Waste Incinerator Service

Sprint info

Sprint name	Sprint 2
Previous sprint	Sprint 1
Next sprint	
QAK model	sprint2.qak, monitoring-device.qak
Developed by	Alessio Benenati Giulia Fattori
Repo Site	WasteIncineratorService

Sprint Starting Condition and Goals

In the previous sprint, we focused on studying the requirements related to the application logic of OpRobot and WIS. **In this sprint, the focus is on the MonitoringDevice**, specifically aiming to **connect the virtual system** produced in sprint 1 **to a real MonitoringDevice** deployed on a physical Raspberry Pi.

Problem Analysis

MonitoringDevice subcomponents

In the previous sprints, we hid the complexity of the monitoring device in a single mock actor without worrying about its subcomponents (LED and Sonar).
A more in-depth study of the component's application logic reveals two possible approaches:

PROF

- Developing a single MonitoringDevice actor responsible for both managing the LED and emitting data from the Sonar.
- Breaking down the MonitoringDevice into two distinct actors, the LED and the Sonar.

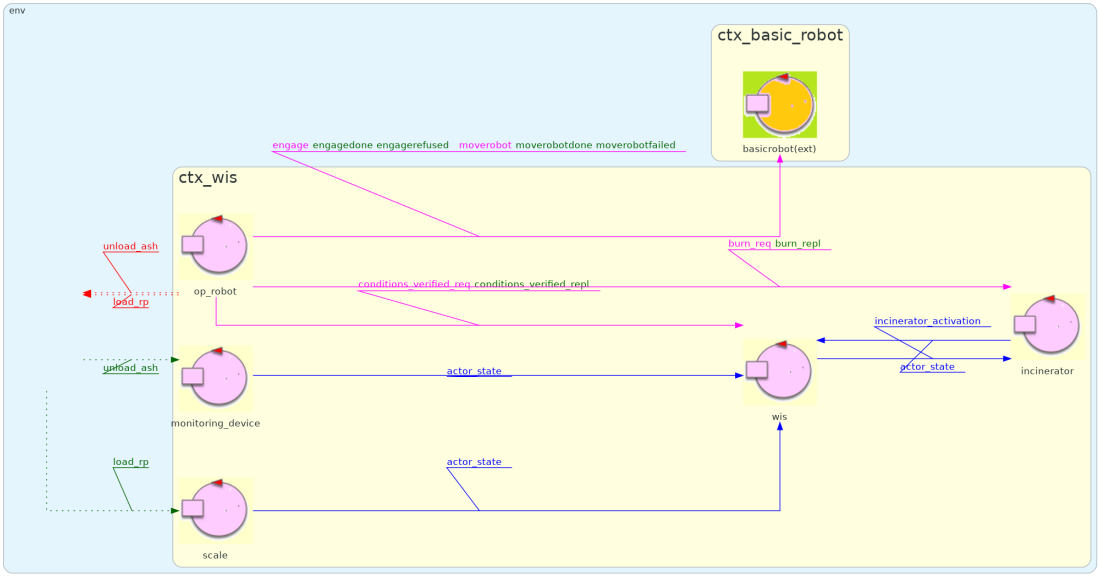
The second solution allows for greater decoupling between the two components, especially considering their different nature (the Sonar is a "producer" of information while the LED acts as a "consumer").
For this reason, it is recommended to **decompose the MonitoringDevice into its two subcomponents (LED and Sonar) and implement them as two independent actors in the same context**.

Project

Project Architecture

Below, we present a comparison between the system architecture derived from the problem analysis in sprint 1 and the one resulting from sprint 2.

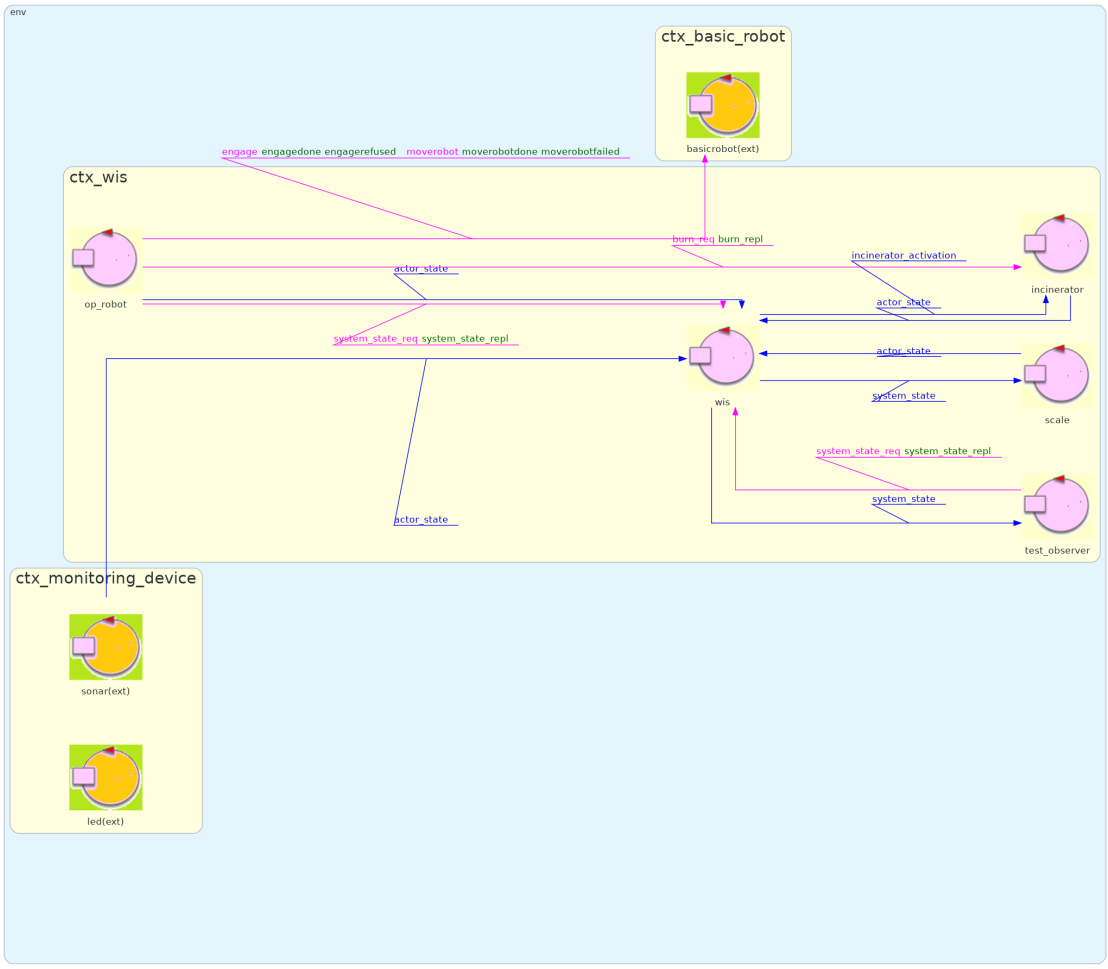
Sprint 1 Architecture:



wis_systemArch

Sprint 2 Architecture:

PROF



wis_systemArch

Implementation and Deployment

PROF

Sonar and Led abstraction

During the implementation, we encountered the **high sensitivity of the Sonar**, which often produces "noisy" data. For this reason, **it became necessary to introduce a "Filtering Pipeline"** to eliminate spurious data.

Specifically, this pipeline is composed of three actors:

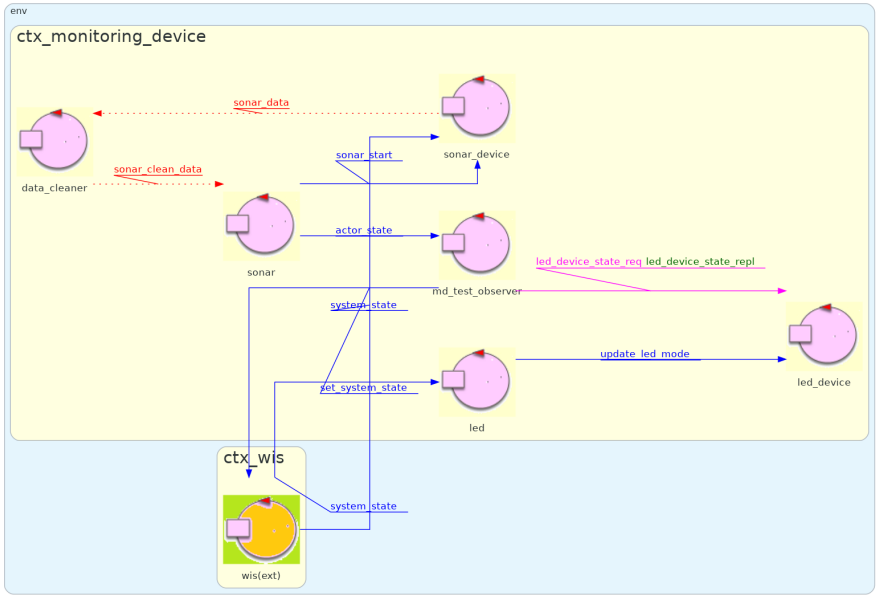
- **SonarDevice**, which handles the actual reading of all data from the physical sonar.
- **DataCleaner**, which monitors the SonarDevice and filters the relevant results for the problem, aiming to minimize the effect of measurement errors.
- **Sonar**, which serves as the "interface" towards the WIS.

In order to decouple the py we decided to split the Led actor into:

- **LedDevice**, which handles the communication with the physical led

- **Led**, which incorporates the Led business logic, deciding when to turn on and off the led

MonitoringDevice context details:



monitoring_deviceArch

System Configurability

During the implementation we faced the problem of the **lack of System Configurability**, so we decided to create a support singleton object **the SystemConfigurator** who is in charge of **loading the main properties of the system from a file during actors'initialization**.

Test Plan

PROF

Test Class: WISTest

Test Name	Initial Condition	Expected Behavior
test00_IncinineratorActivation	WasteStorage contains 4 RP, AshStorge is empty, nobody empties AshStorage, Incinerator is inactive	Once the system is initialized, the Incinerator is active
test01_Ok5RP	WasteStorage contains 5 RP, AshStorge is empty and can contain the ashes of 3 RPs, nobody empties AshStorage	After some time WasteStorage contains 2 RP and AshStorage is full

Test Class: MDTest

Test Name	Initial Condition	Expected Behavior

test00_SonarUnloadAsh	AshStorge is empty, nobody empties AshStorage, Sonar is monitoring AshStorage, OpRobot is unloading ash in the AshStorage	Once OpRobot is done unloading the ash, Sonar detects the ash level change
test01_LedBurning	Incinerator is inactive, Led is off	When the Incinerator is activated, Led starts blinking. When the Incinerator is stopped, Led turns back off
test02_LedEmptyWasteStorage	WasteStorage contains 1 RP, AshStorage is empty, Incinerator is inactive, Led is off	When WasteStorage gets emptied, Led turns on. When WasteStorage gets filled with a new RP, Led turns back off
test03_LedFullAshStorage	AshStorge is empty, WasteStorage is not empty, Incinerator is inactive, Led is off	When AshStorage fills up, Led turns on. When AshStorage gets emptied, Led turns back off

Usage

Basic Robot Activation

To test the system you will have to activate the Virtual Environment first.
To do so, open a terminal in the `Libs/unibo.basicrobot24` folder and type

```
docker compose -f virtualRobot23.yaml up
```

N.B. If you have an older version of docker, you may have to type `docker-compose` instead of `docker compose`

PROF

Next activate the BasicRobot.
It will act as a mediator between the VirtualRobot and the WasteIncineratorService application.
To do so, open another terminal inside the `Libs/unibo.basicrobot24` folder and type

```
gradlew run
```

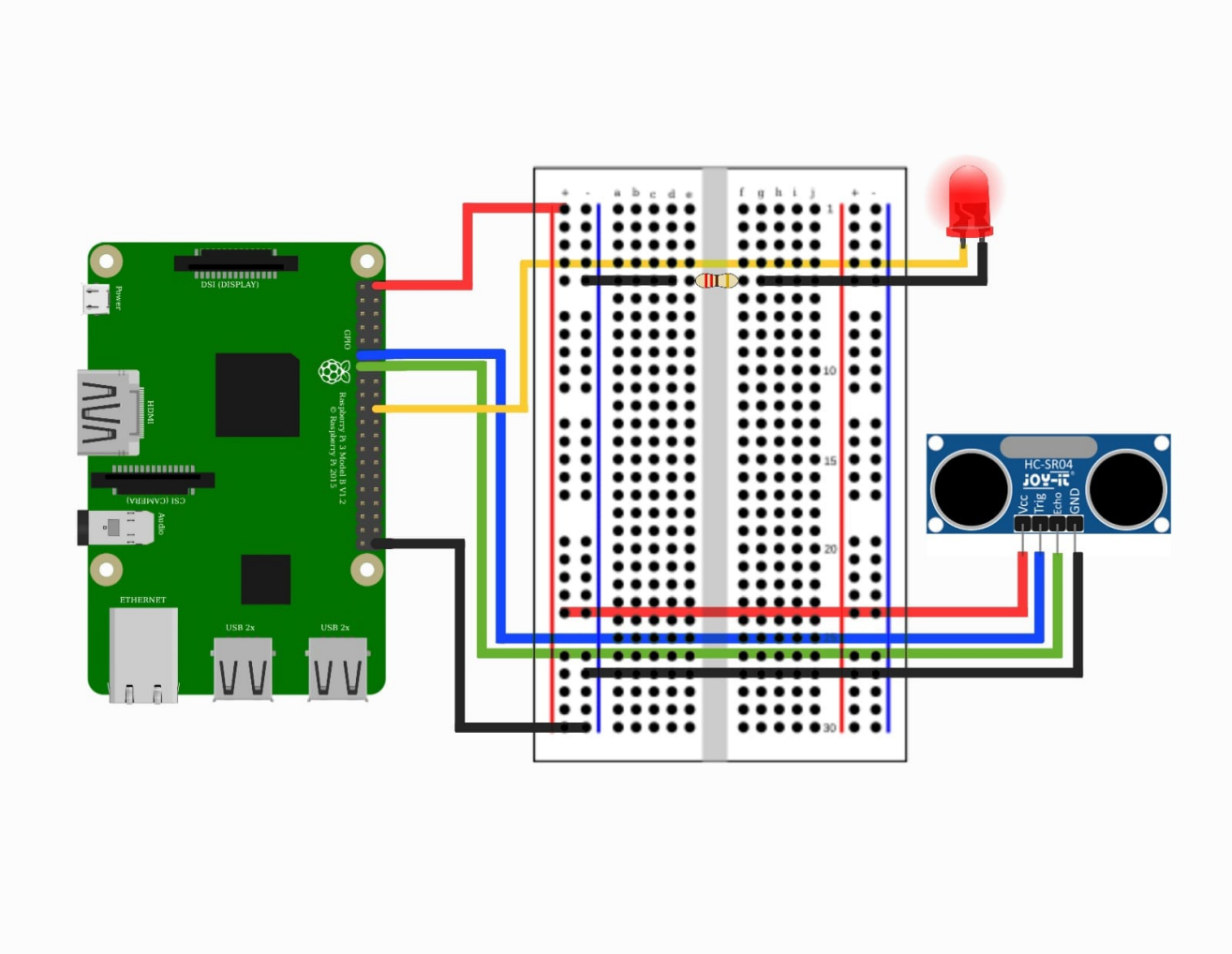
Monitoring Device

After that, you will need:

- a raspberry (we used a raspberry PI 3+)
- a led

- a sonar (HCSR04)
- a 220ohm resistor
- a breadboard

You will have to assemble those elements following this wiring scheme:



PROF

Then you will have to deploy the Monitoring Device control software, to do so, open a terminal inside the `Sprint2/MD` folder run:

```
gradlew build
```

After that, copy the `Sprint3/MD/build/distributions/monitoring_device-1.0.zip` folder inside the raspberry (for instance using `scp`) and unzip it

System activation

Firstly you have to activate the monitoring device, to do so connect to your raspberry via `ssh`, then move inside the `monitoring_device-1.0/bin` folder and run

```
./monitoring_device
```

Lastly, you have to activate the WIS system by opening a third terminal inside the `Sprint1/WIS` folder and running

```
gradlew run
```

N.B. Type `gradlew test` if you want to launch JUnit tests instead of activating the system demo.

Future Sprints

In the next sprint, we will focus on the GUI.

Our goal is to enable the possibility to connect to a web interface and check the state of the system.