Rover control with UML-RT: an introduction

Antonio García-Domínguez

Aston University

CS2020/CS2090/CS4340 — Software Engineering
October 16th, 2019

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

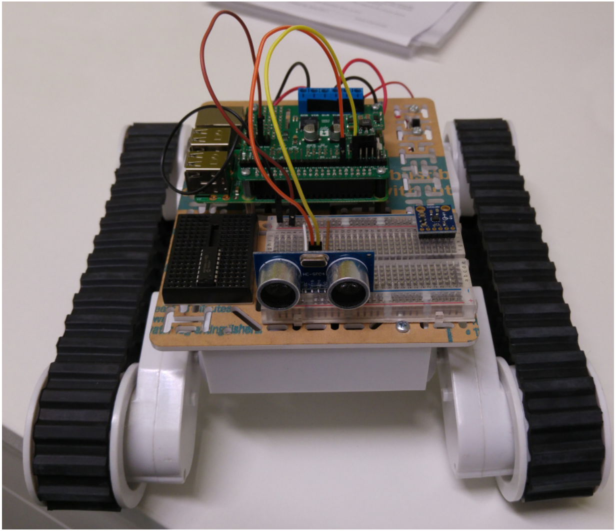
This lecture

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

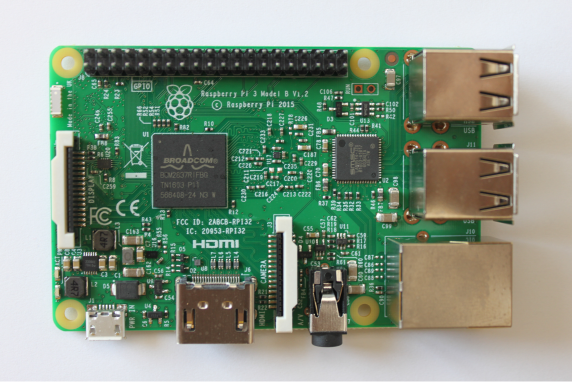
Outline

Outline

Our subject: the Papyrus-IC rover

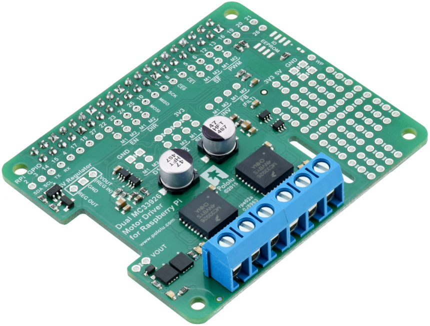


The brain: Raspberry Pi 3



- 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

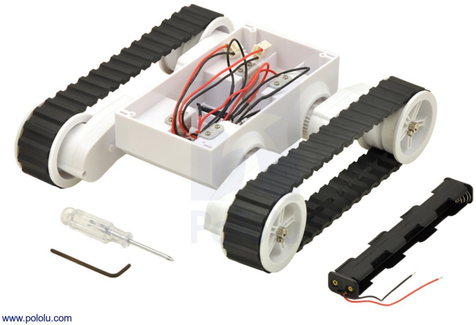
The heart: motor driver board



(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)

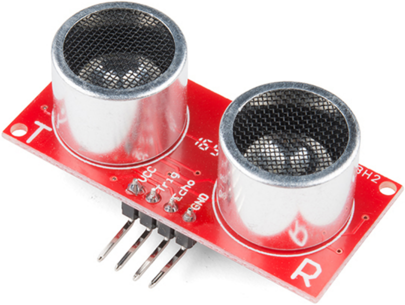
The body: motorized chassis



(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 hardware 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 hardware 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 hardware cleanly

Options

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

Outline

What is UML-RT?

Basic concepts

- UML for Real-Time: time-aware, reactive systems [UML-RT]
- Extension of a part of UML:
  - Composite structures → UML-RT capsule structures
  - State machines → UML-RT uses a subset
- Based on ROOM language by Bran Selic et al. (1994)
- 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?

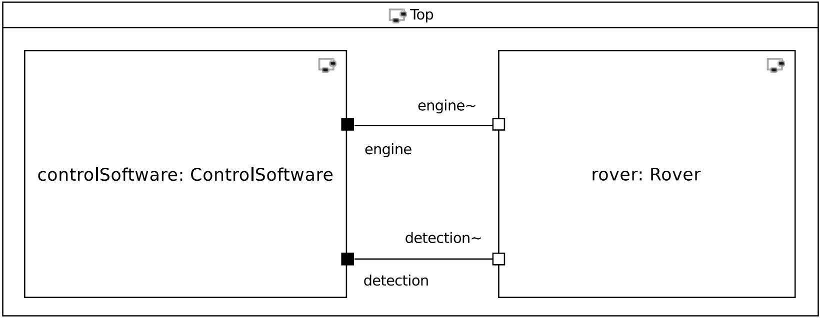
Basic concepts

- UML for Real-Time: time-aware, reactive systems [UML-RT]
- Extension of a part of UML:
    - Composite structures → UML-RT capsule structures
    - State machines → UML-RT uses a subset
- Based on ROOM language by Bran Selic et al. (1994)
- Provides concurrency and execution capabilities

Chosen implementation: Papyrus-RT

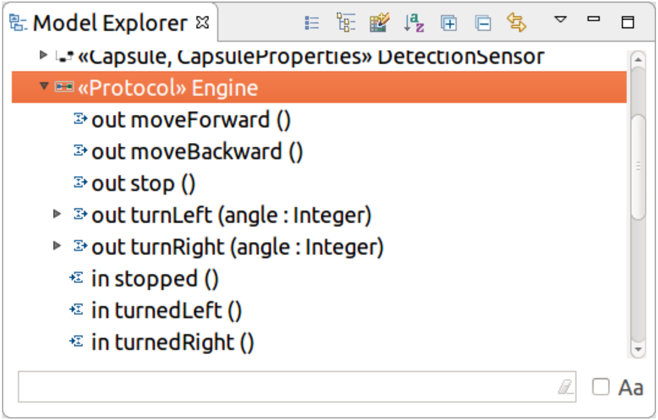
- `https://wiki.eclipse.org/Papyrus-RT`
- Open-source, part of the Eclipse project
- Can generate C++ code from UML-RT models
- We can use C++ code in states and transitions

Top-level capsule



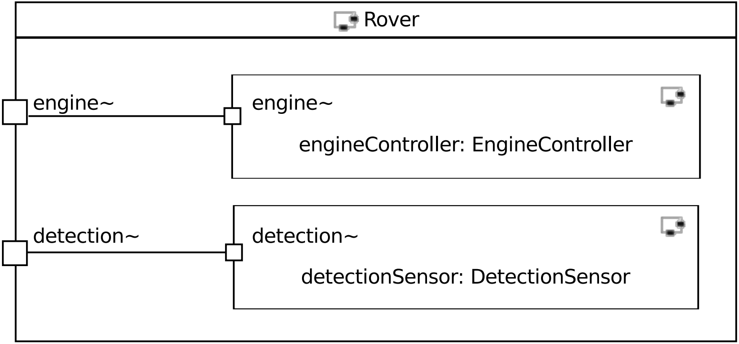
- Capsules: each represents a part of our system
- Capsule ports exchange messages using protocols
- Here, the program is divided into a control algorithm and an abstraction of the rover hardware

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

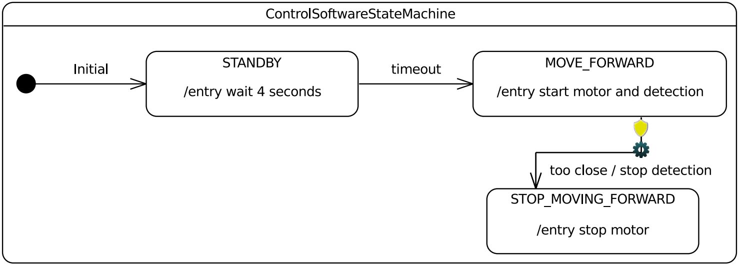
Rover capsule



- 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

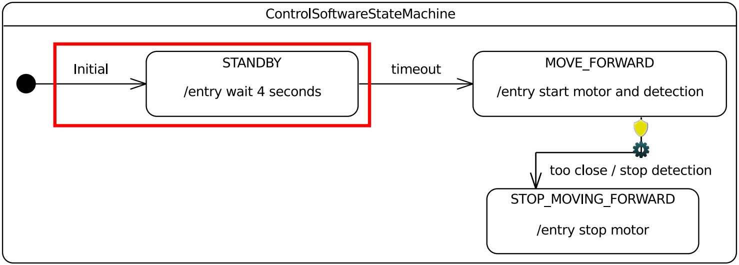
Capsules can have behaviour associated to them through state machines



- 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

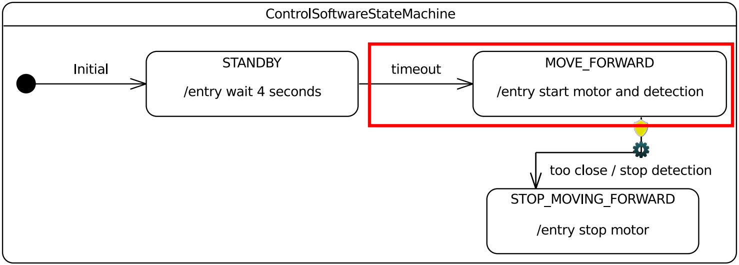
Capsules can have behaviour associated to them through state machines



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

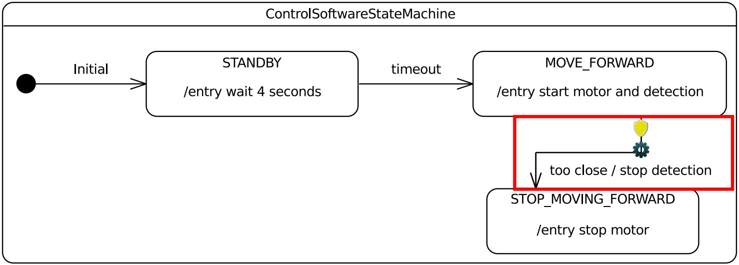
Capsules can have behaviour associated to them through state machines



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 "ControlSoftware" capsule

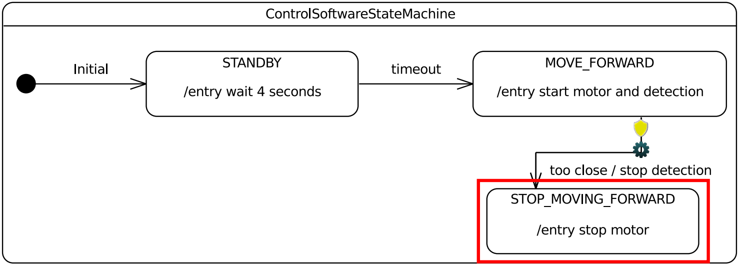
Capsules can have behaviour associated to them through state machines



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

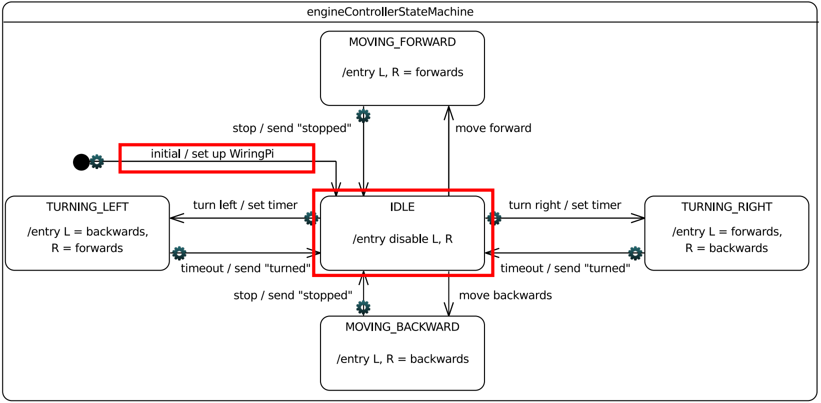
State machine for the "ControlSoftware" capsule

Capsules can have behaviour associated to them through state machines



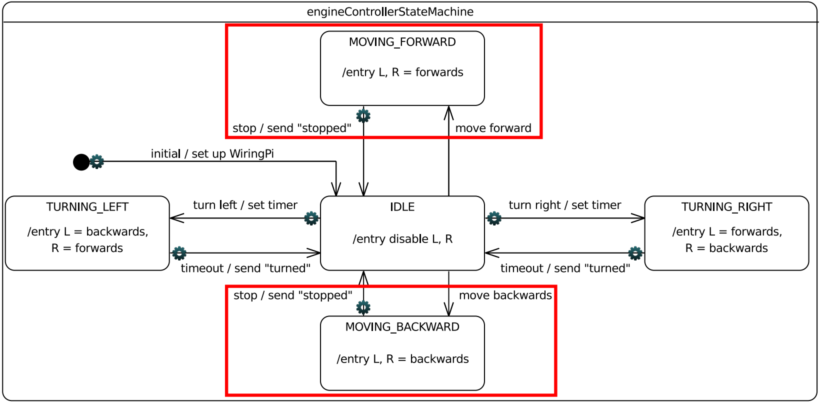
1. Asks "timer" system service for notification in 4 seconds
2. Transitions into "MOVE_FORWARD", asking the "Rover" capsule to "goForward" and "startDetection"
3. "DetectionSensor" sends "obstacleDetected" messages periodically: when too close, controller stops detection and transitions into "STOP_MOVING_FORWARD"
4. Upon entering that state, motor is told to stop

State machine for the "EngineController" capsule



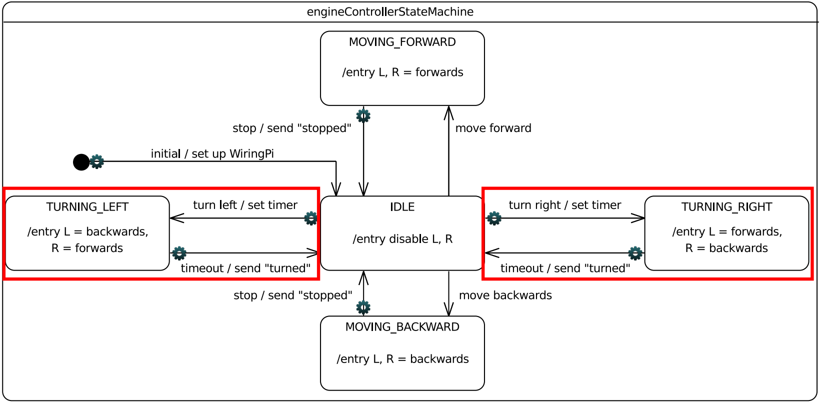
- Starts at the IDLE state after setting up the GPIO pins
- Goes forwards/backwards until told to stop, which it confirms
- A timer turns the rover for duration proportional to angle (!)

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



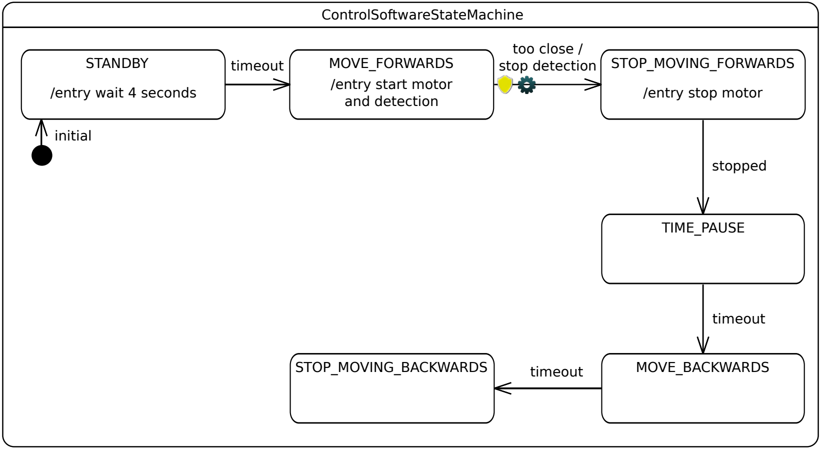
- Starts at the IDLE state after setting up the GPIO pins
- Goes forwards/backwards until told to stop, which it confirms
- A timer turns the rover for duration proportional to angle (!)

Demo time!

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

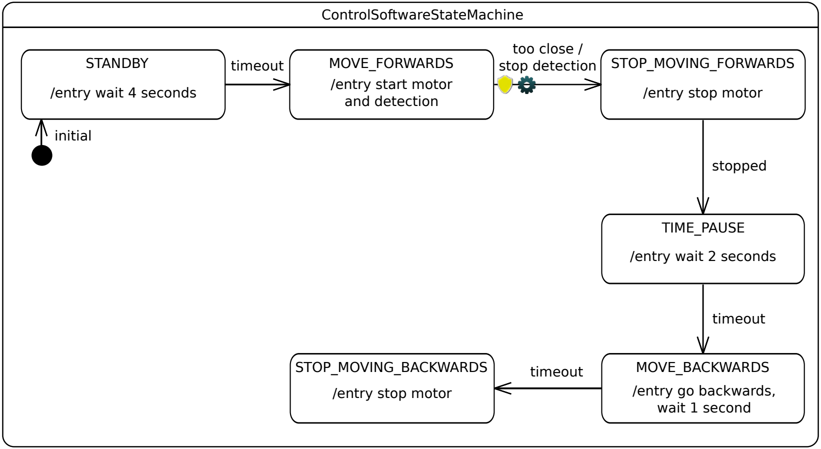
Outline

Backing up: fill in the gaps!

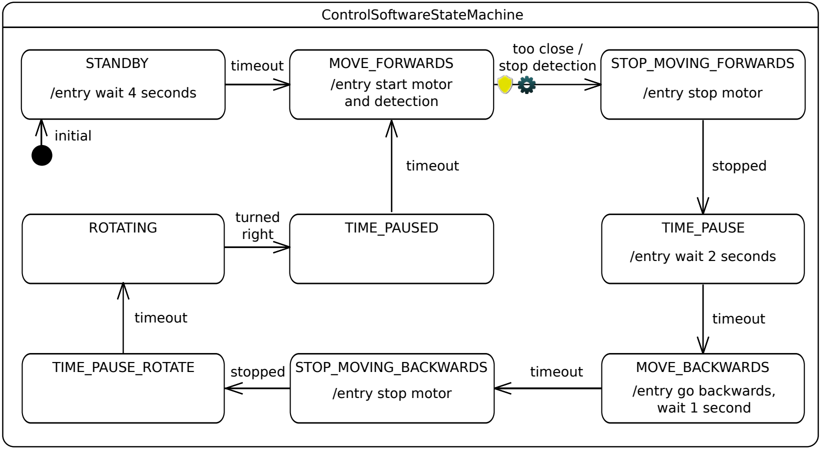


Write down the entry actions for the new states.

Solution for backing up

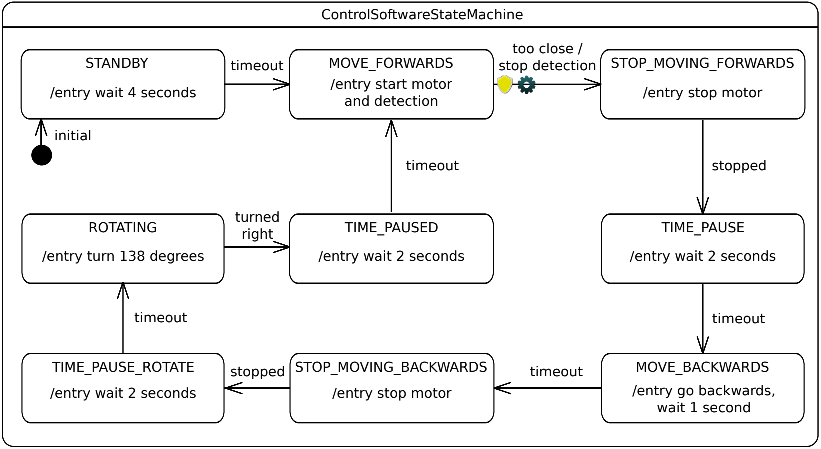


Turning around a bit: fill in the gaps!



Write down the entry actions for the new states.

Solution for backing up



Demo time again!

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

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

Ideas?

Let's brainstorm for a bit! Take into account:

- System complexity
- Total cost of the robot
- Energy use

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

Ideas?

Let's brainstorm for a bit! Take into account:

- System complexity
- Total cost of the robot
- Energy use

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`

A timeline of UML-RT and Papyrus-RT

Summary given by Ernesto Posse (Zeligsoft) @ Papyrus-IC list

1987 Telos project at Bell-Northern Research (acquired by Nortel)

1992 Some people from Telos went on to found ObjectTime. Created the language and tool with the same name.

1994 Telos language was renamed “ROOM” (Real-Time Object-Oriented Modelling). The ROOM book is published, co-authored by Bran, G. Gullekson and P. Ward.

1998 Bran and Jim Rumbaugh made a UML profile for ROOM. They called it UML-RT.

2000 ObjectTime is acquired by Rational. They create Rational RoseRT based on it.

2006 IBM acquires Rational. They create IBM Rational Software Architect Real-Time Edition (RSA-RTE) based on it.

2014 CEA, Zeligsoft and others partner to create Papyrus-RT.

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

B. Selic, G. Gullekson, P. Ward.
Real-time Object-oriented Modeling.
John Wiley & Sons, 1994. ISBN 978-0-471-59917-3.