

# WASTE INCINERATOR SERVICE

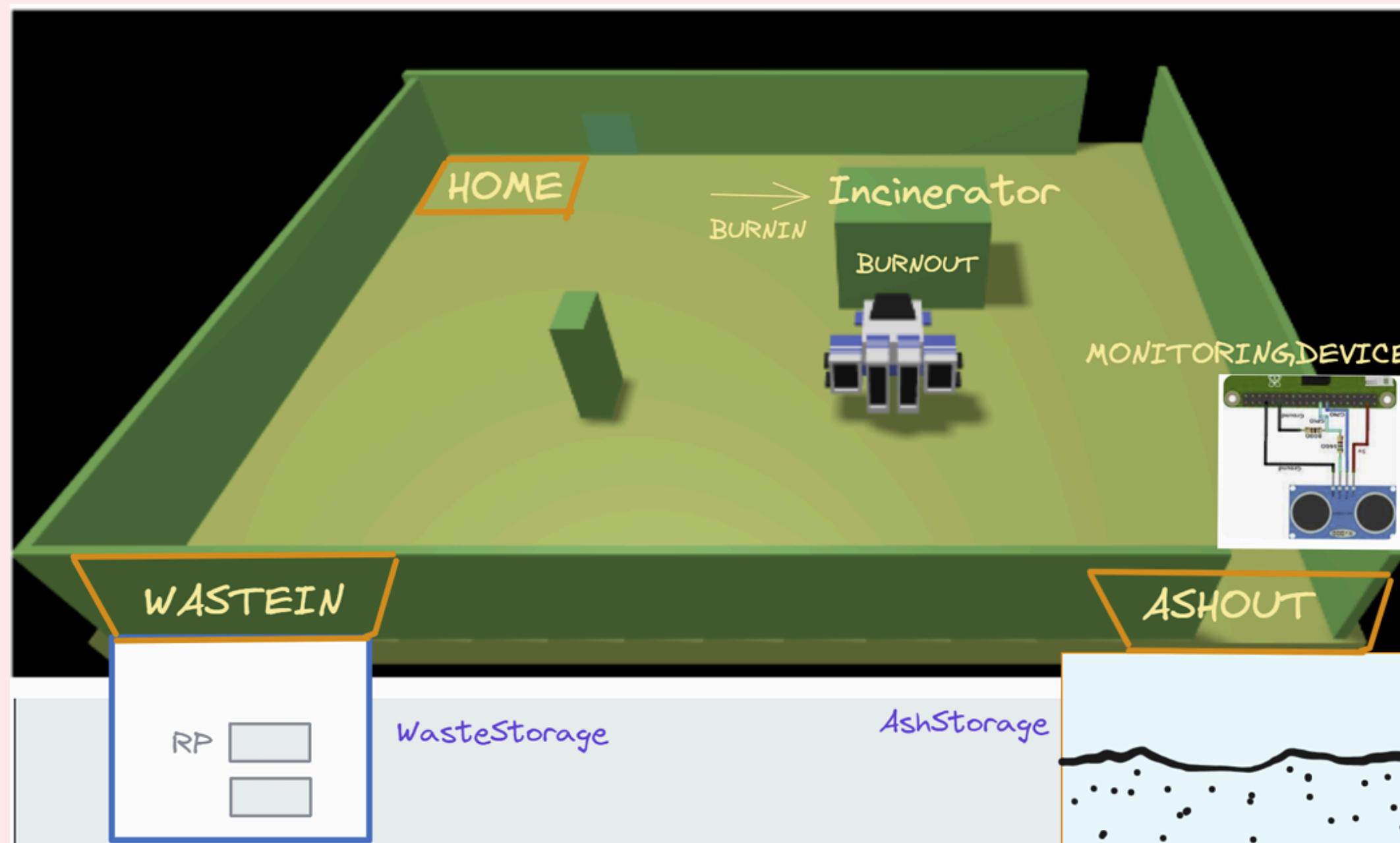
Tema Finale del corso di  
Ingegneria dei Sistemi Software  
2024



A cura di Corsetti Luca, de Respinis Valentina, Sindaco Daniele

# Il Progetto

Una compagnia ha richiesto la progettazione di un WastelIncineratorService (WIS) per trattare lo smaltimento dei rifiuti attraverso incenerimento e richiede un servizio che controlli un robot per lo spostamento dei rifiuti.



# Sprint0 - Analisi Dei Requisiti

Il **WasteStorage** container possiede:

- una bilancia come **weighing device**: riporta il peso dei vari Roll Packets (**RP**) depositati al suo interno.
- porta di ingresso **WASTEIN** all'interno della service area.



# Sprint0 - Analisi Dei Requisiti

L'**AshStorage** container può contenere al massimo le ceneri di **3-4 RP**.

Il container possiede una porta di ingresso **ASHOUT** all'interno della service area.



# Sprint0 - Analisi Dei Requisiti

Il **MonitoringDevice** è composto da

- un Sonar
- un Led

posti su un RaspberryPi.

**Sonar:** misura il **livello delle ceneri** nell'AshStorage.

**DLIMIT:** valore limite al di sotto del quale l'AshStorage container è considerato pieno.

**Led** (warning device): segnala lo **stato dell'Inceneritore** e dell'AshStorage.

Il **Sonar** e il **Led** sono **dispositivi fisici**. La distanza **DLIMIT** sarà quindi una variabile, intera e positiva.

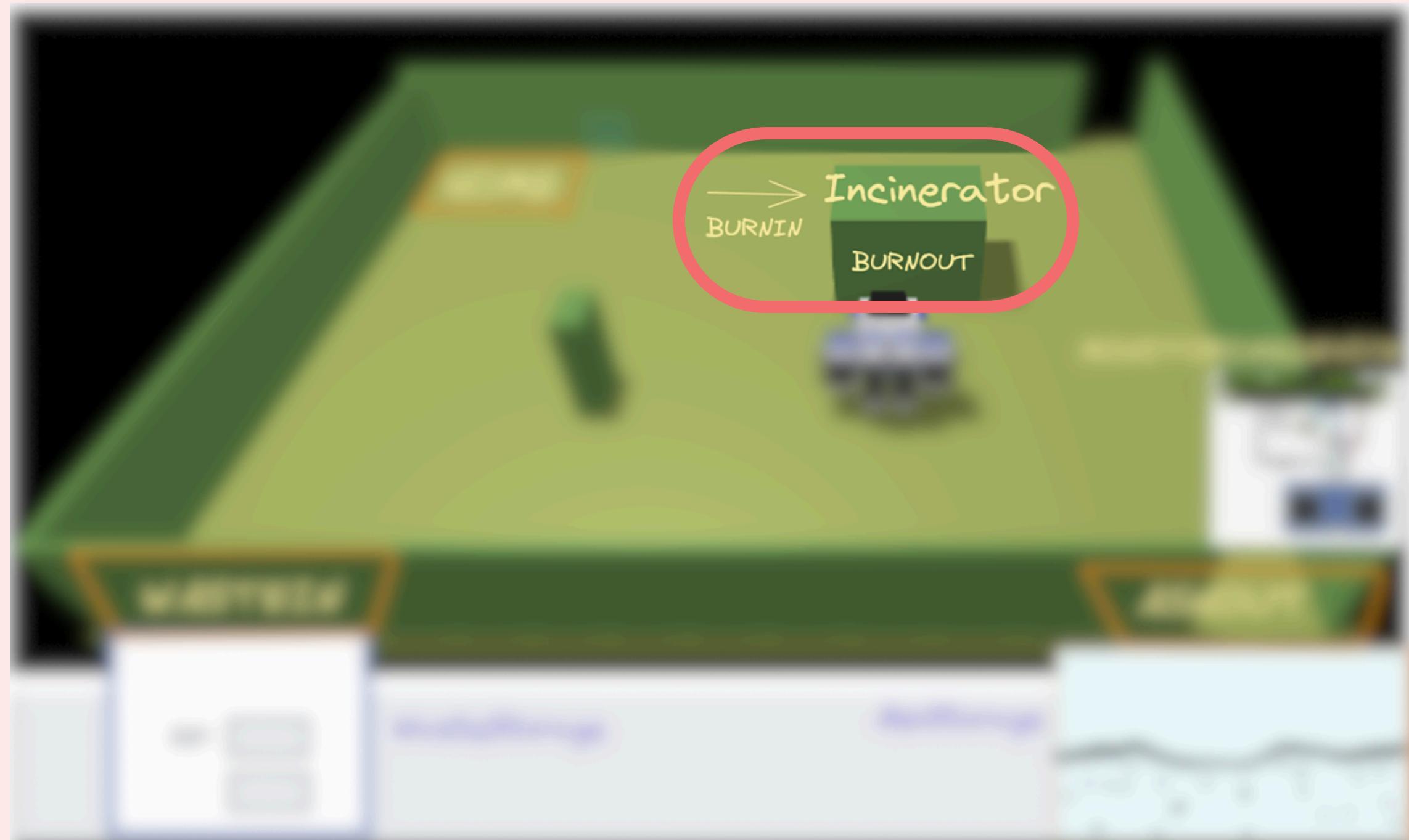


# Sprint0 - Analisi Dei Requisiti

## Incinerator:

- può bruciare un RP alla volta in BTIME secondi,
- possiede due porte, **BURNIN** e **BURNOUT**, che permettono rispettivamente l'ingresso del RP e l'uscita delle ceneri dall'Inceneritore.

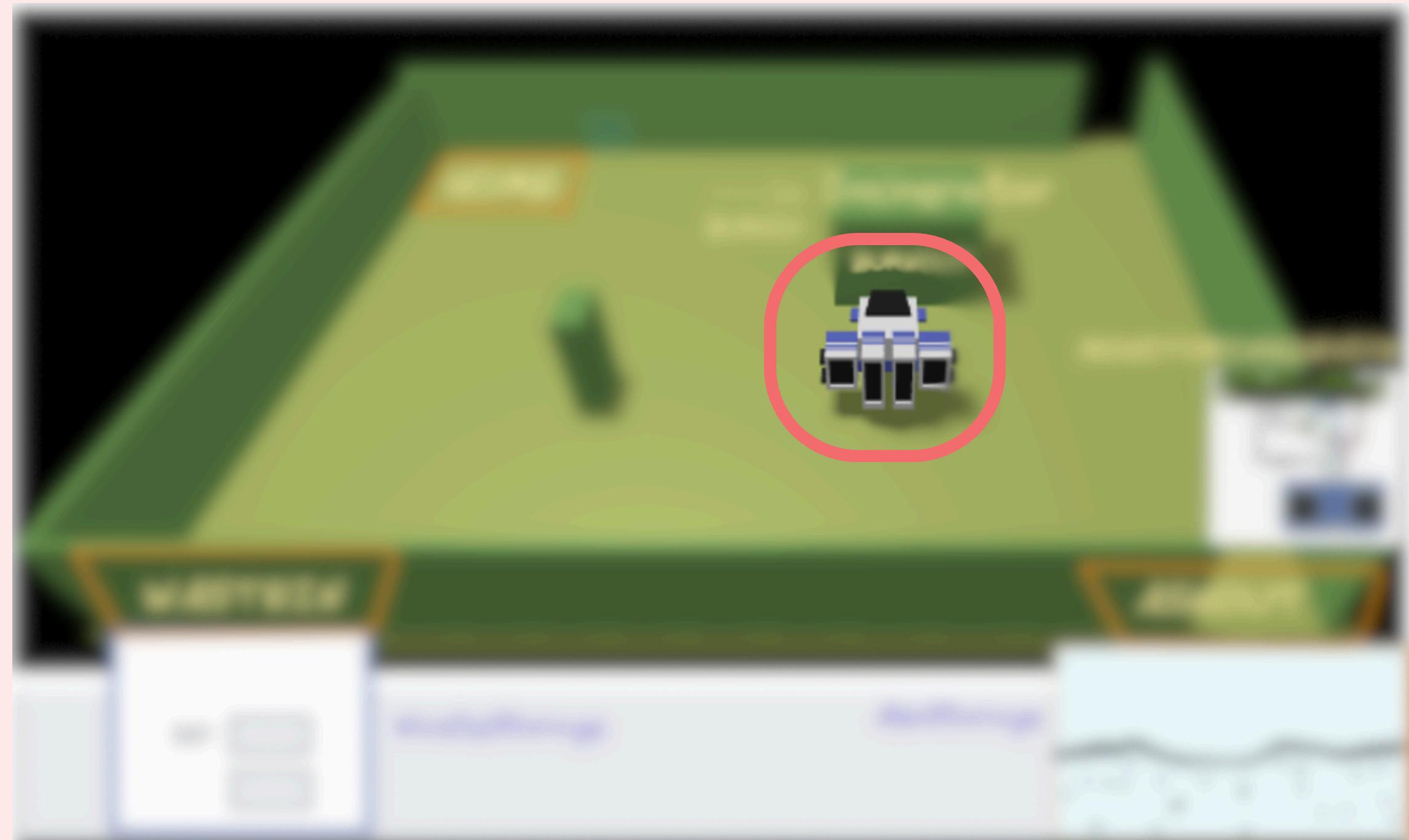
BTIME sarà una variabile, intera e positiva.



# Sprint0 - Analisi Dei Requisiti

## L'OpRobot:

- è un componente proattivo del sistema.
- ha un comportamento specifico in risposta a determinate condizioni.



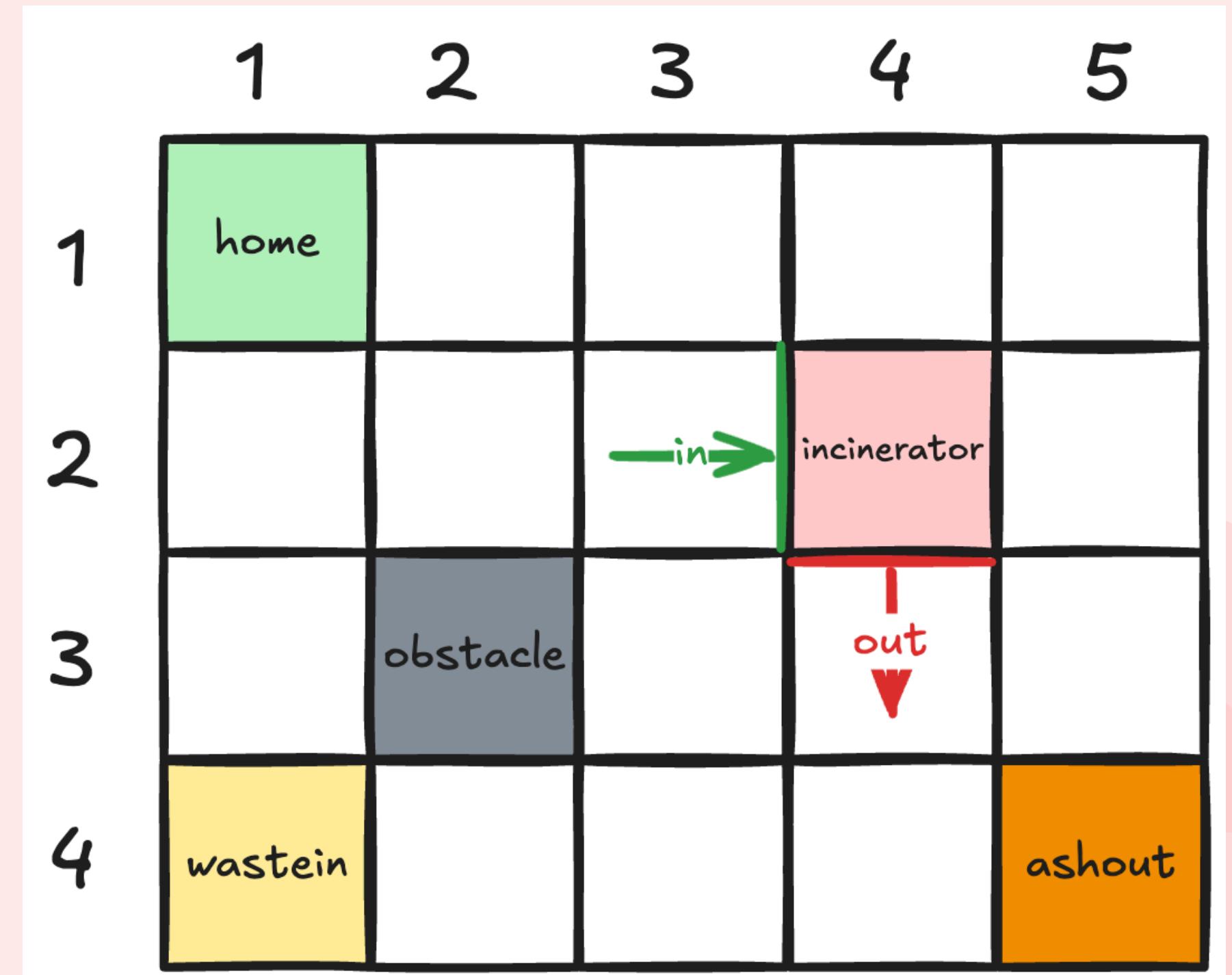
# Sprint0 - Analisi Dei Requisiti

La **service area** è rappresentabile attraverso una matrice di X righe e Y colonne.

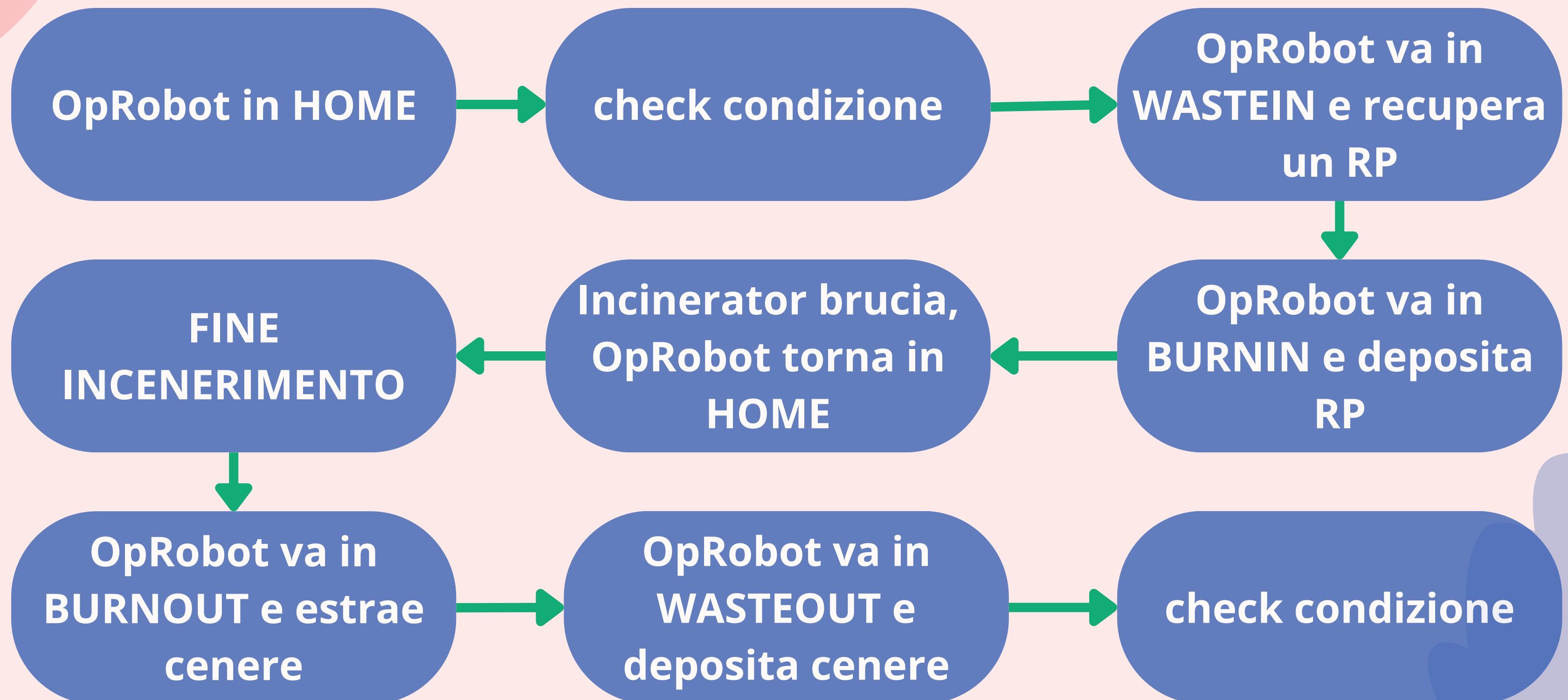
La singola cella avrà dimensione pari a quella del robot, prendiamo come riferimento quindi il DDR Robot.

All'esterno della service area sono presenti:

- il WasteStorage
- l'AshStorage
- il MonitoringDevice.



# User Stories

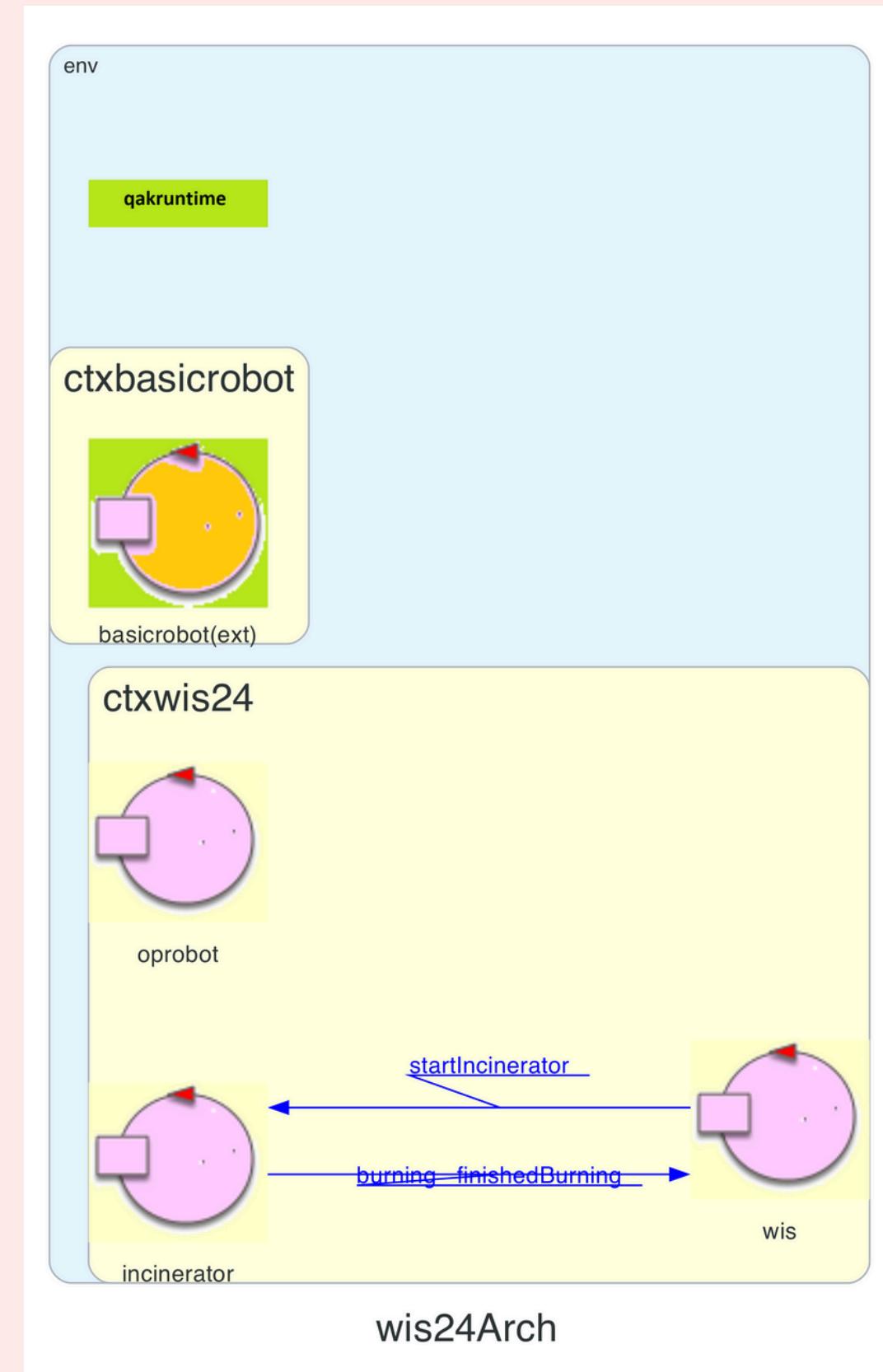


# Sprint0 - Analisi Dei Requisiti

- Il servizio **WasteIncineratorService** non trova una facile corrispondenza in Java.
- Utilizziamo il **QAK language**, un **DSL (Domain Specific Language)**, per colmare l'abstraction gap.
- Il sistema, per essere modellato, necessita del termine di **attore**, ovvero un **componente software** dotato di un **comportamento autonomo**, capace di inviare, ricevere e gestire messaggi in un **contesto distribuito**.

Gli attori **non hanno memoria condivisa** e interagiscono tra di loro attraverso **scambi di messaggi**, che possono essere:

- **dispatch**
- **request**
- **event**



# Divisione del lavoro

## Sprint 1

### GOAL

- Implementazione del **movimento dell'OpRobot** nella Service Area, nell'ordine corretto.

## Sprint 2

### GOAL

- Implementazione **estrazione e deposito** di RP e di Ash.
- Modellazione del MonitoringDevice (**Sonar e Led**) e **Scale**.

## Sprint 3

### GOAL

- Implementazione della **ServiceStatusGUI**.

# Sprint1 - Movimento OpRobot

Necessità di comunicare con l'OpRobot per:

- Recuperare un **RP** dal WasteStorage, quando è soddisfatta la condizione: “*WasteStorage non vuoto, AshStorage non pieno, Incinerator non sta bruciando*”
- **Depositare** il Roll Packet nella **BURNIN** port dell’Incinerator
- Tornare alla posizione iniziale **HOME**
- **Estrarre** la cenere al termine della fase di incenerimento del RP
- **Depositare** la cenere estratta **nell’AshStorage**

# Sprint1 - Analisi del Problema

Problematiche evidenziate durante l'analisi dei requisiti: **Utilizzo di messaggi per comunicare con OpRobot**

Questi messaggi potrebbero essere modellati come

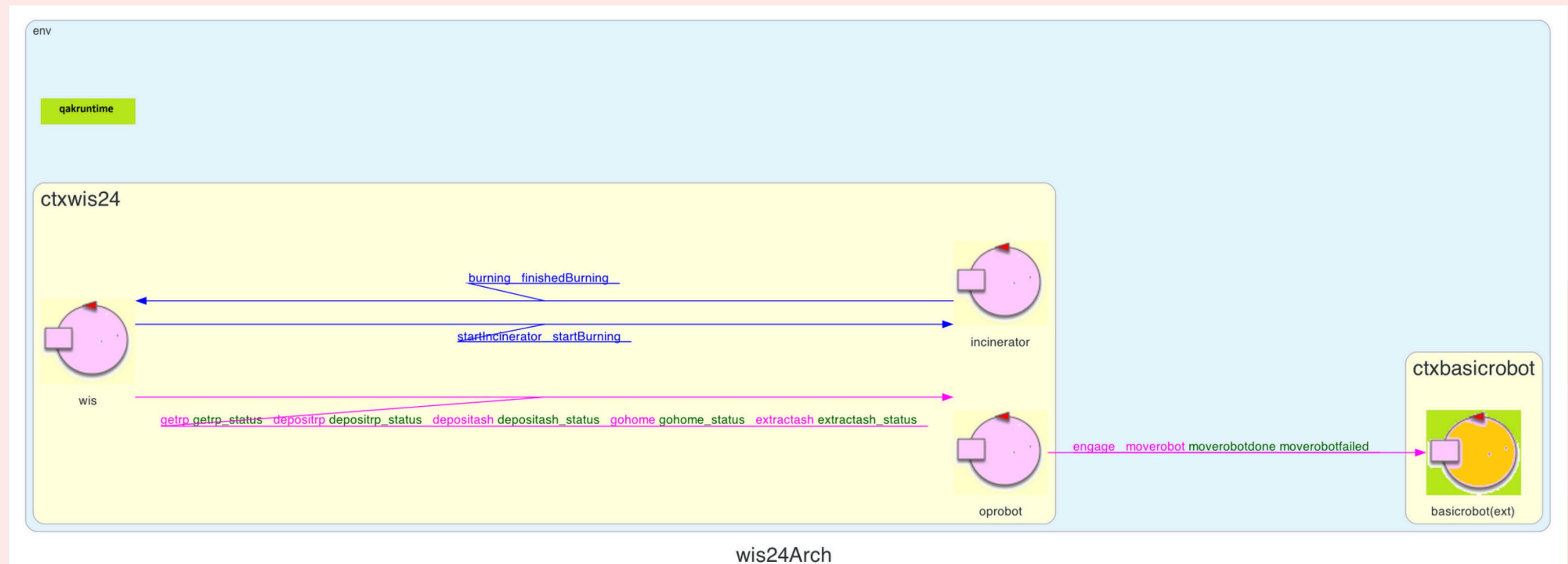
- un Event
- una Dispatch
- una **Request** inviata dall'attore WIS all'attore OpRobot seguita da una **Reply** inviata dall'OpRobot al WIS per notificare il completamento del task.

Preferibile **Request-Reply** per la comunicazione tra WIS e OpRobot, poiché:

- si evita un meccanismo di polling per verificare lo stato del task da parte del WIS.
- questi messaggi sono diretti a singoli componenti del sistema

# Sprint1 - Modello del Sistema

- Più responsabilità all'attore WIS



Source: [Link a Sprint1](#)

# Sprint2 - AshStorage

**MonitoringDevice** composto da un **Led fisico** e un **Sonar fisico**.

- **Sonar**: misura il **livello** della **cenere** nell'**AshStorage**.
- **Led**: segnala lo **stato dell'Incinerator** e **dell'AshStorage**.
  - **Acceso** quando l'Incinerator sta bruciando un RP
  - **Spento** quando l'Incinerator non sta bruciando
  - **Lampeggia** quando l'AshStorage è pieno

**AshStorage**:

- **Considerato pieno**, quando il Sonar misura un **valore minore di DLIMIT**
- ha capienza di 3-4 RP
- È necessario **leggere il valore del Sonar** per determinare lo **stato dell'AshStorage**.
- Il **WIS** dovrà conoscere lo stato **dell'Incinerator** e il **livello misurato dal Sonar** per determinare lo stato d'accensione del Led.

# Sprint2 - WasteStorage

Presenza di una bilancia (**Scale**) nel **WasteStorage**:

- misura il **peso** di **tutti gli RP presenti** nel WasteStorage.

WasteStorage considerato **vuoto** quando il **valore** della Scale è (circa) 0.

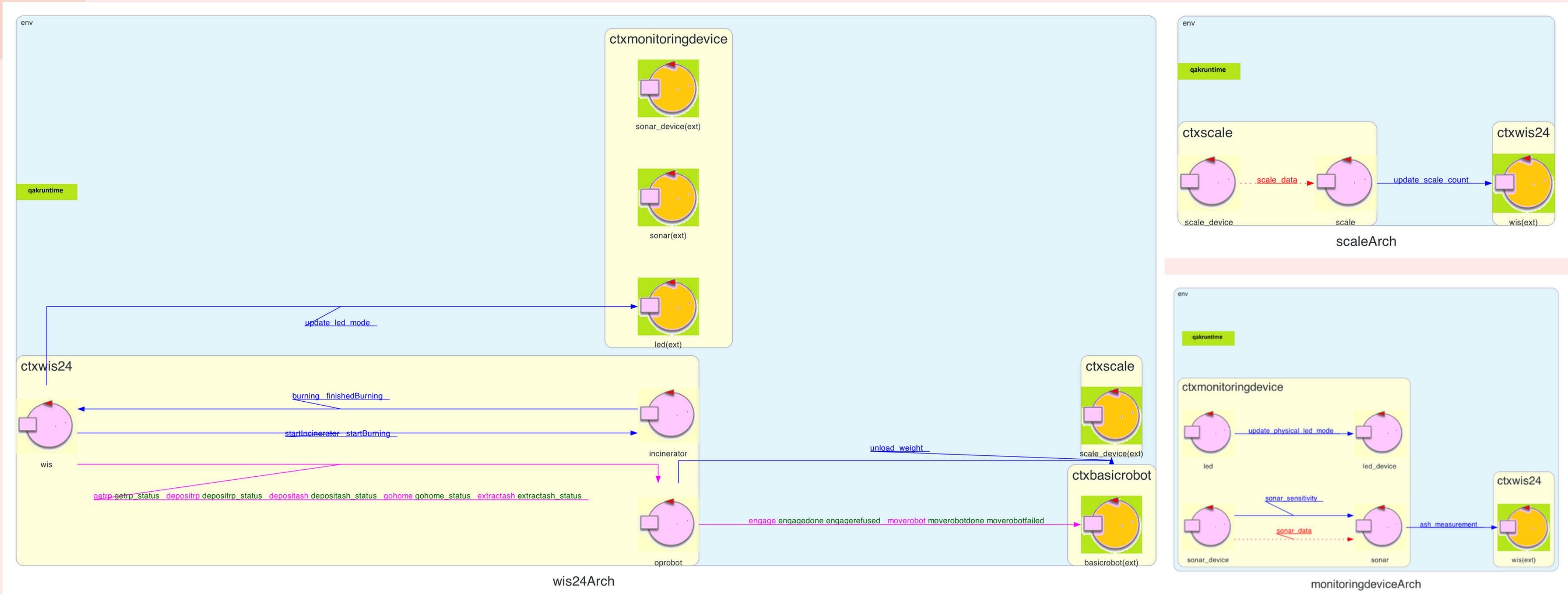
# Sprint2 - Analisi del Problema

- **MonitoringDevice**: Necessaria una connessione per la comunicazione tra RaspberryPi e il pc su cui girerà il WIS.
- **Sonar**:
  - Attore di **alto livello per la logica** e attore di **basso livello per la gestione del dispositivo fisico**.
  - Il livello può cambiare in momenti casuali
    - non è consigliabile un meccanismo di polling da parte del WIS.
    - Preferibile che sia il Sonar ad aggiornare il WIS non appena percepisce un cambiamento nel livello.
- Idonea una Dispatch, non essendo necessaria una risposta ed essendo il messaggio destinato a un singolo attore.
- **Led**:
  - Attore di **alto livello per la logica** e attore di **basso livello per la gestione del dispositivo fisico**. L'attore di alto livello riceverà dal WIS il comportamento da adottare che verrà successivamente inoltrato all'attore che gestirà il dispositivo fisico di pertinenza.

# Sprint2 - Analisi del Problema

- **WeighingDevice:** Il committente non ha fornito nessun dispositivo fisico, quindi si potrebbe
    - inserire l'attore dello Scale nello stesso contesto del WIS
    - creare un nuovo contesto qualora il dispositivo venga fornito in seguito.
  - **Scale:**
    - Attore di alto livello e attore di basso livello. L'attore di alto livello che gestisce la logica si occuperà di convertire il peso ricevuto dall'attore di basso livello che gestisce il dispositivo fisico in RP.
    - Il peso può cambiare in momenti variabili
      - non è consigliabile un meccanismo di polling da parte del WIS.
      - preferibile che sia la Scale ad aggiornare il WIS non appena percepisce un cambiamento di peso.
- Utile una Dispatch, non essendo necessaria una risposta ed essendo il messaggio destinato a un singolo attore.

# Sprint2 - Modello del Sistema



Source: [Link a Sprint2](#)

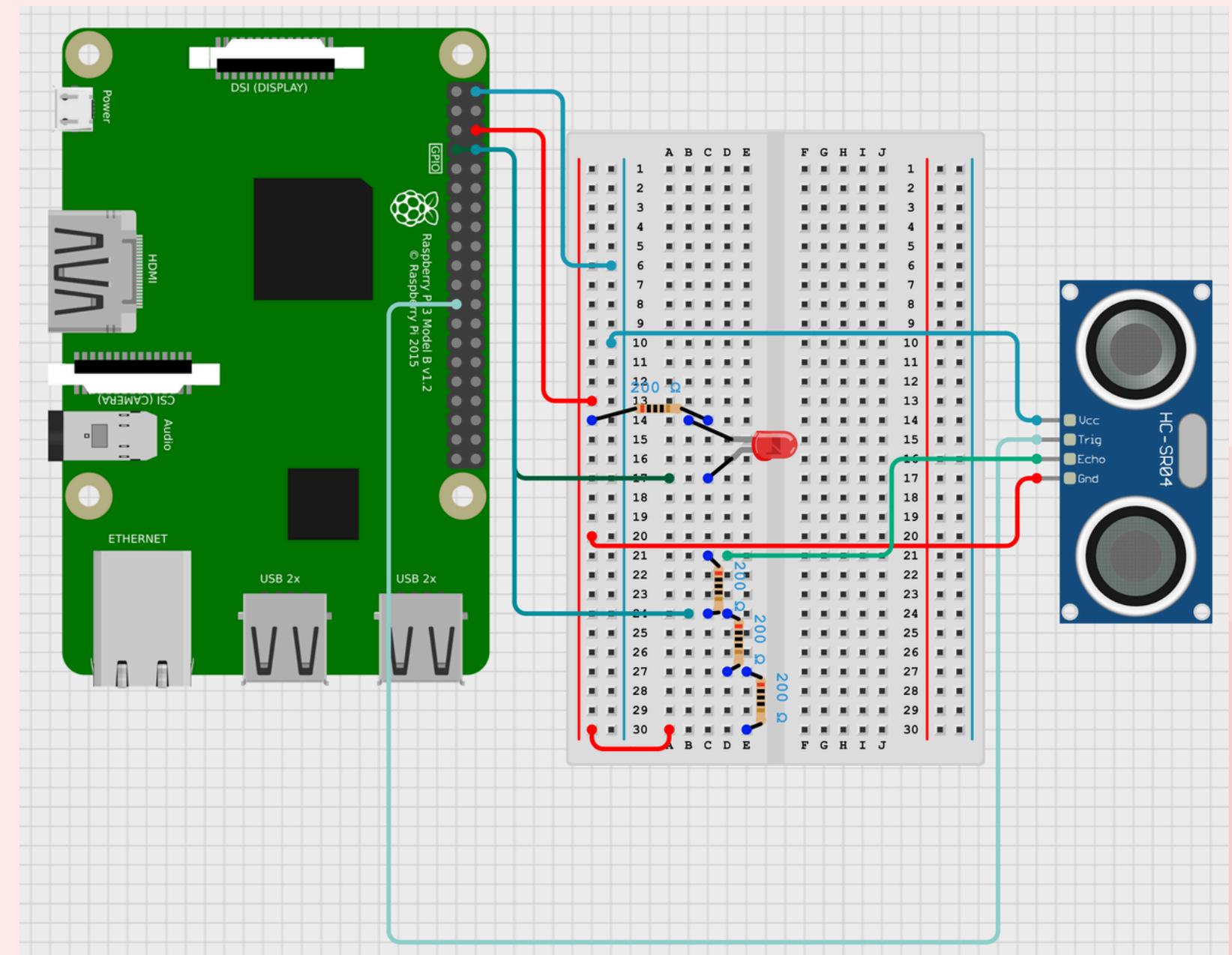
# Sprint3 - Analisi del Problema

- Implementare una **GUI** che permetta di ricevere informazioni dal sistema e far **visualizzare lo stato attuale a un'osservatore esterno**.
- **Non sono richieste interazioni con agenti esterni.**
- **Non sono stati esplicitati metodi di interazione specifici con il sistema.**
- Non essendo specificato il modo in cui la GUI riceve le informazioni dal sistema, sono presenti varie possibilità:
  - La **GUI sfrutta un sistema di Polling** ad ogni attore per acquisire le informazioni.
  - La **GUI riceve informazioni** ad ogni cambiamento di stato degli attori **grazie ad un attore specifico**.
  - La **GUI riceve informazioni** riguardo lo stato degli attori direttamente **dal WIS**.
- Considerando il meccanismo di Request-Reply utilizzato dal WIS, che permette di tenere aggiornato lo stato del sistema, si consiglia la **scelta della terza opzione**.
- La nostra software house ha in passato implementato in un progetto Facade24Start. Non essendo specificate le modalità per l'implementazione della GUI e considerando le similitudini al progetto sopra menzionato, si consiglia di utilizzare la stessa implementazione.

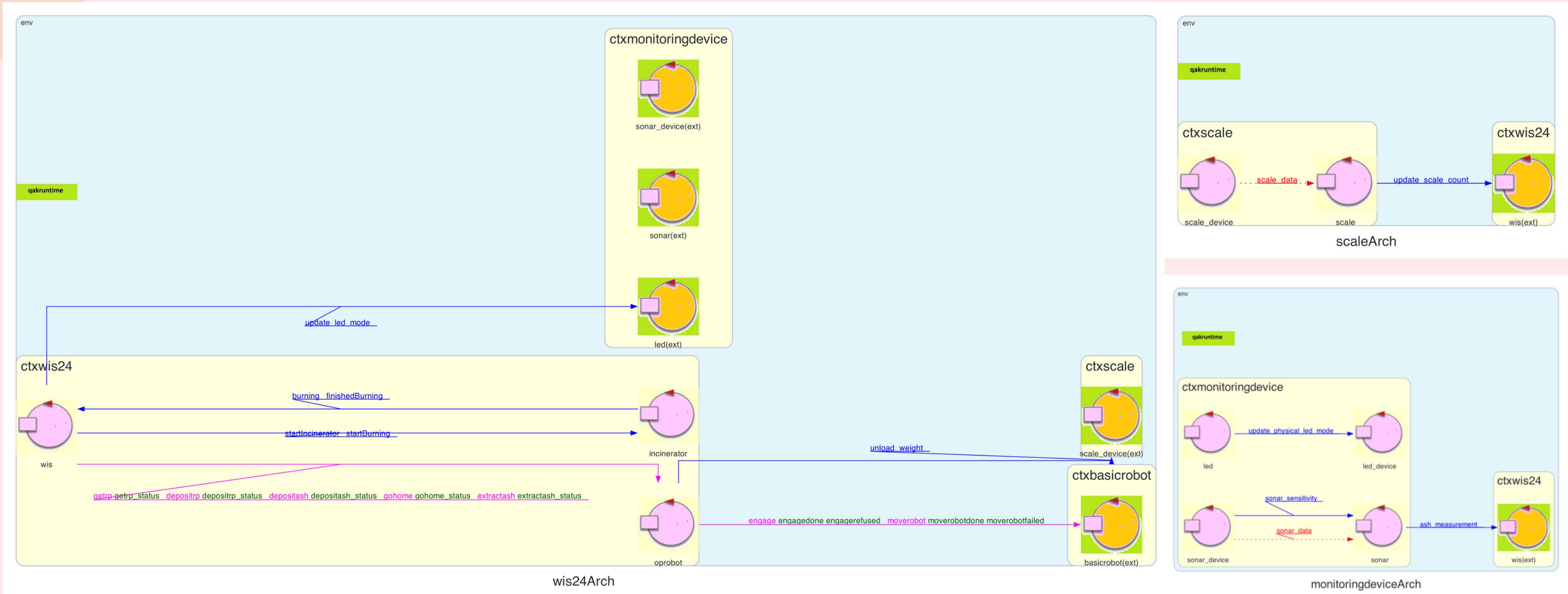
# Sprint3 - Modello del Sistema

Il modello del sistema è uguale a quello risultante dallo Sprint2.

Implementazione fisica del Sistema:



# Architettura Finale



Source: [Link a Sprint3](#)

# Timeline

Dicembre 2024

Gennaio 2025

Febbraio 2025

Marzo 2025

**SPRINT 0**

**SPRINT 1**

**SPRINT 2**

**SPRINT 3**

# Deployment & Esecuzione

- PRE-REQUISITO: gli eseguibili devono essere configurati per comunicare tra di loro, altrimenti le componenti del sistema non comunicheranno tra di loro. È quindi necessario **pre-configurare gli indirizzi ip** degli attori in esecuzione sui dispositivi fisici esterni (Led & Sonar).
- Le nuove versioni dell'applicazione verranno rese disponibili su [github](#)
- Sul raspberry -> `cd monitoringdevice-1.0 && ./bin/monitoringDevice`
- Sul dispositivo scelto per eseguire il sistema **WasteInceneratorService**, deve essere presente Docker per eseguire:
  - il **broker mqtt** -> `cd iss\_project\_2024/project/mqtt && docker-compose up`
  - il **virtual robot** -> `cd iss\_project\_2024/project/unibo.basicrobot24 && docker-compose -f basicrobot24.yaml up`
- Dopodichè, sempre sullo stesso dispositivo, eseguire:
  - **weighingDevice** -> `cd weighingDevice-1.0 && ./bin/weighingDevice`
  - **wis24** -> `cd wis24-1.0 && ./bin/wis24`

Grazie per  
l'attenzione.

