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

- Software product line
- Model editors
- Textual/graphical models
- Code generation / Incremental build process

- Trace analysis
- Model-level unit testing
- Run-time analyses
- Interactive model monitoring

- Execution on embedded platforms
- Simulation environments

Design
Implementation
Observation
Analysis
Execution

Nicolas Hili - December 7, 2017
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

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

Nicolas Hili - December 7, 2017

... 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)

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

...and steering!
Interactive Model-Level Debugging

Mojtaba Bagherzadeh

User-defined model

« M2M Transformation »

Instrumented model

« code generation w/o instrumentation »

Debuggable code

« compilation »

.o file

Debuggable binary files

Model Debugger

« execution »

« interpretation »

Nicolas Hili - December 7, 2017
- Rover models (Harshith)
- Rover product line (Sudharshan)
- Action language for UML-RT (Nicolas)

- Generation for the Rover (Sudharshan)

- Cruise control (Harshith)
- Simulation (Michal)

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