

Sprint0V2

Introduzione

Il goal dello Sprint0

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.

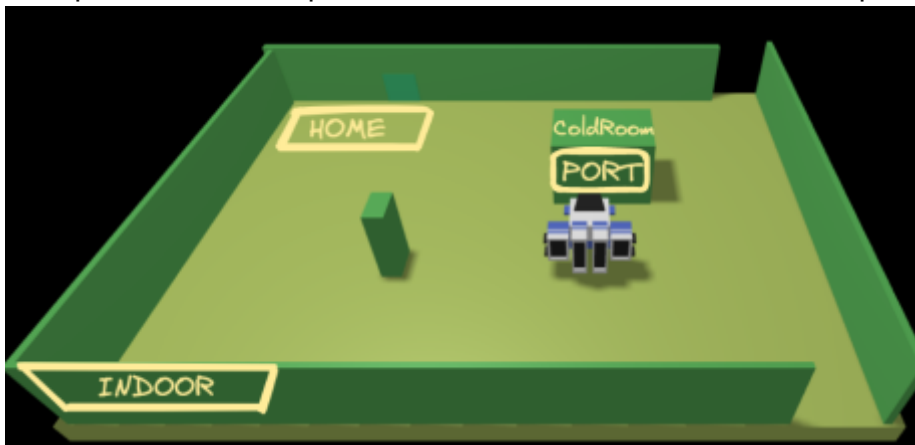
Requisiti

I requisiti sono descritti dal committente nel documento: LINK

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

Analisi dei requisiti

Il committente fornisce la documentazione dell'ambiente WEnv in [VirtualRobot23](#). Questo è rappresentato da una stanza a pianta rettangolare, delimitata da pareti non oltrepassabili e comprensiva di vari ostacoli fissi in varie posizioni.



Al suo interno è possibile pilotare un DDR robot di forma circolare (documentazione a [VirtualRobot23](#)) che risulta possibile muovere in accordo a due modalità :

- Modalità a **step**: il robot ricopre piccoli spazi di lunghezza configurabile (dato il parametro **steptime**) a ogni passo.
- Modalità a **sprint**: il robot procede finché non incontra un ostacolo, e si ferma.

In assenza meccanismi per la misurazione delle distanze nella service area e dopo una discussione con il committente, si considera come unità spaziale la dimensione fisica **RU** del **TRANSPORT TROLLEY** fornito dal committente, misurabile in step di durata fissata. La **Service area** pertanto può essere modellato come uno spazio rettangolare bidimensionale di dimensioni **SA_HxRU** e **SA_WxRU**. Ogni posizione nella scena è identificata da un insieme di coordinate (x,y). Per semplicità indichiamo come origine della **Service area** l'angolo in alto a sinistra, asse x e y corrispondenti rispettivamente alla parete superiore e di sinistra.

Definiamo le seguenti sottoaree come insieme di coordinate:

- **Home.area**: {(0,0)}
- **ColdRoom.area**: {(4,1),(5,1),(4,2),(5,2)}
- **Indoor.area**: {(0,4)}
- **Port.area**: {(4,3),(5,3)}

```
| H, 1, 1, 1, 1, 1, 1 | Y0
| 1, 1, 1, 1, C, C, 1 | Y1
| 1, 1, 1, 1, C, C, 1 | Y2
| 1, 1, X, 1, P, P, 1 | Y3
| I, 1, 1, 1, 1, 1, 1 | Y4
| -, -, -, -, -, -, - | Y5
X0 X1 X2 X3 X4 X5 X6
```

Da requisiti ColdRoom è inoltre caratterizzato da una quantità massima di carico e di conseguenza anche una quantità attuale.

Modelliamo ColdRoom con questi due campi :

- **ColdRoom.maxStorage** come un numero intero positivo con un valore all'inizializzazione del sistema
- **ColdRoom.currentStorage** come un numero intero inizialmente con valore 0, in ogni momento \leq **ColdRoom.maxStorage**

TRANSPORT TROLLEY : entità logica capace di spostarsi nella **Service area**. Fornisce le interfacce logiche al sistema per pilotare un DDR robot, è attiva e nella nostra architettura figura pertanto come un **attore**.

DDR ROBOT: entità attiva che implementa le azioni logiche del transport trolley Il committente ha fornito un software che dispone un interfaccia **BasicRobot** per modellare il DDR-ROBOT

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

L'interazione avviene per mezzo di scambio di messaggi con questi formati su un'architettura potenzialmente distribuita e pertanto lo indichiamo come un **attore** su contesto **External**.

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)
```

ColdStorageService è l'entità che racchiude il core business dell'intero sistema. Dovendo interagire con componenti in un sistema distribuito è necessario che sia modellato come **attore**

ColdRoom potrebbe essere modellato come un **POJO** all'interno del ColdStorageService, ma si è scelto di modellarlo come **attore** per i seguenti motivi:

- ColdStorageService avrebbe troppe responsabilità e quindi deleghiamo la responsabilità in un componente attore separato (principio di singola responsabilità)
- Separiamo la logica di dati dalla logica di business

ServiceAccessGUI è un'entità responsabile di interagire con l'utente umano e di inviare messaggi con il ColdStorageService. Dovendo inviare messaggi ad altri componenti modelliamo ServiceAccessGUI come un **attore**

ServiceStatusGUI è un'entità responsabile di interagire con l'utente umano e di inviare messaggi con il ColdStorageService e la ColdRoom. Analogamente al ServiceAccessGUI deve essere modellato come **attore**

I requisiti introducono inoltre due componenti attive, vale a dire un **Sonar** e un **Led**, destinati alla distribuzione su un nodo fisico potenzialmente indipendente dal resto del sistema. Questi seguono il comportamento generale di un **alarm device** il primo e **warning device** il secondo.

Un alarm device è caratterizzato da un numero reale positivo **DLIMIT** e dalla variabile **CurrentDistance**, misurata a cadenza regolare. (Se questa risulta minore di **DLIMIT**, sarà necessario notificare l'**evento** al transport trolley per l'arresto...analisi del problema)

Un warning device associa la posizione del transport trolley ad un determinato comportamento per la notifica (blink nel caso di un Led).

Entrambi i device sono per il momento descritti nell'ottica di un comportamento **locale**, sul medesimo nodo logico e fisico, destinando ai futuri sprint eventuali distribuzioni su architettura distribuita. I dettagli implementativi verranno sviluppati solamente dallo sprint 2.

Macro componenti

I Macro-Componenti del sistema sono dunque:

- ColdStorageService
- ColdRoom
- TransportTrolley
- BasicRobot
- ServiceStatusGUI
- ServiceAccessGUI
- AlarmDevice

- WarningDevice

Architettura logica

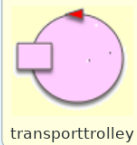
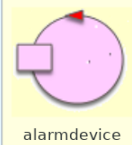
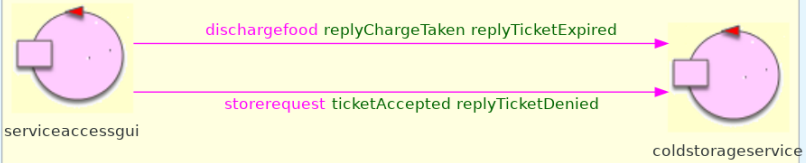
Messaggi:

```
Request storerequest : storerequest(FW)
Reply ticket_accepted :ticket_accepted(TICKETNUMBER)
Reply replyTicketDenied: ticketDenied(ARG)
```

```
Request dischargefood : dischargefood(TICKETNUM)
Reply replyChargeTaken : replyChargeTaken(ARG)
Reply replyTicketExpired: replyTicketExpired(ARG)
```

env

ctxcoldstorageservice



ctxbasicrobot



Piano di Test

Link:

<https://github.com/hjcSteve/coldStorageService/blob/main/sprint0/sprint0V1/test/it/unibo/ctxcoldstorageservice/MainCtxcoldstorageserviceKtTest.kt>

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()
        val responseMessage = conn.request(truckRequestStr)

        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 5 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

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

Componenti Gruppo

By Stefano Jin Cheng Hu, email: stefanojin.hu@studio.unibo.it

By Anna Vandì, email: anna.vandi@studio.unibo.it

By Alessandro Fiorini, email: alessandro.fiorini7@studio.unibo.it

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