EclipseCon 2017:
Developing Robotic Applications Using Model-Driven Engineering Techniques

With UML-RT and Papyrus-RT

Nicolas Hili
June, 2017
Intro
Model-Driven Engineering (MDE)

- Improve productivity, quality, and ability to handle complexity by
  - increasing level of abstraction
    - through use of ‘models’
  - leveraging automation
    - e.g., via code generation from models, model transformation, ...
  - improving analysis capabilities
    - e.g., through constraint solving, simulation, state space exploration, ...

MDE = Abstraction + Automation + Analysis

- Inspired by use of models in engineering and science
MDE: Challenges, Opportunities

- **Challenges** [1],[2]
  - Technical: user experience, model analysis, ...
  - Social: education/training, ...

- **Opportunities**
  - Emerging eco-system: open source, standards, forums, repositories, ...
  - Abstraction, automation, and analysis will continue to be key [3]

---


Domain-specific languages (DSLs) for robotic applications

- Many different DSLs
  - RobotML,
  - SmartSoft,
  - BCM,
  - V3CMM

- Quite domain-specific:
  - Function modelling
  - Mission planning
  - Deployment modelling
Modeling Languages

- Modelica
  - Physical systems
  - Equation-based

- Simulink
  - Continuous control, DSP
  - Time-triggered dataflow

- Stateflow
  - Reactive systems
  - Discrete control
  - State-machine-based

- AADL
  - Embedded, real-time

- UML
  - Embedded, real-time

- UML MARTE
  - Embedded, real-time

- UML-RT
  - Embedded, real-time
  - State-machine-based

Examples in [Voe13, Kel08]

EGGG [Orw00]

Increasing generality

EGGG [Orw00]

Increasing domain-specificity

UML-RT for robotic applications
UML-RT: History

- Real-time OO Modeling (ROOM)
  - ObjecTime, early 1990 ties
- Major influence on UML 2
  - E.g., StructuredClassifier
- “RT subset of UML”
- Tools
  - ObjecTime Developer
  - IBM Rational RoseRT
  - IBM RSA-RTE
  - Eclipse Papyrus-RT

Goal of the Workshop

- **Inform**
  - Intro to robotic development for the PolarSys Rover
  - Intro to MD with UML-RT and Papurys-RT

- **Combine**
  - MDE for robotic applications
  - Application to the PolarSys Rover

- **Inspire**
  - We need more abstraction, automation, and analysis!
  - UML-RT
    - Small, cohesive set of concepts
    - Successful track record, but work needed on, e.g.,
      - static analysis, user experience, deployment, interpretation, testing, verification, simulation, ...
# Overview

1. Intro / MDE (10 mins) (8 slides)
2. Overview (1 min) (1 slide)
3. PolarSys Rover (15 mins) (16 slides)
4. Demo I (5 mins)
5. Hands on session (20 mins) (2 slides)
6. UML-RT: Part I (25 mins) (26 slides)
   - Core concepts
7. Demo II (10 mins) (3 slides)
8. Hands on session (20 mins) (1 slide)
9. UML-RT: Part II (10 mins) (14 slides)
   - More advanced concepts
10. Hackaton (90 mins) (3 slides)
11. Conclusion (5 mins) (2 slides)
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**
  - Raspicam-powered
  - 3D printed model of the traffic light
Raspberry Pi 3 Model B

- 1.2GHz 64-bit quad-core ARMv8 CPU
- 1GB RAM
- Built-in 802.11n Wireless LAN
- Built-in Bluetooth 4.1
- 4 USB ports
- 40 GPIO pins
- HDMI port
- Ethernet port
- Camera interface (CSI) for camera module
- Micro SD card slot
- VideoCore IV 3D graphics core

Source: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
Add-on for Raspberry PI
Can control up to two bidirectional brushed DC motors
All GPIO ports are available
8 GPIOs are used for controlling the two motors (can be re-wired)

Must be supplied with 5V to 28V
Do not power the Raspberry PI (by default)
  - A voltage regulator can be added to power the Raspberry PI
  - Otherwise, the Raspberry must be powered independently through its USB receptacle

Source: https://www.pololu.com/product/2755
Auto calibrating line follower LSS05

- Easy-to-use line follower
- Powered with 5V supply
- 5 pairs of IR transmitter and receiver
- LEDs for visual feedback
- Line detection of 1cm to 3cm
- Auto calibrate
- Dark and bright mode selection

Principle:

Source: https://www.cytron.com.my/p-lss05
Ultrasonic SR04

- Most commonly used sensor for detecting distances
- Unexpansive (less than 5$)
- Range: 2cm to 400cm
- Two GPIOs:
  - trigger (output)
  - echo (input)
- Attention: the echo pin delivers a 5V voltage

Principle:

Source: https://electrosome.com/hc-sr04-ultrasonic-sensor-raspberry-pi/
Camera module

- Camera module for Raspberry PI
- Sony IMX219 8-megapixel sensor
- Connected to the CSI port using a 15cm ribbon cable

Raspberry PI 3 Built-in applications:
- raspistill / raspistillyuv
- raspivid

Different libraries to use:
- Raspicam available for Python / C++
- OpenCV

Sources:
Accessing the Raspberry PI 3

- Common ways for accessing the Raspberry PI:
  - Using a monitor, keyboard and mouse
  - SSH
    - Command-line interface (CLI)
    - Putty for Windows
  - Virtual Network Computing (VNC)
    - VNC viewer, ...
    - Requires a Desktop manager to be installed
  - File Transfer Protocol (FTP)
    - Raspberry PI 3: pure-ftp, ...
    - Client: FileZilla, ...
- SSH, VNC, and FTP are disabled by default on Raspberry PI. To enable them:
  \$ raspi-config
- Also available for this workshop:
  - TTy.js
Connecting your Raspberry PI 3

- Connecting your Raspberry PI 3:
  - Ethernet cable
  - Built-in Wi-fi
    - Wicd applet / Wicd-curse
    - NetworkManager
    - Configuring the Wifi network:
      - `/etc/wpa_supplicant/wpa_supplicant.conf`
  - Discovering your Raspberry PI 3:
    - `$ nmap 192.168.1.0/24`
- Turning your Raspberry PI 3 into a Wi-fi hotspot
  - Hostapd, …

```
... network={
  ssid="MySSID"
  key_mgmt=WPA-EAP
  Identity="...
  Password="...
  id_str="home"
}
...
```

Nmap scan report for line-follower-rover (192.168.1.36)
Host is up (0.010s latency).
Not shown: 997 closed ports
PORT    STATE SERVICE
21/tcp  open  ftp
22/tcp  open  ssh
5900/tcp open  vnc
Your first Application

- Turning a LED on
  - Led connected via a breadboard
  - Different ways for accessing the GPIO
    - System calls:
      
      ```
      echo 11 > /sys/class/gpio/export
      echo out > /sys/class/gpio/gpio11/direction
      echo 1 > /sys/class/gpio/gpio11/value
      ```
    - GPIOClass (no longer maintained ?)
    - Gpio utility (provided by the wiringPI library)

Source: http://wiringpi.com/examples/blink/
WiringPi library

- Provide an easy-to-use access to the GPIOs of the Raspberry PI
- Compatible with Raspberry PI 1, 2, 3 model A and B
- Provides a gpio utility
- Initially developed for C/C++, but some wrappers exist in Python, Java, ...
- Advanced features:
  - Timer, interrupts, delays
  - Support Pulse-Width Modulation (PWM)
    - SoftPWM
  - I2C, SPI libraries
  - ...

Source: http://wiringpi.com
An easy way for testing your system

Different functions:

- `gpio readall`
  - Display a table of GPIO mode and value
- `gpio mode 2 output`
  - Set the mode of GPIO #2 to output
- `gpio write 2 1`
  - Set the GPIO #2 to high

WiringPI uses its own mapping!

- Portability among the different versions of Raspberry PI
- May cause confusion
- Still possible to use the Raspberry PI GPIO pin numbers
```c
#include <wiringPi.h>
int main (void)
{
    wiringPiSetup () ;

    pinMode (0, OUTPUT) ;
    for (;;)
    {
        digitalWrite (0, HIGH) ; delay (500) ;
        digitalWrite (0, LOW) ; delay (500) ;
    }
    return 0 ;
}
```

To compile:

```
$ gcc -Wall -o traffic-light traffic-light.c -lwiringPi
```
```c
#include <wiringPi.h>
#define GREEN 2
#define ORANGE 3
#define RED 0

int main(void)
{
    wiringPiSetup();

    pinMode(GREEN, OUTPUT);
    pinMode(ORANGE, OUTPUT);
    pinMode(RED, OUTPUT);

    for (;;)
    {
        digitalWrite(RED, LOW);
        digitalWrite(GREEN, HIGH);
        delay(4000);

        digitalWrite(GREEN, LOW);
        digitalWrite(ORANGE, HIGH);
        delay(2000);

        digitalWrite(ORANGE, LOW);
        digitalWrite(RED, HIGH);
        delay(4000);

        return 0;
    }
}
```
int main (void) {
    wiringPiSetup () ;
    pinMode(MOTOR_LEFT_ENABLE, OUTPUT);
    pinMode(MOTOR_LEFT_DIRECTION, OUTPUT);
    pinMode(MOTOR_RIGHT_ENABLE, OUTPUT);
    pinMode(MOTOR_RIGHT_DIRECTION, OUTPUT);
    int result = 0;
    softPwmCreate (MOTOR_LEFT_PWM, 0, 100) ;
    softPwmCreate (MOTOR_RIGHT_PWM, 0, 100) ;
    digitalWrite(MOTOR_LEFT_DIRECTION, 0);
    digitalWrite(MOTOR_RIGHT_DIRECTION, 0);
    for (;;)
    {
        delay (5000) ;       // mS
        digitalWrite(MOTOR_LEFT_ENABLE, 1);
        softPwmWrite(MOTOR_LEFT_PWM, 100) ;
        digitalWrite(MOTOR_RIGHT_ENABLE, 1);
        softPwmWrite(MOTOR_RIGHT_PWM, 100) ;
        printf("Accelerating\n");
        delay (5000) ;
        digitalWrite(MOTOR_LEFT_ENABLE, 0);
        softPwmWrite(MOTOR_LEFT_PWM, 0) ;
        digitalWrite(MOTOR_RIGHT_ENABLE, 0);
        softPwmWrite(MOTOR_RIGHT_PWM, 0) ;
        printf("Stopping\n");
    }
    return 0 ;
}
Line Follower

```c
int main (void)
{
    wiringPiSetup () ;

    //LSS05 Auto-Calibrating Line Sensor Pin Setup
    pinMode(LeftSen,INPUT);
    pinMode(LeftMSen,INPUT);
    pinMode(MidSen,INPUT);
    pinMode(RightMSen,INPUT);
    pinMode(RightSen,INPUT);

    for (; ;)
    {
        int leftSen = digitalRead(LeftSen);
        int leftMSen = digitalRead(LeftMSen);
        int midSen = digitalRead(MidSen);
        int rightMSen = digitalRead(RightMSen);
        int rightSen = digitalRead(RightSen);

        if(leftSen == 0 && leftMSen == 0 && midSen == 1 && rightMSen == 0 && rightSen == 0) {
            softPwmWrite(MOTOR_RIGHT_PWM, 80) ;
            softPwmWrite(MOTOR_LEFT_PWM, 80) ;
        }
        else if(leftSen == 0 && leftMSen == 1 && midSen == 1 && rightMSen == 0 && rightSen == 0) {
            softPwmWrite(MOTOR_RIGHT_PWM, 30) ;
            softPwmWrite(MOTOR_LEFT_PWM, 80) ;
        }
        ...
        delay(200);
    }
    return 0 ;
}
```

Reading GPIO value
Resources and References

- **Links**
  - PSysRoverInitialContrib (by Gaël Blondelle):
    https://github.com/gaelblondelle/PSysRoverInitialContrib/tree/master/documentation/c_getting_started
  - WiringPi: http://wiringpi.com
  - How to use GPIOs on Raspberry:
    https://sites.google.com/site/semilleroadt/raspberry-pi-tutorials/gpio
  - Ultrasonic detection sensor SR04:
    https://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi
  - Raspbian + OpenCV:
  - Guide Raspbian Lite with PIXEL/LXDE/XFCE/Mate/i3 gui:
  - VNC: https://www.raspberrypi.org/documentation/remote-access/vnc/
  - FTP: https://www.raspberrypi.org/documentation/remote-access/ftp.md
Overview

1. Intro / MDE (10 mins) (8 slides)
2. Overview (1 min) (1 slide)
3. PolarSys Rover (15 mins) (16 slides)
4. Demo I (5 mins)
5. Hands on session (20 mins) (2 slides)
6. UML-RT: Part I (25 mins) (26 slides)
   • Core concepts
7. Demo II (10 mins) (3 slides)
8. Hands on session (20 mins) (1 slide)
9. UML-RT: Part II (10 mins) (14 slides)
   • More advanced concepts
10. Hackaton (90 mins) (3 slides)
11. Conclusion (5 mins) (2 slides)
Hand-on session

- Connecting to the Raspberry PI 3
  - SSH
    - `$ ssh pi@<Raspberry-IP>`
    - PuTTY for Windows users
  - TTy.js
  - Don’t know the IP of a Raspberry PI?

- Using the gpio utility
  - Displaying the list of GPIOs
  - Turning on a LED (Traffic Light)
  - Reading the detection sensor values (Rover)

- Using wiringPI
  - C/C++ applications folder: ~/examples/
    - trafficlight.c (Traffic light)
    - ultrasonic.c (Rover)
    - engine.c (Rover)
    - line-follower-sensor.c (Rover)

- Please get closer to the Raspberry PI you wanna test
  - When executing an application, please be sure no one else is doing it
  - Concurrent executions may cause unpredicted behaviour and damage the Raspberry PI
Hand-on session (cont’d)

Host: 192.168.1.54
hostname: traffic-light-blue
login: pi / EclipseCon2017

Host: 192.168.1.36
hostname: line-follower-rover
login: pi / EclipseCon2017

Router connection:
SSID: UMLRT-2017
WPA2: EclipseCon2017

Host: ???
hostname: traffic-light-white
login: pi / EclipseCon2017
## Overview

1. Intro / MDE (10 mins) (8 slides)
2. Overview (1 min) (1 slide)
3. PolarSys Rover (15 mins) (16 slides)
4. Demo I (5 mins)
5. Hands on session (20 mins) (2 slides)
6. UML-RT: Part I (25 mins) (26 slides)
   - Core concepts
7. Demo II (10 mins) (3 slides)
8. Hands on session (20 mins) (1 slide)
9. UML-RT: Part II (10 mins) (14 slides)
   - More advanced concepts
10. Hackaton (90 mins) (3 slides)
11. Conclusion (5 mins) (2 slides)
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]
Papyrus-RT: Installation

- Easiest: as RCP
- From web:
  - [https://eclipse.org/papyrus-rt/content/download.php](https://eclipse.org/papyrus-rt/content/download.php)
  - Download RCP for your platform
  - Extract downloaded file into a folder of your choice
- From USB stick:
  - In ‘Papyrus-RT’ folder:
    - Archive: Copy/paste, unpack
  - In ‘Models’ folder:
    - Models: Import in Papyrus-RT
  - In ‘Doc’ folder:
    - Installation instructions
Papyrus-RT: Use

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

- 2 parts
  1. Editing, building the model, generate code
  2. Compiling and running generated code
     - **Linux:** easy
       - [https://wiki.eclipse.org/Papyrus-RT/User/User_Guide/Getting_Started#Execute_the_model]
     - **MacOS:** use VirtualBox/Vagrant
     - **Windows:** use Cygwin, or VirtualBox/Vagrant
       - [https://wiki.eclipse.org/Papyrus-RT/User_Guide/Vagrant_Setup]

Modeling Environment
\[\text{generates code for}\]

Runtime System (RTS)
UML-RT: Characteristics

- **Domain-specific**
  - Embedded systems with soft real-time constraints
- **Graphical, but textual syntax exists**
- **Small, cohesive set of concepts**
- **Strong encapsulation**
  - Actors (active objects)
  - Explicit interfaces
  - Message-based communication
- **Event-driven execution**
  - State machines

---

**Real-time System**
- actors
- state

\[ \text{outputs} = f(\text{state}, \text{inputs}) \]
Core concepts

- Structural modeling
- Behavioural modeling
UML-RT: Core Concepts (1)

- **Types**
  - Capsules (active classes)
    - Capsule instances (parts)
  - Passive classes (data classes)
    - Objects
  - Protocols
  -Enumerations

- **Structure**
  - Attributes
  - Ports
  - Connectors

- **Behaviour**
  - Messages (events)
  - State machines

- **Grouping**
  - Package

- **Relationship**
  - Generalization
  - Associations
UML-RT: Core Concepts (2)

- **Model**
  - Collection of *capsule* definitions
  - ‘Top’ capsule containing collection of *capsule* instances (parts)

- **Capsules**
  - May contain
    - Attributes, ports, or other capsule instances (parts)
  - Behaviour defined by *state machine*

- **Ports**
  - Typed over *protocol* defining *input* and *output* messages

- **State machine**
  - Transition triggered by incoming messages
  - Action code can contain send statements that send messages over certain ports
Capsules (1)

- Kind of active class
  - Attributes, operations
  - Own, independent flow control (logical thread)
- May also contain
  - Ports over which messages can be sent and received
  - Parts (instances of other capsules) and connectors
- Creation, use of instances tightly controlled
  - Created by runtime system (RTS)
  - Cannot be passed around
  - Stored in attribute of another capsule (part)
  - Information flow only via messages sent to ports ⇒ better concurrency control and encapsulation
- Behaviour defined by state machine
Example: Capsules and Capsule Parts

- chute1
- chute2
- switcher
Passive Classes/Data Classes

- Similar to regular classes
- Do not have independent flow of control
- Behaviour defined through operations
- Used to define data structures and operations on them

UML-RT for robotic applications
Protocols

- Provide types for ports
- Define
  - Input messages
    - Services provided by capsule owning port
  - Output messages
    - Services required by capsule owning port
  - Input/output messages
- Messages can carry data
Ports

- “Boundary objects” owned by capsule
- Typed over a protocol P
- Have ‘\textbf{send}’ operation
  - $\text{portName.msg(arg1,\ldots,\text{argn}).send()}$
- Can be
  - \text{base (not conjugated)}
    - Direction of messages is as declared in protocol
    - \textbf{Notation}:
      - \text{q textual: P}
      - q graphical:  ■
  - \text{conjugated}
    - Direction of messages declared in protocol is reversed
    - \textbf{Notation}:
      - \text{q textual: ~P}
      - q graphical:  □

UML-RT for robotic applications
Connectors

- Connect two ports
- Ports must be compatible
  - Both are instances of same protocol
  - Either (asymmetric)
    - one is ‘base’ (i.e., not ‘conjugated’)
      - typically owned by ‘client’
    - and the other is ‘conjugated’
      - typically owned by ‘server’
  - Or (symmetric)
    - only InOut messages
Ports: External, Internal, Relay

- **External behaviour**
  - Provides (part of) externally visible functionality (isService=true)
  - Incoming messages passed on to state machine (isBehaviour=true)
  - Must be connected (isWired=true)

- **Internal behaviour**
  - As above, but not externally visible (isService=false)
  - Connect state machine with a capsule part

- **Relay**
  - Pass external messages to and from capsule parts
Ports: System

- Connects capsule to **Runtime System (RTS)** library via corresponding system protocol
- Provides access to RTS services such as

  - **Timing**: setting timers, time out message
    - timer2Port.informIn(UMLRTTimespec(10, 0));
      // set timer that will expire in 10 secs and 0 nanosecs
    - When timer expires, ‘timeout’ message will be sent over timer2Port
  
  - **Log**: sending text to console
    - logPort.log("Ready to self-destruct")

  - **Frame**: incarnate, destroy capsule instances
Example: PingPong
Example: Rover

```
«Protocol» Engine
  out moveForward ()
  out moveBackwards ()
  out turnLeft (angle : Integer)
  out turnRight (angle : Integer)
  out stop ()
  in turnedLeft ()
  in turnedRight ()
  in stopped ()

«Protocol» Detection
  out startDetection ()
  out stopDetection ()
  in obstacleDetected (distance : Real)
```

EclipseCon 2017, Toulouse, France
Slide courtesy Juergen Dingel
Example: Door Lock System

UML-RT for robotic applications
Core concepts

- Structural modeling
- Behavioural modeling
State Machines

States

• Capture relevant aspects of history of object
• Determine how object can respond to incoming messages
• May have invariants associated with them

Pseudo states

• Don’t belong to description of lifetime of object
  ⇒ object cannot be ‘in’ a pseudo state
• Helper constructs to define complex state changes

Transitions

• Describe how object can move from one state to next in response to message input
States and Pseudo States

- **States**
  - Kinds:
    - Basic
    - Composite (in hierarchical state machines)
  - May contain:
    - Entry action (written in action language)
    - Exit action (written in action language)

- **Pseudo states**
  - Initial
  - Choice point
  - History
  - Entry points
  - Exit point

In composite states only
Transitions

- **Kinds:**
  - Basic
  - **Group** (in hierarchical state machines)

- **Consists of**
  - **Triggers**
    - Transitions out of *pseudo states* (initial, choice) don’t have triggers
    - Transitions out of *non-pseudo state* should have at least one trigger
  - **Guards** (optional, written in action language)
  - **Effect/Actions** (optional, written in action language)
Action Language

- Language used in
  - guards to express Boolean expressions
  - entry action, exit action, transition effects to read and update attribute values, send messages

- Typically: C/C++, Java

⇒ State machines are a **hybrid notation** combining
  - graphical notation for state machines and
  - textual notation for source code in actions

⇒ UML and UML-RT State Machines
  - different from, e.g., Finite Automata
  - closer to ‘**extended hierarchical communicating state machines**’ [5]

Example: Action Code, Timers, Logging

timer.informIn(UMLRITimespec(5,0));
log.log("Switched to red");
Example: Ping Pong

UML-RT for robotic applications

Slide courtesy Juergen Dingel

EclipseCon 2017, Toulouse, France
Example: Timers

```java
logger.log("Processing request");
// compute output
p.response(output).send();

logger.log("Too late!");

logger.log("setting timer");
timer.informIn(UMLRTTimespec(MAX, 0));
```

---

**Diagram Description**

- **ServerStateMachine**
  - Initial state
  - Transition to Waiting state
  - Transition on `request` event
  - Transition on `timeout` event
  - Transition on `Error`

- **Loop**
  - Condition: `n < MAX`
  - Transition on `request(input:inputData)`
  - Transition on `response(output:outputData)`
  - Transition on `set timer` to MAX

- **Error**
  - Transition on `more than MAX seconds later`
  - Transition on `Error` from Client

---

*EclipseCon 2017, Toulouse, France*
*Slide courtesy Juergen Dingel*
Demo II: Models for the PolarSys Rover

- Copying the RTS on the RaspberryPI
  - Locating the RTS in your Papyrus-RT installation
    $ cd ...
  - Copying the RTS into the Raspberry PI 3
    $ scp ...
  - Compiling the RST
    $ make clean && make

- Adding the WiringPI library
  $ cd ...
  $ vim ...

- You Raspberry PI is ready!

- All these steps are already done for this workshop
Overview

1. Intro / MDE (10 mins) (8 slides)
2. Overview (1 min) (1 slide)
3. PolarSys Rover (15 mins) (16 slides)
4. Demo I (5 mins)
5. Hands on session (20 mins) (2 slides)
6. UML-RT: Part I (25 mins) (26 slides)
   * Core concepts
7. Demo II (10 mins) (3 slides)
8. Hands on session (20 mins) (1 slide)
9. UML-RT: Part II (10 mins) (14 slides)
   * More advanced concepts
10. Hackaton (90 mins) (3 slides)
11. Conclusion (5 mins) (2 slides)
Demo II: EngineController
Demo II: DetectionSensor

Application

command

log

timer

detectionSensor: DetectionSensor

DetectionSensorStateMachine

Initial

SETUP

assignGPIOS

IDLE

startDetection

DETECTING

stopDetection

timeout

ApplicationStateMachine

obstacleDetected

Initial

RUNNING

timeout

emergency

EMERGENCY

/entry OpaqueBehavior
Entry

/entry OpaqueBehavior
Entry

EclipseCon 2017, Toulouse, France
Demo II: LineFollower

Application

- log
- timer

command

lineFollower: LineFollower

command

EngineControllerStateMachine

SETUP
assignGPIOs

IDLE

initial
startFollowing

STOP FOLLOWING

timeout

START FOLLOWING

timeout

ApplicationStateMachine

SETUP

timeout

START FOLLOWING

timeout

STOP FOLLOWING

timeout

initial

lostLine

LOST LINE
Hand-on session

host: 192.168.1.54
hostname: traffic-light-blue
login: pi / EclipseCon2017

host: 192.168.1.36
hostname: line-follower-rover
login: pi / EclipseCon2017

Router connection:
SSID: UMLRT-2017
WPA2: EclipseCon2017

Host: ???
hostname: traffic-light-white
login: pi / EclipseCon2017
UML-RT Part II

- More on ports
- More on state machines
Ports: SPP and SAP

- So far, only wired ports
  - Connected automatically when instances are created

- Unwired ports
  - Connected at run-time
  - Publish/subscribe
    - Port on publisher: Service Provision Point (SPP)
    - Port on subscriber: Service Access Point (SAP)
    - Register with RTS using unique service name (manually or automatic)
State Configuration

- States can be active: flow of control resides at state
- If a substate is active, its containing superstate is, too
- **State configuration**: list of active states
- **Stable state configuration**: no pseudo states and ends in basic state
- **Example**: <‘play’, ‘player1Move’, ‘waitForHand’>
1. Machine in **stable state configuration**

2. Message m1 has arrived and is **dispatched**

3. If dispatching enables no transition, m1 is ‘dropped’

4. If dispatching **enables** transition t,
   - source state of t active,
   - message matches trigger of t, and
   - guard evaluates to ‘true’

5. **then transition t executed**
   a. execute exit action of source state of t (if any)
   b. execute action code of t (if any)
   c. execute entry code of target state of t (if any)

6. If target of t is pseudo state
   a. continue by choosing and executing outgoing transition (i.e., goto 5.)

7. Machine in **stable state configuration**
Run-to-Completion

- The event processing of state machines follows ‘run-to-completion’ semantics.
- Dispatching of message triggers execution of possibly entire chain of transitions (Steps 5 and 6 on previous slide).
- Execution lasts until stable state configuration has been reached (last state in transition chain not a pseudo state).
- During transition execution, no other message will be dispatched.

⇒ better concurrency control
- **Source state is** **composite**
- **Example:**
  - Start configuration `<play',player2Move'>`
  - Execute transition `reset`:
    - exit code `player2Move'`, exit code `play'`, effect `reset'`, entry code `idle'`
  - End configuration `<idle'>`
History

- Re-establish full state configuration that was active when containing state was active most recently
- **Example**: from `<play, s>` to `<play, s>` with `reset` `resume1`
Self Transitions

- Source and target states are the same
- 2 kinds: external, internal
- **External**: source state (and all substates) exited and target state entered

UML-RT for robotic applications
Self Transitions: Internal

- Source state (and all substates) remain active; no exit or entry actions executed
Example: Door Lock

```
set timer

"doors open"; "hit key to lock" getchar(); lockPort.lock().send()

"doors locked"; "hit key to open" getchar(); lockPort.unlock().send()

"lock" + i + "locked"; lockPort.lockStatus(true).send
```
Example: Rock/Paper/Scissors
Additional UML-RT Features

- **Structure**
  - Optional capsules
  - Inheritance

- **Behaviour**
  - Junction pseudo state
  - Defer/recall
  - Synchronous communication
  - Message priorities
Additional Papyrus-RT Capabilities

- Generation of multi-threaded code
  - Logical thread
    - Flow of control for capsule instance
  - Physical thread
    - Executes RTS controller
      - Oversees execution of all capsules assigned to physical thread
- Generating single threaded code
  - 1 physical thread executing one controller executing all capsules
- Generating multi threaded code
  - Several physical threads each executing their own controller

- Graphical/textual hybrid modeling (prototype)
  - Fully synchronized

- Legacy model import

- Observer service
Papyrus-RT: What’s Missing?

- Model-level analysis
  - Model execution/interpretation
  - Debugging (ongoing)
  - Testing (ongoing)
  - Static analysis
- Integration with external tools (ongoing)
  - Animation, simulation (Unity)
- Sequence diagram integration
- Graphical/textual hybrid modeling (ongoing)
- Action language (ongoing)
- User experience
- Deployment
Overview

1. Intro / MDE (10 mins) (8 slides)
2. Overview (1 min) (1 slide)
3. PolarSys Rover (15 mins) (16 slides)
4. Demo I (5 mins)
5. Hands on session (20 mins) (2 slides)
6. UML-RT: Part I (25 mins) (26 slides)
   • Core concepts
7. Demo II (10 mins) (3 slides)
8. Hands on session (20 mins) (1 slide)
9. UML-RT: Part II (10 mins) (14 slides)
   • More advanced concepts
10. Hackaton (90 mins) (3 slides)
11. Conclusion (5 mins) (2 slides)
Hackaton !

- Discovery
  - Move forward and take pictures
  - Record a video
  - Detect obstacle and stops
  - Change direction

- Line-following robot
  - Follow a line
  - Stop when detecting an obstacle
  - Brake whenever a front obstacle is too close
  - Stop when the traffic light turns red

- Traffic light
  - Red, green, orange
  - Detect the presence of a vehicle
  - Push button to detect pedestrians
To export a package as a submodel:

- Right click on the package to export in the model explorer
- Click on Create Submodel
- Set the resource URI
- Hit ‘OK’ and save
To import a package in another model:

- Right click on the root element of the model you want to import your submodel in the model explorer.
- Select ‘Import / Import Package from User Model’.
- Browse your workspace and find your submodel.
- Click on the ‘Copy All’ button to import.
Conclusion

- **Intro to**
  - **UML-RT**
    - small, proven subset of UML for real-time systems
  - **Papyrus-RT**
    - open-source MDE tool w/ full code generation
  - **PolarSys Rover / Wiring PI**
- Lots of opportunity to use research, contribute
- More questions?
  - hili@cs.queensu.ca
- Feedback?
# Acknowledgments

**With the active and huge participation of:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Juergen Dingel</td>
<td>Professor at Queen’s University</td>
</tr>
<tr>
<td>Harshith Vasanth Gayathri</td>
<td>MSc at Queen’s University</td>
</tr>
<tr>
<td>Sudharshan Gopikrishnan</td>
<td>MSc at Queen’s University</td>
</tr>
</tbody>
</table>

**Special thanks to:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Position</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaël Blondelle</td>
<td>Director European Ecosystem Development at Eclipse foundation</td>
<td>for helping us bootstrapping the project with the PolarSys Rover</td>
</tr>
<tr>
<td>Aaron Vissier</td>
<td>System analyst at Queen’s University</td>
<td>for his huge help with all the “soldering” and “tooling” questions I have had (and for the future ones :))</td>
</tr>
<tr>
<td>Michal Pasternak</td>
<td>MSc at Queen’s University</td>
<td>for spending hours 3D-printing the extensions for the Rover</td>
</tr>
</tbody>
</table>
Resources and References

- **Links**
  - **Papyrus-RT:** [https://eclipse.org/papyrus-rt](https://eclipse.org/papyrus-rt)
    - Installation, tutorial, etc: [https://wiki.eclipse.org/Papyrus-RT/User](https://wiki.eclipse.org/Papyrus-RT/User)
    - Wiki: [https://wiki.eclipse.org/Papyrus-RT](https://wiki.eclipse.org/Papyrus-RT)
  - **Papyrus:** [https://eclipse.org/papyrus/](https://eclipse.org/papyrus/)
    - Papyrus industrial Consortium: [https://wiki.polarsys.org/Papyrus_IC](https://wiki.polarsys.org/Papyrus_IC)
  - **PolarSys:** [https://www.polarsys.org/](https://www.polarsys.org/)

- **References**


