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.

Requirement analysis

The interaction with the customer has clarified that the customer intends:

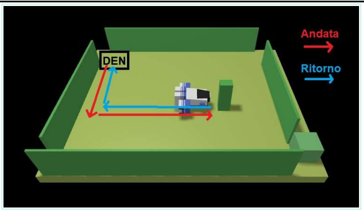
- for **closed environment**: a conventional (rectangular) room of an house, delimited by walls;
- for **sonar**: device in the environment which is able to detect robot presence;
- for **robot**: a device able to execute move commands sent over the network, as described in the document *VirtualRobot2021.html* provided by the customer;
- for **find an obstacle**: bumping in an obstacle, the robot is not able to perform a specific movement ;
- for **den**: a refuge located near the wall where the robot returns when it finds an obstacle
- for **obstacle**: an object in a fixed position of the room or a wall of the room;

The customer does not impose any requirement on the programming language used to develop the application.

User story

As user, I put the robot in its den; afterwards, I activate the **cautiousExplorer** application that moves the robot in the room with the aim to find the fixed obstacles. As soon as the robot finds an obstacle, it return immediately to the den

When the application terminates, the itinerary done by the robot must be that shown in the figure and a proper *TestPlan* should properly check this outcome.



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.

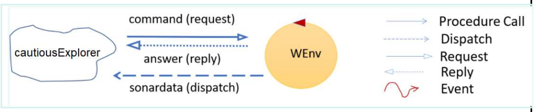
Logical architecture

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. 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 cautiousExplorer as a *dispatch* carrying the distance detected by the sonar,in order to make the robot come back to den



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 patterns as Facade and Factory

- Now we define what the robot has to do in order to reach the goal:

```
initialize empty stack
the robot start in the DEN position
do
        1) send to the robot the request to execute a command
        2) push the move direction in the stack
while answer of the request becomes 'false'
do
        1) pop the move direction from the stack
        2) send to the robot the request to execute a command in the opposite direction of the popped element
while stack is not empty
```

Test plans

To check that the application fulfills the requirements, we could run the previous code and save the first itinerary,the one who makes the robot bump into an obstacle and then coming back to den and then we could redo this itinerary and we can check if redoing the same route it finds the obstacle.

```
initialize two empty stack, firstStack and secondStack
the robot start in the DEN position
do
        1) send to the robot the request to execute a command
        2) push the move direction in firstStack and secondStack
while answer of the request becomes 'false'
do
        1) pop the move direction from the firstStack
        2) send to the robot the request to execute a command in the opposite direction of the popped element
while stack is not empty

do
        1) pop the move direction from the secondStack
        2) send to the robot the request to execute that command
        3) if the answer of the request becomes 'false' and stack is empty we found the previous obstacle meaning that the application fulf
while stack is not empty
```

Project

Testing

Deployment

Maintenance

By Dario Battaglia email: dario.battaglia2@studio.unibo.it