Workplan Automated Car-Parking- Sprint 2

Requirements

The requirements given by the costumers Automated Car-Parking considered in the following SPRINT are:

- A software system, named ***ParkManagerService***, that implements the required automation functions.

User stories

As a **client - parking phase** :

- I intend to use a ***ParkServiceGUI*** provided by the ParkManagerService to notify my interest in *entering* my auto in the parking-area and to receive as answer the number ***SLOTNUM*** of a free parking-slot (1<=SLOTNUM<=6). SLOTNUM**==0** means that no free slot is available.
- If SLOTNUM **>0**, I move my car in front to the INDOOR, get out of the car and afterwards press a ***CARENTER*** button on the ***ParkServiceGUI***. Afterwards, the transport trolley takes over my car and moves it from the INDOOR to the selected parking-slot. The ParkServiceGUI will show to me a receipt that includes a (unique) ***TOKENID***, to be used in the *car pick up* phase.

As a **client - car pick up phase** :

- I intend to use the ParkServiceGUI to submit the request to pick up my car, by sending the TOKENID previously received.
- Afterwards, the transport trolley takes over my car and moves it from its parking-slot to the OUTDOOR-area.
- I move the car, so to free the OUTDOOR-area.

- ***acceptIN***: accept the request of a client to park the car if there is at least one parking-slot available, select a free slot identified with a unique SLOTNUM.
  A request of this type can be elaborated only when the **INDOOR-area is free**, and the transport trolley is at home or working (**not stopped** by the manager). If the INDOOR-area is already engaged by a car, the request is not immediately processed (the client could simply wait or could - optionally - receive a proper notice).
- ***informIN***: inform the client about the value of the SLOTNUM.

If **SLOTNUM>0**:

1. ***moveToIn***: move the transport trolley from its current localtion to the INDOOR ;
2. ***receipt***: send to the client a receipt including the value of the TOKENID ;
3. ***moveToSlotIn***: move the transport trolley from the INDOOR to the selected parking-slot;

4. ***backToHome***: if no other request is present, move the transport trolley to its home location, else ***acceptIN*** or ***acceptOUT***.

If **SLOTNUM==0**:
- ***moveToHome***: if not already at home, move the transport trolley to its home location.

- ***acceptOUT***: accept the request of a client to get out the car with TOKENID. A request of this type can be elaborated only when the **OUTDOOR-area is free** and the transport trolley is at home or working (**not stopped** by the manager). If the OUTDOOR-area is still engaged by a car, the request is not immediately processed (the client could simply wait or could - optionally - receive a proper notice).
  1. ***findSlot***: deduce the number of the parking slot (***CARSLOTNUM***) from the TOKENID;
  2. ***moveToSlotOut***: move the transport trolley from its current localtion to the CARSLOTNUM/parking-slot ;
  3. ***moveToOut***: move the transport trolley to the OUTDOOR ;
  4. ***moveToHome***: if no other request is present move the transport trolley to its home location;
     else ***acceptIN*** or ***acceptOUT***

The following sprint will focus to realize the interection between the parkManagerService and client.

Requirement analysis

Our interaction with the customer has made it clear what he means for:
- SLOTNUM: unique identifier of a parking-slot;
- CARSLOTNUM: unique identifier of a parking-slot retrieved by TOKENID;
- TOKENID : unique string assigned to a client when his entrance request was accepted;
- client: an entity who send messages to the ParkerServiceManager via the network;
- CARENTER: a ParkServiceGui graphic element that allows an user to notify the intention to let the car in;
- request : a message which was sent by the client to the ParkerServiceManager in order to receive a service. The client will wait a reply when the service was complete.

ParkServiceGUI

An user interface that allows a client to with our system. The costumer requires that it will be portable, indipendent from hardware components and from the operative system of the user device so the choise will be to realize a Web GUI.

Problem analysis

The list of requirements for this sprint is:

- the client request to enter in the parking-area using the ParkServiceGui (2);
- the client request to exit from the parking-area using the ParkServiceGui (2);

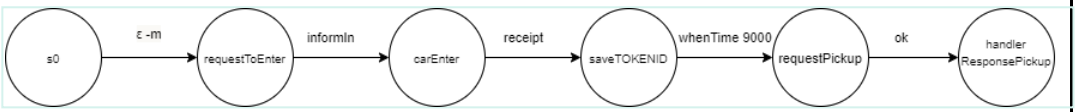
To carry out the operations requested by the client (acceptIN, informIN, receipt, acceptOUT, findSlot) it is necessary that the application, in addition to being able to perform the basic transport-trolley moves, is capable of:

- handle the TOKENID, SLOTNUM, CARSLOTNUM;
- the client send/receive information or command to/from client;
- check the availability of resources, INDOOR-area, OUTDOOR-area, parking-slot, verification of the presence of client requests;
- tell to the robot how it should behave given the customer's requests;
- realize the ParkServiceGui.

To destroy the abstraction gap is possible to use QACTOR which is a modelling language and defines a work model of the system based on actors behavior.

Components

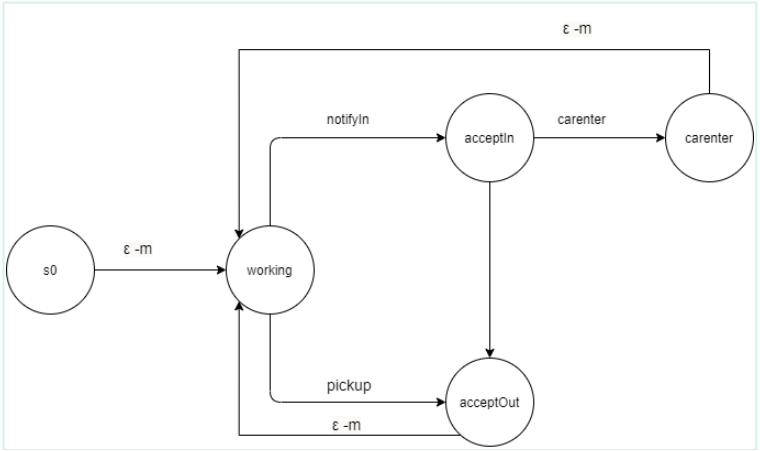
ParkServiceGui

The choise will be to realize the Web-application GUI in the following way: - they could be HTML pages implemented to recive information, INPUT sections (e.g. CARENTER) and to send information, OUTPUT sections (e.g. TOKENID). The techology should be SpringBoot to reduce development time. Right now we just confine its behavior to a mock-qactor. Its complete realization will be done in the project phase.	

The ParkManagerService

The ParkManagerService, in this sprint, will be a part of the WebApplicationPms. The ParkManagerService will be modelled as a QActor to simulate one part of requirements:

- It should manage the requests from the clients, excecute them and elaborate the replies.
- The ParkManagerService is modelled to manage the requests of more clients. For example if two clients notify the interest to enter, to recognise the correct client who pressed CARENTER, the client should send the SLOTNUM previously sending by the ParkManagerService.
  The Weightsensor and Outsonar are implemented to simulate the behavior and manage more clients.
- The commands MovetoSlotIn and MoveToSlotOut can be group in one command MoveToSlot+SLOTNUM where the transport-trolley should go because the transport-trolley hasn't any interest to know if is an In or Out request.
- The management of the SLOTNUM, CARSLOTNUM and TOKENID is implemented in the file kotlin Slotnum.kt.
- The TOKENID could be modeled with the SLOTNUM+TIMESTAMP to be unique and to easly retrieve the CARSLOTNUM.



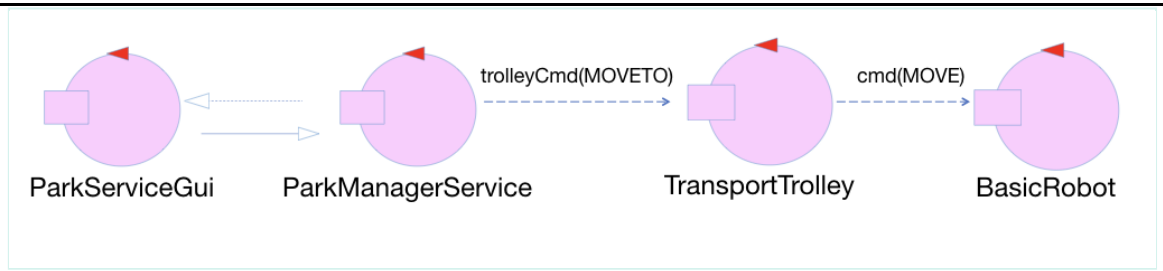
The transport-trolley

The transport-trolley was modelled like in the sprint1.html to excecute the ParkManagerService tasks.

The basicrobot

The basicRobot was modelled like in the sprint1.html. It will use to communicate between the DDR robot and the transport-trolley.

Components comunication



ParkingServiceGui/ParkManagerService

The interaction follow the request/response model because when the client send a command the application must answer with helpful informations (e.g. SLOTNUM, TOKENID...)

```
Request carenter : carenter(C)
Reply receipt : receipt(I)

Request notifyIn : notifyIn(N)
Reply informIn : informIn(S)

Request pickup : pickup(TOKENID)
Reply ok : ok(O)

Context ctxparkingarea ip [host="localhost" port=8021]

QActor parkingservicegui context ctxparkingarea{
        ...
        request parkmanagerservice -m notifyIn : notifyIn(A)
        ...
        request parkmanagerservice -m carenter : carenter ($SLOTNUM)
        ...
        request parkmanagerservice -m pickup : pickup($TOKENID)
}
QActor parkmanagerservice context ctxparkingarea{
        ...
        replyTo notifyIn with informIn : informIn($SLOTNUM)
        ...
        replyTo carenter with receipt : receipt($TOKENID)
        ...
        replyTo pickup with ok : ok($OUTFREE)}
```

The components communication parkManagerServic/Transport-trolley and Transport-trolley/BasicRobot are described in sprint1.html

> The executable model is the following file: sprint2.qak

Testplan

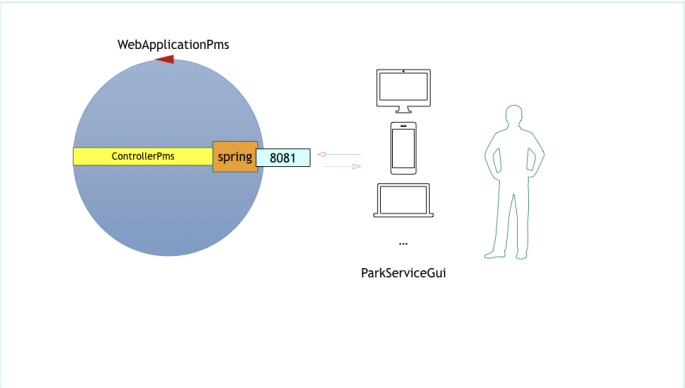
Testplan: we should check if the Slotnum.kt return the correct number (SLOTNUM from 1 to 6) or all the slot are occupied (SLOTNUM 0) .

The code is the following file: TestPlan.kt

To realize an authomatized test we will use JUNIT.

Project

WebApplicationPms

The WebApplicationPms will be a system of actors and other components that implements the required automation functions. It is a distributed service accessible via HTTP: `http://localhost:8081` The WebApplication is in the following folder: it.unibo.WebApplicationPms	

ControllerPms

The ControllerPMS represents the WebApplication controller that responsible for processing incoming requests, preparing a model, and returning the view to be rendered as a response. It defines an API with the purpose of creating a standard communication so that external components can communicate with our application and without knowing its implementation aspects.
With the respect of clean architecture the ControllerPMS should implement the view managing of the Web application.
The ControllerPms is the following file:
ControllerPms.kt

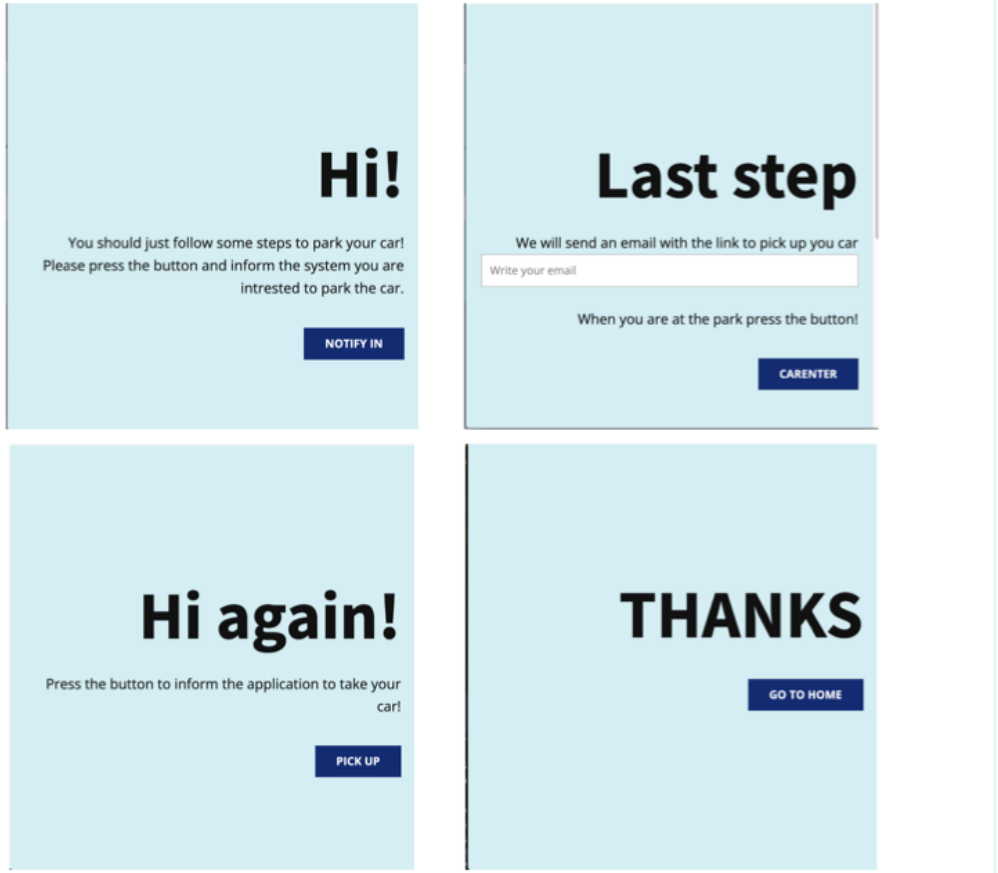
```
        //example of implementation
 @RequestMapping("/carenter", method = arrayOf(RequestMethod.GET))
   fun carenter(button: Button, slotnum: Slotnum): String {
       var slot = getSlotNum()
       if (!slot.equals("0")) {
           button.enterdisabled = true
           slotnum.slotnum = slot.toInt()
           println(slotnum.slotnum)
       } else {
           button.enterdisabled = false
       }
       return "carenter.html"
   }

    // used to connecting the WebApplication to the service
fun createconnection(messageId: String, content: String): String{
       connToPms = connQakBase.create(ConnectionType.TCP)
       connToPms.createConnection()
       val msg = MsgUtil.buildRequest("parkmanagerserviceProxy", messageId, content, qakdestination)
       answer = connToPms.requestWithRensponse(msg)
       print(answer)
       return answer
   }
```

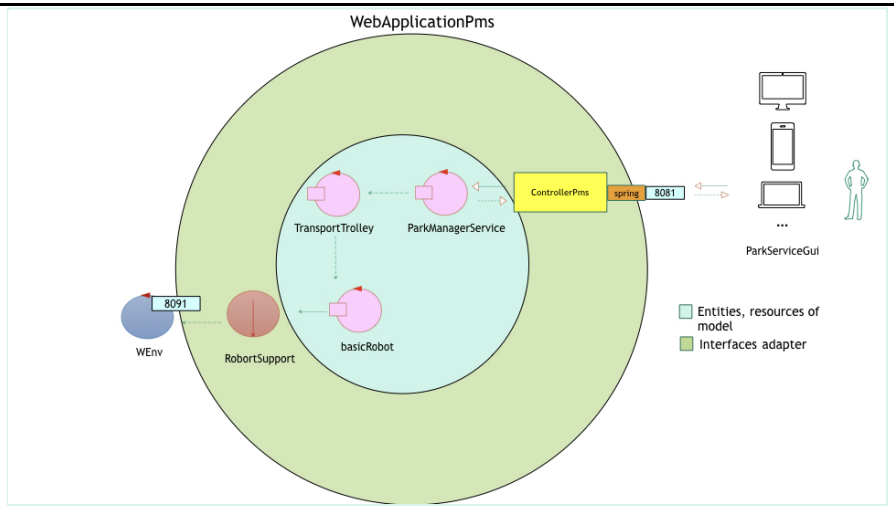
ParkServiceGui

The ParkServiceGui will be a series of responsive HTML pages with the aim to show what operations the client could be.

- parking phase:it is composed by two pages:
  - notify in: an html page with a button "notifyIn" pressed by the user to inform the system his intrest to use the service.
  - carenter: an html page with a input form where the client should put his email and a button pressed by the user to inform the system that the robot could take the car. The system will use the email to send to the user the link for go to the pick up phase.
- pickup phase: it is composed by one page and it is accessible only using the email sending previously. There is only one button "pickup" used to inform the system that the client want take its car. If the system cannot process the command it will inform the client to try later. The EmailService was implemented to guarantee the right correlation between client and his TOKENID. If the client picked up his car the system is able to recognised and avoid the client to use two times the same link.



Logical Architecture



The QActor excecutable model after project phase is the following file: sprint2_project.qak

Deployment

The project is located in the repository: https://github.com/noemival/ParkManagerService_2021/tree/main/it.unibo.parkManagerService

Testing

This functional test want to verify the correct implementation of the requirements specified in this sprint. In particular, the requirements were verified:

- NotifyIn: notify my interest in entering my auto in the parking-area
- CarEnter: press a CARENTER button on the ParkServiceGUI thus the car cod be transported by the trolley to reach the correct slot
- Pickup: submit the request to pick up my car.

To verify that the system goes into the expected states (acceptIn, handleCarEnter, acceptOut) an observer has been created with the aim of capturing the changing states of the parkmanagerservice actor. Since the requirements are linked together, they will be tested following a specific order:

1. NotifyIn
2. CarEnter
3. Pickup.

The code is the following file: Testing.kt

SPRINT REVIEW

By studentName email: antonio.iacobelli@studio.unibo.it 

By studentName email: noemi.valentini5@studio.unibo.it