Waste Incinerator Service

Sprint info

Sprint name	Sprint 0
Previous sprint	
Next sprint	Sprint 1
QAK model	sprint0.qak
Developed by	Alessio Benenati Giulia Fattori
Repo Site	WasteIncineratorService

QAK

The majority of the project has been modeled using QAK (Quasi Actor Kotlin), a meta-model created at UNIBO. QAK has its own DSL developed using xText that compiles directly into Kotlin code.
QAK allowed us to design the application with a higher level of abstraction, introducing the following main concepts:

- Actor: active entity modelled as finite state machines capable of sending and receiving messages.
- Context: an environment that contains some actors and abilitates them to communicate with other actors both in the same or in another context
- Interactions: abstractions of the main communications strategies (requests, dispatches, events).

We chose to use QAK because it helps bridge the abstraction gap, allowing us to maintain a higher level of technology independence during the initial phases of development.
You can find a detailed description of QAK here.

Requirements Analysis

Structure

Analyzing the natural language requirements text, we identified the following entities that should be modeled:

- ServiceArea
  - Home
  - BurnIn port
  - BurnOut port
  - WasteIn
  - AshOut
- OpRobot
- DDRRobot
- WIS
- Incinerator
- WasteStorage
  - Scale
  - RP
  - WRP
- AshStorage

    - MonitoringDevice
    - Sonar
    - Led
- ServiceStatusGUI

Interaction

From the requirements, we inferred the following messages that need to be modeled:

Information	Source	Destination	Description	Model
activationCommand	WIS	Incinerator	message sent by WIS during system initialization to activate the Incinerator	dispatch (according to customer)
endOfBurning	Incinerator	WIS,OpRobot	wireless signal sent by Incinerator when it ended burning	event (according to customer)
ashLevel	Sonar	unspecified	current state of the AshStorage	dispatch || event

Behavior

Incinerator

Incinerator is a component of unspecified dimensions and is positioned within the service area; its precise coordinates are currently unknown, but the area it occupies will not be accessible to other agents.
There are two doors, BURNIN and BURNOUT, located on the left side and the lower side of the Incinerator, respectively. Coordinates will be assigned to these doors based on the final position of the Incinerator itself.
According to the requirements, the Incinerator must communicate with the rest of the system by emitting network signals whenever there is a change of state; similarly, it must be able to receive and process communication signals emitted by other system components and, if necessary, modify its own state.
Before it can operate, the Incinerator must be activated via a wireless signal.

WIS

WIS (WasteIncineratorService) is a software system service that encompasses all the functionalities desired by the client.
According to the requirements, it must be able to monitor the status of the Incinerator, WasteStorage, and AshStorage to ensure that the system requirements are met before activating OpRobot.
Additionally, it is convenient to assign WIS the task of sending the activation signal to the Incinerator; this way, it is ensured that every time the system is activated, the Incinerator is also activated.

SSGUI

SSGUI is the graphical interface from which the user can monitor the status of the service and the system itself.

OpRobot

OpRobot is an entity defined by requirements to perform a series of tasks in a precise sequence, necessitating the ability to receive signals and updates from the rest of the system to modify its state and position.
The requirements do not specify where to place the business logic, so at this stage, we cannot define its exact responsibilities; these will be discussed during the Sprint1 problem analysis.
According to the requirements, OpRobot utilizes the DDRRobot provided by the customer.

MonitoringDevice

The MonitoringDevice is a component formed by the combination of a Sonar and an LED on a RaspberryPi, positioned above the ASHOUT door. The sonar measures the level of ashes present in AshStorage and emits an alarm when this exceeds a

certain value DLIMIT.
The LED is also used as an indicator, according to the legend specified in the requirements.
The MonitoringDevice does not run on the same node as the application, so to interact with it, it must send/receive messages, in addition to performing its own task.

WasteStorage

Outside the service area, at the WASTEIN door, there is the WasteStorage container, dedicated to the storage of material. The waste is grouped in the form of Roll Packets, each characterized by a specific weight not known in advance, which will be transferred exclusively from outside to inside the service area through WASTEIN; the Roll Packets represent the unit of measure of what enters the system.
WasteStorage has a scale (Scale) to measure the weight of the Roll Packet currently in the container, delivered periodically by an external agent; when the scale value is zero, WasteStorage can be considered empty. It has a maximum capacity of 50 kg, and if it is already full, it cannot accept further deliveries until it is emptied by OpRobot.
This element does not have its own behavior and does not communicate with the rest of the system; the only information of interest is that provided by the Scale.

AshStorage

At the ASHOUT door, there is the AshStorage container, into which the ash from the Incinerator is transported.
The container's capacity is equivalent to the ashes from 3-4 Roll Packets, and it is emptied periodically by an external agent.
The level of ash present in AshStorage at any given time is measured by the MonitoringDevice, so this container also does not have its own behavior or need to communicate with the rest of the system.

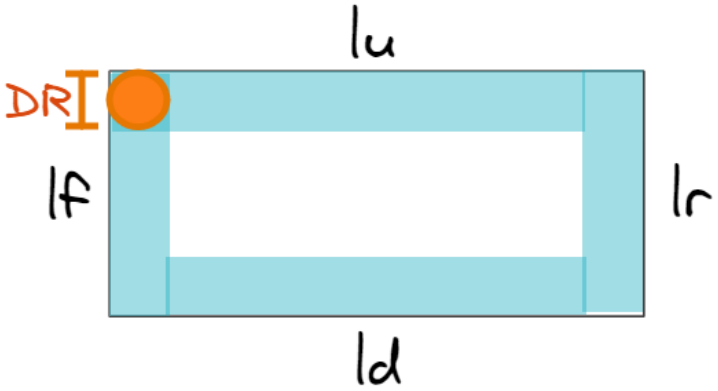
Models

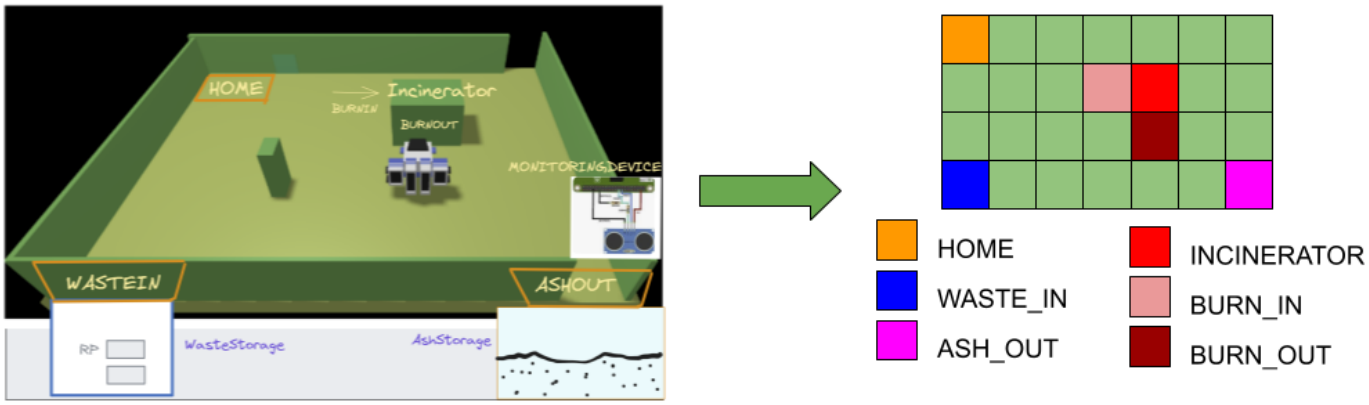
DDRRobot Model

The **OpRobot**, defined in the requirements as the robot controlled by the WIS, makes use of a DDRRobot (and its control software) provided by the customer. We link the detailed definition of DDRRobot and its qak control software.

Service Area Model

Using the DDRRobot given by the customet we have that the service area is modeled as a Euclidean space enclosed by boundaries, flat and rectangular, with sides of length `lf == lr` and `lu == ld`, measured in RD, a unit corresponding to the radius of the circle within which the provided robot (DDR Robot) can be contained.
To refer to any point in the room, a Cartesian coordinate system is used: two axes are identified, with `x` along `lu` and `y` along `lf`, originating from the top-left corner, which is the initial position of the robot, HOME, having coordinates (0, 0).
Having a measurement unit directly provided by the client allows us to formalize the space using this same unit as a reference.
Consequently, we can model a map of the service area by dividing the space into cells, each of length RD, and use this reference to position other elements within the room.
We will make the assumption that each port in the ServiceArea has dimension of one cell.





Incinerator

Given its behavior and its necessity to exchange messages, the Incinerator will be modeled as an **Actor**.

WIS

According to requirements, WIS is a service, so it will be modeled as an **Actor**.

MonitoringDevice

Given its behavior and its necessity to exchange messages, the MonitoringDevice will be modeled as an **Actor**.

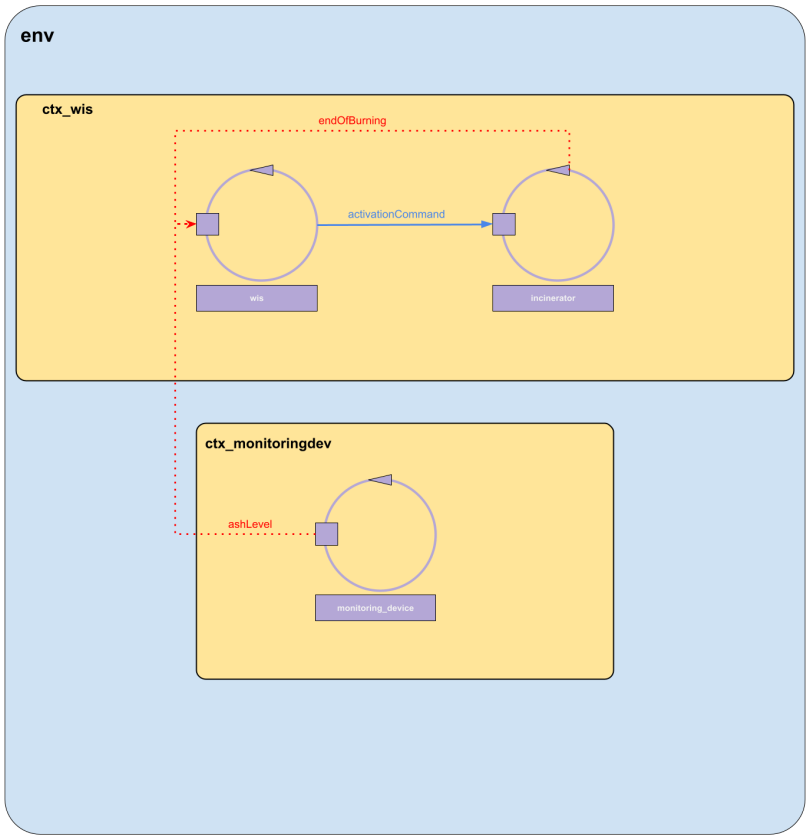
OpRobot

According to requirements, we do not have enough information to decide how to model the OpRobot (it could be both an Actor or a POJO) so we defer the discussion to future sprints.

WasteStorage, AshStorage, and Scale

Upon analyzing the requirements, we found that WasteStorage and AshStorage do not have any particular responsibilities, making it unnecessary to model them. The only entity that truly needs to be modeled is the Scale. However, at this stage of the project, we have yet to determine whether it will be modeled as a POJO, an Actor, or a variable. Therefore, this discussion will be deferred to future sprints.

Requirements Architecture



Functional Test Plan

Test Name	Initial Condition	Expected Behavior
testIncinineratorActivation	WasteStorage contains 4 RP, AshStorge is empty, nobody empties AshStorage, Incinerator is inactive	Once the system is inictialized, Incinerator is active
TestOk4Rp	WasteStorage contains 4 RP, AshStorge is empty and can contain the ashes of 3 RPs, nobody empties AshStorage	After some time WasteStorage contains 1 RP and AshStorage is full

Future Sprints

In the next sprint, we will focus on the OpRobot's behavior, analyzing its relations with the other components.
In particular, we will focus on the interactions between OpRobot and WIS, defining the responsibilities of both.
Our goal is also to connect the OpRobot to a virtual environment (the 'VirtualRobot' provided by the customer) so that it will be simple to switch to a physical OpRobot at any time by only changing a configuration parameter.