

# Lab ISS | the project resumableBoundaryWalker

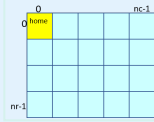
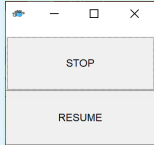
## Introduction

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 <b>HOME</b> position, as shown in the picture on the right	
the application presents to the user a <b>consoleGui</b> similar to that shown in the picture on the right	
when the user hits the button <b>RESUME</b> the robot <b>starts or continue to walk</b> along the boundary, while updating a <b>robot-moves history</b> ;	
when the user hits the button <b>STOP</b> the robot stop its journey, waiting for another <b>RESUME</b> ;	
when the robot reaches its <b>HOME</b> again, the application <i>shows the robot-moves history</i> on the standard output device.	

## Delivery

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

cognome\_nome\_resumablebw.pdf

## Requirement analysis

Our **interaction with the custom** ha clarified that the customer intends:

- for **room**: a conventional (rectangular) room of an house;
- for **boundary**: the perimeter of the room, that is physically delimited by solid **walls**;

- 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 **user control**: the user decides when the robot has to walk its path and when it has to stop;
- for **consoleGui**: software application that allows the user to exercise the control over the robot;
- for **robot-moves history**: a representation of the path walked by the robot;
- for **walk**: the robot moves forward, close to the room walls.

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

## User story

A **user story** is specified in the Requirements part.

## Test plans

The main aspects to check are:

- when the robot reaches its **HOME** again, the path shown in the robot-moves history is the one expected;
- when the robot is stopped by the user, the robot-moves history must be consistent.

The tests must be carried out via software.

## 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 protocols.

However, the problem does introduce an **abstraction gap at the conceptual level**, since **the required logical interaction** is always a **request-response**, regardless of the technology used to implement the interaction with the robot.

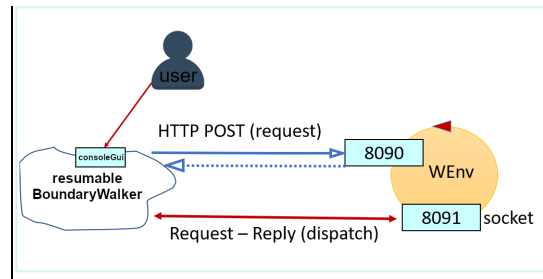
## Logical architecture

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

Moreover, there is a third entity, the **user**, who interacts with the `resumableBoundaryWalker` through the `consoleGui`.

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 `resumableBoundaryWalker` software is left to the designer. However, we can say here that it is **not a database, or a function or an object**.
- To make our `resumableBoundaryWalker` software **as much as possible independent** from the underlying communication protocols, the designer could make reference to proper design pattern
- It is quite easy to define **what the robot has to do** to meet the boundary walk requirement:

```
let us define enum direction {UP,DOWN,LEFT,RIGHT}
the robot start in the HOME position, direction=DOWN
for 4 times:
    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'
    2) send to the robot the request to execute the turnLeft
For every move the robot-moves history is updated.
```

- To check the user control requirement it is recommended to run a part of the application (e.g. the robot walks a part of its path) and to check if the robot-moves history is consistent.

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 if the path shown in the robot-moves history is the one expected when the robot reaches its **HOME** again, a possible test could be:

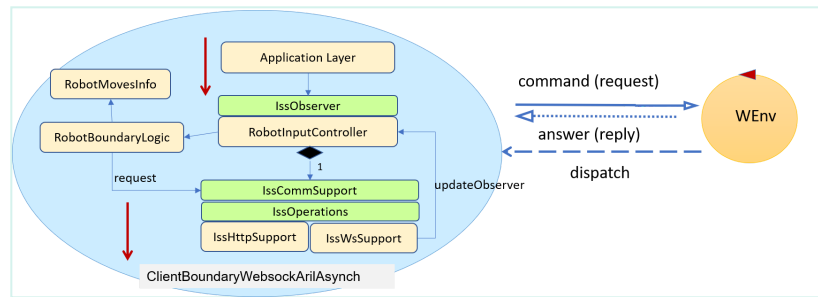
By setting the time for the moves in order to obtain movements of robot-units, we incrementally build a map of the territory after each move. In this way we could know where the robot is after each command and therefore we could check the path taken at the end of the application; for example (1 represents the 'cells' traveled by the robot)

At the end of the application, the map must be something like this:

```
|r, 1, 1, 1, 1,
|1, 0, 0, 0, 1,
|1, 0, 0, 0, 1,
|1, 1, 1, 1, 1,
```

## Project

This part starts with the same considerations made in the [BoundaryWalk project](#) and continue with other considerations made in [VirtualRobotClients file](#) up to the following structure:



As explained at the end [VirtualRobotClients](#) there is a useful `mapUtil` which is recommended to use to represent the robot-moves history. To use it, it's necessary to implement an aril based communication, that we have already done in the first part of this project.

The provided `ConsoleGui.java` file allows the user to start, stop (by sending an halt move request) and resume the boundary walk according to its own needs.

The **controller** of the application is associated to the `ConsoleGui` as an observer, so when a `ConsoleGui` button is pressed, the controller can manage the event with the right handler.

## 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/gianlu598/soavigianluca> ).

The final commit commit has done after **4** hours of work.

## Maintenance

By Gianluca Soavi email: [gianluca.soavi@studio.unibo.it](mailto:gianluca.soavi@studio.unibo.it)

