Requirements

Design and build a software system (***basicrobot***) that is able to receive via Internet commands (represented in textual form), so that:

- the commands **w | s | h | a | d** move a *differential drive robot* (***robot***) respectively **forward | backward | stop | left | right** (**req-cmd**).
- the command **step** moves the ***robot*** forward for a prefixed time (e.g. 2 secs) (**req-step**).

Moreover, the ***robot*** must be always able to react 'immediately' to the **stop** command, by stopping any ongoing movement (**req-stop**) .

Requirement analysis

An external actor is being placed as a logical representation of the ***robot*** entity to keep the model completely technology-independent. The implementation is not attached as it is not meaningful in the scope of Requirement Analysis.

```
System basicrobot

Dispatch step   : step(T)  // req-step
Dispatch stop   : stop(X)  // req-stop
Dispatch cmd    : cmd(X)   // req-cmd

Context ctxBasicRobot   ip [ host= "localhost"   port= 8023 ]

// placeholder mock-up actor logically representing the robot.
ExternalContext ctxMockRobot  ip [ host= "10.201.116.57"   port= 8020 ]
ExternalQActor robot context ctxRealRobot

QActor basicrobot context ctxBasicRobot {
    State s0 initial {
        println("init")
    }
    Goto idle

    State idle {
        println("idle")
    }
    Transition tWork
        whenMsg step -> sStep
        whenMsg cmd -> sHandleCmd
        whenMsg stop -> sStopStep

    // REQUIREMENT: req-cmd
    State sHandleCmd {
        println("sHandleCmd")
        onMsg (cmd : cmd(X)) {
            forward robot -m cmd : cmd(X)
        }
    }
    Goto idle

    // REQUIREMENT: req-step
    // req-step problem complexity demands further analysis
    // not suitable to the Requirement Analysis
    State sStep {
        println("sStep")
        forward robot -m step : step(X)
    }
    Goto idle

    // REQUIREMENT: req-stop
    State sStopStep {
        println("sStopStep")
        forward robot -m cmd : cmd(h)
    }
    Goto idle
}
```

Problem analysis

I decided to logically divide the model in two parts: a Mind, responsible for behaviour control, and a Body that acts as a translator from meta-model defined messages to a technology dependent codification.

Body - message translator

```
System basicrobot

Dispatch cmd : cmd(X)

Context ctxBasicRobot   ip [ host= "10.201.116.57"   port= 8020 ]

QActor basicrobot context ctxBasicRobot {
    State s0 initial {
        solve(consult("basicRobotConfig.pl"))
        solve(robot(R, PORT))
        ifSolved {
            println("USING:${getCurSol(\"R\")},port=${getCurSol(\"PORT\")}")
            run itunibo.robot.robotSupport.create( myself, @R, @PORT )
        }
    }
    Goto idle

    State idle{
        println("robot idle")
    }
    Transition t0 whenMsg cmd -> handleCmd

    State handleCmd{
        printCurrentMessage
        onMsg(cmd : cmd(X)) {
            run itunibo.robot.robotSupport.move(payloadArg(0))
        }
    }
    Goto idle
}
```

Mind

The req-step problematic brings an intrinsic proactive behavior. This is solved by expanding the language expressive power, introducing a timer that emits an event when time is up. The FSA is then able to ***react*** to the event with a state transition.

```
System robotmind

Dispatch step   : step(T)  // req-step
Dispatch stop   : stop(X)  // req-stop
Dispatch cmd    : cmd(X)   // req-cmd

Context ctxMind ip [ host= "localhost"   port= 8023 ]
ExternalContext ctxBasicRobot  ip [ host= "10.201.116.57"   port= 8020 ]

ExternalQActor basicrobot context ctxBasicRobot

QActor robotmind context ctxMind {
    ["var StepTime = 0L;"]

    State s0 initial {
        println("init")
    }
    Goto idle

    State idle {
        println("idle")
    }
    Transition tWork
        whenMsg step -> sStep
        whenMsg cmd -> sHandleCmd

    // REQUIREMENT: req-cmd
    State sHandleCmd {
        println("sHandleCmd")
        onMsg (cmd : cmd(X)) {
            forward basicrobot -m cmd : cmd(X)
        }
    }
    Goto idle

    // REQUIREMENT: req-step
    State sStep {
        println("sStep")
        onMsg (step : step(T)){
            ["StepTime = payloadArg(0).toLong()"]
            forward basicrobot -m cmd : cmd(w)
        }
    }
    Transition tStop
        whenTimeVar StepTime -> sEndStep
        whenMsg stop -> sEndStep

    // REQUIREMENT: req-stop
    State sEndStep {
        println("sEndStep")
        forward basicrobot -m cmd : cmd(h)
    }
    Goto idle
}
```

Deployment

Build both BasicRobot and RobotMind into deployable zip files with the commands: gradle -b build_ctxMind.gradle distZip gradle -b build_ctxBasicRobot.gradle distZip Copy any *.pl file into the bin sub directory and execute the executable scripts.

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