Bottom-Up FSA implementation - Michele Proverbio

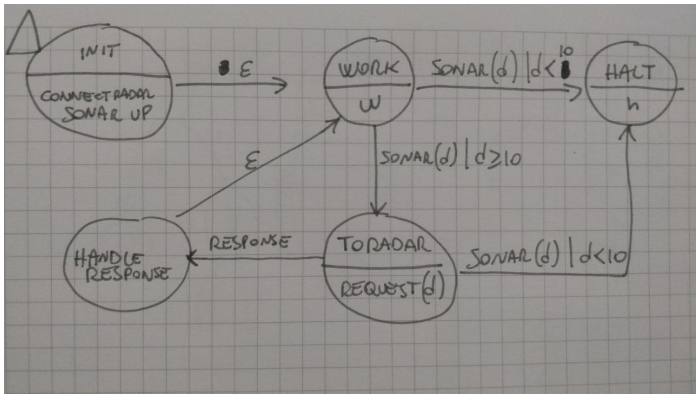
Requirements

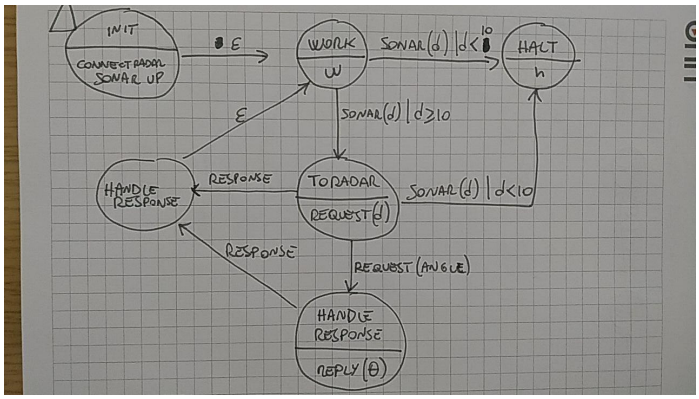
Build a bottom-up implementation of a FSA describing the behaviour of a robot with those specifics:

- the robot moves forward until an obstacle is detected
- the robot must send sonar data to a radar with a request-response pattern

Expand the previous requirements: the radar might request an angle to the robot before responding to the request sending the sonar data.

Requirement analysis





Problem analysis

```
import pyqak
from pyqak import *
from transitions import *
from context import Context, ExternalContext

ctx_radar = ExternalContext('localhost', 8020)
ctx_radar.external_actor('radar')

ctx = Context('localhost', 8030)
ctx.actor_scope('robot')


@initial
@state
def init(self, t):
```

```
    print('robot | init')
    self.transition('work', Epsilon)


@state
def work(self, t):
    print('robot | work')
    print('w')  # fire motors
    self.transition('halt', WhenEvent, 'sonar', lambda s: int(s) < 10)
    self.transition('toradar', WhenEvent, 'sonar', lambda s: int(s) >= 10)


@state
def halt(self, t):
    print('robot | halt')
    print('h')  # stop motors


@state
def toradar(self, t):
    print('robot | toradar')
    polar_msg = str(t['msg'].payload) + ',0'
    self.request('radar', 'sonar_val_req', polar_msg)
    self.transition('halt', WhenEvent, 'sonar', lambda s: int(s) < 10)
    self.transition('handle_response', WhenReply, 'sonar_val_req')


@state
def handle_response(self, t):
    print('robot | handle_response')
    self.transition('work', Epsilon)


ctx.actor_scope('sonar')


@initial
@state
def run(self, t):
    self.emit('sonar', 20)
    self.emit('sonar', 20)
    self.emit('sonar', 20)
    self.emit('sonar', 10)


pyqak.run(ctx)
```

By Michele Proverbio email: michele.proverbio@studio.unibo.it