Lab ISS | the project cautiousExplorer

Introduction

This case-study starts to deal with the design and development of proactive/reactive software systems that use aynchronous exchange of information.

Requirements

Design and build a software system that allow the robot described in *VirtualRobot2021.html* to exibit the following behaviour:

- the robot lives in a closed environment, delimited by walls that includes one or more devices (e.g. sonar) able to detect its presence;
- the robot has a **den** for refuge, located near a wall;
- the robot works as an *explorer of the environment*. Starting from its **den**, the robot moves (either randomly or - preferably - in a more organized way) with the aim to find the fixed obstacles around the **den**. The presence of mobile obstacles is (at the moment) excluded;
- since the robot is *'cautious'*, it returns immediately to the **den** as soon as it finds an obstacle. Optionally, it should also return to the **den** when a sonar detects its presence;
- the robot should remember the position of the obstacles found, by creating a sort of 'mental map' of the environment.

Delivery

The customer requires to receive the completion of the analysis (of the requirments and of the problem) by **Friday 12 March**. Hopefully, he/she expects to receive also (in the same document) some detail about the project.
The name of the file (in pdf) should be:

```
cognome_nome_ce.pdf
```

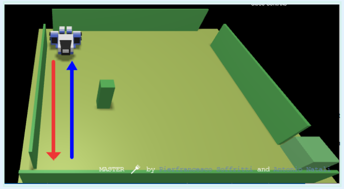
Requirement analysis

Our **interaction with the customer** has clarified that the customer intends:

- for **robot**: a device able to execute move commands sent over the network, as described in the document *VirtualRobot2021.html* provided by the customer;
- for **closed environment**: a room delimited by walls;
- for **sonar**: a device which detect and notify the presence of an object when the object is located in a certain part of the closed environment;
- for **den**: the robot starting position, located along the room boundary, and the position to which it must return when certain situations occur, as described in the requirements;
- for **fixed obstacle**: an object, placed in a certain fixed position in the room, which is an impediment to a robot movement;
- for **mental map**: a representation of the closed environment where the robot is located;

- for ***detect its presence***: to know, with the help of specific sensors, if an object is located in a certain part of the closed environment;
- for ***move***: the robot can move forward, backward and turn left and right;
- for ***find the fixed obstacles***: when the robot detects an obstacle, and its impact sensors emit collision data, as described in ***VirtualRobot2021.html*** provided by the customer;
- for ***returns immediately to the den***: as soon as the robot finds an obstacle, it must move until it returns to its starting position;
- for ***remember***: the robot should have and should be able to consult a representation of the closed environment where it is placed.

User story

As user, I put the robot in its **den** (an arbitrary position along the boundary of the closed environment); afterwards, I activate the **cautiousExplorer** application that moves the robot inside the room until it finds an obstacle or a sonar detects it. The application cannot be interrupted by any user-command. When the application terminates, I expect that the robot has returned to its den (starting position) and it has memorized its path in a 'mental map'.	

Test plan

It is necessary to verify that the robot, after encountering an obstacle or after being detected by the sonar, returns to the starting point (den).	The verification must be done via software, without the intervention of a human user.

Problem analysis

Main aspects

In the VirtualRobot2021.html the customer states that the robot can receive move commands in two different ways:

- by sending messages to the port 8090 using **HTTP POST**
- by sending messages to the port 8091 using a **websocket**

Another problem, related to the communication, arises when we consider the optional requirement *"Optionally, it should also return to the den when a sonar detects its presence"*, which says that the robot has to return to its starting position when the sonar detects it.
When a **request-response communication can be used to send movement commands to the robot**, as far as sonar is concerned, that type of communication is not suitable anymore.
The sonar requirement requires a communication that uses dispatches, since the information from the sonar arrives unpredictably and asynchronously.
In this case, considering the type of communication available, a websocket communication must be used to implement a solution for the requirement.

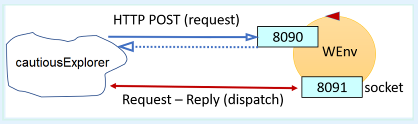
With respect to the technological level, there are many libraries in many programming languages that support the required protcols.

Logical architecture

We must design and build a **distributed system** with two software macro-components:

1. the **VirtualRobot**, given by the customer
2. our **cautiousExplorer** application that interacts with the robot

A first scheme of the logical architecture of the systems can be defined as shown in the figure (for the meaning of the symbols, see the legenda)



We observe that:

- The specification of the exact 'nature' of our *cautiousExplorer* software is left to the designer. However, we can say here that is it **not a database, or a function or an object**.
- To make our *cautiousExplorer* software **as much as possibile independent** from the underlying communication protocols, the designer could make reference to proper design pattern
- A basic test to meet the main requirements could be (the sonar and the mental map requirements are not satisfied):

```
let us define emum direction {UP,DOWN,LEFT,RIGHT}
the robot start in the den position, direction=DOWN
        1) send to the robot the request to execute the command moveForward
        and continue to do it, until the answer of the request becomes 'false'
        (collision with an obstacle);
        2) send to the robot the request to execute the moveBackward
        as many times as the command moveForward has been executed
        to bring the robot to the starting position.
```

Test plans

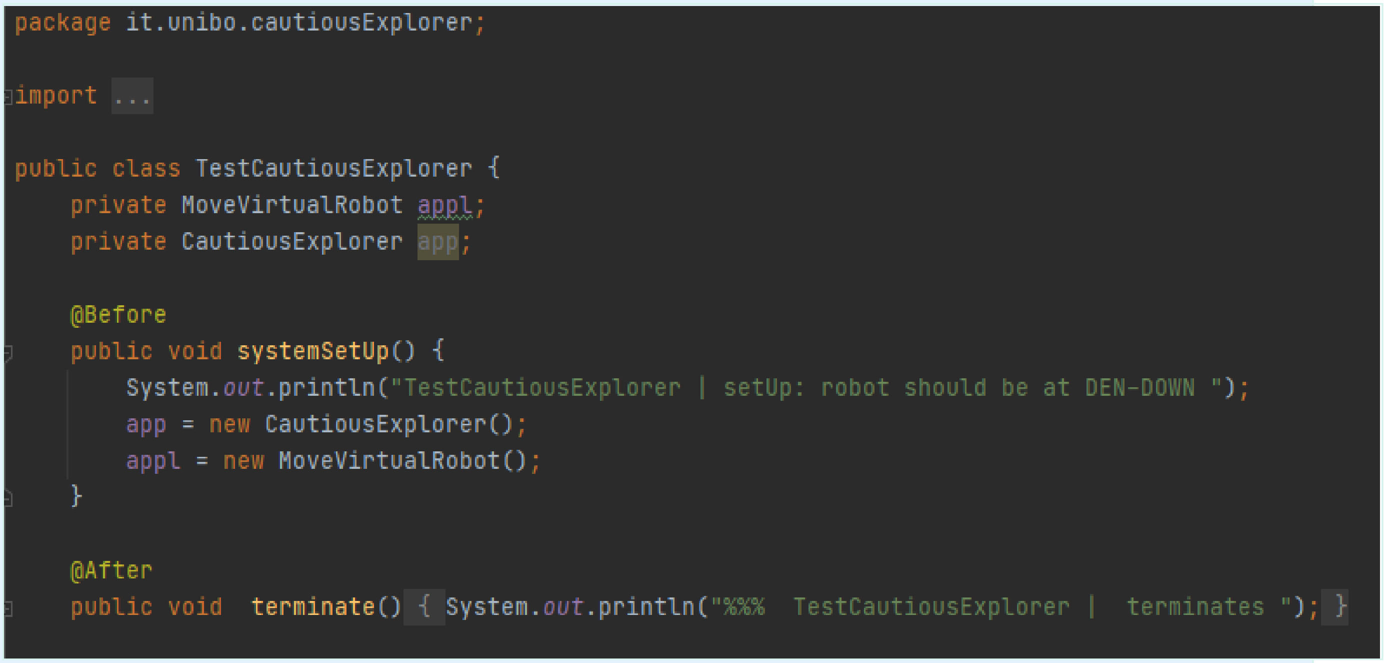
To check that the basic test was successful, we could keep track of the moves done by the robot. For example:

```
...
let us define String moves=""; and int counter=0;
        1) send to the robot the request to execute the command moveForward;
        if the answer is 'true' append the symbol "w" to moves, increase counter
        and continue to do 1) until the answer of the request becomes false;
        2) for counter times:
                send to the robot the request to execute the command moveBackward
                and append the symbol "b" to moves until the answer is false;
```

In this way, when the application terminates, the string **moves** should have the typical structure of a regular expression, that can be easily checked with a TestUnit software:

moves : "w*b*"   * : repetion N times(N>=0 and N is the same for "w" and "b")

JUnit code



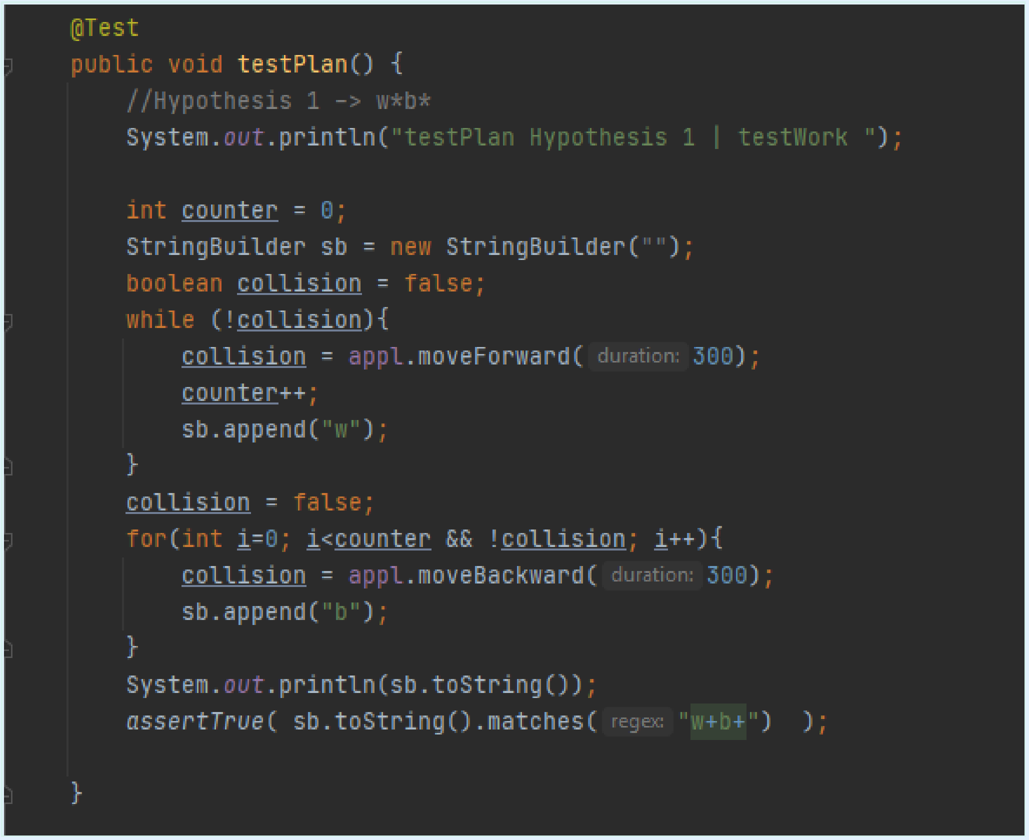
```
package it.unibo.cautiousExplorer;

import ...

public class TestCautiousExplorer {
    private MoveVirtualRobot appl;
    private CautiousExplorer app;

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

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



```
    @Test
    public void testPlan() {
        //Hypothesis 1 -> w*b*
        System.out.println("testPlan Hypothesis 1 | testWork ");

        int counter = 0;
        StringBuilder sb = new StringBuilder("");
        boolean collision = false;
        while (!collision){
            collision = appl.moveForward( duration: 300);
            counter++;
            sb.append("w");
        }
        collision = false;
        for(int i=0; i<counter && !collision; i++){
            collision = appl.moveBackward( duration: 300);
            sb.append("b");
        }
        System.out.println(sb.toString());
        assertTrue( sb.toString().matches( regex: "w+b+")  );

    }
```

By Gianluca Soavi email: gianluca.soavi@studio.unibo.it