

# Sprint0V1

## Introduzione

### Il goal dello Sprint0

<https://github.com/anatali/issLab23/blob/main/iss23Material/html/TemaFinale23.html>

Lo Sprint0 si concentra sull'analisi dei requisiti forniti dal committente, riportando dettagli e chiarimenti circa il TemaFinale23, al fine di eliminare eventuali ambiguità.

Successivamente, si può procedere delineando la struttura complessiva dei macro-componenti.

A conclusione dello Sprint0, verrà definito un piano di lavoro che schedulerà gli Sprint successivi.

### Analisi dei requisiti

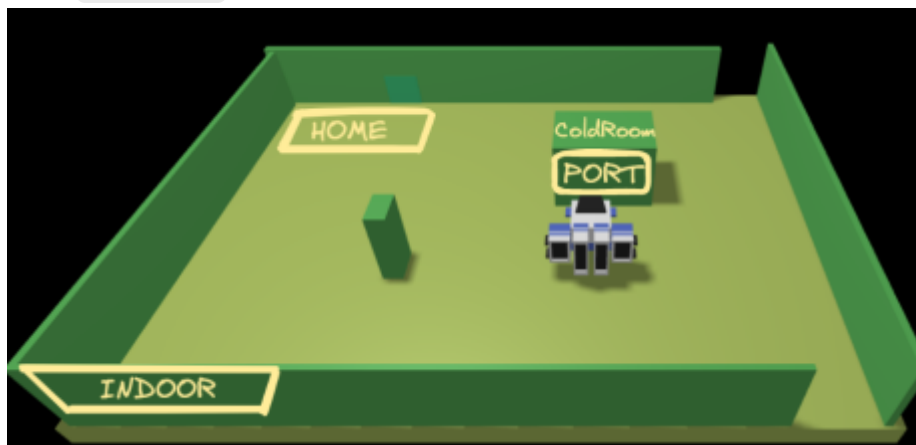
Il sistema **ColdStorageService** è composto dai seguenti componenti.

#### Transport trolley

- Viene fornito un **DDR robot** che implementa un **transport trolley**.
  - Inizialmente, il **transport trolley** è situato alla posizione **HOME** come in figura
  - Il **transport trolley**
    - ha la forma di un quadrato con i lati di lunghezza **RD**
    - deve essere capace di raccogliere il cibo da **Fridge truck** situato nel **INDOOR**
    - deve essere capace di trasportare il cibo dal **INDOOR** alla porta della **ColdRoom**
    - deve essere capace di depositare il cibo nella **ColdRoom**
- Le dimensioni dei componenti descritti in seguito sono espresse in termini di misure robotiche **RD**.

#### Area di servizio

- La **Service area** è uno spazio rettangolare e piatto di dimensioni  $h\_area$  **RD**,  $b\_area$  **RD**. Essa include:
  - una porta **INDOOR** per immettere il cibo. La porta INDOOR è posizionata in coordinate (IndoorX, IndoorY)
  - un contenitore **ColdRoom** destinato a contenere il cibo, fino ad una quantità massima di **MAXW** kg (**MAXW** numero reale positivo definito dal committente). La **ColdRoom** è costituita da una porta in posizione non specificata.
- La **ColdRoom** è posizionata nella **Service area** come nella figura



#### Service Access GUI

- Deve essere fornita un'interfaccia grafica (GUI) per permettere agli utenti di interagire con il sistema **ColdStorageService**
- La **GUI** deve mostrare il **peso corrente** del cibo depositato nella **ColdRoom**
- Gli utenti (**Fridge Truck drivers**) devono essere capaci di mandare una **richiesta di deposito** con una nuova quantità **FW** kg di cibo (**FW** variabile intera positiva)
- La **richiesta di deposito** può essere
  - **accepted** se la **ColdRoom** ha la capacità sufficiente per depositare **FW** kg di cibo
  - **rejected** se la **ColdRoom** non possiede la capacità sufficiente.
- Se la richiesta viene accettata, la GUI deve generare un **Ticket**
  - Il **ticket** ha una scadenza di **TICKETTIME** secondi a partire dalla sua generazione
  - Il **ticket** scaduto non è più **valido**.
- LA **GUI** deve avere un campo per immettere un **ticket number** quando un **Fridge truck** è arrivato a **INDOOR**

## Service Status GUI

- Un **Service Manager** deve monitorare lo **stato del servizio** attraverso un'interfaccia grafica **Service Status GUI**
- Per **stato del servizio** si intende:
  - il **peso corrente** del cibo depositato nella **ColdRoom**
  - lo **stato attuale** e la **posizione** del **transport trolley** nella **Service area**
- La **GUI** deve tenere traccia del **numero di richieste di deposito rejected** a partire dall'inizio del servizio

## Alarm requirements

Il committente specifica requisiti di comportamento del sistema al verificarsi di determinate situazioni, includendo a tal scopo altri componenti

- Il sistema deve includere un **Sonar** e un **LED** connesso a un Raspberry Pi
- Il **Sonar** si deve comportare come un **alarm device**
- Un **alarm device**
  - misura la distanza periodicamente
  - se la distanza è inferiore ad una distanza prefissata **DLIMIT**, il **transport trolley** deve fermarsi finché l'**alarm device** non rileva una distanza superiore a **DLIMIT**
- Il **LED** deve comportarsi come un **warning device**
- In particolare, secondo questo schema:
  - **LED OFF** quando il **transport trolley** è a **HOME**
  - **LED BLINKING** quando il **transport trolley** è in movimento
  - **LED ON** quando il **transport trolley** è fermo
- Quando il **transport trolley** è in movimento, l'**Alarm requirements** deve essere soddisfatto.
- Il **transport trolley** non deve essere fermato se non sono passati **MINT** millisecondi dal precedente stop

## Service users story

Un caso d'uso può essere riassunto in questi punti:

- Un Fridge truck driver utilizza la Service Access GUI per mandare una richiesta di deposito per il suo carico di **FW** kg
  - Se la richiesta viene accettata, il driver guida il suo truck all'**INDOOR** del servizio in un tempo **TICKTTIME**
- Quando il truck è all'**INDOOR** del servizio, il driver usa la Service Access GUI per immettere il **ticket number** e aspetta fino a quando non compare il messaggio **charge taken** sul Service Access GUI. A questo punto, il truck deve lasciare l'**INDOOR**
- Quando il servizio accetta un **ticket**, il **transport trolley** deve:
  - raggiungere l'**INDOOR**
  - raccogliere il cibo
  - mandare il messaggio **charge taken**
  - raggiungere la **ColdRoom** per depositare il cibo
- Quando l'azione di deposito è terminata, il transport trolley accetta un altro **ticket** (se c'è) oppure ritorna a **HOME**

## Architettura logica

Viene fornito un **DDR robot** che implementa un **transport trolley**.

Il **DDR robot** è una entità attiva. L'interazione avviene per mezzo di scambio di messaggi.

Il software per la modellazione del **DDR robot** è fornito dal committente.

unibo.basicrobot23:

<https://github.com/anatali/issLab23/tree/b04de6a7f33fcfabaf93f9e06b46feb31931fa83/unibo.basicrobot23>

Questi sono i messaggi definiti nella documentazione.

## System basicrobot23

```
Dispatch cmd      : cmd(MOVE)      //MOVE=w|s|d|a|r|l|h
Dispatch end      : end(ARG)

Request step      : step(TIME)
Reply stepdone    : stepdone(V)
Reply stepfailed  : stepfailed(DURATION, CAUSE)

Event sonardata   : sonar( DISTANCE )      //percepito da sonarobs/engager
Event obstacle    : obstacle(X)

Request doplan    : doplan( PATH, STEPTIME )
Reply doplandone  : doplandone( ARG )
Reply doplanfailed : doplanfailed( ARG )

Dispatch setrobotstate: setpos(X,Y,D)
Dispatch setdirection : dir( D ) //D =up|down!left|right

Request engage    : engage(CALLER)
Reply engagedone  : engagedone(ARG)
Reply engagerefused : engagerefused(ARG)

Dispatch disengage : disengage(ARG)

Event alarm       : alarm(X)
Dispatch nextmove : nextmove(M)
Dispatch nomoremoves : nomoremoves(M)

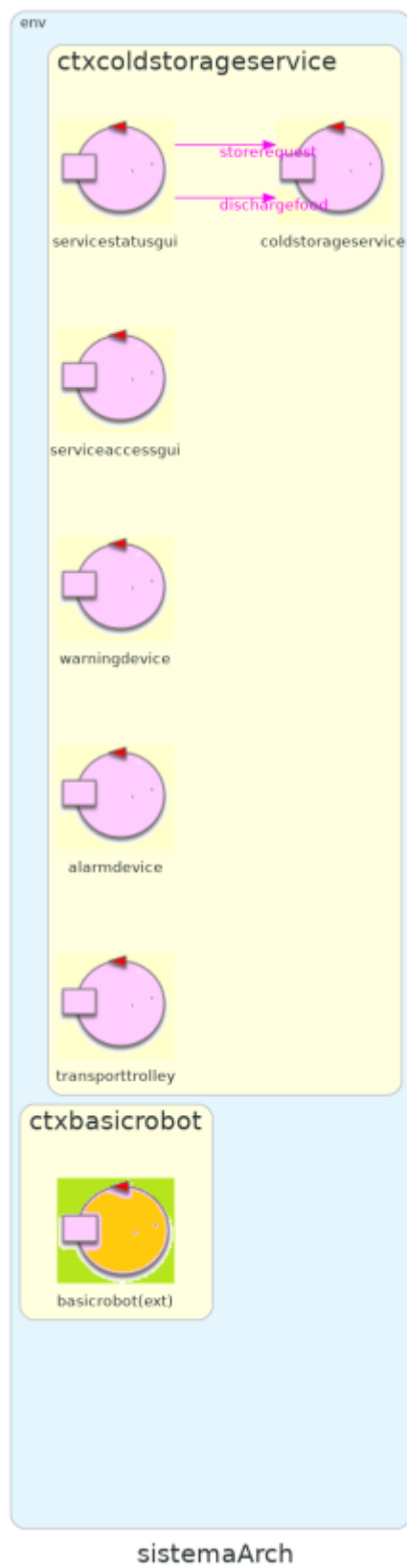
//Endosimbiosi di robotpos
Request moverobot  : moverobot(TARGETX, TARGETY)
Reply moverobotdone : moverobotok(ARG)
Reply moverobotfailed : moverobotfailed(PLANDONE, PLANTODO)
```

I Macro-Componenti del sistema sono:

- ColdStorageService
- TransportTrolley
- BasicRobot
- ServiceStatusGUI
- ServiceAccessGUI
- AlarmDevice
- WarningDevice

Link architettura logica: [#todo](#)

Modello non definito nei requisiti, non vincolante.



## Piano di Test

Scenario di Test 1: Richiesta con Cold Room Vuota

```
@Test
fun `test store request`() {
    //mandiamo la request
    val truckRequestStr = CommUtils.buildRequest("tester", "storerequest", "storerequest(10)",
"coldstorageservice").toString()
    println(truckRequestStr);
    val responseMessage = conn.request(truckRequestStr)
    println(responseMessage)

    assertTrue("TEST___ il ticket accettato ",
        responseMessage.contains("ticketAccepted"));
}
```

### Scenario di Test 2: Richiesta con Cold Room Piena

```
@Test
fun `test store request 1000`() {
    //mandiamo la request
    val truckRequestStr = CommUtils.buildRequest("tester", "storerequest", "storerequest(1000)",
"coldstorageservice").toString()
    println(truckRequestStr);
    val responseMessage = conn.request(truckRequestStr)
    println(responseMessage)

    assertTrue("TEST___ il ticket rifiutato",
        responseMessage.contains("ticketDenied"));
}
```

**\*\*Scenario di Test 3: Ticket accettato per richiesta di scarico**

```
@Test
fun `test discharge request no expired`() {
    //mandiamo la request
    val truckRequestStr = CommUtils.buildRequest("tester", "dischargefood", "dischargefood(NOEXPIRED)",
"coldstorageservice").toString()
    println(truckRequestStr);
    val responseMessage = conn.request(truckRequestStr)
    println(responseMessage)

    assertTrue("TEST___ charge taken",
        responseMessage.contains("replyChargeTaken"));
}
```

**\*\*Scenario di Test 4: Ticket rifiutato in quanto scaduto**

```
@Test
fun `test discharge request expired`() {
    //mandiamo la request
    val truckRequestStr = CommUtils.buildRequest("tester", "dischargefood", "dischargefood(EXPIRED)",
"coldstorageservice").toString()
    println(truckRequestStr);
    val responseMessage = conn.request(truckRequestStr)
    println(responseMessage)

    assertTrue("TEST___ ticket expired",
        responseMessage.contains("replyTicketExpired"));
}
```

## Piano di lavoro

Si è valutato di suddividere il sistema in 4 step di avanzamento

### Sprint 1

#### Obiettivo:

- Prototipo del **coldStorageService** (corebusiness del sistema)
- Prototipazione interazione con le GUI
- Dettaglio interazione con il **DDR**

- Architettura dettagliata
- TEMPO STIMATO:** 36h di lavoro/uomo

## Sprint 2

**Obiettivo:** **Service Access GUI**  
**TEMPO STIMATO:** 30h di lavoro/uomo

## Sprint 3

**Obiettivo:** **Service Status GUI**  
**TEMPO STIMATO:** 24h di lavoro/uomo

## Sprint 4

**Obiettivo :** Contesto raspberry  
**TEMPO STIMATO:** 12h di lavoro/uomo

## Sprint 5

**Obiettivo:** deployment sul robot fisico e distribuzione del sistema  
**TEMPO STIMATO:** 10h di lavoro/uomo

*Lo sprint 2, 3 e 4 possono essere realizzati in parallelo.*

## Componenti Gruppo

By Stefano Jin Cheng Hu, email: [stefanojin.hu@studio.unibo.it](mailto:stefanojin.hu@studio.unibo.it)

By Anna Vandì, email: [anna.vandi@studio.unibo.it](mailto:anna.vandi@studio.unibo.it)

By Alessandro Fiorini, email: [alessandro.fiorini7@studio.unibo.it](mailto:alessandro.fiorini7@studio.unibo.it)

Git Repo: <https://github.com/hjcSteve/coldStorageService>