Lab ISS | the project cautiousExplorer with Actors

Introduction

This case-study deals with the design and development of proactive/reactive software systems based on the concept of Actor, as introduced in LabIss2021 | wshttp support with ActorBasicJava observers.

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 rectangular room, delimited by walls that includes one or more devices (e.g. sonar) able to detect the presence of obstacles, including the robot itself;
- the robot has a **den** for refuge, located in the position shown in the picture cautiousExplorer
- the robot works as an *explorer of the room*. Starting from its **den**, the goal of the robot is to create a map of the room that records the position of the fixed obstacles. The presence of mobile obstacles in the room is (at the moment) excluded;
-
- since the robot is '*cautious*', it returns immediately to the **den** as soon as it finds an obstacle. It also stops for a while (e.g. 2 seconds), when it 'feels' that the sonar has detected its presence.

Delivery

The customer requires to receive at **12 noon on April 6** a file named

```
cognome_nome_cea.pdf
```

including a (synthetic) description of the project (preceded by a proper analysis of the problem) based on components of type ActorBasicJava and a reference to a *first working prototype* (even with limited capabilities) of the system.

Meeting

A SPRINT-review phase with the customer is planned (via Teams) at **5.15 pm on April 6**.

Requirement analysis

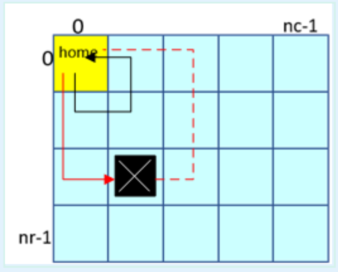
Our **interaction with the customer** has clarified that the customer intends:

- 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 **rectangular room**: a closed environment delimited by 4 walls;
- for **sonar**: a device which detect and notify the presence of an object when the object is located in a certain part of the closed environment;
- for **den**: the robot starting position, located along the room boundary (shown in the Requirements picture), and the position to which it must return when certain situations occur, as described in the requirements;
- for **explorer of the room**: this termi is clearly specified in the Requirements section;
- for **fixed obstacle**: an object, placed in a certain fixed position in the room, which is an impediment to a robot movement;
- for **map**: a representation of the rectangular room (and its obstacles) where the robot is located;

- for ***move***: the robot can move forward, backward and turn left and right;
- for ***find the fixed obstacles***: when the robot detects an obstacle, and its impact sensors emit collision data, as described in *VirtualRobot2021.html* provided by the customer;
- for ***returns immediately to the den***: as soon as the robot finds an obstacle, it must move until it returns to its starting position;
- for ***stop for a while***: the robot is stationary for a certain period of time (e.g. 2 seconds);
- for ***detect the presence***: to know, with the help of specific sensors, if an object is located in a certain part of the closed environment;
- for ***remember***: the robot should have and should be able to consult a representation of the closed environment where it is placed;

- for **at the moment excluded**: the presence of mobile obstacles (obstacles whose position in the room changes over time) can be a possible future development.

User story

As user, I put the robot in its **den**; afterwards, I activate the **cautiousExplorer** application that moves the robot inside the room until it explores the whole room. The exploration is done in an organized way. The robot explores the environment making "concentric circles", with ever larger radius, around its den (as shown in the picture). If it finds an obstacles, it records the obstacle position, returns immeditely to its den and then starts an other "concentric	 Legenda

circle journey". If a sonar detects the robot during the exploration, the robot stop itself for 2 seconds and then resume its journey. The application cannot be interrupted by any user-command. When the application terminates, I expect that the robot has returned to its den (starting position) and it has memorized a map of the rectangular room, with every single fixed obstacle.	

Problem analysis

Main aspects

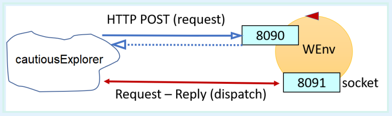
In the VirtualRobot2021.html the customer states that the robot can receive move commands in two different ways:

- by sending messages to the port 8090 using **HTTP POST**
- by sending messages to the port 8091 using a **websocket**

Another problem, related to the communication, arises when we consider the requirement *"It also stops for a while (e.g. 2 seconds), when it 'feels' that the sonar has detected its presence"*, which says that the robot has to stop its journey for a period of time when the sonar detects it.
When a **request-response communication can be used to send movement commands to the robot**, as far as sonar is concerned, that type of communication is not suitable anymore.
The sonar requirement requires a communication that uses dispatches, since the information from the sonar arrives unpredictably and asynchronously.
In this case, considering the type of communication available, a websocket communication must be used to implement a solution for the requirement.

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:	 HTTP POST (request) 8090 WEnv cautiousExplorer 8091 socket Request – Reply (dispatch)

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 (for the meaning of the symbols, see the legenda)

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 underlying communication protocols, the designer could make reference to proper design pattern

Available resources

Since we already have working code, the following classes and libraries could be usefully exploited to reduce the development time of a first prototype of the application:

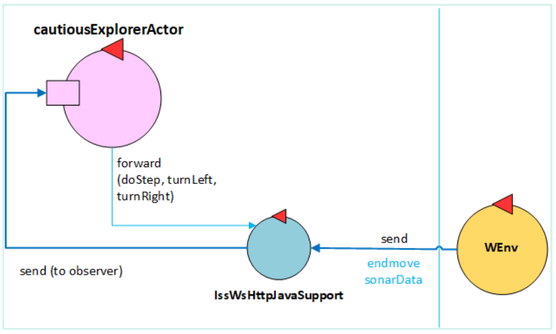
- the RobotMovesInfo.java that provides an utility to represent the robot journey as a string or as a map;
- the IssWsHttpJavaSupport.jar that is able to send/receive messages via http/ws to the **WEnv** and is able to send information to ActorBasicJava actors registered as abservers.

Test plans

We should simulate the user story explained in the Requirement analysis and check if the final map is correct.

Project

Considering the available resources, the requirements and every single consideration, made in the ResumableBoundaryWalker project about the aril based communication, the useful mapUtil and the **javaActorSupport**, we arrived at the following structure:

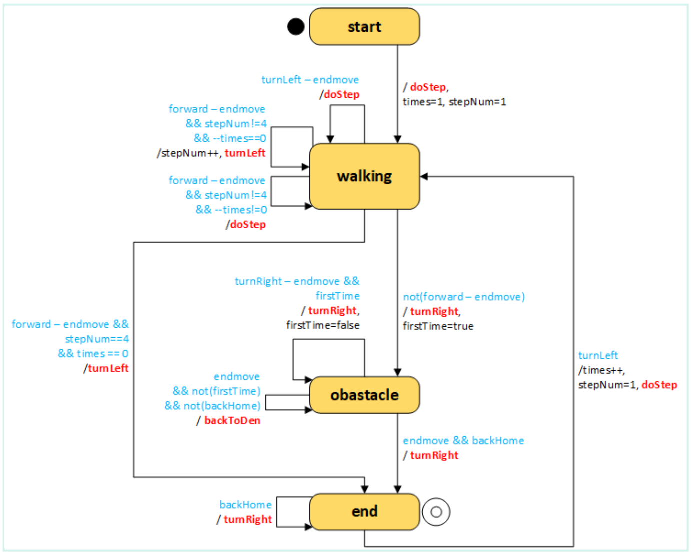


MainRobotActorJava.java:
the cautiousExplorer application as a system composed of ActorBasicJava.java components.

The cautiousExplorer as a Finite State Machine

CautiousExplorerActor.java:
the cautiousExplorer application logic designed and implemented as a Mealy Machine.

The first protoype is a partial prototype. The robot does not create the entire map of the rectangular room, but it create a partial map until it finds an obstacle. It makes "concentric circles" around its den and when it finds an obstacle, it returns to its starting position and stops the room exploration.
The future developments of this initial prototype include a better use of the mapUtil to reach the main goal (room entire map).



- times variable is the radius of the "concentric circle journey", expressed in robot-unit. It is increase when a "concentric circle journey" is correctly done;
- firstTime variable is initially set to false;
- the backToDen function takes the just made path, read it in reverse (e.g "wwl" -> "lww"), and call the **turnRight** when a "l" move is found or the **doStep** when the "w" move is found;

- bacHome variable is initially set to **false**, and it is set to **true** when the backToDen function has read the entire reverse path.

Testing

Deployment

The deployment consists in the commit of the application on a project named it.unibo.cautiousExplorer of the MY GIT repository (https://github.com/gianlu598/soavigianluca).

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

Maintenance

By Gianluca Soavi email: gianluca.soavi@studio.unibo.it