Part 1: Specification and Verification in Uppaal

José Proença & David Pereira & Eduardo Tovar
{pro,drp,emt}@isep.ipp.pt

Formal Verification of Critical Appicaitons – 2021/2022

To do

Develop the requested Uppaal specification and requirements, and produce a report in PDF, including screenshots of the requested timed automata.

What to submit

The *PDF report* **and** the developed *Uppaal models*.

How to submit using git

1. Create a private git repository in your favouring host (e.g., github or bitbucket).
2. **Name it `FVOCA22-g<group number>`.**
3. Add `pro@isep.ipp.pt` and `drp@isep.ipp.pt` as members of the group (read-permissions are enough).
4. Include all the files to be submitted in the repository.

Note that **all students should push commits.**

Deadline

Extended: 8 May 2022 @ 23:59 (Sunday)

Motor controller in a Railway system

A railway company produces signalling systems that are used in critical systems. These must provide enough evidence over their reliability and correctness over time to comply with the heavy certification processes.

This assignment is a simplification of an ongoing use-case of an European project, depicted in Fig. 1. In this use-case a critical motor that rotates left and right interacts with a controller. This controller runs on a resource-constrained device with a real-time OS. In turn, a remote dashboard sends commands and receives updates to/from the controller. The goal of this assignment is to analyse the behaviour mainly of the controller, interacting with the motor and the dashboard.

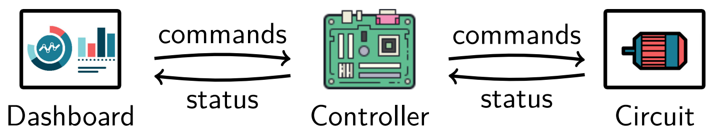

Figure 1: Architecture of the motor controller system under verification

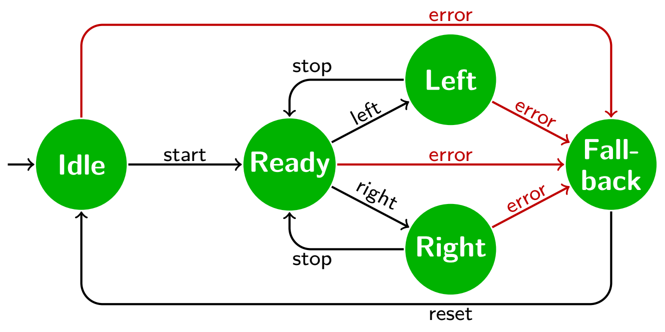

Figure 2: General behaviour of the controller component

Controller behaviour The overall behaviour of the controller is summarised in Fig. 2. Initially the controller is idle, where it must remain for at least 1000ms to perform some bootstrap self-tests. It can then receive a start command to become ready to actuate. Once it is ready, it can receive a left (resp. right) command from the dashboard, which will trigger an instruction move-left (resp. move-right) to the motor.

The controller then checks every 400ms if the motor reached the movement limit or if it detected some error. This check is made using a shared register between the motor and the controller. If the controller detects that the limit is reached between 4000ms and 5000ms after the move instruction is sent, it becomes idle and notifies the dashboard. Otherwise it raises an error (more info on this below) and goes to a fallback state. If the motor is misbehaving, it should not take more than 6000ms to raise an error since the move-instruction is sent. While in a fallback state, the controller waits for the dashboard to send a reset command to become idle again.

Raising errors Everytime an error is raised by the controller an error message must be sent to the dashboard and a stop message must be sent to the motor.

Critical implementation with 2 controllers A more complex version of this system uses 2 controllers for redundancy, both interacting with the dashboard and the motor. Every 100ms each of the two controllers should check if the other controller is in the same state, using shared variables. After 3 failed attempts this controller should raise an error and go to its fallback state.

To be able to test this pair of controllers, the dashboard can also send a fail command to any of the controllers, rendering it useless. Once a controller is faulty, it should not take more than 6000ms to raise an error since a successful *start*, *left*, or *right* instruction is sent, or a movement *limit* is reached.

Exercise 1. Write a set of 5 or more desirable requirements following the EARS approach based on the description of this system. If you want, you can make new assumptions not described above, making it explicit what is new.

Exercise 2. Model this system as an UPPAAL model **without** the extension of the second controller.

Include three concrete scenarios of this controller, i.e., assume that the dashboard performs three different fixed sequences of instructions at specific points in time. This is an open task: you should implement at least a controller, a motor, and a dashboard, and you may use extra automata if needed. You can also include new assumptions or functionality, provided that you explain them.

Your model should be parameterised by (at least) the following global constants:

- Minimum and maximum time at the idle and start phase;
- Minimum and maximum time to expect the limit to be reached;
- Periodicity to read the status of the the motor;

Describe this UPPAAL model and the 3 scenarios in your report, justifying clearly your decisions (assumptions and abstractions) made in this modelling exercise.

Exercise 3. Create an updated model based on the previous one that uses 2 controllers, as explained above.

Model three concrete scenarios as before, but now also considering fault injections.

Your model should be further parameterised by (at least) the following global constants:

- Number of attempts to incorrectly detect a wrong state before raising an error;
- Periodicity to check for the state.

Describe this new UPPAAL model and the 3 scenarios in your report, justifying clearly your decisions (assumptions and abstractions) made in this modelling exercise.

Exercise 4. Use UPPAAL's CTL logic to express and verify properties.

4.1. Specify all properties from the requirements in Ex. 1.

4.2. Express a property in UPPAAL's CTL logic for each item below. Fix a particular set of parameters (c.f. Ex. 2), and say if each property holds or not (or if it takes too much time), and explain why.

1. It is possible to move the engine 3 time to the left in less than 15000ms;
2. Whenever a fallback state is reached, it must take at least 6000ms until the motor can turn again to the left.
3. Until a scenario is finished, the system does not deadlock.

4.3. Select 5 properties from Ex. 4.1, and for each of them find different values for the parameters that satisfy and that reject them. Use the model from Ex. 3 if available, or from Ex. 2 otherwise. If a given property is always true of false, justify informally why.