

# Rover control with UML-RT: an introduction

Antonio García-Domínguez

Aston University

CS2020/CS2090/CS4340 — Software Engineering  
October 19th, 2017

# Why this lecture?

## Ways to use UML

So far, we have used UML in two ways:

- As throwaway sketches to communicate between developers
- As analysis / design documentation to be maintained

But there is a third:

- As executable artefacts

## This lecture

- We will show that you **can run** UML(-RT) models
- In fact, they provide useful abstractions here  
... also, robots are fun :-)

# Why this lecture?

## Ways to use UML

So far, we have used UML in two ways:

- As throwaway sketches to communicate between developers
- As analysis / design documentation to be maintained

But there is a third:

- As executable artefacts

## This lecture

- We will show that you can run UML(-RT) models
- In fact, they provide useful abstractions here
- ... also, robots are fun :-)

# Why this lecture?

## Ways to use UML

So far, we have used UML in two ways:

- As throwaway sketches to communicate between developers
- As analysis / design documentation to be maintained

But there is a third:

- As executable artefacts

## This lecture

- We will show that you can run UML(-RT) models
- In fact, they provide useful abstractions here
- ... also, robots are fun :-)

# Outline

1 Rover

2 UML-RT

3 Exercises

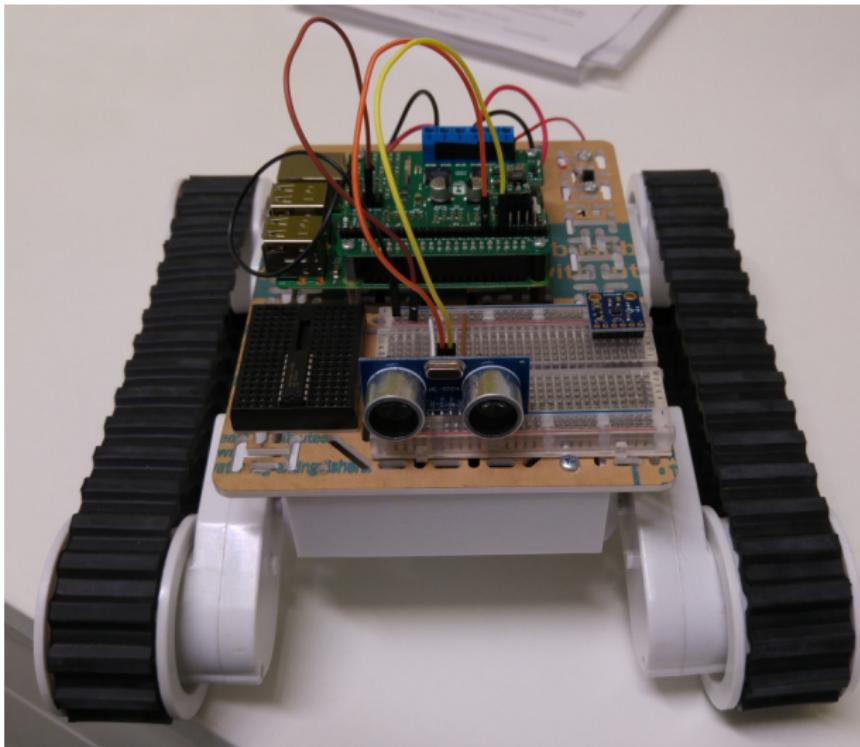
# Outline

1 Rover

2 UML-RT

3 Exercises

# Our subject: the rover



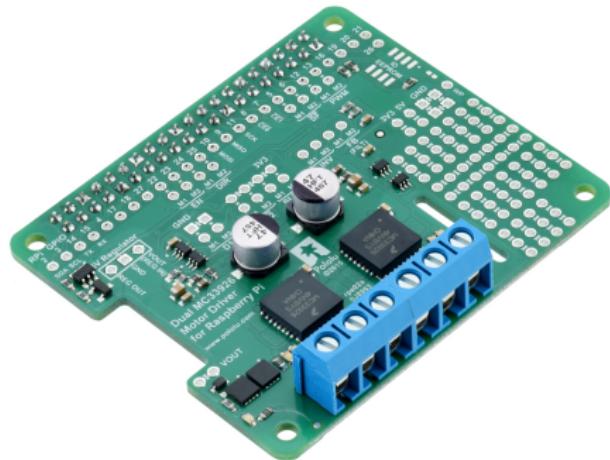
## The brain: Raspberry Pi 3



(Wikimedia Commons, CC-BY-SA 2.0)

- Full ARM-based computer with 1GB RAM/BT/WiFi for £30
  - Can run GNU/Linux with a desktop environment (Raspbian)
  - General Purpose I/O pins to talk to sensors and actuators

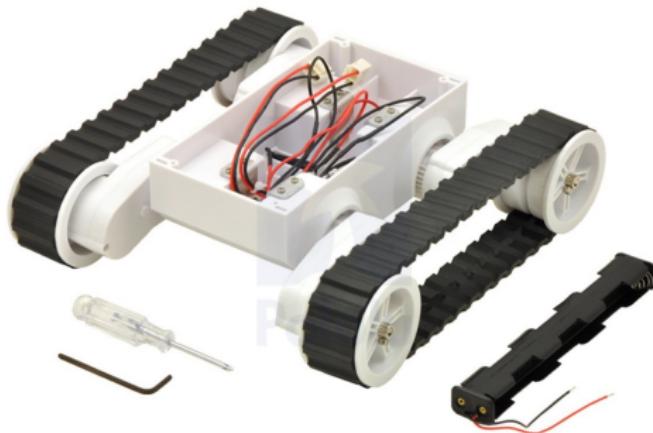
# The heart: motor driver board



(pololu.com store)

- Dual Pololu MC33926 motor driver add-on board (£30)
- Allows the Pi to use GPIO pins to control two DC motors
- Can enable/disable, control direction, and regulate speed
- Can feed the Pi through a voltage regulator (we use one)

# The body: motorized chassis

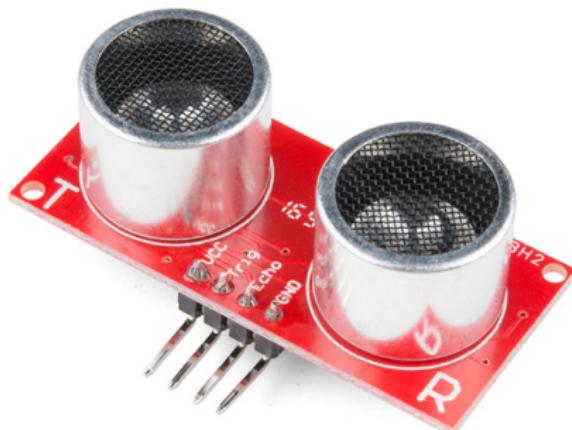


[www.pololu.com](http://www.pololu.com)

(pololu.com store)

- Pololu Rover5 chassis (£40)
- 2 treads with built-in DC brushed motors
- This version has no encoders — can't tell how fast it's going...

# The “eyes”: ultrasonic range finder



(sparkfun.com store)

- HC-SR04 ultrasonic range finder (£4–£8)
- Sends ultrasonic pulse, measures time for echo
- Nice for longer distances, but not very good at an angle...
- Can be noisy as well sometimes: needs a bit of filtering

# How do you code for the rover?

## Available languages and libraries for the Pi

- Python and C/C++ are the most popular
- Libraries: pigpio, WiringPi, RPi.GPIO...
- Pi runs a full operating system (unlike Arduino)

## Complexities in writing the code

- There are many things going on at once in a robot
- Keeping track of everything and reacting **in time** is crucial
- We also want to integrate/swap components cleanly

## Options

- Write a good bit of multithreaded code by hand (hard!)
- Use a model to generate it, providing snippets with specifics

# How do you code for the rover?

## Available languages and libraries for the Pi

- Python and C/C++ are the most popular
- Libraries: pigpio, WiringPi, RPi.GPIO...
- Pi runs a full operating system (unlike Arduino)

## Complexities in writing the code

- There are many things going on at once in a robot
- Keeping track of everything and reacting **in time** is crucial
- We also want to integrate/swapping components cleanly

## Options

- Write a good bit of multithreaded code by hand (hard!)
- Use a model to generate it, providing snippets with specifics

# How do you code for the rover?

## Available languages and libraries for the Pi

- Python and C/C++ are the most popular
- Libraries: pigpio, WiringPi, RPi.GPIO...
- Pi runs a full operating system (unlike Arduino)

## Complexities in writing the code

- There are many things going on at once in a robot
- Keeping track of everything and reacting **in time** is crucial
- We also want to integrate/swap components cleanly

## Options

- Write a good bit of multithreaded code by hand (hard!)
- Use a model to generate it, providing snippets with specifics

# Outline

1 Rover

2 UML-RT

3 Exercises

# What is UML-RT?

## Basic concepts

- UML for Real-Time: time-aware, reactive systems [UML-RT]
- Extension of a part of UML:
  - Component diagrams → UML-RT capsule structure diagrams
  - State machine diagrams → UML-RT uses a subset of those
- Proposed in 2006 by Bran Selic
- Provides concurrency and execution capabilities

## Chosen implementation: Papyrus-RT

- <https://wiki.eclipse.org/Papyrus-RT>
- Open-source, part of the Eclipse project
- Can generate C++ code from UML-RT models
- We can use C++ code in states and transitions

# What is UML-RT?

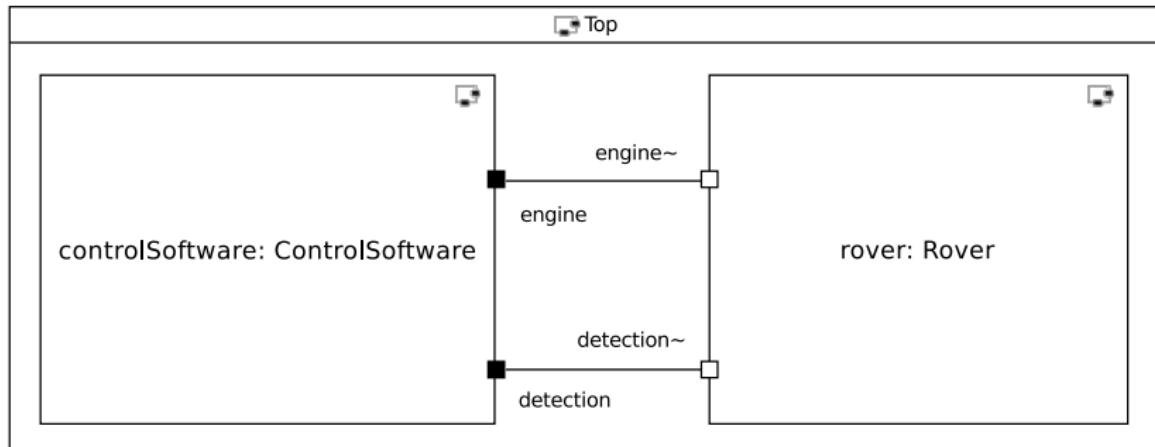
## Basic concepts

- UML for Real-Time: time-aware, reactive systems [UML-RT]
- Extension of a part of UML:
  - Component diagrams → UML-RT capsule structure diagrams
  - State machine diagrams → UML-RT uses a subset of those
- Proposed in 2006 by Bran Selic
- Provides concurrency and execution capabilities

## Chosen implementation: Papyrus-RT

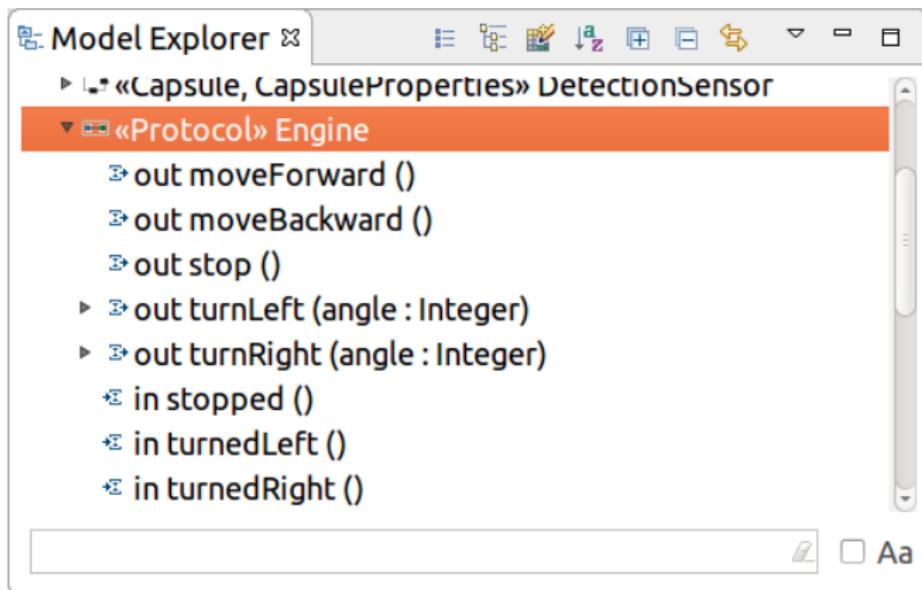
- <https://wiki.eclipse.org/Papyrus-RT>
- Open-source, part of the Eclipse project
- Can generate C++ code from UML-RT models
- We can use C++ code in states and transitions

# Top-level capsule



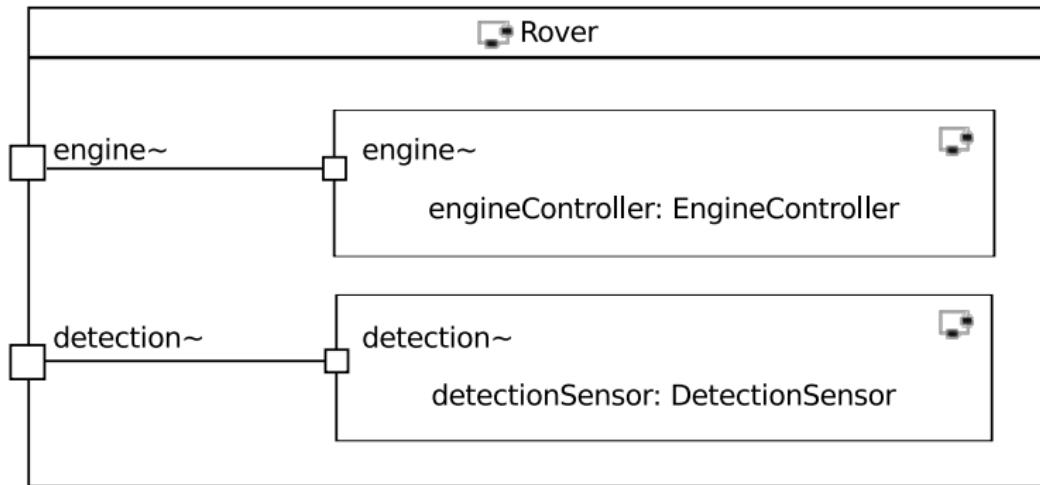
- Capsules: pretty much components as you know them
- Capsule ports exchange messages using protocols
- Here, the program is divided into a control algorithm and an abstraction of the rover hardware

# Protocols for capsule communication



- “Out” messages: black port → white (“conjugated”) port
- “In” messages go the other way
- Out messages are commands, in messages are notifications

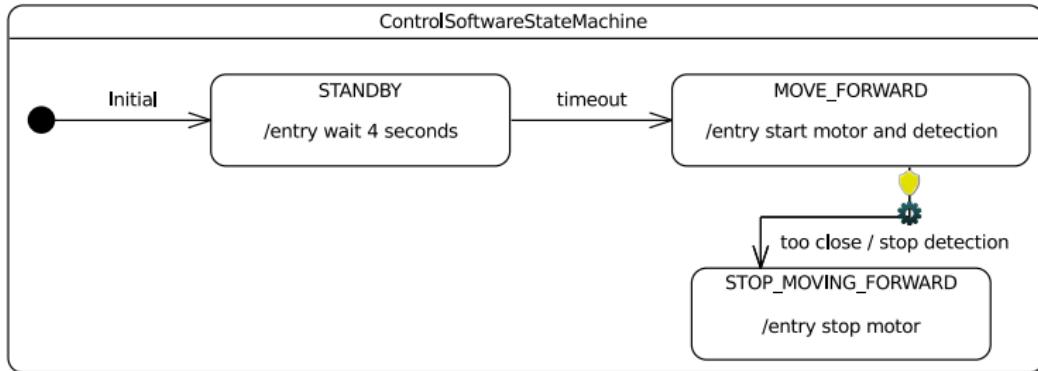
# Rover capsule



- You can nest capsules inside others
- “Rover” contains “EngineController” and “DetectionSensor”
- Inner capsules handle messages from the “Rover” ports

# State machine for the “ControlSoftware” capsule

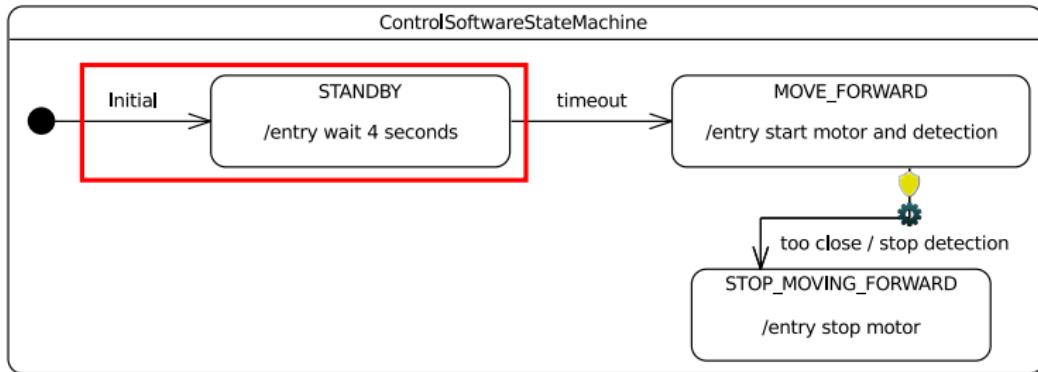
Capsules can have behaviour associated to them through state machines



- Same notation as plain UML state machines
- Some UML state machine elements are not available
- In Papyrus-RT, guards are yellow shields, and gears are effects
- Labels only for readability: all behaviour is in C++ snippets

# State machine for the “ControlSoftware” capsule

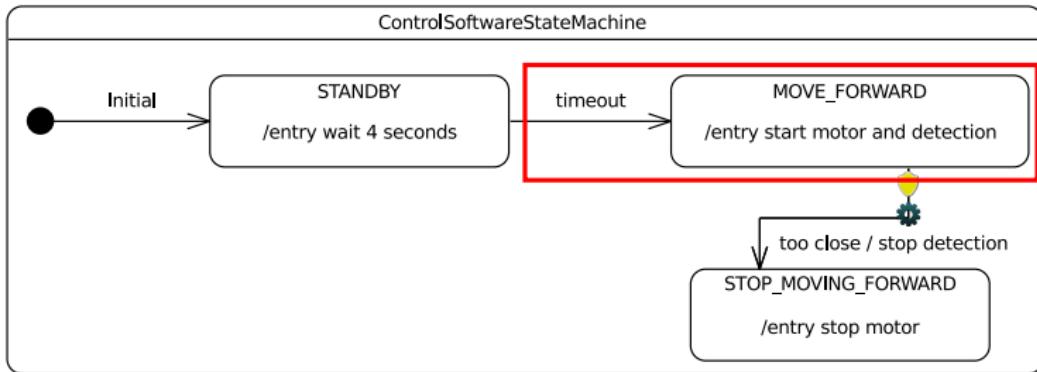
Capsules can have behaviour associated to them through state machines



- ➊ Asks “timer” system service for notification in 4 seconds
- ➋ Transitions into “MOVE\_FORWARD”, asking the “Rover” capsule to “goForward” and “startDetection”
- ➌ “DetectionSensor” sends “obstacleDetected” messages periodically: when too close, controller stops detection and transitions into “STOP\_MOVING\_FORWARD”
- ➍ Upon entering that state, motor is told to stop

# State machine for the “ControlSoftware” capsule

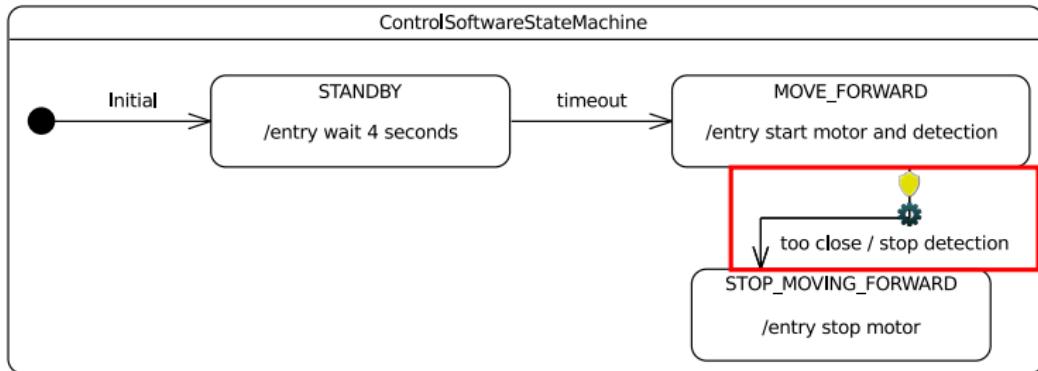
Capsules can have behaviour associated to them through state machines



- ➊ Asks “timer” system service for notification in 4 seconds
- ➋ Transitions into “MOVE FORWARD”, asking the “Rover” capsule to “goForward” and “startDetection”
- ➌ “DetectionSensor” sends “obstacleDetected” messages periodically: when too close, controller stops detection and transitions into “STOP MOVING FORWARD”
- ➍ Upon entering that state, motor is told to stop

# State machine for the “ControlSoftware” capsule

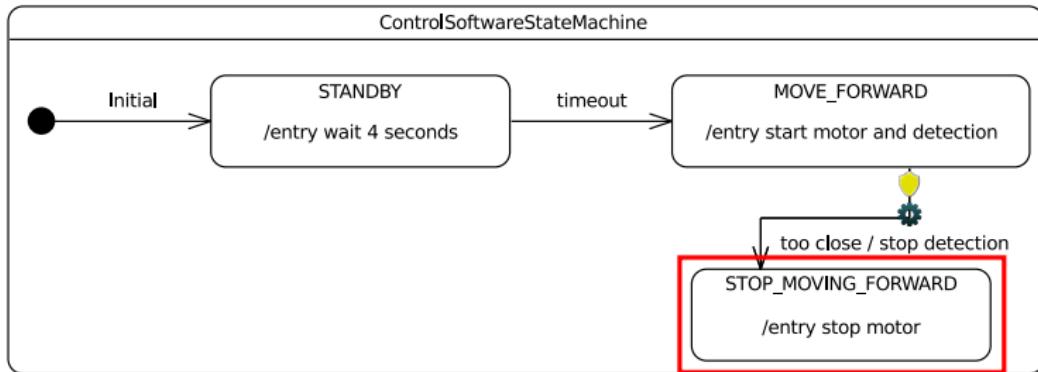
Capsules can have behaviour associated to them through state machines



- ① Asks “timer” system service for notification in 4 seconds
- ② Transitions into “MOVE\_FORWARD”, asking the “Rover” capsule to “goForward” and “startDetection”
- ③ “DetectionSensor” sends “obstacleDetected” messages periodically: when too close, controller stops detection and transitions into “STOP\_MOVING\_FORWARD”
- ④ Upon entering that state, motor is told to stop

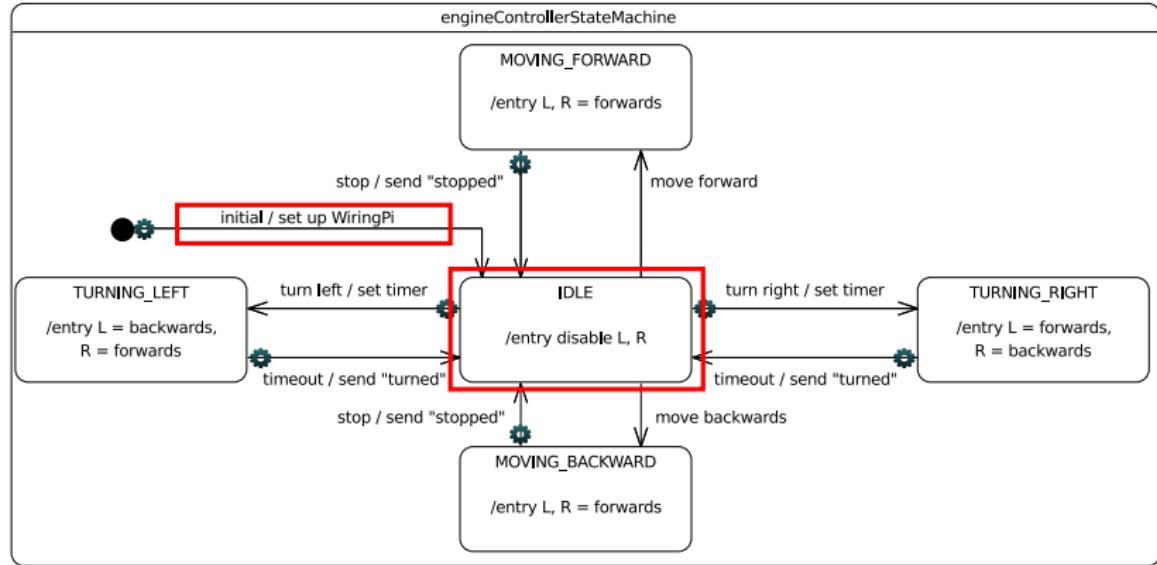
# State machine for the “ControlSoftware” capsule

Capsules can have behaviour associated to them through state machines



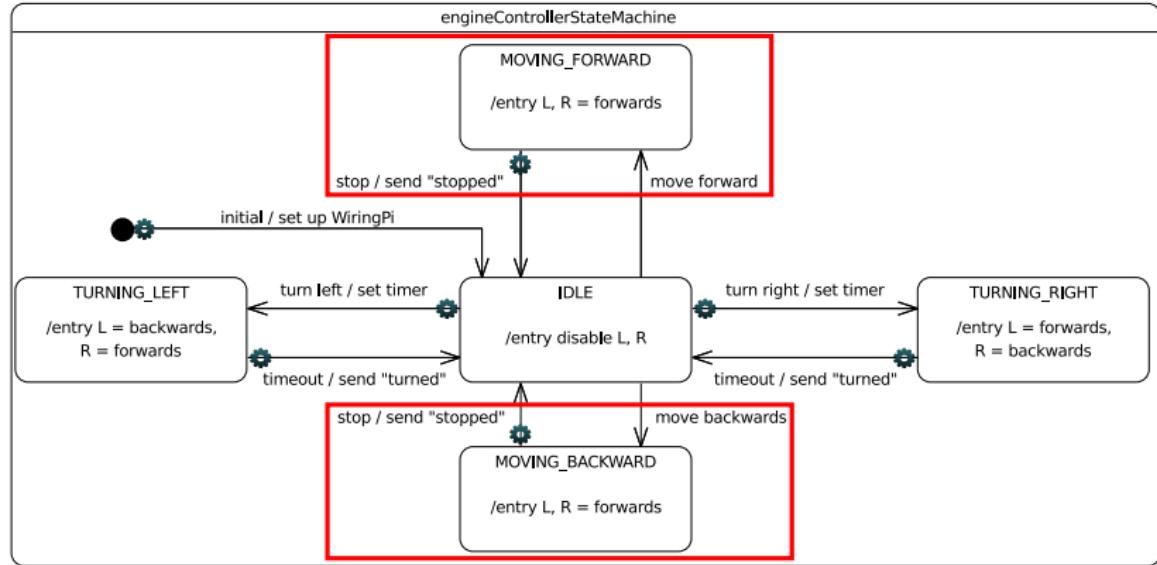
- ① Asks “timer” system service for notification in 4 seconds
- ② Transitions into “MOVE\_FORWARD”, asking the “Rover” capsule to “goForward” and “startDetection”
- ③ “DetectionSensor” sends “obstacleDetected” messages periodically: when too close, controller stops detection and transitions into “STOP\_MOVING\_FORWARD”
- ④ Upon entering that state, motor is told to stop

# State machine for the “EngineController” capsule



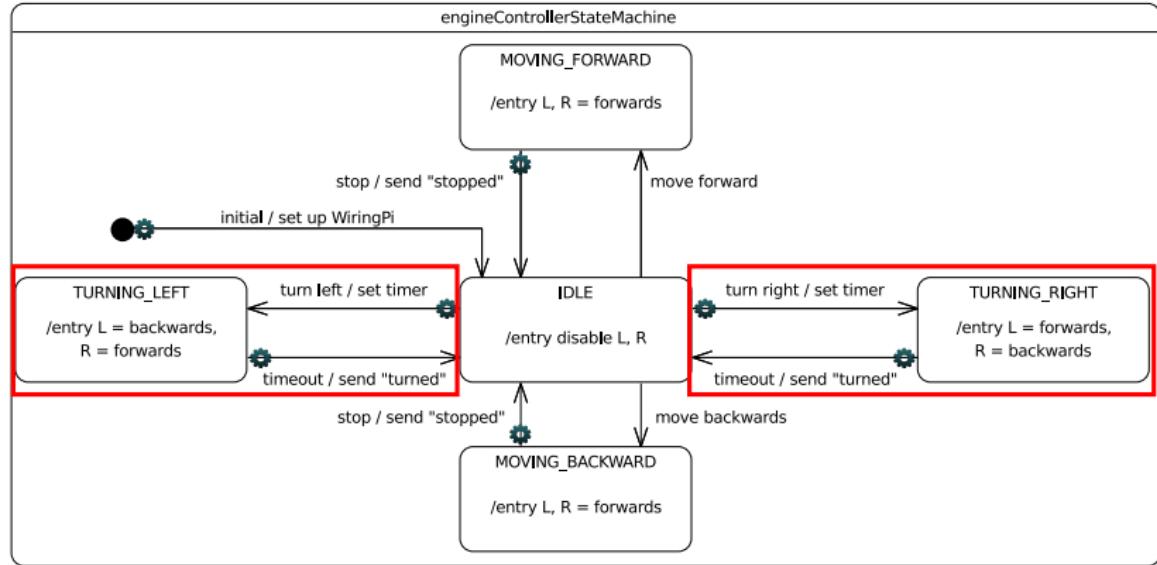
- Starts at the IDLE state after setting up the GPIO pins
- Goes forwards/backwards until told to stop, which it confirms
- A timer turns the rover for duration proportional to angle (!)

# State machine for the “EngineController” capsule



- Starts at the **IDLE** state after setting up the GPIO pins
- Goes forwards/backwards until told to stop, which it confirms**
- A timer turns the rover for duration proportional to angle (!)

# State machine for the “EngineController” capsule



- Starts at the IDLE state after setting up the GPIO pins
- Goes forwards/backwards until told to stop, which it confirms
- A timer turns the rover for duration proportional to angle (!)

Demo time!

Let's see how it's done with Papyrus-RT, and run the simple state machine in the rover.

# Outline

1 Rover

2 UML-RT

3 Exercises

# Exercise: make the rover back up and rotate

Modify the “ControlAlgorithm” state machine. After getting too close and stopping, it should:

- ① Wait 2 seconds
- ② Go backwards for 1 second
- ③ Turn right 138 degrees
- ④ Stop
- ⑤ Go forward again

Do it first in English, then think of the messages you should be sending and triggering on.

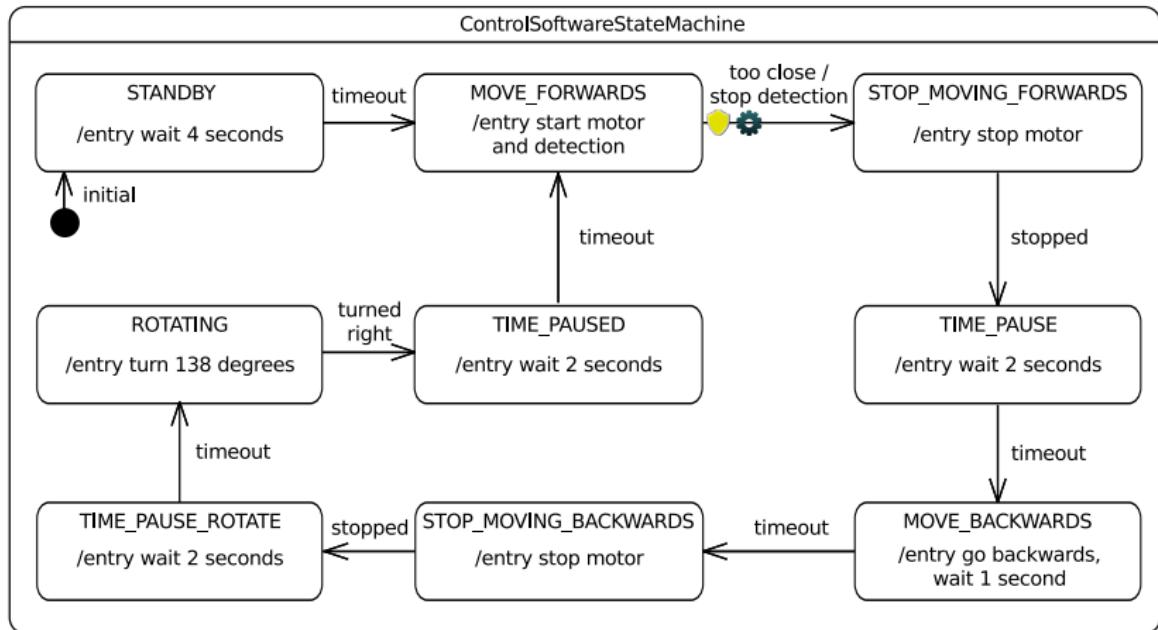
## Engine protocol

- out moveForward()
- out moveBackward()
- out stop()
- out turnLeft(angle)
- out turnRight(angle)
- in stopped()
- in turnedLeft()
- in turnedRight()

## Timer protocol

- out informIn(secs, nanosecs)
- in timeout()

# Solution for the exercise



Demo time again!

Let's switch to the branch with this solution and try it out on the rover.

# Exercise: how would you improve the rover?

## Problems with the latest model

- Turning angle is fixed, regardless of obstacle
- Speed is also fixed, regardless of distance
- We are limited by having only one sensor

## In groups

- Come up with an idea to improve the rover's behaviour
- You can change the protocols if you like
- You can add extra ultrasonic range finders, too
  - Where would you put them, though?
- How would you implement your idea in UML-RT?

# Exercise: how would you improve the rover?

## Problems with the latest model

- Turning angle is fixed, regardless of obstacle
- Speed is also fixed, regardless of distance
- We are limited by having only one sensor

## In groups

- Come up with an idea to improve the rover's behaviour
- You can change the protocols if you like
- You can add extra ultrasonic range finders, too
  - Where would you put them, though?
- How would you implement your idea in UML-RT?

## One last demo!

We have one branch where speed is regulated by the distance to the obstacle, and we turn until the obstacle is far enough. Let's try it out.

# In closing

Models can provide useful abstractions

- With UML-RT, we think about real-time control differently
- Collaborating state machines **vs** manual multithreaded coding
- The models become C++ code, which we can run on the Pi
- This is within the wider area of **executable modelling**

Papyrus-RT: your thoughts?

- Con: it's not as easy as just drawing
- Pro: **we can do more things with these models**
- Roadmap: sequence/class/activity diagrams
- Generating code for other languages is also planned

# In closing

Models can provide useful abstractions

- With UML-RT, we think about real-time control differently
- Collaborating state machines *vs* manual multithreaded coding
- The models become C++ code, which we can run on the Pi
- This is within the wider area of **executable modelling**

Papyrus-RT: your thoughts?

- Con: it's not as easy as just drawing
- Pro: **we can do more things with these models**
- Roadmap: sequence/class/activity diagrams
- Generating code for other languages is also planned

Hope you enjoyed it!

# Questions? Feedback?

Source code for slides and models:

<https://github.com/bluezio/rover-demo>

Step-by-step instructions from scratch here:

<https://goo.gl/1jeJ8w>

# Bibliography



B. Selic.

Using UML for modeling complex real-time systems.

Languages, Compilers, and Tools for Embedded Systems,  
LNCS vol. 1474, pp. 250–260.

<https://link.springer.com/chapter/10.1007/BFb0057795>