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

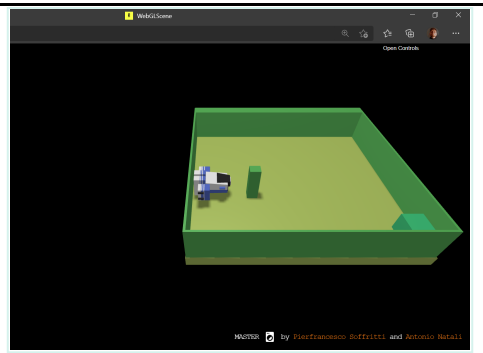
The interation with the client made it clear that he associates the following meaning with nouns:

- **robot**:a device capable of moving by receiving commands by the network as reported in ***VirtualRobot2021.html***
- **closed enviroment**:space surrounded by walls with no way out
- **den**: the place where the robot starts and then returns
- **walls**: constitute the perimeter of the environment
- **obstacles**: any element material or non-material which opposes and constitutes an impediment to an action or an activity or a movement
- **cautious**:it means that the robot has to go back to his den

For the actions(verbs):

- **detects**: the robot passed in front of the sonar

A first user story

As a user, I place the robot in its det (facing back to the wall) 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 is done. At the end of the execution of the system, I expect that the robot has found an obstacle and returned to its det.	

Problem analysis

We highlight that:

1. In the VirtualRobot2021.html: commands 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**

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

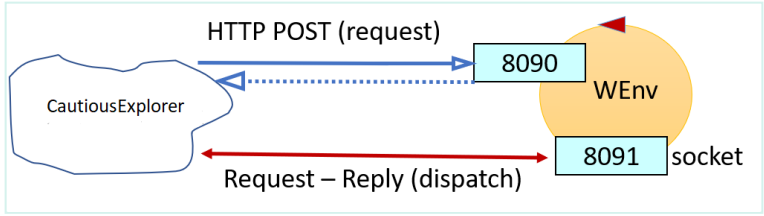
However, the problem does introduce an **abstraction gap at the conceptual level**, since **the required logical interaction** is not always a ***request-response***, **regardless of the technology** used to implement the interaction with the robot.

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 with a *request-response* pattern

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 undelying communication protocols, the designer could make reference to proper ***design pattern***, e.g. **Adapter**, **Bridge**, **Facade**.
- It is quite easy to define **what the robot has to do** the meet the requirements:

```
the robot start in the DEN position, direction= back to the wall:
        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'
        or until the sonar does not detect the presence of the robot and starts to sound
        2) send to the robot the request to execute the moveBackward to the den
```

Test plans

To check that the application fulfills the requirements, we could keep track of the moves done by the robot. For example:

```
...
let us define String moves="";
        1) send to the robot the request to execute the command moveForward;
```

```
        if the answer is 'true' append the symbol "w" to moves and continue to do 1);
        2) when the answer of the request becomes 'false',
        send to the robot the request to execute the command moveBackward and append the symbol "b" to moves
        and continue to do 2) until the robot returns to the den
```

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)
```

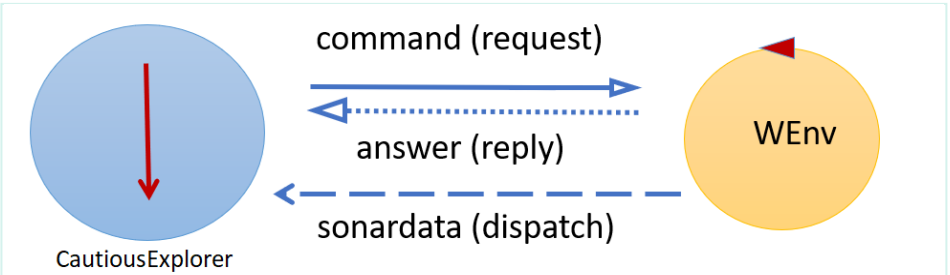
Moreover the number of **w** must correspond to the number of **b** in order to verify the requirements.

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.
Our CautiousExplorer application should:

- send a *request* to WEnv for the execution of a robot-move command
- handle the *reply* sent by WEnv to the robot-move command request
- handle the information possibly sent by WEnv to boundaryWalk as a *dispatch* carrying the distance detected by the sonar

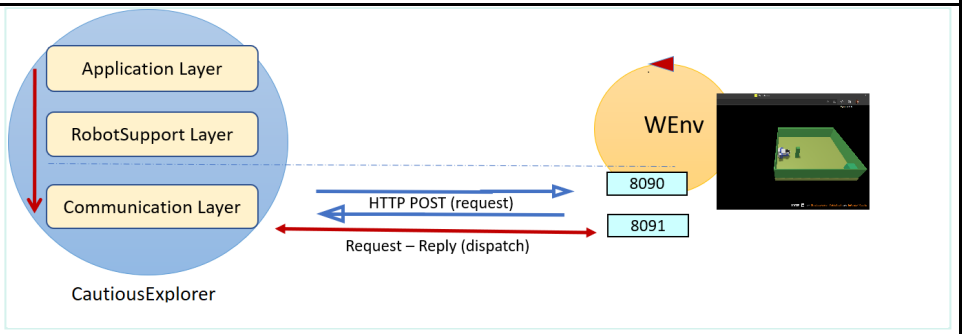


Zoom in the CautiousExplorer architecture

The code of the program is structured according to a conventional *layered architecture*, which is the simplest form of software architectural pattern, where the components are organized in *horizontal layers*.
We introduce the following layers:

1. **Application layer**:

2. **RobotSupport layer**: this layer provides a support that helps the design and the implementation of robot-based applications.



3. **Communication layer**: this layer provides a support for using the protocols required by the application.	

Testing

Deployment

Maintenance

By Antonio Franzese email: antonio.franzese4@studio.unibo.it