

Introduction

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

Requirements

Design and build a software system that allow the robot described in [VirtualRobot2021.html](#) to exhibit 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 requirements 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

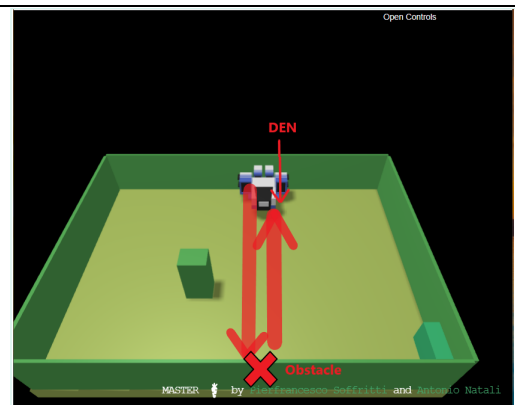
In the **interaction with the customer** we clarified that the customer intends:

- for **robot**: a device able to execute move commands sent over the network and communicate collision, as described in the document [VirtualRobot2021.html](#) provided by the customer;
- for **lives**: robot moves and **work** in this **closed environment**
- for **closed environment**: a conventional (rectangular) room of an house, physically delimited by solid **walls**;
- for **den**: a secure position, near one of the four **wall**, where the robots starts his **work** and where he returns after he **finds an obstacle**;
- for **cautious**: describes that the robot will return to the den position when **finds an obstacle**;
- for **finds an obstacle**: robot detect a collision; Robots detect a collision like specified in [VirtualRobot2021.html](#);
- for **fixed obstacles**: obstacle that ara fixed in place, not moving
- for **mobile obstacles**: obscales that changes their position (there will be not)
- for **return to the den**: come back to the then position, from wherever he is, avoiding obstacles;
- for **devices(sonar)**: devices located out of the wall that can **detect** the **robot** presence in a strict line in front of them
- for **detect**: sends message like specified in [VirtualRobot2021.html](#) provided by the customer;
- for **mental map**: data structure that represent the position of the **obstacles** in the room, referencing the **den**;
- for **work**: robot moving (randomly or in a organized way) searching all possible obstacles in the enviroment around the den;

User Story (without Sonar)

As user, I put the robot in its **den position** orientend with its back to the wall (south oriented); Afterwards, I activate the **cautiosExplorer** application that moves the robot towards until it finds a obstacle, memorize it, and go backwards returning to the **den position**.
The application cannot be interrupted by any user-command.

When the application terminates we epect that the coordinates of the obstacle founded by the robot correspond to te position of the point of the wall we indicated in the image and the path done by the robot is the same indicatend in the image. A proper [TestPlan](#) should properly check this outcome.

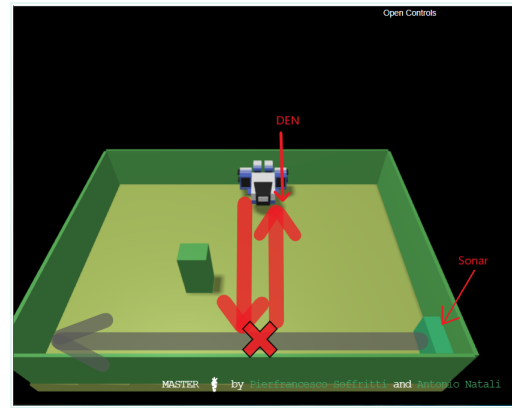


User Story (with Sonar)

As user, I put the robot in its **den position** orientend with its back to the wall (south oriented); Afterwards, I activate the **cautiosExplorer** application that moves the robot towards until it will be detected by the sonar, memorize it, and go backwards returning to the **den position**.
The application cannot be interrupted by any user-command.

When the application terminates we epect that the coordinates of the obstacle founded by the robot correspond to te position of the point of the wall we indicated in the image and the path

done by the robot is the same indicated in the image. A proper [TestPlan](#) should properly check this outcome.



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

Problem analysis

We highlight that:

- 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](#)

And we can receive information about Sensor and Sonar only from the websocket in an asynchronous way.

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

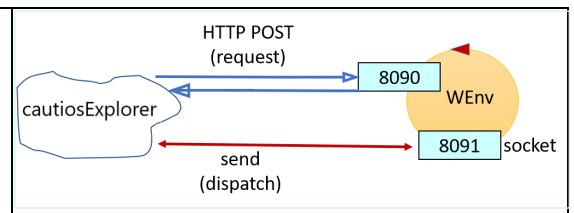
However, the problem does introduce an **abstraction gap at the conceptual level**, since the **required logical interaction** is not always a [request-response](#). Wenv can [dispatch](#) some information at any time of the execution, even while the application is executing commands ([request-response](#)) on the robot.

3. Logical Architecture

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

- the [VirtualRobot](#), given by the customer
- our [cautiosExplorer](#) application that interacts with the robot with asynchronous communications

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 [cautiosExplorer](#) software is left to the designer. However, we can say here that it is **not a database, or a function or an object**. And to properly handle the **asynchronous communication** on the websocket the designer could make reference to the [Observer](#) design pattern or a [Concurrency pattern](#) like [Monitor Object](#).
- To make our [cautiosExplorer](#) software **as much as possible independent** from the underlying communication protocols, the designer could make reference to proper [design pattern](#), e.g. [Adapter](#), [Bridge](#), [Facade](#).
- As we are in a [VirtualEnvironment](#) request of execution of a command to the robot should reference the [request-response](#) communication pattern, while asynchronous messages sends by the robot representing information on Sensor and Sonar should reference the [dispatch](#) communication method

We can define in a simple way a possible behaviour of the robot to meet the requirements:

```
Let us define the den Position in the map relative to axis (x,y);
The robot start in the den position, oriented always back to the wall;
- Send to the robot the request to execute the command moveForward until returns false or
  the Forward Collision Sensor communicate a collision or a Sonar communicate a true detection;
  and count the moveForward number of executions;
- Save the number of moveForward execution and the type of Obstacle founded (solid or Sonar);
- Send request of moveBackward until robot returns a collision
- Return the obstacle position and the type of obstacle
```

Test plans

To check that the application fulfills the requirements, we could keep track of the moves done by the robot. For example, supposed we define a the time of execution as the robot will move a distance pair to its length ([RobotUnit](#)). We also divide the map of using the [RobotUnits](#) (length of the robot) and we know the obstacle position:

```
...
set int obstacleD, denX, denY to our real value (obstacle distance expected in Robot unit, den position x and y)
let us define String moves=""; int moveW=0, moveB=0
for 4 times:
  1) send to the robot the request to execute the command moveForward;
  if the answer is 'true' append the symbol "w" to moves, increment moveW and continue to do 1);
  2) when the answer of the request becomes 'false' or the Sonar communicate a detection go to 3)
  3) send to the robot the request to execute the command moveBackward;
  if the answer is 'true' append the symbol "b" to moves, increment moveB and continue to do 3);
end;
```

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**"      *: repetition N times(N>=0)
```

Also **moveW** has to be equal to **moveB**, easy to check:

```
moveW==moveB
```

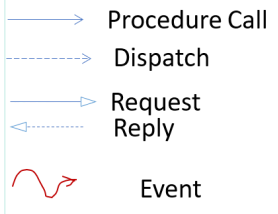
Also **moveW** has to be equal to **obstacleD** if the robot finds the right obstacle position:

```
moveW==obstacleD
```

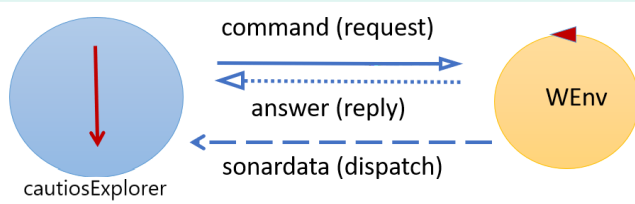
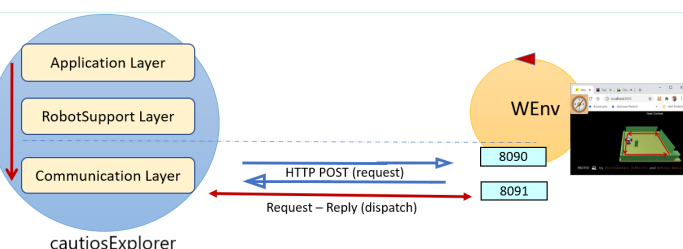
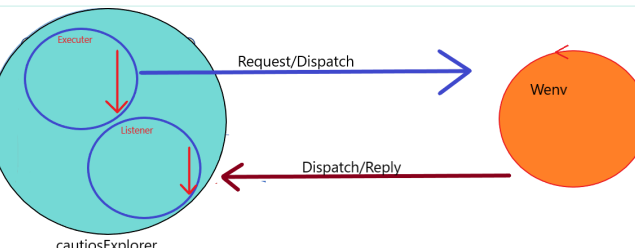
Project

Abstraction: Towards message-based interaction

Since we intend to develop distributed applications and to **gradually perform a transition** from component-interaction based on **procedure-calls** to the usage of **distributed** components that interact using **messages**, let us introduce here a more general representation of messages.

<p>From the point of view of the application designer, we do introduce the following terminology:</p> <ul style="list-style-type: none">• Dispatch: a message 'fire and forget': the sender does not expect any answer.• Request: the sender expects an answer sent using a Reply.• Invitation: the sender expects an ack.• Event: the sender 'emits information' without specifying any receiver.	
---	--

Project Architecture

<p>Nature of the application component</p> <p>The cautiosExplorer application is a conventional Java program, represented in the figure as an object with an internal thread. Our cautiousExplorer application should</p> <ul style="list-style-type: none">• 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 sent by WEnv to cautiousExplorer as a dispatch carrying the detection by the sonar and collision by sensors	
<p>Zoom-In of the cautiousExplorer architecture</p> <p>Referencing the Project of BoundaryWalk 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 <i>horizontal layers</i>.</p> <p>We introduce the following layers:</p> <ol style="list-style-type: none">1. Application layer2. 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.	
<p>Application Layer Structure</p> <p>The application layer will be structured with two THREADS to handle asynchronous communications:</p> <ul style="list-style-type: none">• One requesting commands to te Robot (Wenv)• A listener handling all the messages received from the robot(Wenv) <p>This way the application can handle messages coming from the Wenv at any time, handling response of the robot and information messages from the Sonar or Sensors to REACT suddenly and in the right way all the messages coming from Wenv without any inconsistency or wasting of time.</p>	

Testing

Deployment

Maintenance

