LABORATORIO DI INGEGNERIA DEI SISTEMI SOFTWARE

Introduction

Our motto:
there is no code without a project, no project without problem analysis and no problem without requirements.

Requirements

Design and build a software system that makes a robot is able to walk along the boundary of a rectangular, empty room.

Requirement analysis

The **interaction with the client** made it clear that he or she associates the following meanings with nouns listed below:

- **room**: a conventional room,such as those found in all buildings
- **boundary**: the perimeter of the room, physically enclosed by solid walls (**walls**)
- **robot**: a device capable of moving by receiving commands via network, as reported in VirtualRobot2021.html

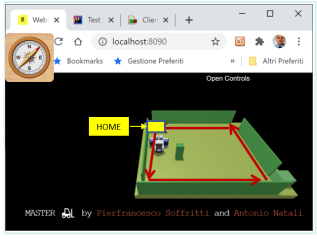
Regarding actions (verbs):

- **walk**: the robot must move forward, along the walls of the room.

Requirements

- our application must communicate with the robot via network using a web socket or through the http protocol
- the robot controlled by our application must completely around the room, moving around the perimeter

A user story

As a user, I place the robot in the HOME cell (facing south) and then activate a system that sends movement commands to the robot (via wifi network). As a user I cannot interrupt the execution: the system must terminate autonomously, once the task has been performed. At the end of the execution of the system, I expect that the robot has completed (only once) the path shown in the shown on the right.	

Verification of expected results (Test plans)

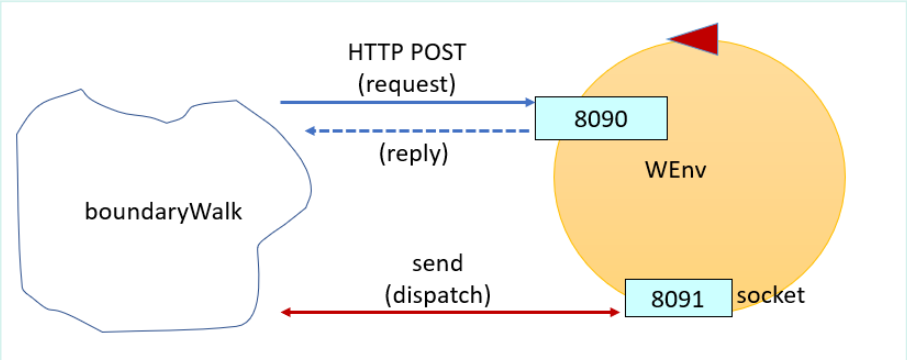
It is necessary to verify that the path completed by the robot is that which is expected	The verification of the congruence of the path must be carried out via software, without the need for the intervention of a human user.

Problem analysis

Relevant aspects

1. We need to build a **distributed system** consisting of two macro-components:
   - the (virtual) robot supplied by the client
   - our application (boundaryWalk) that sends commands to the robot to satisfy the requirements
2. The robot can be controlled via network in two different ways, as described in VirtualRobot2021.html: commands:
   - sending messages to the port 8090 with HTTP POST protocol
   - sending messages to the port 8091 using a websocket
3. Since there are numerous libraries in many programming languages that allow these command to be sent, no significant abstraction-gap is identified on the operational level.
4. This type of communication implies that both the application and the robot are connected to an internet network (wi-fi)
5. We estimate that the first prototype of the application should be able to be built in one working day (at most).

Logical architecture



Legend: legenda.pptx

Problems identified

Given the information on how to use the robot, we can prefigure tthat the solution to the problem can be represented by following algorithm:

```
the robot starts from the HOME position, facing south (DOWN)
for 4 times:
        send  moveForward  commands to the robot until the robot hits the opposite wall
        and then I send a  turnLeft  rotation command
```

Possible strategies for the TestPlan: 'infer knowledge' from commands

Based on this analysis, we can define some different testing strategies. To make the discussion easier, we introduce the following command abbreviations cril:

```
w : expresses the move {"robotmove":"moveForward", "time": 600}
s : expresses the move {"robotmove":"moveForward", "time ":600}
h : expresses the move {"robotmove":"moveForward", "time ":100}
l : expresses the move {"robotmove":"turnLeft ",  "time ":300}
r : expresses the move {"robotmove":"turnRight",  "time ":300}
```

The different testing strategies can be summarized as follows:

1. **HYPOTHESIS-TESTPLAN 1**: for each successful move, we add the move identifier to a **moves** string (initially empty).
   We set the **time** value for the **w, s** moves in order to move the robot in 'small steps', so that a **wall** is reached in **N> 1** moves.
   In this case, at the end of the application, the string must take the following form:

   ```
   moves:  "w*lw*lw*lw*l"        * : repeat 0 or more times
   ```

2. **HYPOTHESIS-TESTPLAN 2**: We set the **time** value for the **w, s** moves so that the robot travels a distance equal to its length.
   Therefore,one can measure the length of each side in **'robot-units** and check at the end of the application that the length of the perimeter expressed in' robot-units' is equal to the distance traveled by performing the indicated algorithm .
3. **HYPOTHESIS-TESTPLAN 3**: by setting the **time** for the **w, s** moves to obtain **robot-unit** moves, we incrementally build a **map of the territory** after each move. Doing so, one can know where the robot is after each command and therefore, check the path taken at the end of the application; for example (**1** represents the 'cells' traveled by the robot):

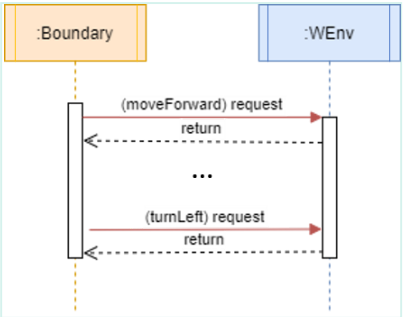
   ```
   |r, 1, 1, 1, 1,
   |1, 0, 0, 0, 1,
   |1, 0, 0, 0, 1,
   |1, 0, 0, 0, 1,
   |1, 1, 1, 1, 1, X,
   |X,
   ```

Project

The structure of our system consists of two components:

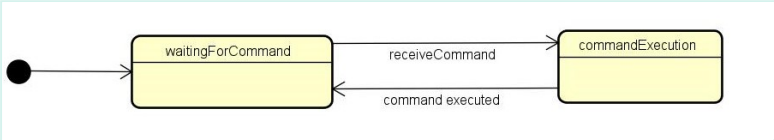
- boundaryWalk
- wenv (service)

The interaction between Boundary Walk and WEnv can take place in the following way:

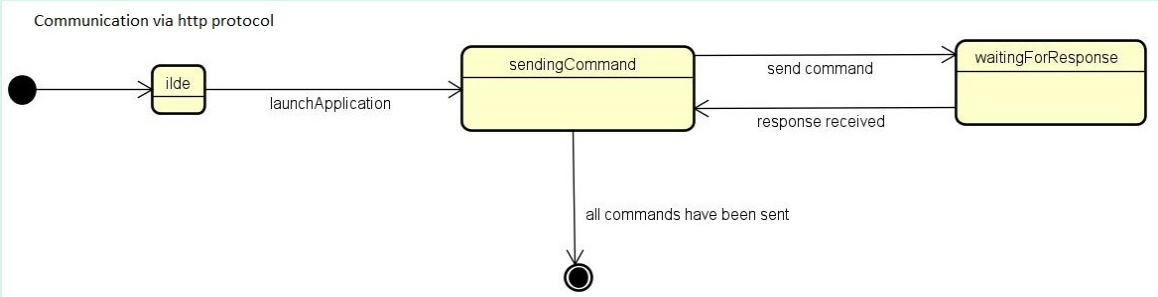


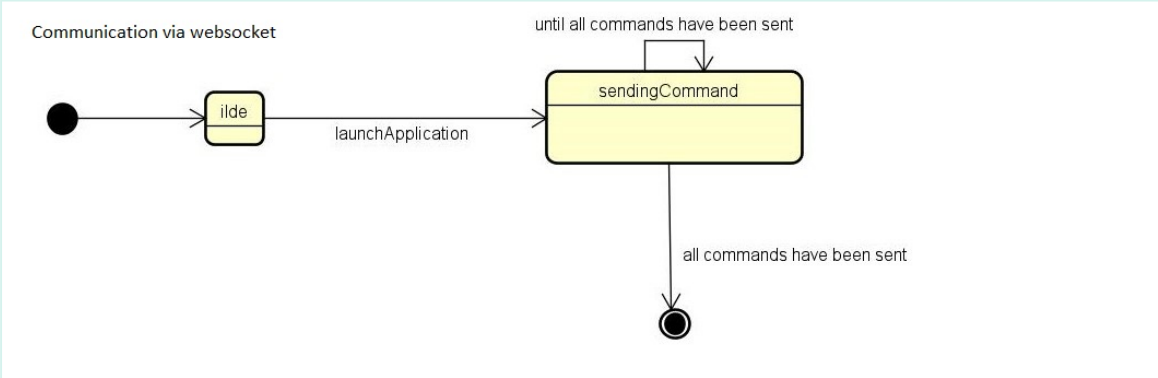
The behavior of boundarywalk and WEnv can be represented as follows (state chart diagrams):

- WEnv:



- boundaryWalk:





Legend: legenda.pptx

Testing

We will use JUNIT tests to formalize the expected behavior of the sistem. The following figure shows the test class.

```
package it.unibo.boundaryWalk;
import ...

public class TestMoveVirtualRobot {
    private MoveVirtualRobot appl;

    @Before
    public void systemSetUp() {
        System.out.println("TestMoveVirtualRobot | setUp: robot should be at HOME-DOWN ");
        appl = new MoveVirtualRobot();
    }

    @After
    public void  terminate() { System.out.println("%%%  TestMoveVirtualRobot |  terminates "); }

   @Test
    public void testBoundaryWalk() {
       String s = new String();
       String s1 = "w*lw*lw*lw*l";
       boolean moveFailed;
       int i;
        System.out.println("TestMoveVirtualRobot | testBoundaryWalk ");
       for(i = 0; i<4; i++){
            do{
                moveFailed = appl.moveForward( duration: 600);
            }while (moveFailed == false);
           assertTrue(moveFailed);
           moveFailed = appl.moveLeft( duration: 1000);
           assertTrue(!moveFailed );
           s = s + "w*l"; }
       assertTrue(s.equals(s1));
    }
}
```

Report: report.html

By studentName email: antonio.iacobelli@studio.unibo.it