Final task ISS-2021 Bologna: Automated Car-Parking

Requirements

Automated Car-Parking

A company intends to build an *automating parking service* composed of a set of elements:

- A software system, named ***ParkManagerService***, that implements the required automation functions.
- A **DDR** robot working as a ***transport trolley***, that is intially situated in its ***home*** location. The **transport trolley** has the form of a square of side length ***RD***.
- A ***parking-area*** is an empty room that includes;
  - an ***INDOOR*** to enter the car in the area. Facing the **INDOOR**, there is a ***INDOOR-area*** equipped with a ***weigthsensor*** that measures the ***weigth*** of the car;
  - an ***OUTDOOR*** to exit from the **parking-area**. Just after the **OUTDOOR**, there is ***OUTDOOR-area*** equipped with a ***outsonar***, used to detect the presence of a car. The **OUTDOOR-area**, once engaded by a car, should be freed within a prefixed interval of time ***DTFREE***;
  - a number **N (N=6)** of ***parking-slots***;
  - a ***thermometer*** that measures the temperature ***TA*** of the area;
  - a ***fan*** that should be activated when **TA > TMAX**, where ***TMAX*** is a prefixed value (e.g. **35**)

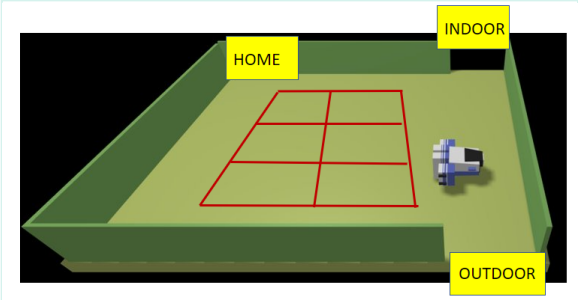
  A ***map*** of the parking area, represented as a grid of squares of side length ***RD***, is available in the file parkingMap.txt:

```
|r, 0, 0, 0, 0, 0, 0, X,
|0, 0, X, X,  0, 0, 0, X,
|0, 0, X, X,  0, 0, 0, X,
|0, 0, X, X,  0, 0, 0, X,
|0, 0, 0, 0, 0, 0, 0, X,
|X, X, X, X, X, X, X, X,
```

The map includes the positions of the **parking-slots** (marked above with the symbol **X**) and of the ***fixed obstacles*** in the area (the walls marked with the symbol **X**).

The area marked with **X** is a sort of 'equipped area' upon which the **transport trolley** cannot walk. Thus, to get the car in the **parking-slot (2,2)**, the **transport trolley** must go in cell **(1,2)**.

The proper scene for the WEnv is reported in: parkingAreaConfig.js



- a ***parking-manager*** (an human being) which supervises the state of the **parking-area** and handles critical situations.

The job of our company is to design, build and deploy the ***ParkManagerService***.

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**.

As a **parking-manager**:

- I intend to use the ***ParkServiceStatusGUI*** provided by the **ParkManagerService** to observe the ***current state*** of the **parking area**, including the value **TA** of the temperature, the state of the **fan** and the state of the **transport trolley** (**idle, working or stopped**).
- I intend to ***stop*** the **transport trolley** when **TA > TMAX**, activate the **fan** and wait until **TA < TMAX**. At this time, I stop the **fan** and resume the behavior of the **transport trolley**. Hopefully, the **start/stop of the fan** could also be automated by the **ParkManagerService**, while the **start/stop of the transport trolley** is always up to me.
- I expect that the **ParkManagerService** sends to me an ***alarm*** if it detectes that the **OUTDOOR-area** has not been cleaned within the **DTFREE** interval of time.

Requirements

The **ParkManagerService** should create the **ParkServiceGUI** (for the client) and the **ParkServiceStatusGUI** (for the manager) and then perform the following tasks:

- *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*

- *monitor*: update the **ParkServiceStatusGUI** with the required information about the state of the system.

- *manage*: accept the request of the manager to stop/resume the behavior of the **transport trolley**.

About the devices

All the sensors (**weigthsensor**, **outsonar**, **thermometer**) and the **fan** should be properly simulated by mock-objects or mock-actors.

When available a Raspberry and a sonar

The **outsonar** could be a real device. We can simulate the presence/absence of a car.

Non functional requirements

1. The ideal work team is composed of **3 persons**. Teams of 1 or 2 persons (**NOT** 4 or more) are also allowed.
2. The team must present a **workplan** as the result of the requirement/problem analysis, including some significant **TestPlan**.
3. The team must present the sequence of **SPRINT** performed, with appropriate motivations.
4. Each **SPRINT** must be associated with its own 'chronicle' (see templateToFill.html) that presents, in concise way, the key-points related to each phases of development. Hopefully, the team could also deploy the system using docker.
5. Each team must publish and maintain a GIT-repository (referred in the templateToFill.html) with the code and the related documents.
6. The team must present (in synthetic, schematic way) the **specific activity of each team-component**.

Requirements analysis

Glossary

- **Transport trolley:**
  - a squared robot of side **RD** able to pick up a car and transport it from a point of the map to another;
  - the robot picks up and puts down cars from the square in front of it;
  - while the car is on the robot, they fill the same space on the map;
- **Home:** the home location of the **trolley** is the north-western corner of the parking-area, facing south;
- **Parking-area:** rectangular empty room containing the **parking-slots**, the home location and some maneuvering space for the **trolley**;
- **INDOOR-area:** the area where cars wait for the **trolley** to pick them up; it is not part of the **parking-area**;
- **OUTDOOR-area:** the area where the **trolley** leaves the cars to be picked up by their owners; it is not part of the **parking-area**;
- **Parking-slot:** a squared portion of the **parking-area** of side **RD** where parked cars are stored by the **trolley**; there are six of them and they are identified by a **SLOTNUM**;
- **Fan:** device able to lower the temperature of the **parking-area**;
- **Fixed obstacles:** the parts of the **parking-area** where the **trolley** is not able to pass through;
- **Hopefully / Optionally:** the corresponding requirements should be implemented and employable as an alternative to the default behavior.

User stories

The user stories provided by the customer are sufficiently precise and complete and they do not need immediate elaboration.

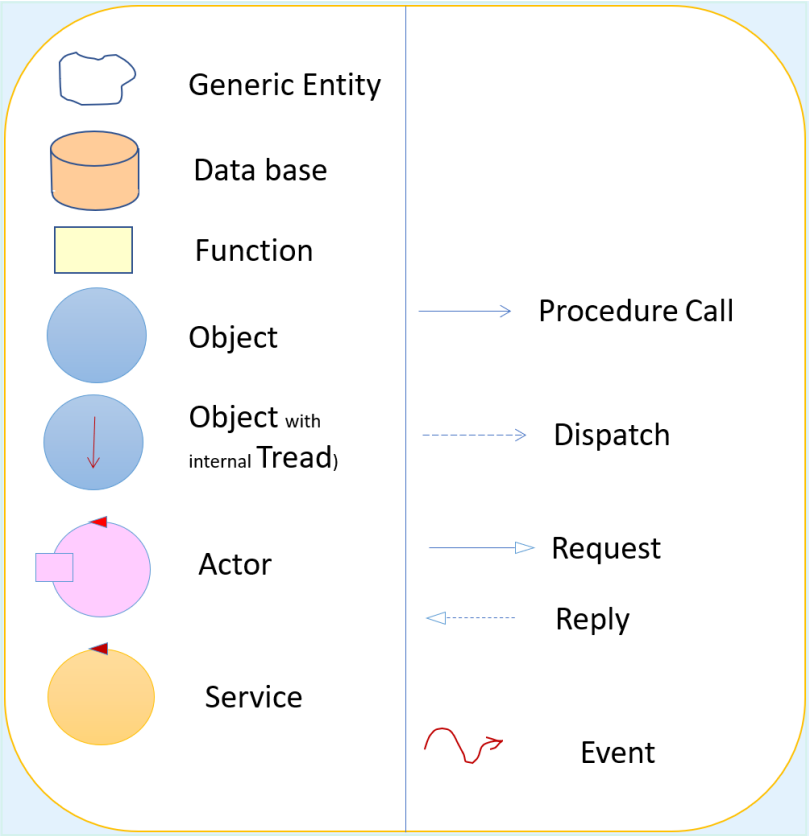
Informal test plan

The requirement details provided by the customer are accurate enough to act as an informal test plan on their own.

Problem analysis

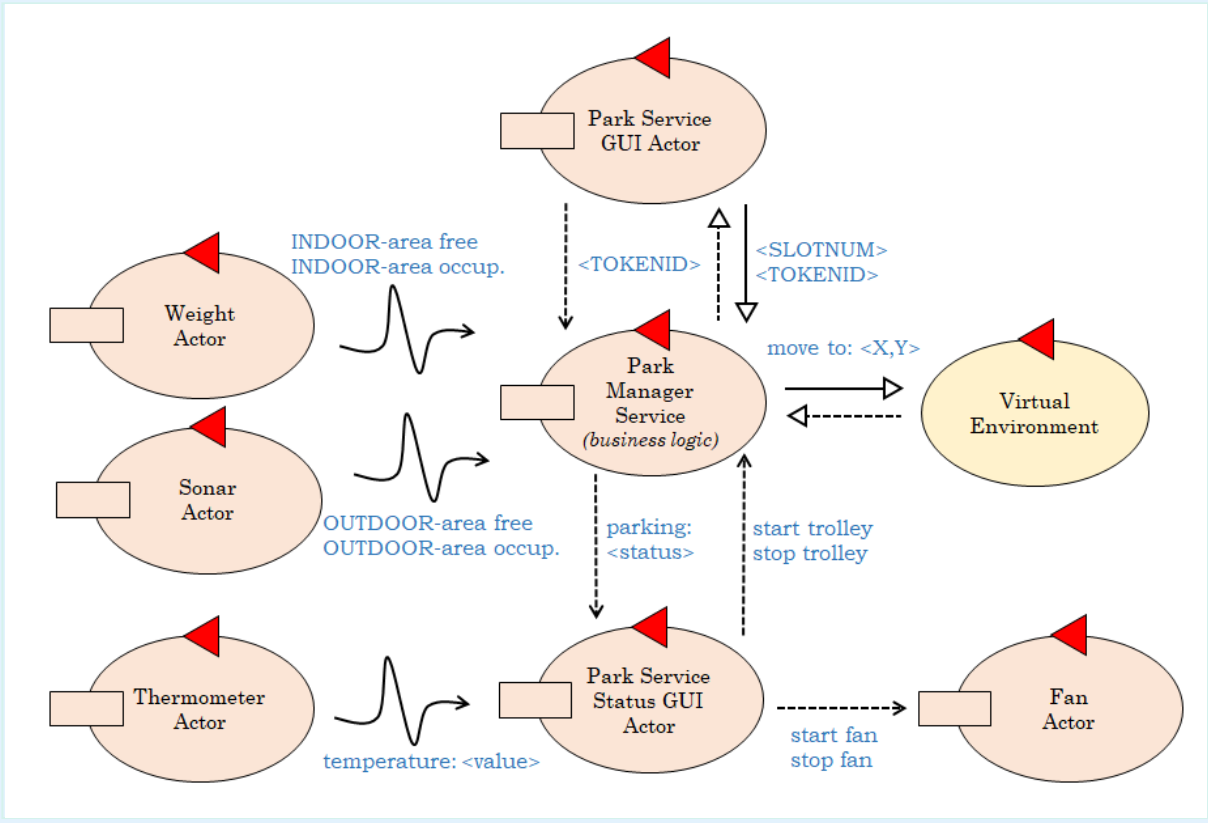
Relevant aspects

1. The system to be built will be a distributed system, consisting of several macro-components:
   - the **transport trolley** (and its virtual environment, provided by the customer);
   - the **weight sensor**;
   - the **thermometer**;
   - the **fan**;
   - the **outsonar**;
   - our **application** (ParkManagerService, complete with its two GUIs), which sends commands to the trolley in order to meet the requirements.
2. This distributed system, made of several heterogeneous components, would benefit greatly from an **actor-based** framework with support for **message-passing** interaction; for this purpose, we should leverage the available QAK meta-model to build executable models;
3. Since the available map measures the **parking-area** in squares of side length **RD** (same as trolley size), the communication with the virtual robot should follow the aril convention;
4. There isn't any conceptual abstraction gap for this problem, however since we proposed to exploit the **QAK meta-model**, we are put in front of an **abstraction gap** regarding the use of languages because QAK relies on Java and Kotlin to work. However, QAK itself manages to fill a large part of this limitation, as it is supplied with its own domain specific language and because it was designed specifically for heterogeneous distributed systems;
5. The **QAK meta-model** provides compatibility with the communication protocols **TCP**, **MQTT** and **CoAP**, equally valid for the interaction between actors; regarding the communication with the **virtual environment**, both supported interaction models (**HTTP POST** and **WebSocket**) are fit for the task.
6. We will use the following **legend** for all the diagrams in the document:



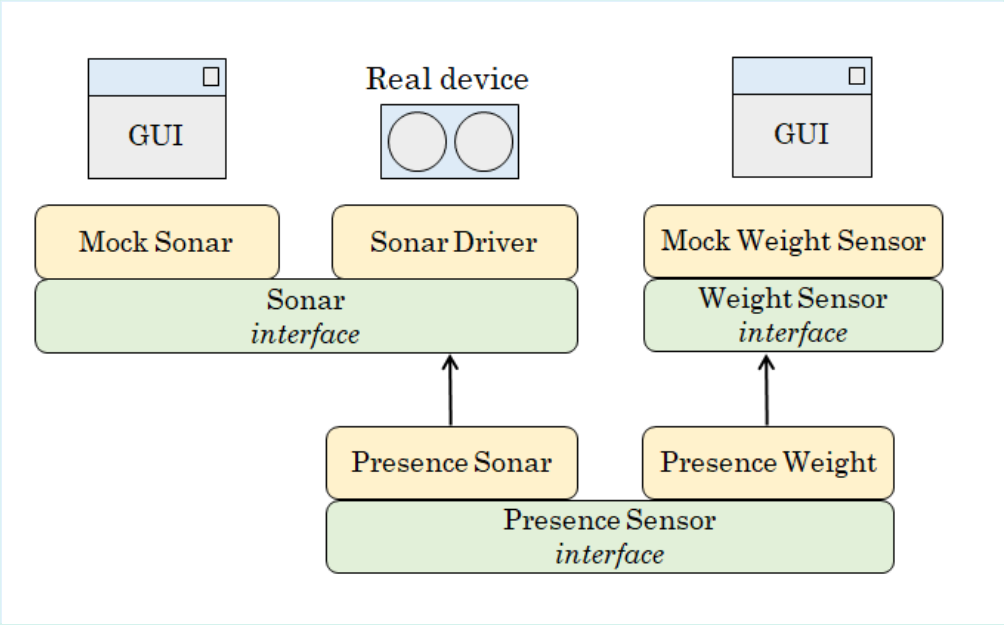
Logical architecture

Each logical component should be modeled as an actor or split in two or more actors as expressed by the following general architecture (plus some minor actors introduced later):

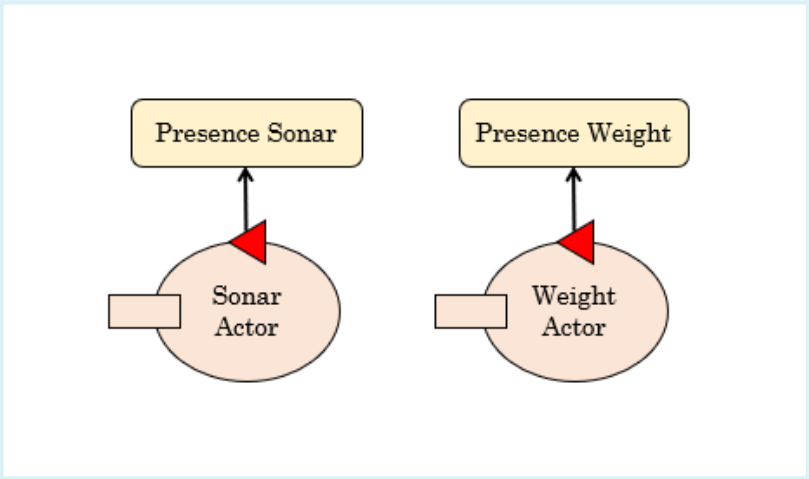


The presence sensors

The **INDOOR-area** and the **OUTDOOR-area** are respectively equipped with a **weightsensor** and an **outsonar**. While being two physically different components, they serve the same logical function and thus they can be both modeled as **presence sensors** with appropriate **adapters**. We report hereunder a diagram to illustrate the conceptual taxonomy:



The sensor entity should be queryable as to know if it does or does not detect a presence. The actor using it (one for each sensor) should poll the presence within a fixed interval of time and fire an **event** each time the status changes from present to not-present (or the other way around). These actors thus present **proactive** behavior.

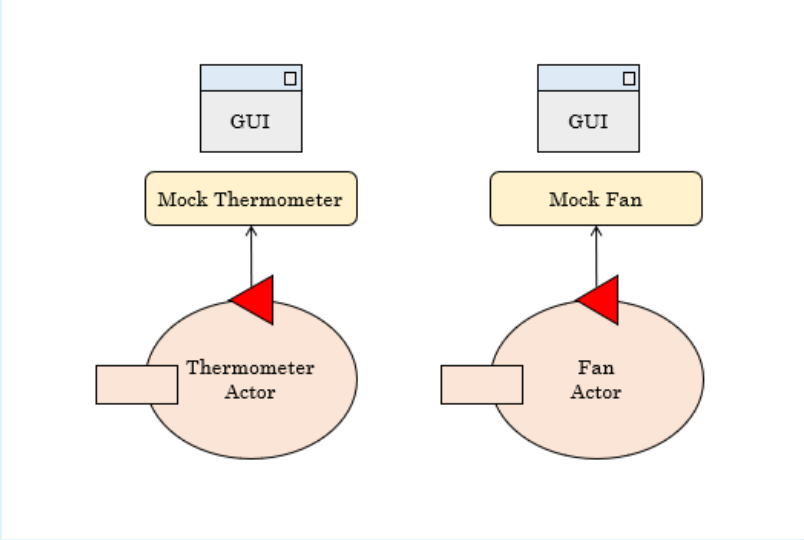


The thermometer and the fan

Much like the sonar and the weight sensor, the **thermometer** should be a queryable component encapsulated into a **proactive** actor that polls the temperature and fires an **event** with the new temperature value each time it varies **significantly**. Concerning the significance of a variation in temperature, it is worth noting that most general purpose sensors have a sensibility of **0.1 °C**.

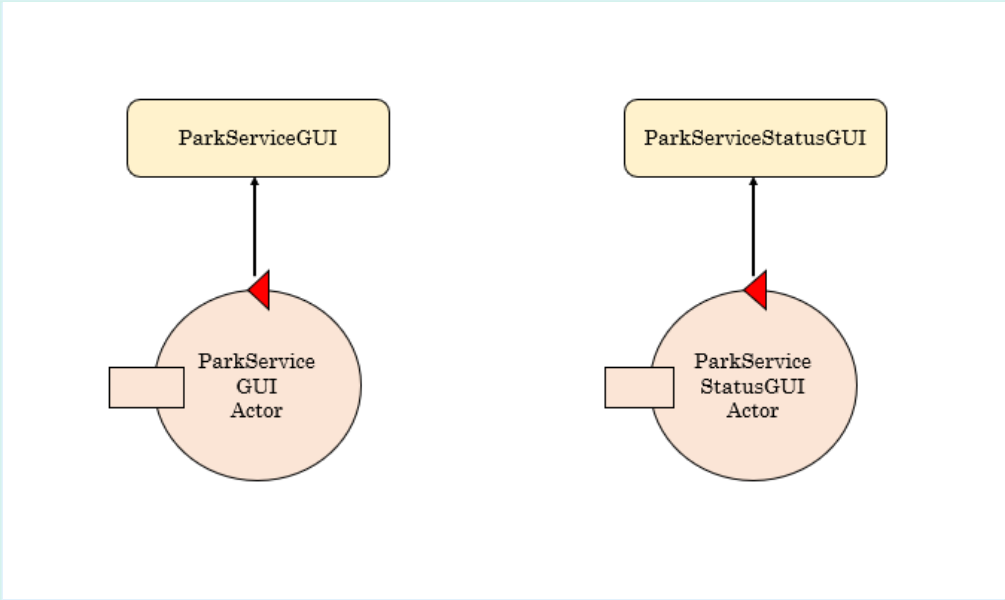
The **fan** should be a passive component able to start and stop the device when notified to do so. Its actor is **reactive** in nature and should be able to receive

dispatches and update the status of the fan according to them.



The user interfaces

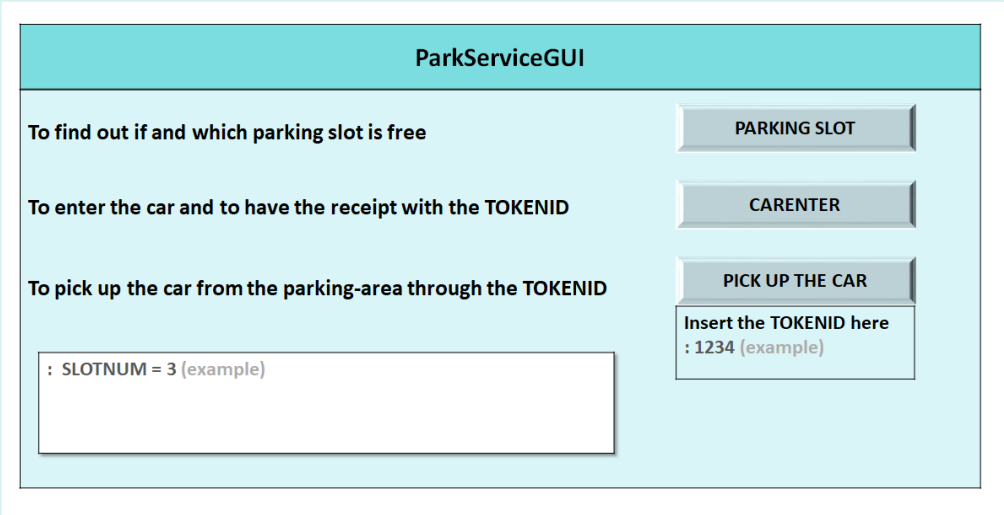
The **ParkManagerService** app is composed of two GUIs: the **ParkServiceGUI**, used by clients, and the **ParkServiceStatusGUI**, used by the parking-manager. The GUIs should be controlled by two different and independent actors because they may not be hosted on the business logic node. They will probably not even run on the same node, because the physical device used by the client will likely not be the same device used by the manager. If the designer chooses to build them with a Web framework, then the GUIs will act as websites and this constraint will probably not be required.



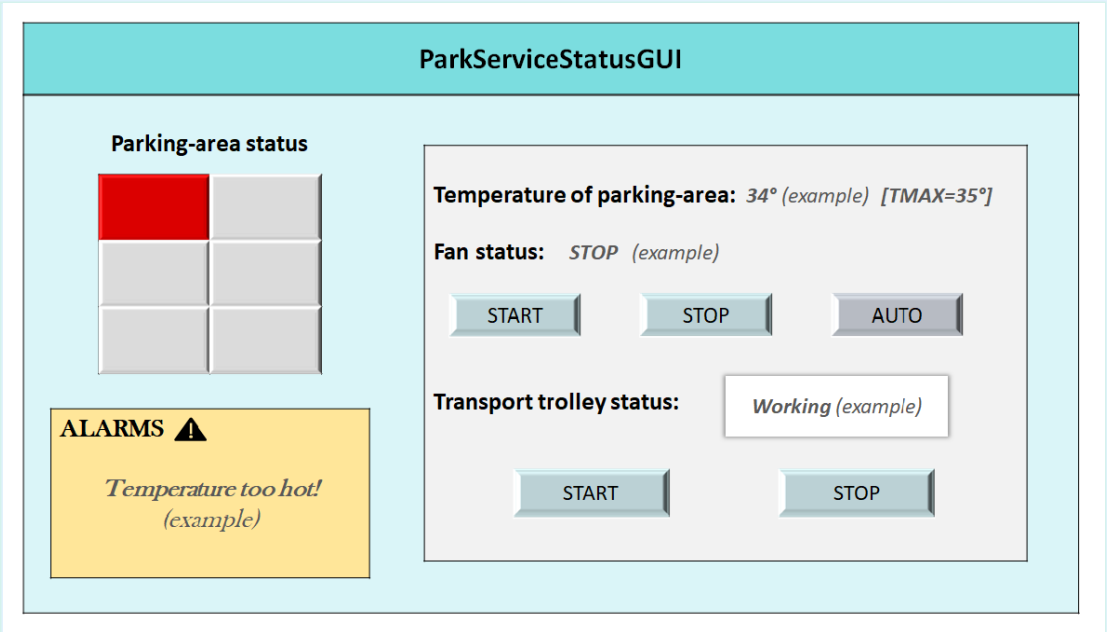
The **proactive** behavior (toward the system) is prevalent on both GUIs, since their main purpose is to take in user input. Especially the **ParkServiceStatusGUI** presents however also a **reactive** behavior in how it handles alarms.

The **ParkServiceGUI** has three buttons:

1. The **PARKING SLOT** button is needed to notify the interest by a client in **entering** its car in the **parking-area** and to receive the number **SLOTNUM** of a free **parking-slot** (between 1 and 6). If **SLOTNUM** is equal to 0, it means that no free slot is available.
2. The **CARENTER** button can be pressed only if **SLOTNUM** is greater than 0, so that the **transport trolley** may take the car from the **INDOOR** to the assigned **parking-slot**. Then it would show the client a receipt that includes the unique **TOKENID** that will be needed later to pick up the car.
3. The **PICK UP THE CAR** button is needed to **pick up** the car with the **TOKENID** received before, which needs to be written in its specific section under the button. The client's car will then be moved to the **OUTDOOR-area**.

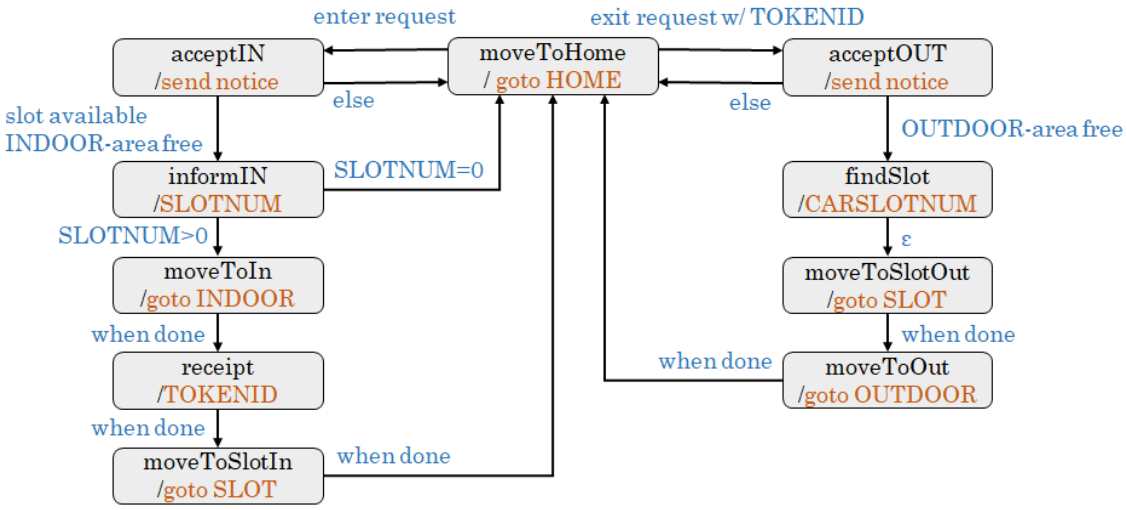


The **ParkServiceStatusGUI** has a grid with the **parking-area** status (red rectangles are occupied **parking-slots**), a section with the **transport trolley** status (idle, working or stopped) and a dedicated section for the **temperature**, in which you can see and manage the **fan** status when it is necessary with the use of three buttons: **start**, **stop** and **auto** (to automate the behavior of the fan); here you can also suspend and resume the behavior of the **trolley** with the **start** and **stop** buttons. At last, there is also a rectangle in which **alarms** appear.



The business logic

The core business logic can be conveniently described as a finite-state machine to be enclosed into a proper actor. Such actor controls the main state of the application and presents a purely **reactive** behavior, since it just reacts to external stimuli.



The **goto** outputs should be delivered to an entity able to perform the implied set of elementary actions on the **trolley**.

The additional task **manage** must ensure that, regardless of the current state, the execution of the automaton gets **stopped** whenever the manager gives the relative signal. When the manager gives the resume signal, the automaton must resume its evolution from where it was interrupted. Moreover, the additional task **monitor** must update the **ParkServiceStatusGUI** with the current status of the system.

In states **acceptIN** and **acceptOut**, if the conditions to process the relative requests are not met, their processing must be postponed. While the **trolley** is stopped, new enter and exit requests by clients are not postponed, they are instead refused (with proper notice).

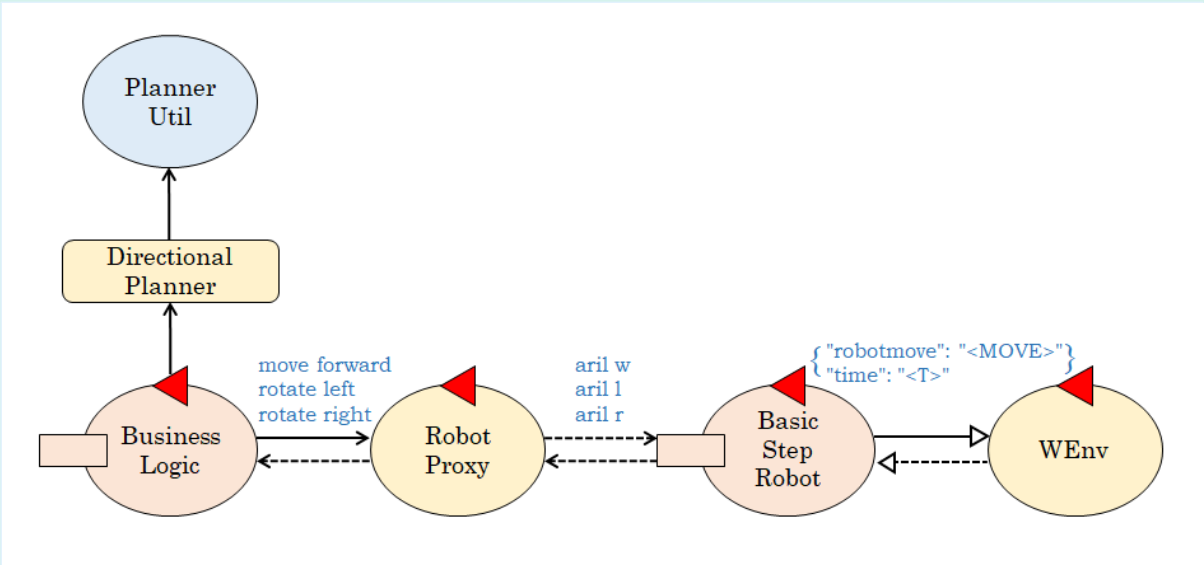
Driving the trolley

To access the **WEnv** (the virtual environment provided by the customer), it should be considered the usage of a more sophisticated tool, already available, called Basic Step Robot Actor, which would be able to avoid interferences among overlapping commands and to natively receive movement orders expressed as steps. A new service, for now called **Robot Proxy**, may be required to ensure compatibility between our system and the legacy tool **Basic Step Robot Actor**.

An already available singleton object called Planner Util is able to plan the route to drive a robot to a specified spot avoiding obstacles, if provided with a map such as the one made available by the customer. This tool may be extended by a new component, which we will call **Directional Planner**, to add the functionality to

plan the desired orientation of the robot alongside its desired position (so when the movement is complete, the **trolley** is also oriented toward the desired direction).

The diagram hereunder presents a possible architecture taking advantage of the suggested tools:



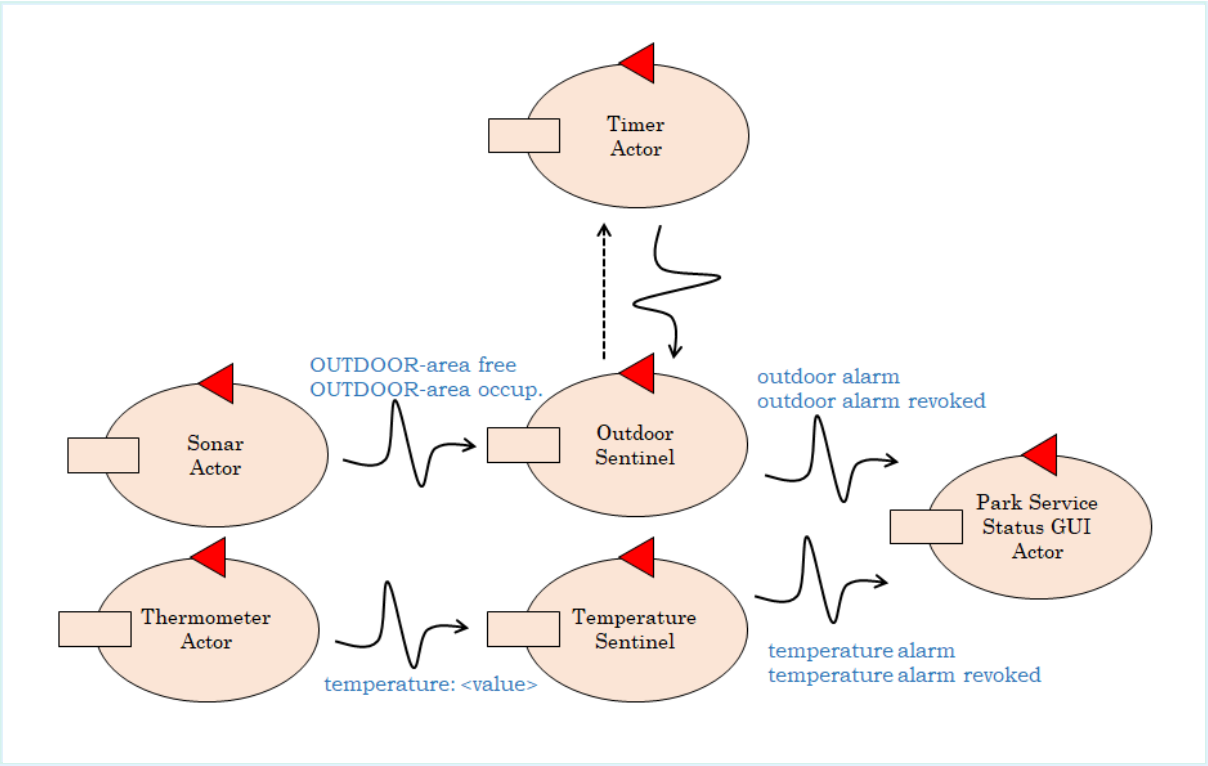
The **WEnv** requires a **request-reply** communication paradigm. Even if some intermediate steps will probably favor different approaches, the replies from the **WEnv** should nevertheless be always brought back to the actor that originated the relative requests.

The **WEnv** itself presents both proactive and reactive behavior, however it is only required its **reactive** nature for this application, since there are no virtual sonars in the scene and since robot collisions are theoretically excluded thanks to the map.

The alarms

The system requires the presence of a timed alarm for when the **OUTDOOR-area** has not been cleared within **DTFREE** time from the beginning of its occupation. The system also suggests the presence of a second alarm for when **TA** exceeds **TMAX** and some kind of notification for when **TA** goes back into its normal low temperature range.

We suggest the presence of two **proactive** actors, one for the **OUTDOOR** and one for the temperature, that fire an **event** each time they have an alarm or a notification to report. For the actor requiring temporization, it is available a legacy component called TimerActor that may be leveraged for this purpose. The diagram hereunder presented reports this conceptual interaction:



Workplan

Based on what was discussed in this initial phase, it was concluded that there will be needed **4 sprints** for the design of this application.

Sprint 1

- Main business logic with just one parking-slot;
- Communication with the virtual robot;
- ParkServiceGUI for the clients;
- As soon as the system starts, the trolley goes to the INDOOR, takes in the car, parks it and goes back to its home; then, after a prefixed amount of time, the trolley takes the car from the parking-slot, brings it to the OUTDOOR and goes back to its home (there are no sensors at this point);

This sprint will start on **July the 19th** and we expect it to be completed within **3 working days**.

Sprint 2

- Weightsensor, outsonar and thermometer;
- Fan with manual control only;
- ParkServiceStatusGUI for the parking-manager (the buttons to start and stop the trolley are disabled).

We expect this sprint to be completed within **4 working days**.

Sprint 3

- Possibility to start and stop the behavior of the trolley;
- Support for all the six parking-slots.

We expect this sprint to be completed within **3 working days**.

Sprint 4

- OUTDOOR-area alarm and temperature alarm;
- Option to automate the control of the fan;
- Support for the real sonar device.

We expect this sprint to be completed within **4 working days** and the whole project to be finished by **August the 4th**.

Refined test plan

User story 1 - parking phase

- In case of occupied INDOOR-area, when the client presses the **PARKING SLOT** button he receives a notification that no request can be sent.
- The INDOOR-area is free and the client uses the **PARKING SLOT** button, the business logic processes the request and the SLOTNUM is displayed in the GUI itself.
- If SLOTNUM > 0, then the client presses the **CARENTER** button and the transport trolley positions itself in the INDOOR-area and then in the slot previously indicated by SLOTNUM. Then that specific parking slot changes status and is shown as occupied in the ParkServiceStatusGUI table. Finally, the transport trolley returns to its home. At that point, a receipt with a TOKENID is shown to the client on the screen.
- If SLOTNUM = 0, a message is shown on the screen indicating that there is no space in the parking-area and, if the client decides to wait for its turn, as soon as SLOTNUM > 0 then his request is processed as in the previous case.
- If the transport trolley status is stopped and the client presses the **CARENTER** button, the request is rejected and the client is notified.

User story 2 - car pick up phase

- When the client presses the **PICK UP THE CAR** button, the request to enter the TOKENID is shown on the screen. If the code entered exists and is correct and the OUTDOOR-area is free, the transport trolley goes to the parking-slot (associated to that TOKENID) and then to the OUTDOOR-area. Finally it

returns to home. Then that specific parking-slot will change status and will be shown as free in the ParkServiceStatusGUI table.

- If the OUTDOOR-area is occupied, when the client presses the **PICK UP THE CAR** button, he receives a notification that no request can be sent. As soon as the area is free, his request will be processed as in the previous case.
- If the entered code does not exist or is not correct, then an error message will be shown on the screen and the client will have to re-enter the TOKENID.
- If the transport trolley status is stopped and the client presses the **PICK UP THE CAR** button, the request is rejected and the client is notified.

User story 3 - parking-manager

- When the parking manager presses the **START** button in the fan area, the mock of the fan declares to have turned on and the fan status changes to START. When the **STOP** button is pressed and the fan declares that it has turned off, the fan status changes to STOP.
- When the **AUTO** button has been pressed and the temperature rises above TMAX, the fan activates itself and the fan status changes to START. When the temperature value returns below TMAX, the fan turns off and the fan status changes to STOP.
- When the parking manager presses the **START** button in the transport trolley area, the latter stops working and its status in the ParkServiceStatusGUI changes to START. When he then presses the **STOP** button, the transport trolley resumes its work and its status becomes STOP.
- When the OUTDOOR-area remains occupied more than the DTFREE time interval, then an alarm message is shown on the screen. As soon as the OUTDOOR-area is free again, the error message is no longer shown.
- When the temperature is higher than TMAX, an alarm message is shown on the screen. As soon as the temperature is again lower than the threshold value, the error message is no longer shown.

By: Giacomo Fantazzini and Claudia Badalamenti
Email: giacomo.fantazzini2@studio.unibo.it - claudia.badalamenti@studio.unibo.it