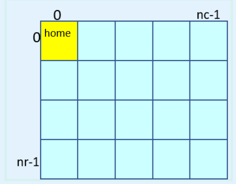
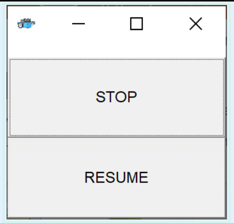
Lab ISS | the project resumableBoundaryWalker

Introduction

This case-study starts to deal with the design and development of proactive/reactive software systems which work under user-control.

Requirements

Design and build a software system (named from now on 'the application') that leads the robot described in *VirtualRobot2021.html* to walk along the boundary of a empty, rectangular room under user control.
More specifically, the **user story** can be summarized as follows:

the robot is initially located at the **HOME** position, as shown in the picture on the rigth	
the application presents to the user a **consoleGui** similar to that shown in the picture on the rigth	
when the user hits the button **RESUME** the robot **starts or continue to walk** along the boundary, while updating a **robot-moves history**;	
when the user hits the button **STOP** the robot stop its journey, waiting for another **RESUME** ;	
when the robot reachs its **HOME** again, the application *shows the robot-moves history* on the standard output device.	

Delivery

The customer **hopes to receive** a working prototype (written in Java) of the application by **Monady 22 March**. The name of this file (in pdf) should be:

```
cognome_nome_resumablebw.pdf
```

Requirement analysis

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

- **room**: a conventional (rectangular) room of an house;
- **boundary**: the perimeter of the room, that is physically delimited by solid **walls**;
- **robot**: a device able to execute move commands sent over the network, as described in the document *VirtualRobot2021.html* provided by the customer;
- **Home**: the position where the robot starts;
- **consoleGui**: **GUI** stands for **Graphical User Interface**. GUI permits users to use the graphics to interact with an operating system;
- **robot-moves history**: a map that traces all the movements of the robot;
- **journey**: path of the robot;

For the verbs:

- **walk**: the robot moves forward, close to the room walls;
- **reachs its HOME again**: the robot complete the boundary retourning in the **Home** position;

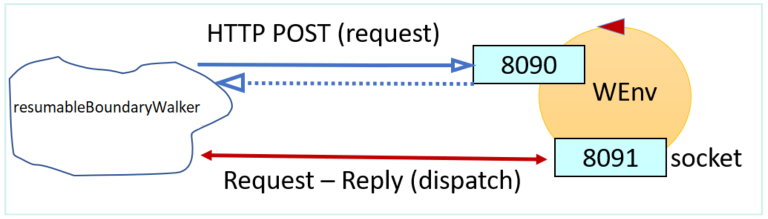
Problem analysis

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 **resumableBoundaryWalker** 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)	 HTTP POST (request) 8090 WEnv resumableBoundaryWalker 8091 socket Request – Reply (dispatch)

We observe that:

- The specification of the exact 'nature' of our ***resumableBoundaryWalker*** 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 ***resumableBoundaryWalker*** 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:

```
let us define emum direction {UP,DOWN,LEFT,RIGHT}
the robot start in the HOME position, direction=DOWN
        1) send to the robot the request to execute the command moveForward with the resume button
        and continue to do it, until the answer of the request becomes 'false' or the user press the stop button
        2) if the user has pressed the stop button, the robot waits until the user presses the resume button
        3) send to the robot the request to execute the turnLeft only in the case that the last answer is 'false'
```

The following resources could be usefully exploited to reduce the development time of a first prototype of the application:

1. The Consolegui.java (in project it.unibo.virtualrobotclient)
2. The RobotMovesInfo.java (in project it.unibo.virtualrobotclient)
3. The RobotInputController.java (in project it.unibo.virtualrobotclient)

The expected time required for the development of the application is (no more than) 6 hours.

Test plans

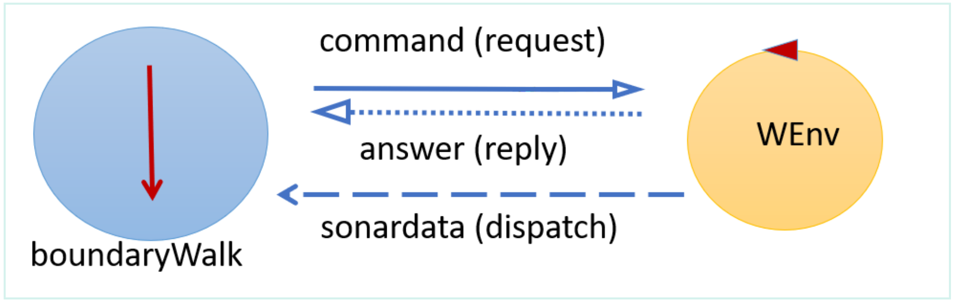
To check that the application fulfills the requirements, we could keep track of the moves done by the robot. For example:

```
...
let us define String moves="";
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 and continue to do 1);
    2) when the answer of the request becomes 'false',
    send to the robot the request to execute the command turnLeft and append the symbol "l" to moves
```

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:

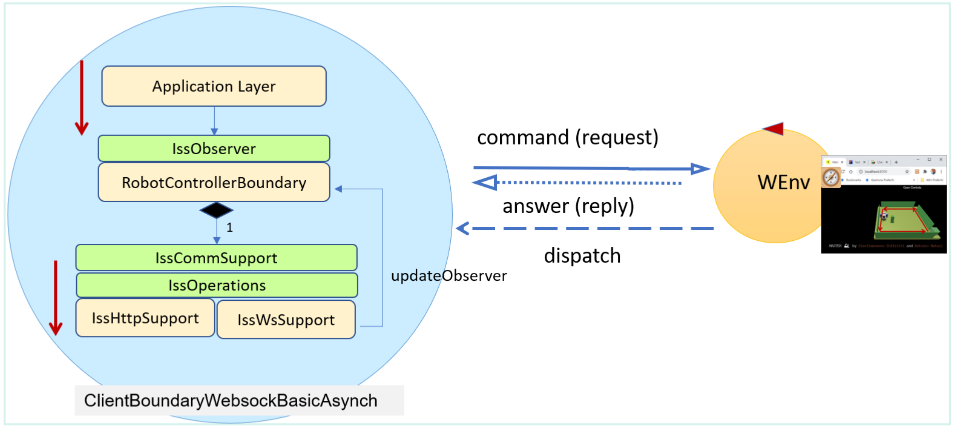
```
moves:  "w*lw*lw*lw*l"        * : repetion N times(N>=0)
```

Project

Nature of the application component The *resumableBoundaryWalker* application is a **conventional Java program**, represented in the figure as an object with an internal thread. Our 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	 command (request) answer (reply) sonardata (dispatch) WEnv boundaryWalk
Zoom in the resumableBoundaryWalker architecture	

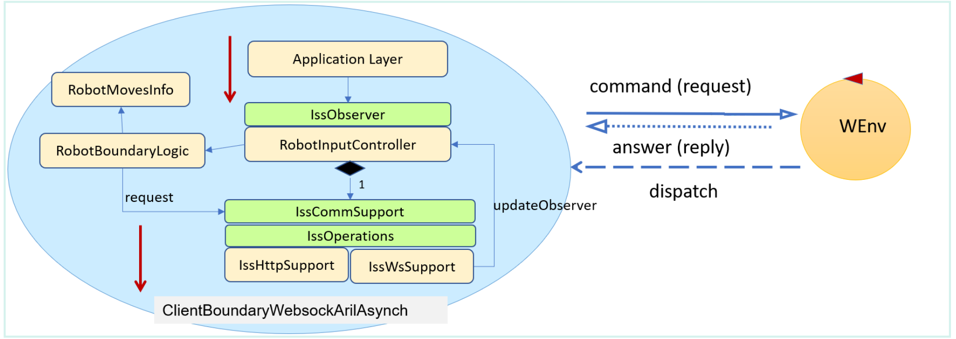
To make the 'business code' as much independent as possibile from the technological details of the interaction with the virtual robot (and with any other type of robot in the future), let us structure the code according to a conventional ***layered architecture***, which is the simplest form of software architectural pattern, where the components are organized in horizontal layers.

The architecture of this project make refernce to the **BoundaryWalk_Project** and the **VirtualRobot_Project**. The first layered structure is reported in the **BoundaryWalk**. the observer pattern is also used to manage asynchronous communication, as explained in **VirtualRobot_Client** document. We can see the architecture in the image on the left.



Adding isolated business logic to save the robots moves history

Referencing **VirtualRobot_Client** we isolate the business logic to mantain the robot moves info as in the image, adding two components **RobotBoundaryLogic** and **RobotMovesInfo**.



User Interface components:ConsoleGui

The user interface have to start, stop and resume the Robot. To do this,we add the **Consolegui.java** as an observer observing the robot controller **RobotInputController**. This observer will be updated by the User Interface (Gui) and will send the action from the gui to the **RobotInputController** that will handle them and stop or resume the robot. In this way we do not have to change any component functionality.

Testing

Deployment

The deployment consists in the commit of the application on a project named **iss2021_resumablebw** of the MY GIT repository (https://github.com/Afranz1/FranzeseAntonioRoot).

The final commit commit has done after **5** hours of work.

Maintenance

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