Waste Incinerator Service

Sprint info

Sprint name	Sprint 1
Previous sprint	Sprint 0
Next sprint	Sprint 2
QAK model	sprint1.qak
Developed by	Alessio Benenati Giulia Fattori
Repo Site	WasteIncineratorService

Sprint Starting Condition and Goals

In the previous sprint, we focused on requirements analysis and produced a simple base architecture of what could be inferred from the assignment text.
In this one we will focus on the relationship between WIS and OpRobot, our goals are:

- finding the best way to divide the business logic between the OpRobot and the WIS actor
- consequently choosing the right model (**Actor** or **POJO**) for the OpRobot
- producing a simple prototype of the system reproducing the functioning of these two entities

Problem Analysis

WIS and system observability

Based on the requirements, the user interacts with the WIS not to change the system's state but to monitor it. From this, it can be deduced that the WIS must be able to retrieve information on the state of each system component. For this purpose, **it makes sense to make the WIS an observer of each component**.

PROF

WIS and OpRobot

Regarding the OpRobot the requirements do not provide enough information to determine with certainty how to model it. In particular, it is stated that the behavior actuator of the OpRobot is the DDRRobot, which is provided by the client as a **service** (BasicRobot). However, it is not specified whether this should be controlled by an autonomous actor or whether the WIS itself could control the BasicRobot.

At first glance, one might think that having the WIS control the DDRRobot could be a good idea because the execution cycle of the OpRobot requires observing the system's state to verify the initial conditions, and this information is already present in the WIS as it acts as an observer.

However, a more in-depth analysis reveals that OpRobot actually needs to verify the initial conditions only at two specific moments (at the beginning and the end of an execution cycle) and would not gain

significant advantages from continuously observing the state of the entire system (which would significantly increase the complexity of the WIS actor, which would have to both control the DDRRobot and update its internal representation of the system's state).

For these reasons, it is more convenient to **apply the Single Responsibility Principle by incorporating the logic for controlling the DDRRobot into a dedicated actor, the OpRobot**, which communicates with the WIS to ensure that the initial conditions are verified.

LoadRP and UnloadAsh

In a real system, **the OpRobot should be able to load and unload the RPs and their ashes**, and such changes to the system would be detected by the respective sensors (Scale and MonitoringDevice) without the need to exchange messages at the software level.

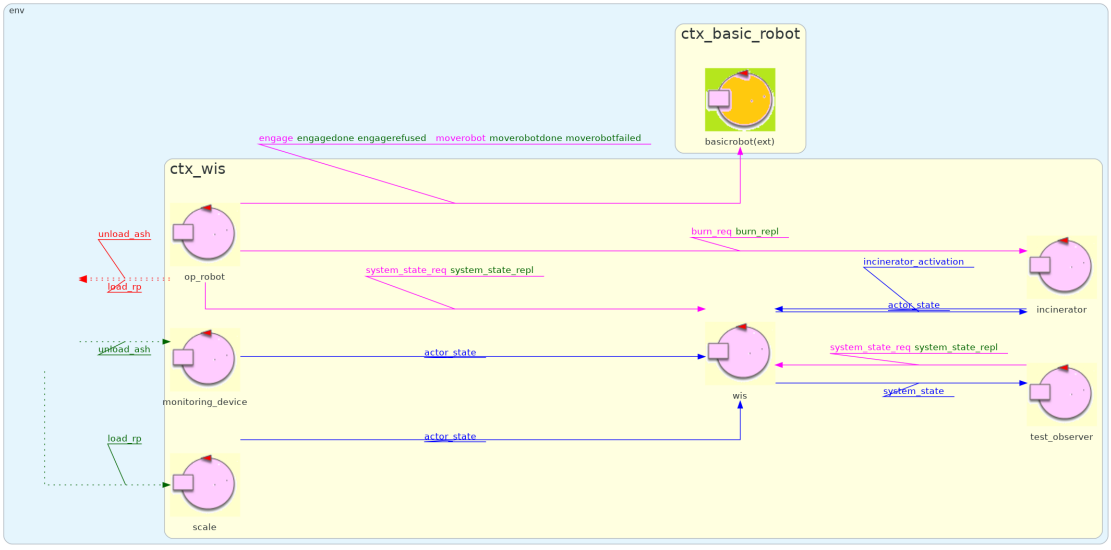
However, **since the current prototype operates in a purely virtual environment, it is necessary to simulate these two actions** by sending appropriate messages.

For this reason, we have decided to introduce two specific events, LoadRP and UnloadAsh, which modify the state of the Scale and MonitoringDevice, respectively.

Project

System Architecture

Based on the Problem Analysis carried out previously, we implemented an executable version of the system covering the discussed features; we attach here a visual representation of the system architecture:



wis_systemArch

PROF

Implementation and Deployment

During the implementation phase no particular need of additions to the inital project emerged, so the system architecture remained unchanged.

Test Plan

Test Class: WISTest

Test Name	Initial Condition	Expected Behavior
testIncinineratorActivation	WasteStorage contains 5 RP, AshStorge is empty, nobody empties AshStorage, Incinerator is inactive	Once the system is initialized, Incinerator is active
testOk5Rp	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

Usage

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`

After that, you will have to activate the BasicRobot, which 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

PROF

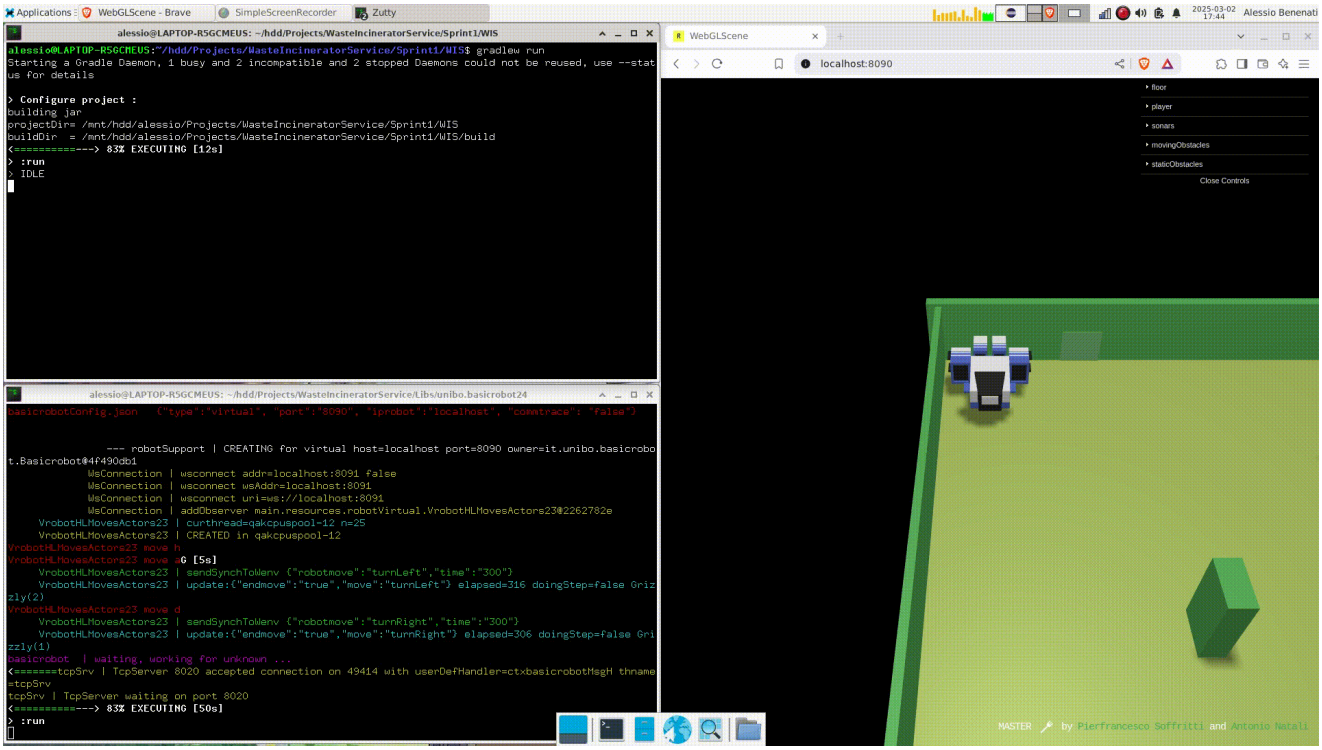
```
gradlew run
```

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.

Here we show the working system:



Future Sprints

In the next sprint, we will focus on the MonitoringDevice's behavior.

Our goal is to connect the actual prototype of the system to a real monitoring device deployed on a real raspberry.