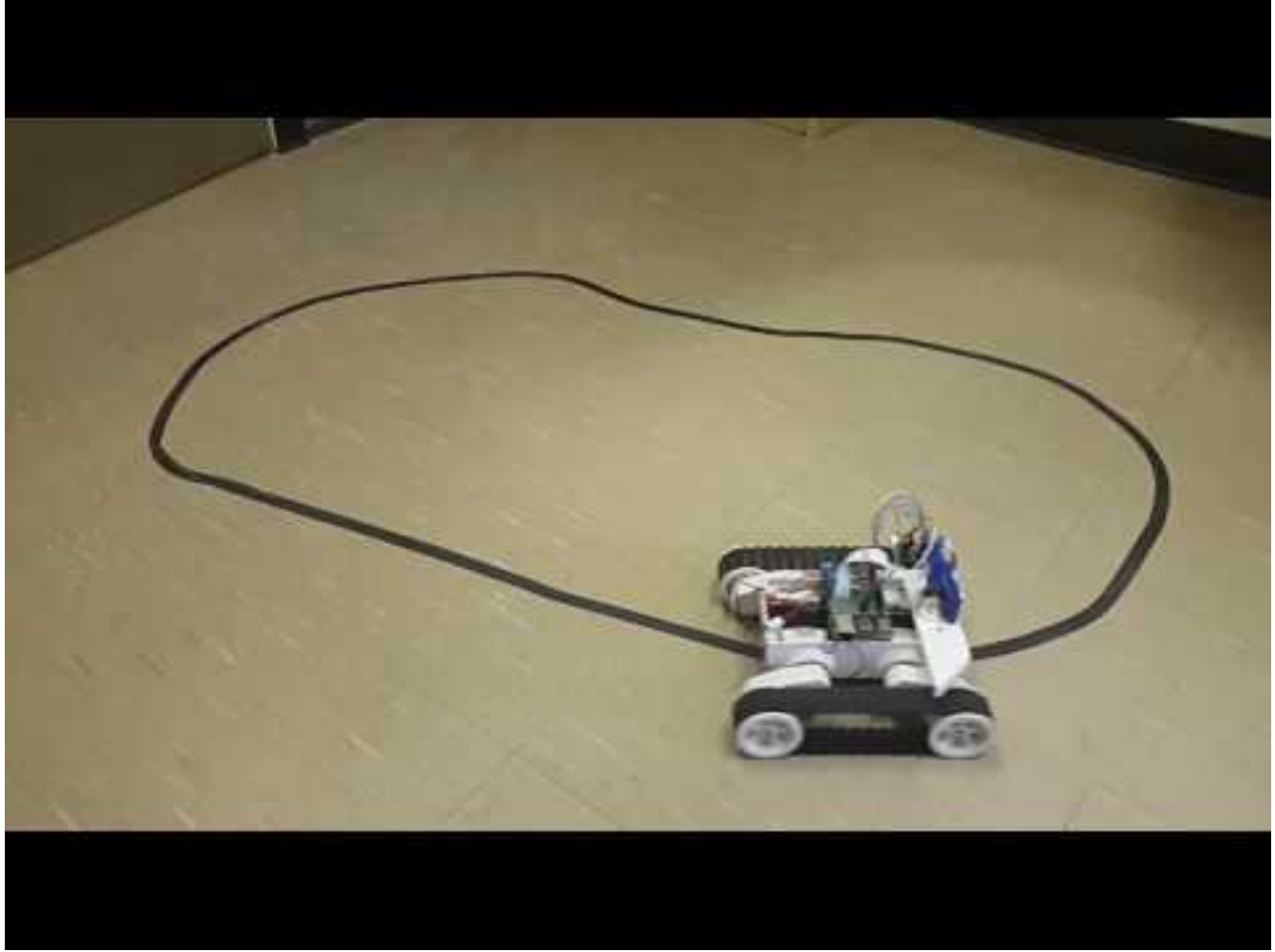
Line Follower







PolarSys Rover Execution



PolarSys Rover: feedback

Use of MDE for the Rover ☺

Writing code more intuitive at first (fast prototyping, hardware testing…)

But models really helpful when complexity is increased (state machine more intuitive than a thousand of LoC!)

Need to follow some guidelines (naming convention, structuring the model into packages…)

Modularity and reuse!

Still some limitation ☹

C++ code embedded inside the model

Make it hard to debug the model

Make it hard to customize the model for ≠ platforms

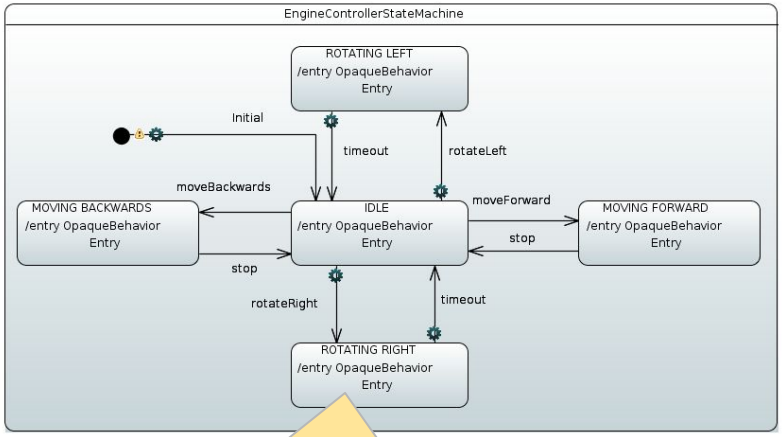
Reduce analysis capabilities

Different solutions

- ➔ (Language-independent) action semantics
- ➔ Run-time observation
- ➔ Rover product line (Sudharshan)
- ➔ Simulation (Michal)

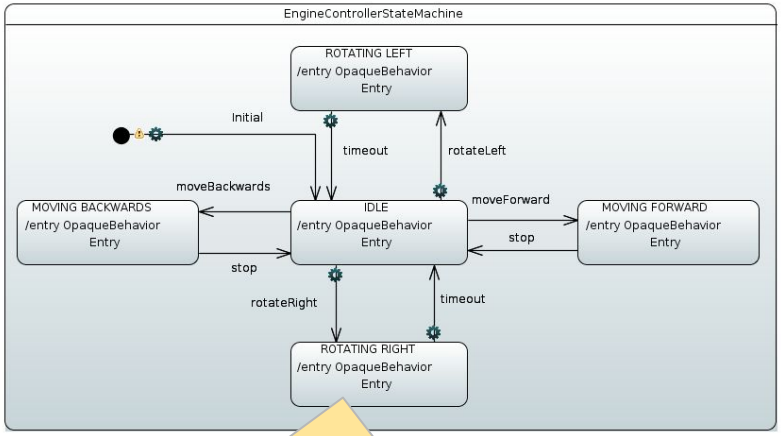
PolarSys Rover Models:

Action semantics




```
...
this.timerId  = timer.informIn(UMLRTTimespec(
    s, 1000000*m);
...
```

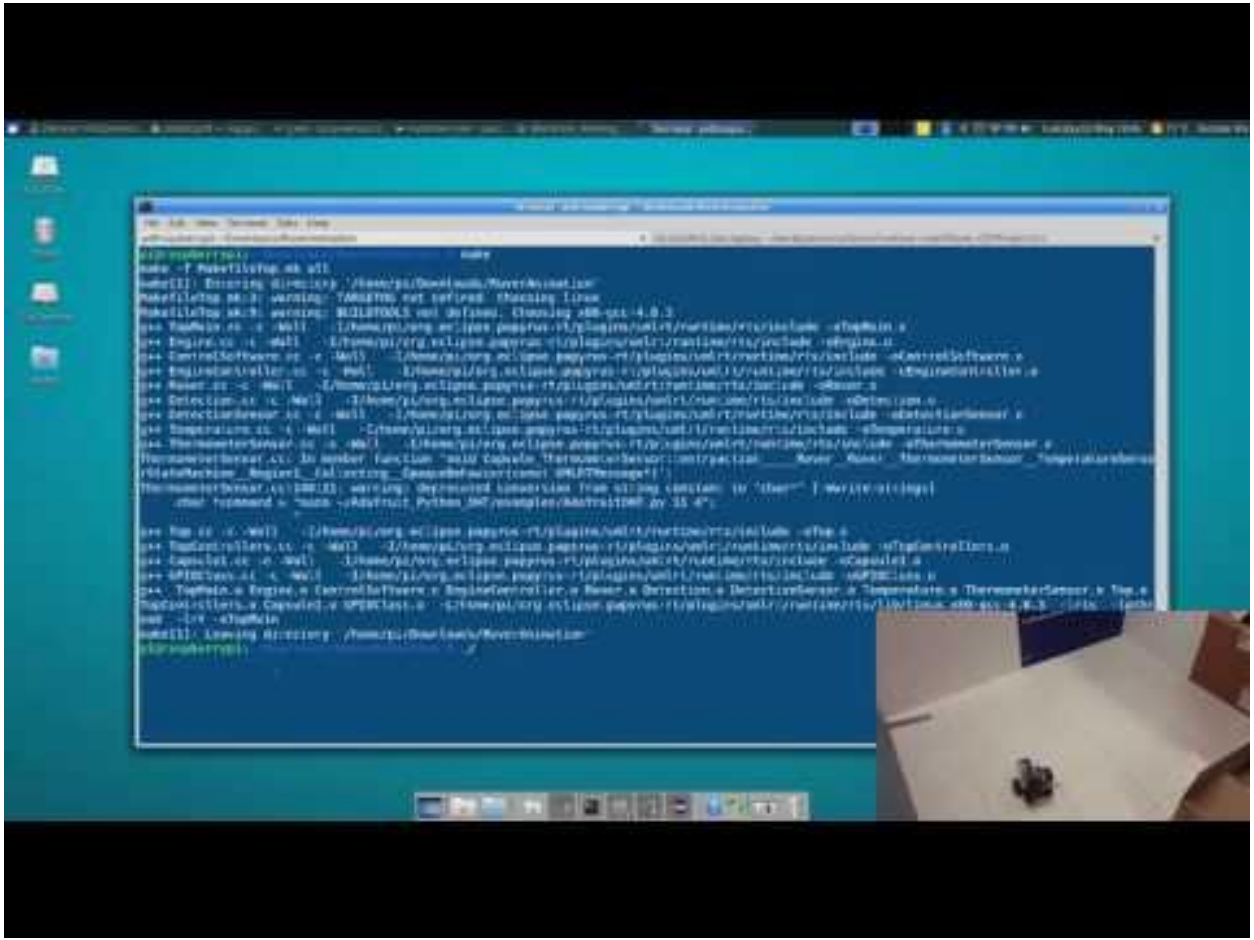
- ☹ C++ specific (Primitive types, RTS…)
- ☹ No validation/checking (errors detected at compile-time)
- ☹ No ‘smart’ features (content-assist, syntax highlighting, quick fixes)

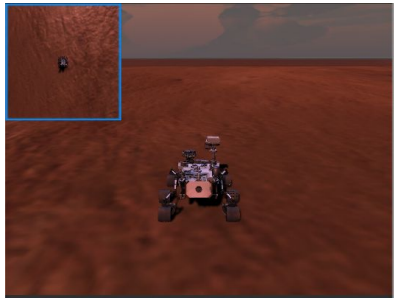



```
...
inform in s seconds  and m milliseconds;
...
```

- ☺ C++ independent
- ☺ Validation/checking (errors detected at design-time)
- ☺ ‘smart’ features

PolarSys Rover Observation...

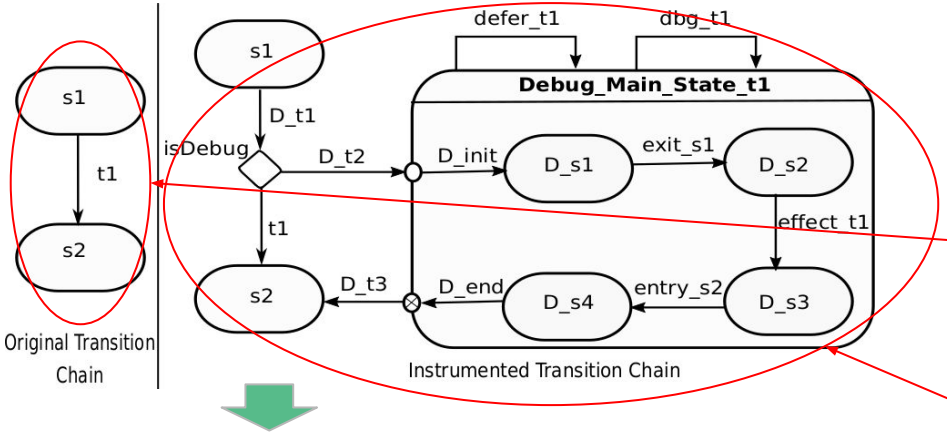


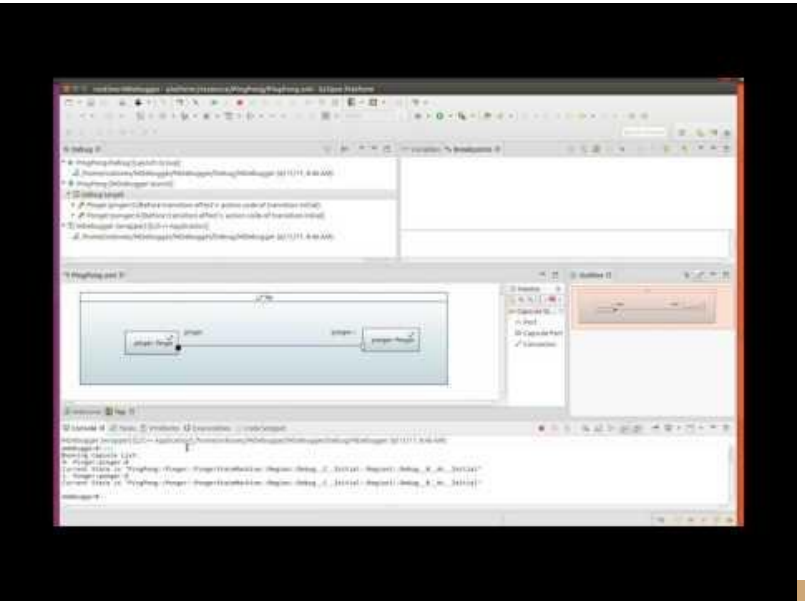


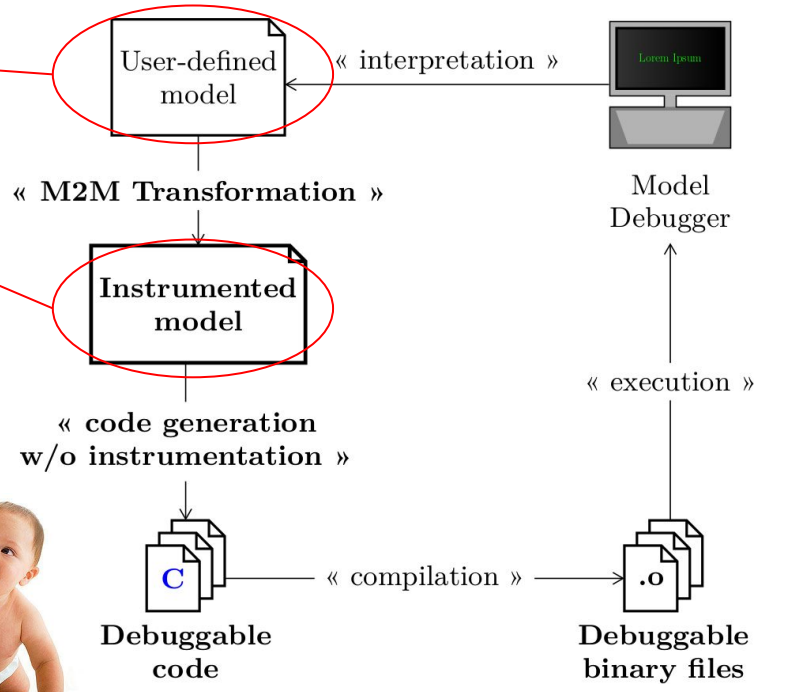
...and steering!

Interactive Model-Level Debugging

Mojtaba Bagherzadeh

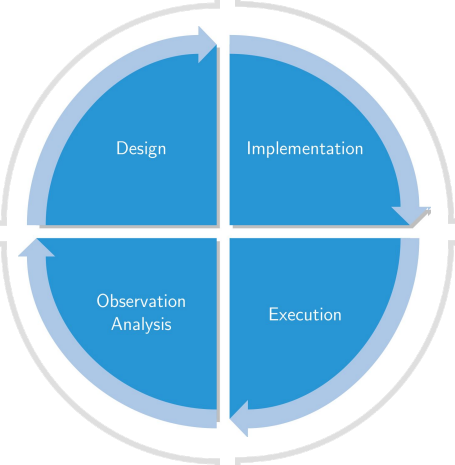






- ➔ Rover models (Harshith)
- ➔ Rover product line (Sudharshan)
- ➔ Action language for UML-RT (Nicolas)

- ➔ Generation for the Rover (Sudharshan)

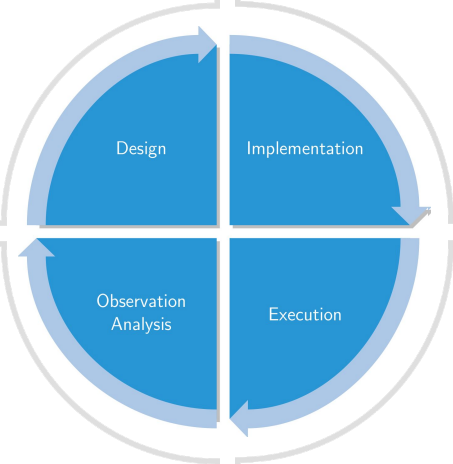


- ➔ Run-time observation and steering (Nicolas & Mojtaba)

- ➔ Cruise control (Harshith)
- ➔ Simulation (Michal)

- Rover models (Harshith)
- Rover product line (Sudharshan)
- Action language for UML-RT (Nicolas)

- Generation for the Rover (Sudharshan)
- Incremental code generation (Kanchan)
- Smart CPS/Internet of Things (Karim)



- Run-time observation and steering (Nicolas & Mojtaba)
- Unit-testing (Reza)
- Model-level debugging (Mojtaba)

- Cruise control (Harshith)
- Simulation (Michal)

Design - Implementation - Execution - Analysis

An Incremental Process