LABORATORIO DI INGEGNERIA DEI SISTEMI SOFTWARE

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:

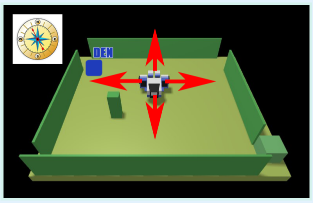
- the robot lives in a closed environment, delimited by walls that includes one or more devices (e.g. sonar) able to detect its presence;
- the robot has a **den** for refuge, located near a wall;
- the robot works as an *explorer of the environment*. Starting from its **den**, the robot moves (either 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.

Requirement analysis

EXPLANATION OF THE MEANING of the words or expressions used by the customer to define the requirements.

- **environment:** a rectangular room, bounded by walls.
- **den:** a specific starting position located near a wall or corner.
- **the robot moves in organized way:** The robot's movements can be based on established logic.
- **fixed obstacles:** In the environment (room) there are some physical obstacles that the robot cannot cross. walls are also considered obstacles.
- **mental map:** An abstract object in which the obstacle positions are saved and which is updated every time the robot finds a new obstacle. The robot must have access to this information.

USER STORY

Considering a rectangular room as the environment, the user places the robot in its den *(e.g. the blue blox in the figure)*. The **cautiousExplorer** application is executed, the robot starts to move around the room, returning to the den whenever it encounters a wall or obstacle and saving its position in the *"mental map"*. When the application stops, the robot should be at the den and the mental map should show the positions of the obstacles encountered and the path walked by the robot.	

Problem analysis

EVALUATION OF PROBLEMS and THE ABSTRACTION GAP

Random or organized movements

The robot, during exploration, can move either randomly or following an established logic:

- **Random movements:** it's possible to choose the robot's movements randomly (by taking them from the set of moves the robot can perform).
    - **Advantages:** By executing the application several times, the robot may find new obstacles.
    - **Disadvantages:** This option does not optimise the time spent searching for obstacles.
- **Organized movements:** it is possible to move the robot according to an established movement pattern.
- **Advantages :** this option minimises the number of moves (and time cost) required to find obstacles around the den.
- **Disadvantages:** by running the application several times, the robot will always find the same obstacles The use of resources is greater.

Abstraction gap evaluation:
Most programming languages (e.g. Java) have libraries that allow both options to be implemented without problems.

Return to the den

It is necessary that the robot does not encounter any obstacles on its way back to the den. This capability requires that the robot remembers the route it took until it encountered the obstacle, and retraces it in reverse.

> **Abstraction gap evaluation:**
> To achieve this feature the use of all main programming languages does not lead to an abstraction gap (e.g. It's possible to use a string to remember the moves performed).

Mental map

The robot should save the position of obstacles it encounters. It is possible to create a data structure (e.g. a matrix) in which each element corresponds to a portion (cell) of the environment and some symbols are used to indicate whether a cell is free or occupied by an obstacle.

> **Abstraction gap evaluation:**
> In according to the technical specifications of the robot *VirtualRobot2021.html*, it is possible to set the duration of a move so that the robot moves only one cell per move, so there is no abstract gap.

Interaction with the robot

The robot can receive move commands in two different ways:

- by sending messages to the *port 8090* using **HTTP POST:**
  - the usage of HTTP implies a synchronous exchange of messages and therefore the application waits to receive the response to the previous command before sending a new command.
- by sending messages to the *port 8091* using a **WEBSOCKET:**
  - The usage of WS implies an asynchronous exchange of messages and therefore the application does not wait before sending a new command (it is necessary to impose a waiting time by software). This implies that if the duration of a move exceeds the wait time, the status of the next move will be "*endmove: notallowed*"
  - Using a WS, however, the client will receive information from the sonar, so that the optional requirement could be satisfied (*see the* ***requirements***).

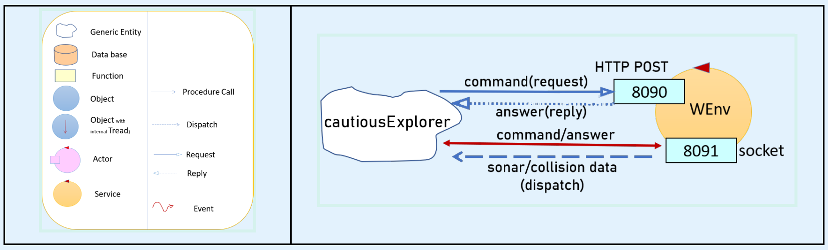
> **Abstraction gap evaluation:**
> The main programming languages have libraries to support both protocols.

LOGICAL ARCHITECTURE

We want to obtain a distributed system consisting of two entities:

- The **virtual robot**
- The application that the constumer has commissioned (**cautiousExplorer**).

At this stage we can consider our application as a generic entity that communicates with WEnv to make the robot move.



- HTTP protocol is more simple to adopt, but it works only in synchronous mode.
- The usage of Websocket permits to receive spontaneously (without request) informations from the virtual robot (*sonar* and *collision data*).

It would be useful to implement the system so that it is independent of the communication protocol adopted, because it could be changed in the future.

It is possible to express in a generic way what the robot will have to do to satisfy the requirements:
We set the parameter time for the moves **w** and **b** in order to obtain displacements of a length equal to the length of the robot (*robot-unit* or *cell*)
Let us define the moves that the robot can perform:

- **w**: moveForward
- **b**: moveBackward
- **l**: turnLeft
- **r**: turnRight

String **path** can contains only **w|b|r|l**
Starting from **den**, facing south.

```
String path= ""; String pathTrue=""; Matrix map;

For n times:
1. randomly or following a pattern the content of path is produced.
2. For each character x in path:
   ○ send to the robot the request to execute the command x
   ○ if the answer is "true":
      ▪ append the symbol x to path
      ▪ reading next x and continue 2);
```

```
    ○ if the answer is "false":
        ▪ add the position of obstacle to map
        ▪ exit from 2)

3. For each character x in pathTrue:
    ○ send to the robot the request to execute the opposite command of x
    ○ reading next x and continue 3);
```

Test plans

It is necessary to obtain an automatizable **TestPlans**, in this way we could verify the correct operation of the application automatically.

Test plan

We could use the *"mental map"* that the application produces to check that the robot has returned to the **den** after finding an obstacle. The *mental map*could be a matrix similar to the following.

The *mental map* could be a matrix similar to the following.
|r, 1, 1, 0, 0,
|0, 0, 1, 0, 0,
|0, 0, 1, 0, 0,
|0, 0, x, 0, 0,
|0, 0, 0, 0, 0,

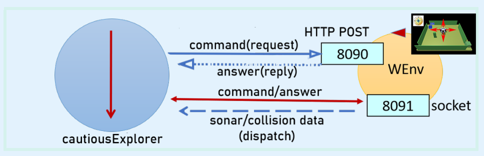
Legend:
1: cells crossed by the robot
0: cell not crossed by the robot
r: current robot position
x: obstacle

Once the application has been completed, a **test plan** could consist of checking the following conditions:

- the current position of the robot is in the **den**
- the cells crossed by the robot (with the symbol **1**), make a path from the den to the obstacle and back again.

Project

Nature of the application component

The **cautiousExplorer** application is a conventional Java program, represented in the figure as an object with an internal thread.	 HTTP POST command(request) 8090 answer(reply) WEnv command/answer 8091 socket sonar/collision data (dispatch) cautiousExplorer

Layered architecture

The application can communicate with the robot either using the HTTP protocol or Websocket. It is important to make the functioning of the application independent of the protocol used. This can be achieved using a layered architecture, in particular by introducing an additional level referring to the **communication layer** of the **buondaryWalk project**.

Testing

Deployment

Maintenance

By studentName email: student@studio.unibo.it