VacuumCleaner - ISS M 2022

Introduction

Homework for 25/04/2022.

Carry out the requirement analysis, project analysis and test plan of a software that manages the movements of an automatic vacuum cleaner.
The vacuum cleaner is represented by a virtual robot that receives messages over HTTP or websocket and move/behave accordingly to those. The space in which the robot can move is surrounded by 4 walls that makes the room rectangular, and there might be static obstacles with which the robot collides.

Requirements

Make a **software application** that gives a specific sequence of moves to a robot, so that it moves covering the whole surface of a rectangular room (i.e. the robot must travel accross every piece of the room).
The room is surrounded by 4 walls, and the robot has sensors to detect when a collision with an obstacle occurs.

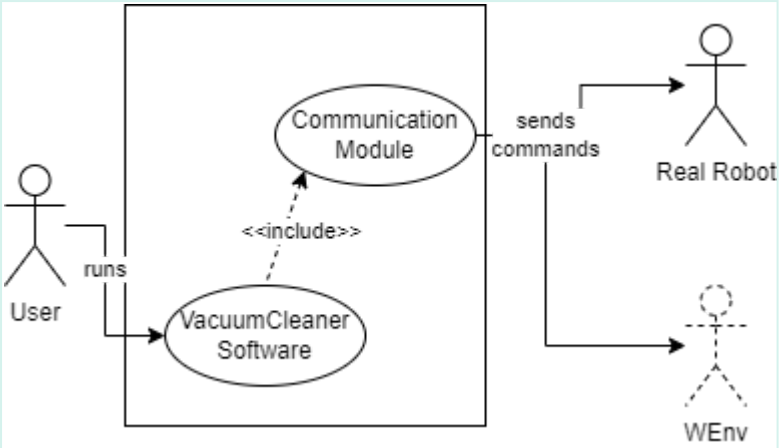
Requirement analysis

The customer wants a software application which, communicating via HTTP or websocket with a module provided by him (Virtual Environment - **WEnv**), sends a sequence of commands that the robot must perform in order to "clean the room completely".
By "cleaning the room completely" the customer means that the robot must travel through the whole surface of the room, which is surrounded by 4 static walls.

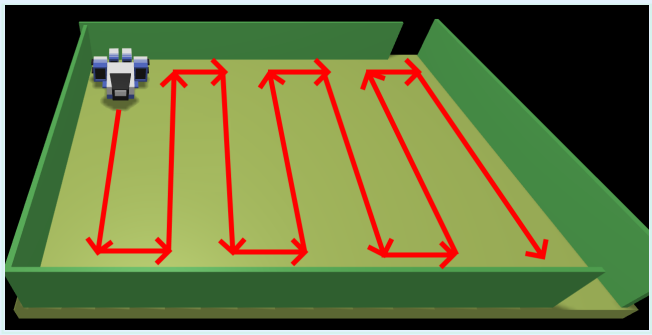
Therefore, it is necessary to create a ***VacuumCleaner*** component which sends a sequence of commands to the WEnv, which makes the robot move according to those specific instructions, by covering the whole area of the room.

- The entity involved for the moment has no nature, and so we consider it to be a **generic entity**.
- The application and the virtual robot can run on different devices, but initially it's sufficient to work locally on a single machine.
- There are not technologic constraint but it's possible to use libs provided by the customer/company (**unibolibs**) that can be useful to complete the task (e.g. the communication module).
- In order to being able to implement the communication properly, it is necessary to respect the interaction model with which the WEnv is able to communicate, as described in this document.

Use Case



User Story

The user places the robot in a fixed and well known spot (home position, e.g. the upper-left corner), then he runs the VacuumCleaner application. The application will communicate with the WEnv or the real robot, making it move following a certain pattern that we are sure is effective for cleaning the whole room, without leaving dirty (blank) spots.	

Problem analysis

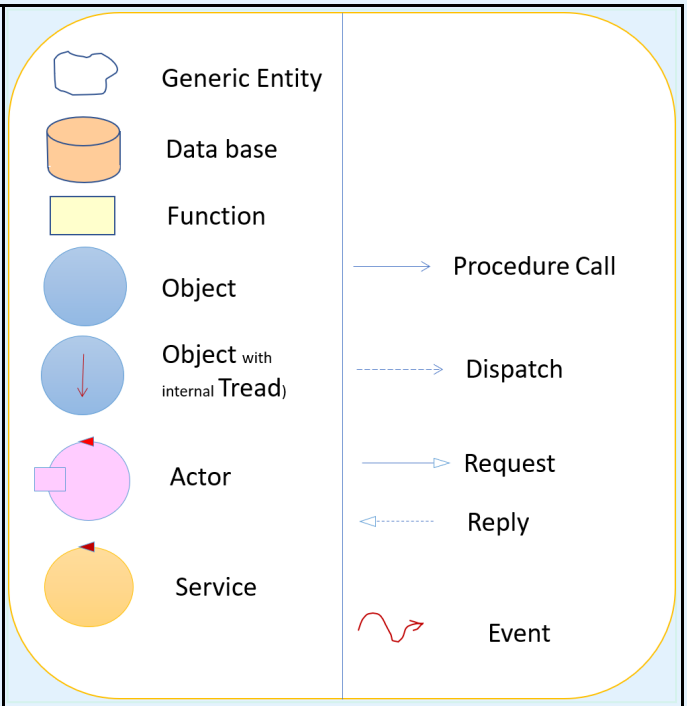
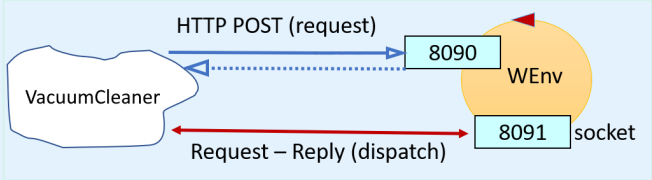
The requirements locate several **potential issues**:

1. **COMMUNICATION**: according to VirtualRobot2021, provided by the customer, the robot can receive move commands in two different ways:
   - by sending **HTTP POST** requests on port 8090;
   - by sending messages on a **websocket** on port 8091.

   Which one should we use? For simplicity, we initially choose to **communicate synchronously**, since we expect a response to each message sent to the robot (to check if a collision occurred). If in the future the customer will ask for an additional stop/resume mechanism, we would then be bound to use an asynchronous communication, since we will want to be able to stop the robot at any time.
2. **PATH AND ALGORITHM**: the choice of the path the robot must follow. For simplicity, for the moment we take for granted that there are no obstacles inside the room, except the 4 walls that surround it. Moreover, let's suppose that the robot starting position is fixed and well known, for example the upper-left corner, which we can call "den". In this way, the path choice becomes trivial, as it is possible to adopt a "**zig-zag pattern**": the robot moves forward until it encounters an obstacle (i.e. the wall), then rotates by 90° in order to face the area of the room yet to be explored (in our example that direction is the right). Then the robot should move by a step, and rotate again by 90° in the opposite direction. Then it can start moving forward again until it collides with a wall and so on.
   In this regard, we can exploit the robot collision detection mechanism, to know when a wall has been collided with, and rotate the robot accordingly.

3. **STATE**: we must ensure that the robot doesn't leave blank spots. It is possible to dynamically build a data structure, such as a **map**, marking the spots where the robot has already passed. Assuming that the room can be represented in a Cartesian plane, we can think that each space is identifiable by 2 coordinates X and Y, and is equal to the dimensions of the robot (length and width). As the robot progresses, we insert a particular element in the map that indicates the status of the box, for example a different character wheter it has already been visited or is still to be visited. When the robot hits the first bottom wall, the we know the map row number.
   Once the map is completed, we can simply check if each element of the map is filled with the "visited" character, to test the correctness of the solution.
4. **DISTANCE**: it is necessary to calculate the distance to be covered by the robot for each step. This way we can make the robot move for an amount of time which corresponds exactly to the robot length.

Logical Architecture

Legend	
We must design and build a distributed system with 2 software macro-components: 1. the Virtual Robot, given by the customer 2. the ***vacuumCleaner*** application that interacts with the robot A first scheme of the logical architecture of the systems can be defined as shown in the figure.	

Test plans

To test the behaviour of the robot we can track its movement by saving each completed command into a String. For example, if the robot moves forward for 3 times and then rotates left, we will have

a string equal to "fffl".

```
Moves legenda:
moveForward = f, rotateLeft = l, rotateRight = r
```

```
public class TestVacuumCleaner {
    private final static int DOWN = 0, RIGHT = 1, UP = 2, LEFT = 3;
    private final String localHostName = "localhost";
    private final int port = 8090;
    private final String HttpURL = "http://" + localHostName + ":" + port + "/api/move";

    private Interaction2021 conn;
    private String cleaned;
    private int curX, curY, heading;

    @Before
    public void init() {
        // setup string
        cleaned = "";

        // set robot position to DEN
        curX = 0;
        curY = 0;
        heading = DOWN;

        // enstablish connection
        conn = HttpConnection.create(localHostName + ":" + port);
    }

    @Test
    public void testCleaning() throws Exception {
        boolean obstacle = false;
        String answer = "";
        while(true)
        {
            while(!obstacle)
            {
                answer = conn.request( "{\"robotmove\":\"moveForward\", \"time\":50}");
                obstacle = answer.contains("collision");
                cleaned += obstacle ? "f" : "";
            }
            obstacle = false;
            if(heading == DOWN)
            {
                // rotate left by 90°
                conn.request( "{\"robotmove\":\"turnLeft\", \"time\":300}");
                cleaned += "l";

                // step forward
                answer = conn.request( "{\"robotmove\":\"moveForward\", \"time\":50}");
                if(answer.contains("collision"))
                    break;
                else cleaned += "f";

                // rotate right by 90°
                conn.request( "{\"robotmove\":\"turnRight\", \"time\":300}");
                cleaned += "r";

                continue;
            }
```

```
            if(heading == UP)
            {
                // rotate right by 90°
                conn.request( "{\"robotmove\":\"turnRight\", \"time\":300}");
                cleaned += "r";

                // step forward
                answer = conn.request( "{\"robotmove\":\"moveForward\", \"time\":50}");
                if(answer.contains("collision"))
                    break;
                else cleaned += "f";

                // rotate left by 90°
                conn.request( "{\"robotmove\":\"turnLeft\", \"time\":300}");
                cleaned += "l";

                continue;
            }

        }

    }

    @After
    public void printResults() {
        System.out.println(cleaned);
    }
}
```

Project

Testing

Deployment

Maintenance

By **Michele Righi**

ISS repo: mikyll/righimichele/iss2022/

Contacts:
e-mail: michele.righi5@studio.unibo.it
GitHub: mikyll
LinkedIn: Michele Righi