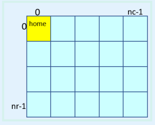
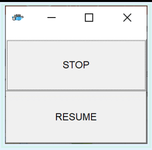
Lab ISS | the project resumableBoundaryWalker

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

After interviewing client, meanings he associates with nouns have been clarified:

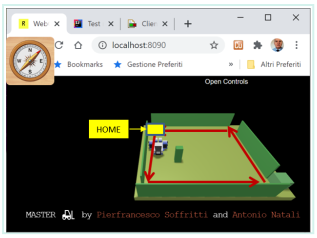
- **room**: a conventional room, as found in all buildings
- **boundary**: room's perimeter, physically delimited by solid wall
- **robot**: a device able to moving after receiving commands by network, as reported in VirtualRobot2021.html

- **consoleGUI**: a graphic interface used by user to impart order to the robot. It's composed by two buttons, **RESUME** and **STOP**.
  - **RESUME** button per to **start or continue the robot walk** around room's boundary, while updating a **robot-moves history**.
  - **STOP** button **interrupt the robot walk** in his actual position.
- **home**: the initial position of the robot in where he start his walk.
- **robot-moves hystory**: during the robot run, the application saves and remembers all moves performed.

Regarding to actions (verbs):

- **walk**: the robot have to move forward, near room's walls

A first user story

As mentioned in the requirements, when user start the application, this present a GUI in which are prensent two button **RESUME** and **STOP**. At the start the robot are positioned in the **HOME** cell. To start the robot run, user must press button **RESUME**. Once the button is pressed, robot start his walk around the room boundary. In any moment of execution, the user can press button **STOP** . In that case the robot stop his walk in the position in where he is. To resume the walk, user must press the button **RESUME**. During robot walk, the application remember all moves that robot performs. Once the robot reach again **HOME** position, execution is finished and all robot-moves history will print on the standard output device. In this case user have total control of the robot walk, through the GUI, he can start/stop/resume its execution when he wants. At the end of system's execution, I expect that robot performed (only one time) path shown in figure.	 localhost:8090 Open Controls HOME MASTER by Pierfrancesco Soffritti and Antonio Natali

Problem analysis

Aspect noted

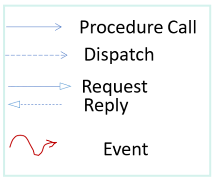
1. System must provide both ***proactive*** (that is bondary walk) and ***reactive*** (the need to react to **STOP/RESUME** buttons pressure) behaviour.
2. **STOP** button provide to interrupt the robot journey after completion of last given command

3. **RESUME** button provide at the application start to begin journey around room boundary, instead after an interruption, it provide to restart the journey from point where it stopped
4. Buttons are implemented in a **GUI**, user have to interact with this component to effectively change behaviour of the robot.

Logical Architecture

We do introduce the following terminology:

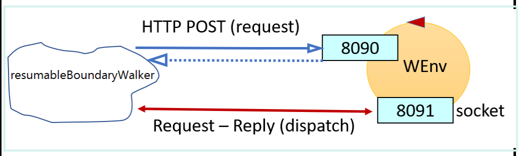
- **Dispatch**: a message '*fire and forget*': the sender does not expect any answer.
- **Request**: the sender expects an answer sent using a **Reply**.
- **Invitation**: the sender expects an ack.
- **Event**: the sender '*emits information*' without specifying any receiver.



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 *websocket technology*. We use this technology as we want that the communication between application and robot be continuous. This decision was made since the user can have a continuos interaction with application. Indeed he can interact with robot moves pressing **RESUME/STOP** commands

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)



Available resources

Since we already have a working code that could be usefully exploited to reduce devolopement time of our firt prototype of this application. Those resources are:

- ***Proactive*** behaviour is already analyzed and designed in a previous project ClientBoundaryWebsockArilAsynch.
- The Consolegui.java (in project it.unibo.virtualrobotclient)
- The RobotMovesInfo.java (in project it.unibo.virtualrobotclient)
- The RobotInputController.java (in project it.unibo.virtualrobotclient)

Following the **Single Responsibility Principle** , the application component is composed of:

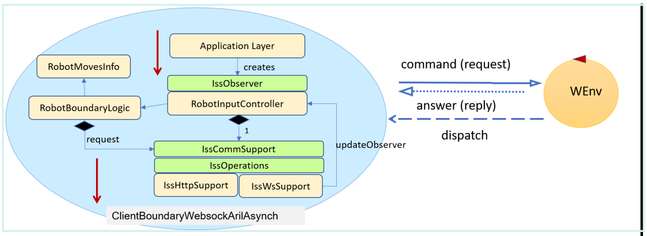
- the controller RobotInputController: it is a **POJO** that implements the IssObserver interface and works as an **Observer** of the underlying ws-based communication support.
  Thus, it is called by the underlying support, and handles the endmove messages sent by the WEnv on the wsconnection by executing the proper operation of the RobotBoundaryLogic that performs the boundary walk ***without interruptions*** .

- the RobotBoundaryLogic, that provides the following operations:

  - doBoundaryInit: called at the system start-up
  - boundaryStep: called when the controller handles a endmove message
  - doBoundaryGoon: an internal operation that calls a single-step robot move

  These operations are called by the RobotInputController that handles the endmove messages sent by the WEnv on the wsconnection as answers to the execution of a robot-move done by the RobotBoundaryLogic itself.

- The RobotMovesInfo that provides an utility to represent the robot journey as a string or as a map.



Moreover, our 'company' has already defined:

- The Consolegui.java class: it provides a GUI with the buttons STOP and RESUME and already accepts a controller that implements the IssObserver interface

Test plans

We should simulate a user command and check the effects of the user commnds STOP/RESUME.

Project

Considering all resources that we already have, without create other classes, we can customize those resources in order to adapt them to this problem.

1. Consolegui.java connected to the RobotApplInputController.java. This class implements **Observer** class in order to catch every	

interaction with button on the GUI.

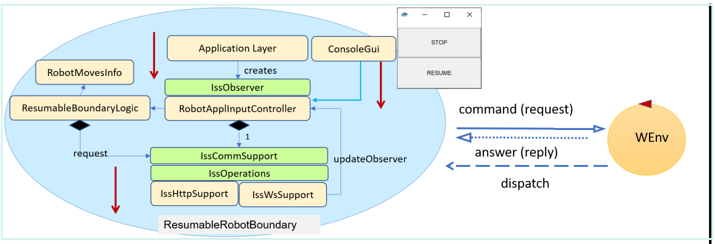
2. RobotApplInputController.java an object that handles the JSON messages

   ```
   {"robotcmd":"STOP"}   {"robotcmd":"RESUME"}
   ```

   and calls ResumableBoundaryLogic.java.

3. ResumableBoundaryLogic.java that consider the fact that the journey could have been stopped.

   The difference between other locig class is that here we must handle the case in which the robot moves is halted. To do this the function **buondary step** make a controll on a new variable **jouneyHalted** that change according to user actions on GUI



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/LeoManto/Mantovani_Leonardo**).

Maintenance

My git repo: **https://github.com/LeoManto/Mantovani_Leonardo**

By Leonardo Mantovani email: leonardo.mantovani2@studio.unibo.it