SPRINT1

Introduzione:

I requisiti del committente esposti nello Sprint precedente, necessitano nel presente Sprint1 di ulteriori chiarimenti. Successivamente, si procederà con l'analisi degli stessi definendo il **core business** funzionante del sistema. Con questo si intende un sistema di trasporto coerente con le specifiche che adotti le librerie e programmi disponibili, un meccanismo per la generazione del ticket e una gestione efficace della ColdRoom. Contestualmente si provvede alla creazione di un debito piano di test.

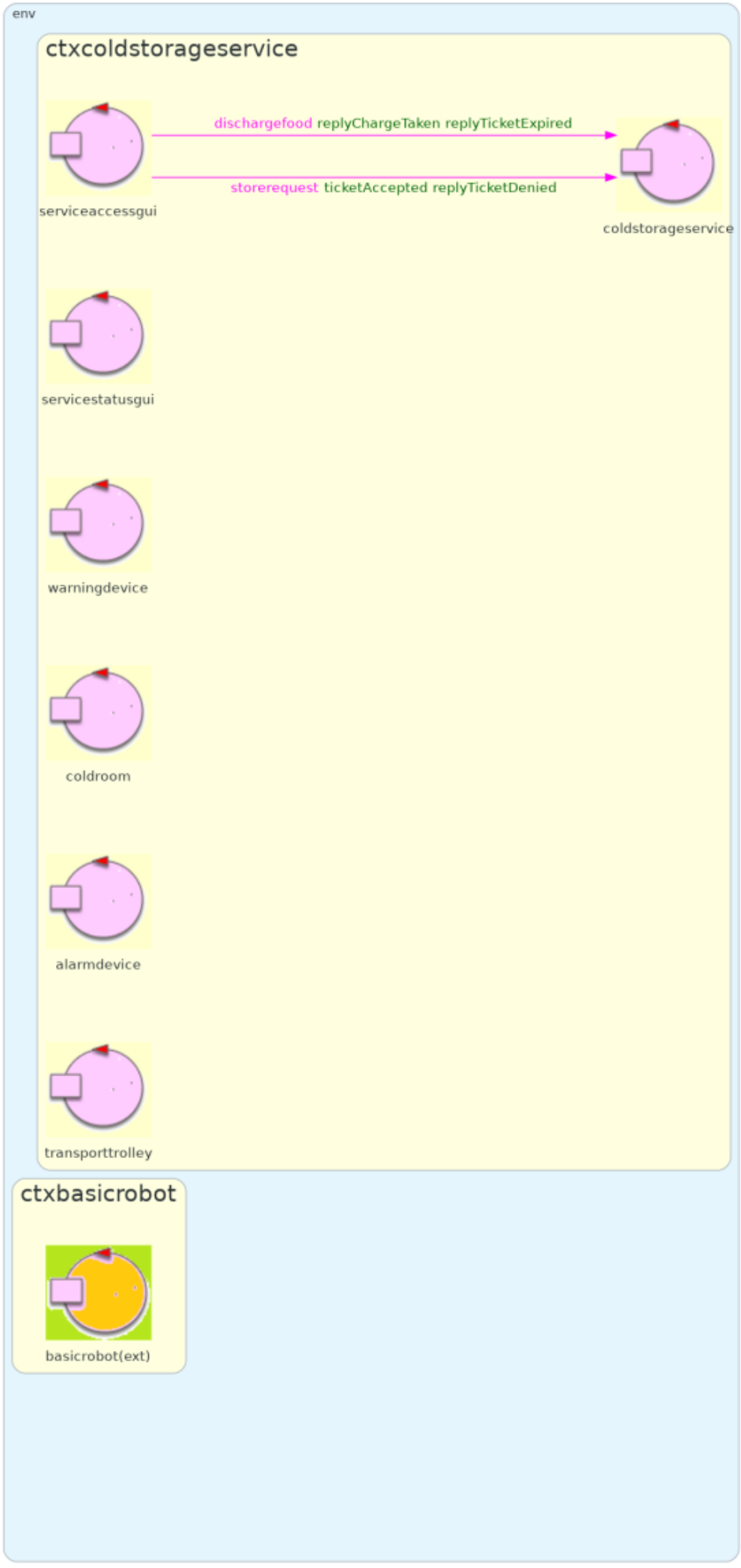
Chiarimenti del committente:

In seguito ai recenti confronti col committente, si sono delineati dei nuovi requisiti:

- Un truck alla volta può occupare l'Indoor, e dunque essere servito dal Transport Trolley.
- La Cold Room è unica, così come unica è la risorsa concorrente Transport Trolley.
- Non è richiesto di gestire l'eventualità che il Transport Trolley non sia in grado di gestire l'intero carico di un Truck in un solo tragitto.
- Si ipotizza la coerenza tra il carico dichiarato per il rilascio del ticket e quello effettivamente depositato all'Indoor.

Analisi dei requisiti nello sprint precedente:

Al termine dello Sprint0, l'analisi dei requisiti ha portato a definire le componenti più idonee per il caso d'uso richiesto. Ne riproponiamo lo schematico:



sistemaArch

Di queste, ai fini dello sprint1, verranno definite le interazioni tra una GUI prototipale, il ColdStorage, il Transport Trolley e la ColdRoom, già proposti come **attori**.

Analisi del problema:

CtxColdStorageService

Per quanto riguarda le GUI, in una prima fase si sono descritte con degli attori sullo stesso nodo fisico ma, come già esposto, è importante che siano in grado di operare su architetture distribuite. Le escludiamo pertanto dal contesto che ora mira ad esprimere, in via il più generale possibile, il nodo fisico corrispondente all'esecuzione del solo core business. Escludiamo per il momento anche gli alarm device, anch'essi espressamente destinati all'esecuzione su un Raspberry Pi (assunto come nodo diverso da quello d'esecuzione del core business). Il contesto di nostro interesse è il **CtxColdStorageService** e comprende gli attori:

- ColdStorage
- Transport Trolley
- ColdRoom

Alla creazione del contesto, viene generato un MessageHandler col compito di gestire e smistare ad ogni membro i destinati ad esso destinati. Si necessita di una porta da esporre per la ricezione dei messaggi, abbiamo scelto la **8055**.

Responsabilità e user story della ColdStorage:

L'attore ColdStorage è cruciale e riassume in sè molteplici responsabilità:

- Deve essere raggiungibile dalle due GUI per gestirne le richieste e tenere nota di eventuali richieste sospese in attesa di risposta (**Queued**).
- Deve gestire la logica dei ticket, inglobando al suo interno una base di conoscenza persistente, che corrisponde ai ticket emessi in precedenza. In questa sede, i ticket vengono semplicisticamente descritti da numeri interi.
- Deve tenere nota dello stato della service area, inteso come posizione corrente del trolley e stato del trolley (**Busy** / **Idle**).

Il principio di singola responsabilità non risulta dunque in questo stadio rispettato. Si decide di posticipare alla fase di progettazione l'eventuale risoluzione del problema. Il comportamento della ColdStorage è schematicamente riassunto dal seguente diagramma degli stati, dettagliato in sede di analisi ed espresso dal codice sprint1Robot/src/analisi.qak:

```
QActor coldstorageservice context ctxcoldstorageservice{
[#
var Trolley_is_working : Boolean = false;
var KgtoLoad : Int = 0;
var Expiration : Long = 10000;
var servingTicket : Int = 0;
var QUEUED_TICKET : Int = 0;
var TICKETNUM : Int = 0;
#]
//INITIALIZE THE COLD STORAGE SERVICE
State s0 initial {
        println("$name ) waiting for a new message...") color black
        discardMsg Off
} Transition t0
whenRequest storerequest->handleStoreRequest
```

```
whenRequest dischargefood->handleDischargeRequest
whenRequest discharged_trolley ->handleTrolley_atColdroom
whenMsg trolley_isindoor -> clearIndoor
whenMsg chargeTaken-> handleChargeTaken

//ASK COLD ROOM HOW MANY KG ARE ALREADY INSIDE
State handleChargeTaken{
        onMsg(chargeTaken : chargeTaken(TICKETID)){
                println("il TT ha preso il carico, dico al Truck che puo andare via ")
color     black
                replyTo dischargefood with replyChargeTaken : replyChargeTaken(ARG)
        }
}Goto s0

//ASK COLD ROOM HOW MANY KG ARE ALREADY INSIDE
State handleStoreRequest{
        println("$name ) handle store request") color black
        onMsg(storerequest:storerequest(KG)){
        [#
        KgtoLoad = payloadArg(0).toInt();
        #]
        println("$name ) asking to coldRoom") color black
        request coldroom -m spaceCheck : spaceCheck($KgtoLoad)
        }
} Transition t0
whenReply space_reserved -> handleTicketGeneration
whenReply space_insufficient -> refuseStoreReq

State refuseStoreReq{
        replyTo storerequest with replyTicketDenied : ticketDenied(D)
        println("$name ) ticket denied, not enough space.")
}Goto s0

//TICKET GENERATION AND CURRENT LOAD CONTROL
State handleTicketGeneration{
        println("$name ) handle ticket gen") color black
        onMsg( space_reserved : space_reserved(D) ){
        [#
        var TICKETCODE = "TICKET_NEW";
        var TIMESTAMP = "timestamp";
        #]

        replyTo storerequest with ticketAccepted :
ticketAccepted($TICKETCODE,$TIMESTAMP)
        println("$name ) ticket accepted! Ticket $TICKETCODE emitted. ") color
magenta
        }
} Goto s0

//DISCHARGE AND TICKET CHECKING
State handleDischargeRequest{
        println("$name ) handle discarge req") color black
        onMsg( dischargefood : dischargefood(TICKETNUM) ){
                [#
                TICKETNUM = payloadArg(0).toInt();
                var Expired : Boolean = (if (TICKETNUM == 0 ) true else false);
                #]
                if [# !Expired && !Trolley_is_working#]{
```

```
                println("$name ) Sending food to the cold room, lazzaro alzati e
cammina") color magenta
                [#
                Trolley_is_working=true;
                servingTicket = TICKETNUM;
                #]
                forward transporttrolley -m dischargeTrolley :
dischargeTrolley($TICKETNUM)
                }
                else {
                if [# !Expired #]{
                println("$name ) Truck is already serving another truck, let's queue the
ticket $TICKETNUM") color magenta
                [#
                QUEUED_TICKET=TICKETNUM;
                #]
                println("$name ) ticket in coda $QUEUED_TICKET") color magenta
                }
                else { println("$name ) The ticket has expired... sending notification
to SAGui") color magenta
                replyTo dischargefood with replyTicketExpired: replyTicketExpired(ARG)
                }
                }
        }
} Goto s0

State handleTrolley_atColdroom{
println("$name ) handle at coldroom") color black
onMsg( discharged_trolley : discharged_trolley(TICKETNUM) ){
        println("il tt ha finito di scaricare ho ticket in coda $QUEUED_TICKET")
color magenta
        [#
        TICKETNUM = payloadArg(0).toInt();
        val KG : Int = 10;
        #]
        if [# QUEUED_TICKET !== 0 #]{
                [#
                servingTicket = QUEUED_TICKET;
                #]
                replyTo discharged_trolley with serve_newtruck :
serve_newtruck($QUEUED_TICKET)
                println("servi un altro ticket con id $QUEUED_TICKET") color magenta
                [#
                QUEUED_TICKET=0;
                #]
        }
        else {
                println("non ho un altro ticket da darti, torna a casa ") color magenta
                [#
                Trolley_is_working=false;
                #]
                replyTo discharged_trolley with idle_trolley : idle_trolley(D)
                }
                forward coldroom -m stored_food : stored_food($KG)
        }
}Goto s0

State clearIndoor{
```

```
        println("$name ) handle clear indoor") color black
        onMsg( trolley_isindoor : trolley_isindoor(D) ){
        println("il TT ha preso il carico, dico al Truck che puo andare via ") color
black
        replyTo dischargefood with replyChargeTaken : replyChargeTaken($TICKETNUM)
        }
} Goto s0
}
```

Responsabilità e user story del Transport Trolley

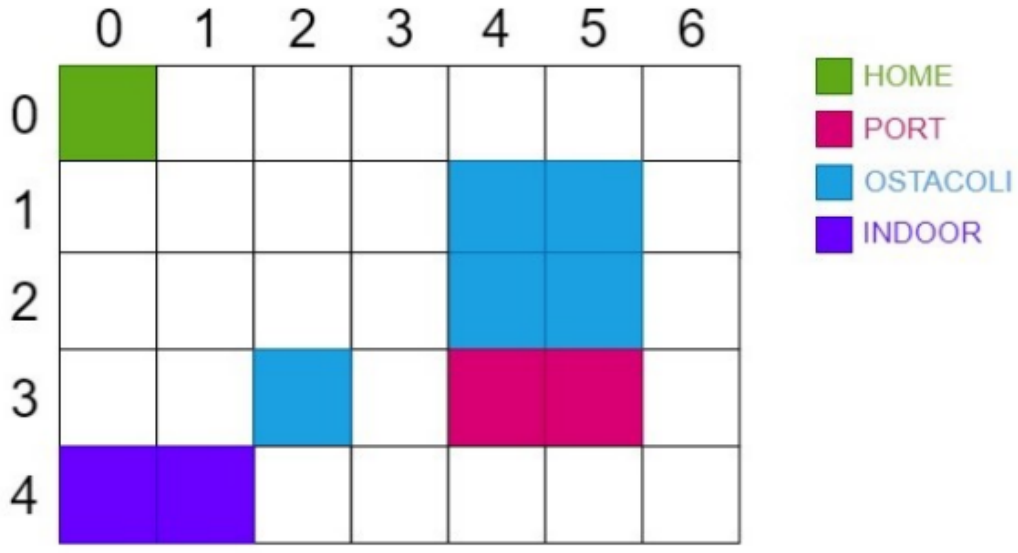
L'attore Transport Trolley è l'unità preposta unicamente ad un'efficace gestione del robot. Si interfaccia con il BasicRobot (libreria fornita dal committente) e quindi con l'ambiente virtuale della Service Area, per muovere e servire **una richiesta di scarico alla volta.** Come minuziosamente descritto nell'Analisi dei Requisiti dello Sprint0, è importante che in ordine vengano eseguite le operazioni:

- Il robot entra in **Busy** mode. Movimento da Home a Indoor all'arrivo di un Truck all'Indoor (supponendo che il Trolley fosse nello stato di Idle, quindi in Home).
- Carico dell'intero ammontare di Kg depositati dal truck, in un tempo arbitrario.
- Movimento dall'Indoor alla ColdRoom.
- Scarico del transport trolley in un tempo arbitrario. In seguito, è necessario che il robot si consulti con la ColdStorage per notificare lo scarico avvenuto e richiedere se ha altre richieste **Queued** da servire.
- In caso affermativo, il robot deve tornare all'Indoor, ripartendo poi dal punto 2. Alternativamente, il robot tornerà allo stato di Idle in Home.

Il Transport Trolley ha lo scopo di:

- Comunicare col Basic Robot, una volta concordate le coordinate delle varie destinazioni al fine di far muovere fisicamente il robot.
- Eseguire un flusso di stati coerente con la user story.
- Avvisare il trucker una volta che il carico è stato prelevato dall'indoor.
- Avvisare il coldstorageservice che il deposito è stato effettuato, per poi aggiornare il conteggio dei chili della coldroom.

Per quanto riguarda le coordinate da rispettare negli spostamenti del robot, coerentemente a quanto detto nell'Analisi dei Requisiti, il basic robot fornisce la seguente documentazione:



Per necessità di chiarezza, identifichiamo d'ora in avanti le seguenti con le coordinate:

- **Home: {0,0}**
- **Indoor: {0,4}**
- **Coldroom: {4,3}

Si prevede inoltre da requisito che si possa gestire l'eventualità di un imprevisto avvenuto in qualsiasi momento che provoca un'interruzione immediata del transport trolley. Questa logica deve necessariamente introdurre uno stato persistente **Error** e uno di **Recovery** per proseguire l'esecuzione inconclusa dello stato precedente l'arrivo dell'alarm, all'arrivo di un comando di tipo "**Resume**". In maniera esplicita, ecco di seguito il codice scaturito dall'analisi del problema:

```
QActor transporttrolley context ctxcoldstorageservice{
[# var lastState: String = ""
        var ticketID:Int= 0
#]
State s0 initial {
        println("$name | init e engage basicrobot") color magenta
        request basicrobot -m engage :
engage(transporttrolley,125)
}
Transition      t0      whenReply engagedone -> engaged
                                whenReply engagerefused -> quit

State engaged {
        println("$name | basicrobot engaged") color magenta
}Goto atHome

State atHome{
        [# lastState = "atHome" #]

        println("$name | basicrobot at Home") color magenta
        forward basicrobot -m setdirection : dir(down)


}
Transition t0 whenMsg dischargeTrolley -> goingIndoor

State goingIndoor {
        [# lastState = "goingIndoor" #]
        println("$name | vado all'INDOOR") color magenta
        request basicrobot -m moverobot : moverobot (0,4)
}
Transition t0 whenReply moverobotdone -> atIndoor
whenEvent alarm -> stopped

State atIndoor {
        [# lastState = "atIndoor" #]

        println("$name | sono in INDOOR") color magenta
        println("$name | carico il cibo") color magenta
}
Transition t    whenTime 3000 -> loadDone               // simula azione di carico

State loadDone {
        forward coldstorageservice -m chargeTaken : chargetaken(CIAO)
}
Goto goingColdroom
```

```
State goingColdroom {
        [# lastState = "goingColdroom" #]
        println("$name | vado verso la cold room") color magenta
        request basicrobot -m moverobot : moverobot (4,3)
}
Transition t0 whenReply moverobotdone -> atColdroom
whenEvent alarm -> stopped

State atColdroom {
        [# lastState = "atColdroom" #]
        println("$name | sono in Cold Room") color magenta
}
Transition t0 whenTime 3000     -> chargeStored // simula azione deposito

State chargeStored {
        [# lastState = "chargedStored" #]
        println("$name | terminato deposito. Aspetto istruzioni") color magenta
        request coldstorageservice -m discharged_trolley:
discharged_trolley(TICKETID)
}
Transition t0 whenReply idle_trolley -> goingIndoor
                                whenReply serve_newtruck -> goingHome

State goingHome{
        [# lastState = "goingHome" #]
        println("$name | vado alla posizione HOME") color magenta
        request basicrobot -m moverobot : moverobot (0,0)
}
Transition t0 whenReply moverobotdone -> atHome
        whenEvent alarm -> stopped



State stopped {
        discardMsg On
        println("$name | Sono fermo per ostacolo sonar") color magenta
}
Transition t0 whenEvent resume and [# lastState == "atHome" #]          -> atHome
                          whenEvent resume and [# lastState == "goingIndoor" #]         ->
goingIndoor
                          whenEvent resume and [# lastState == "atIndoor" #]    ->
atIndoor
                          whenEvent resume and [# lastState == "goingColdroom" #]       ->
goingColdroom
                          whenEvent resume and [# lastState == "atColdroom" #]  ->
atColdroom
                          whenEvent resume and [# lastState == "chargeStored" #]        ->
chargeStored
                          whenEvent resume and [# lastState == "goingHome" #]           ->
goingHome

State quit {
        forward basicrobot -m disengage :
disengage(transporttrolley)
        [# System.exit(0) #]
}
}
```

Responsabilità e user story della ColdRoom

L' attore ColdRoom ha un'unica responsabilità: deve in ogni momento mantenere aggiornata la conoscenza dello stato del frigorifero. Questa viene esaustivamente espressa da una serie di quantità numeriche:

- Max_storage: Intero massimale della capienza.
- Current_storage: Intero compreso tra 0 e Max_storage, **aggiornato solo a scarico avvenuto**.
- Reserved_storage: Intero che esprime il quantitativo di Kg riservati mediante rilascio di ticket, ma **non ancora depositati**.

La logica di gestione degli spazi in ogni momento dell'esecuzione risiede nella seguente:

$$Maxstorage \geq Currentstorage + Reservedstorage$$

```
QActor coldroom context ctxcoldstorageservice{

[#
val MAX_STG : Int =Configuration.conf.MAX_STG;
var current_STG : Int =0;
var reserved_STG : Int =0;
#]
State s0 initial {

        println("$name) I am started with a Maximum $MAX_STG, currently
$current_STG!")
}Goto waiting_state

State waiting_state {
        println("$name) Okay then, the storage is $current_STG kg, with a reserved
$reserved_STG kg")
} Transition t2
whenRequest spaceCheck  -> check_state
whenMsg stored_food -> store_state


State check_state{
        onMsg( spaceCheck : spaceCheck(KG)){
                [#
                var Kg : Int = payloadArg(0).toInt();
                #]
                if[# MAX_STG >= current_STG + reserved_STG + Kg#]{
                        [#
                        reserved_STG+=Kg;

                        #]
                        replyTo spaceCheck with space_reserved : space_reserved($Kg)

                }else{
                        println("$name) not enough space for reservation...")

                        replyTo spaceCheck with space_insufficient : space_insufficient(D)
                }
        }
} Goto waiting_state

State store_state{
        onMsg( stored_food : stored_food(KG)){
```

```
                [#
                var Kg : Int = payloadArg(0).toInt();
                reserved_STG -= Kg;
                current_STG += Kg;
                #]
                println(" $name) Performed the load with success!")
        }
}Goto waiting_state
}
```

Definizione dei messaggi

Messaggi tra il Coldstorage e il ServiceAccessGUI

Quando il truck driver fa una richiesta di store con un certo peso

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

Quando il truck driver arriva e chiede di scaricare immettendo il TICKETNUMBER

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

Messaggi tra il Coldstorage e il Transport Trolley

Il ColdStorage manda trolley a servire un truck (da idle a serving) cok messaggio dischargeTrolley esplicitando il ticket number della prenotazione. Quando il robot è all'Indoor, il Transport trolley risponde con ChargeTaken, liberando l'indoor per l'arrivo di un nuovo Truck.

```
Dispatch dischargeTrolley : dischargeTrolley(TICKETID)
Dispatch chargeTaken : chargeTaken(TICKETID)
```

Quando il trolley è alla ColdRoom, dopo aver scaricato il peso deve comunicare al ColdStorage l'adempimento di tale operazione e chiede se ci fosse un nuovo ticket da servire oppure può tornare a Home. Il ColdStorage avrà la responsabilità di comunicare con il ColdRoom l'aggiornamento del peso e rispondere alla richiesta del Transport trolley la disponibilità di uno ticket da gestire.

```
Request discharged_trolley : discharged_trolley(TICKETID)
Reply idle_trolley : idle_trolley(D) for discharged_trolley
Reply serve_newtruck : serve_newtruck(D) for discharged_trolley
```

Come anticipato, si prevedono due eventi:

```
//Eventi di stop e resume per il transport trolley
Event alarm:alarm(X)
Event resume:resume(ARG)
```

Alla ricezione del primo, il robot si deve **bloccare** se si trova in uno stato di movimento (GoingIndoor, GoingColdroom, GoingHome), e riprende a muoversi solo se si riceve il l'evento resume.

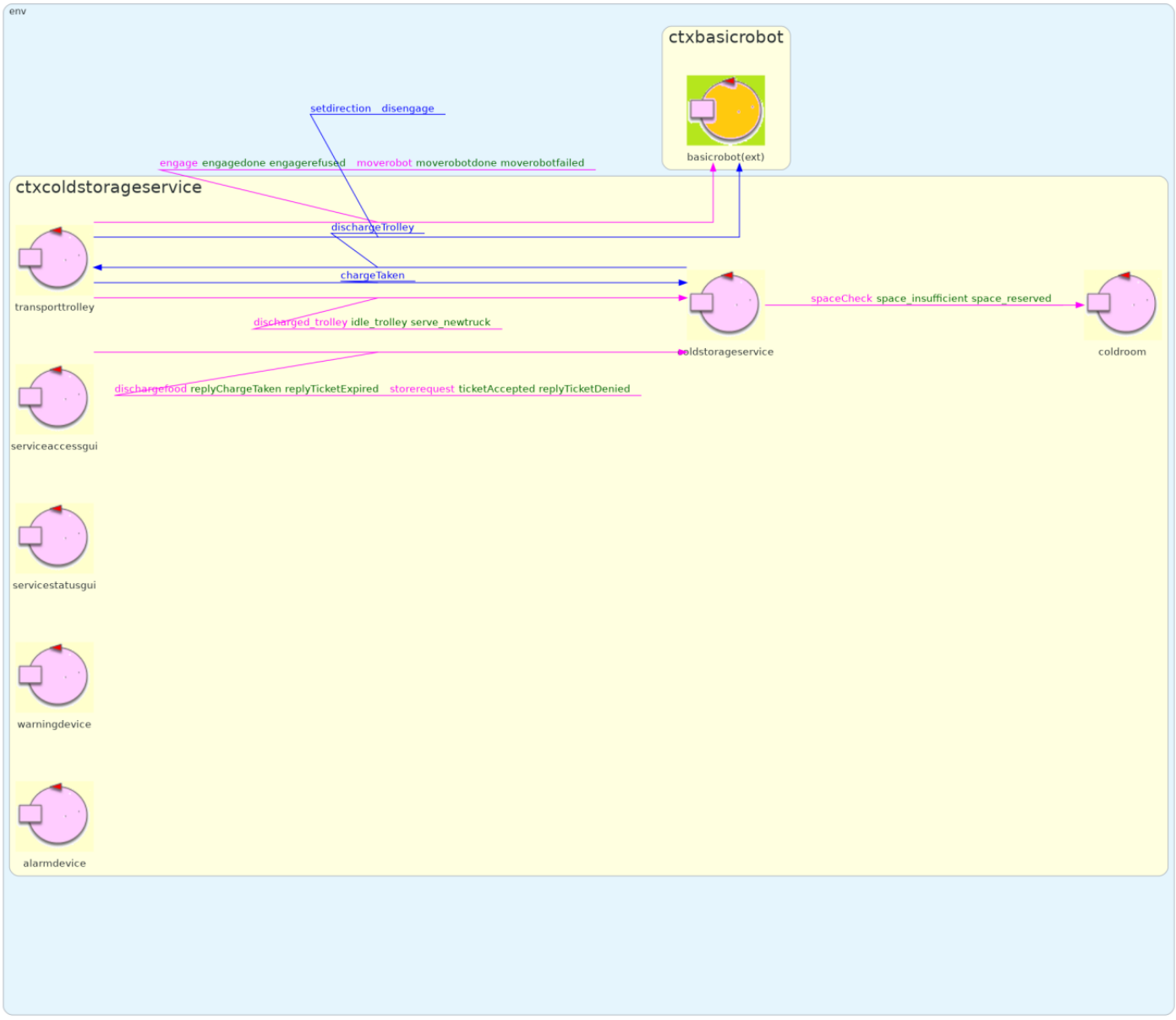
Messaggi tra il ColdStorage e il ColdRoom

Tutte le operazioni che interessano il ColdRoom avvengono tutte attraverso l'intermediario del ColdStorageService. Durante lo stato handleTicketGeneration si avrà bisogno di verificare lo stato interno del ColdRoom e da tale esite generare o no il ticket.

```
Request spaceCheck : spaceCheck(KG)
Reply space_insufficient : space_insufficient(D) for spaceCheck
Reply space_reserved : space_reserved(D) for spaceCheck

Dispatch stored_food : stored_food(KG)
```

Architettura Logica:



coldstorageserviceArch

Piano di test:

Ai fini del test del comportamento del **CtxColdStorageService**, ci avvaliamo delle funzionalità di JUnit per la realizzazione di una **MockGui.**

Questa verifica il corretto funzionamento a fronte di vari scenari di interesse:

Scenario di Test 1: Riscossione di due ticket sequenzialmente, con accodamento del secondo per Trolley Busy

```
@Test
fun ` test discarge request with new ticket in queue`(){
    //mandiamo la request

    val truckRequestStr = CommUtils.buildRequest("tester", "dischargefood",
"dischargefood(1)", "coldstorageservice").toString()
    println(truckRequestStr);

    val responseMessage = conn.request(truckRequestStr)
    println(responseMessage)
    assertTrue("TEST___ charge taken",
            responseMessage.contains("replyChargeTaken"));
```

```
    val truckRequestStr2 = CommUtils.buildRequest("tester", "dischargefood",
"dischargefood(2)", "coldstorageservice").toString()
    println(truckRequestStr2);

    val responseMessage2 = conn.request(truckRequestStr2)
    assertTrue("TEST___ charge taken",
            responseMessage2.contains("replyChargeTaken"));
}
```

Scenario di Test 2: Richiesta di scarico con inoltro di un evento Alarm tra l'Indoor e Coldroom

```
@Test
fun ` test discarge request with new ticket in queue con alarm`(){
    //mandiamo la request

    val truckRequestStr = CommUtils.buildRequest("tester", "dischargefood",
"dischargefood(3)", "coldstorageservice").toString()
    println(truckRequestStr);

    val responseMessage = conn.request(truckRequestStr)
    assertTrue("TEST___ charge taken",
            responseMessage.contains("replyChargeTaken"));

    val alarm = CommUtils.buildEvent("tester", "alarm", "alarm(X)").toString();
    conn.forward(alarm)

    Thread.sleep(2000)

    val resume = CommUtils.buildEvent("tester", "resume",
resume(ARG)").toString();

    conn.forward(resume)
    println(responseMessage)

}
```

Progettazione:

In sede di progettazione, si sono sviluppati diversi elementi volti a garantire una migliore adattabilità del sistema.

File di configurazione

Si è deciso, per ottenere maggiore estensibilità,di fare leggere i parametri del sistema (ad esempio `MAX_KG`) da un file di configurazione `AppConf.json` all'avvio del sistema.

```
object Configuration{

    var conf : Conf = Conf()
    init {
        val filePath = "AppConf.json"
        val file = File(filePath)
        val jsonString = file.readText()
        val res = Gson().fromJson(jsonString, Conf::class.java)
        conf = res;
    }
}
```

```
data class Conf(val MAX_STG: Int = 100)
```

Progettazione Coldstorage:

Per gestire al meglio l'entità **TICKET** si è scelto di creare una **classe Java** specifica per la sua gestione e integrata nel modello qak come risorsa esterna.

```
public class TicketList {
    private List<Ticket> tickets;
    private int lastNumber;
    private long expirationTime;

    public TicketList(long expirationTime) {
        tickets = new ArrayList<>();
        //fetchTicktesformFIle
        lastNumber = 0;
        this.expirationTime = expirationTime;
    }
    public synchronized Ticket createTicket(int kgToStore) {
        Ticket ticket = new Ticket();
        ticket.setKgToStore(kgToStore);
        lastNumber++;
        ticket.setTicketNumber(lastNumber);
        ticket.setStatus(0);
        //ticket.setTicketSecret(generateSecret(7));
        ticket.setTimestamp(Instant.now().toEpochMilli());
        tickets.add(ticket);
        return ticket;
    }

    public synchronized Ticket getTicket(int ticketNumber) {
                                ...
    }
    public synchronized void  removeTicket(int ticketNumber) {
                                ...
    }
    public synchronized int getTotalKgToStore() {
                                ...
    }
    public synchronized boolean isExpired(Ticket ticket) {
                                ...
    }
    public String toString(){
                                ...
    }
}
```

La suddetta classe, estende il concetto di **Ticket** e **lista di Ticket**, come un POJO rappresentato da una lista Java. I metodi definiti permettono una gestione efficiente e scalabile di una lista potenzialmente enorme, e soprattutto permettono operazioni di controllo sulla validità del ticket. Ciascun ticket può infatti essere Expired o Non-expired, in virtù di un parametro di costruzione di nome **Expiration** rappresentato da un long integer, che

definisce la durata in millisecondi di ciascun biglietto. Questo, avvalorando l'utilità del File di configurazione, può essere modicato a piacimento mediante `AppConf.json`.

Per quanto riguarda le problematiche legate al principio di singola responsabilità sollevate in precedenza, si è deciso in fase di progettazione che il disaccoppiamento delle funzioni di smistamento dei messaggi, di gestione dei ticket e intermediazione tra Transport trolley e Coldroom mediante introduzione di due nuovi attori porterebbe ad una non necessaria complicazione dell'architettura logica. Distribuire le informazioni sui ticket, sullo stato del robot e sui messaggi in arrivo dai trucker necessiterebbe di nuovi messaggi specializzati per realizzare microservizi per le query di dati, con un traffico eccessivo. Si ritiene pertanto più adeguato procedere con l'architettura già formulata, accentrando in ColdStorageService il core business logico e l'accesso ai dati.

Le modifiche al codice presentato in analisi sono le seguenti:

```
var Expiration : Long = Configuration.conf.Expirationtime;
var List = tickets.TicketList(Expiration);
var servingTicket = tickets.Ticket();
var queuedTicket = tickets.Ticket();
```

Per accedere agli attributi della classe ColdStorageService.kt.
I metodi forniti dalle classi Ticket e TicketList vengono invece richiamati in lettura in questa maniera:

```
var ticket=List.createTicket(KgtoLoad);
var TICKETCODE = ticket.getTicketNumber();
var TIMESTAMP = ticket.getTimestamp();
```

Oppure in scrittura per modificare lo stato (0 valid, 1 reclaimed, 2 expired).

```
ticket.setStatus(1)
```

Progettazione Transport Trolley:

Si prevede di introdurre la possibilità di configurazione delle coordinate all'interno della scena direttamente dal file AppConf.json, eventualità che al momento, si è deciso di non realizzare e pertanto il Transport Trolley risulta del tutto inalterato.

Progettazione Cold Room:

Essenzialmente, la sola modifica apportata riguarda l'estrazione del parametro MAX_STG dal file di configurazione mediante la riga:

```
val MAX_STG : Int =Configuration.conf.MAX_STG;
```

Osservabilità degli stati interni

Dopo un'attenta analisi del problema, è emersa la necessità di validare il corretto funzionamento del sistema, inclusa l'esposizione degli stati interni degli attori coinvolti, con particolare attenzione agli stati interni del transport trolley.

Al fine di massimizzare l'espressività del meta-linguaggio QAK, si è optato per l'implementazione dell'osservabilità degli attori tramite il protocollo MQTT.

Pertanto, nell'ambito della fase di progettazione, è stato inizialmente implementato un broker MQTT, configurato come un container Docker utilizzando eclipse-mosquitto, eseguito sulla stessa macchina del sistema, sulla porta

1833.

Per avviare il broker MQTT, è stato eseguito il seguente comando:
`sudo docker run -p 1833:1833 eclipse-mosquitto`
Successivamente, ciascun attore interessato a pubblicare il proprio stato tramite il broker, dovrà farlo mediante un insieme definito di topic di interesse.

```
Event coldstorage_state:coldstorage_state(X)
Event trolley_state:trolley_state(X)
```

```
        State atIndoor {
                [# LASTSTATE = "atIndoor" #]
                publish "trolley_state" -m trolley_state:trolley_state($LASTSTATE)

        }Transition t   whenTime 3000 -> loadDone
```

Posso certamente assisterti nella creazione di un piano di test che sfrutti MQTT in combinazione con JUnit. Possiamo definire una serie di testcase che verifichino il corretto funzionamento delle sistema tramite MQTT utilizzando JUnit come framework di testing.

```
@Test
fun ` discarge request-ticket 1 OBSERV mqtt`() {
    try {
        println("connession mqtt:" + mqttConn.connect(mqSender, mqttBrokerAddr))
        println("set call back mqtt:" + mqttConn.setCallback(myMqttCallback))
        println("set subscribe to " + topic + " :" + mqttConn.subscribe(topic))
    } catch (e: Exception) {
        println(e)
    }
    //mandiamo la request per avere il ticket
    val ticketRequestStr = CommUtils.buildRequest("tester", "storerequest",
"storerequest(35)", "coldstorageservice").toString()
    val ticketresponseMessage = conn.request(ticketRequestStr)
    assertTrue("TEST___ il ticket accettato ",
            ticketresponseMessage.contains("ticketAccepted"));
    //mando una richiesta di discarge food
    val truckRequestStr = CommUtils.buildRequest("tester", "dischargefood",
"dischargefood(1)", "coldstorageservice").toString()
    val responseMessage = conn.request(truckRequestStr)
    assertTrue("TEST___ charge taken",
            responseMessage.contains("replyChargeTaken"));
    //verifico gli stati osservati
    val messagesMQTT = myMqttCallback.getMessagesMQTT();
    assertTrue("TEST___ transport_state",
            messagesMQTT[messagesMQTT.size-1] == "trolley_state(goingColdroom)"
                    && messagesMQTT[messagesMQTT.size-2] == "trolley_state(atIndoor)"
    )
}
```

Deployment

1. Avviare il container nella sottodirectory unibo.basicrobot23 mediante il comando

```
sudo docker-compose -f basicrobot23.yaml up
```

Questo creerà:

- l’ambiente virtuale con il robot in localhost, alla porta 8090
- Il basicrobot23 sempre in localhost sulla porta 8020

2. Avvio il broker mqtt nella stessa macchina locale del contesto coldstorage
   `sudo docker run -p 1833:1833 eclipse-mosquitto`
3. Nella directory sprint1Robot lanciare da terminale il comando `./gradlew run` per avviare il contesto **CtxColdStorageService.** Questo espone **la porta 8055** per eventuali connessioni.
4. In intellij avviare il test MainCtxcoldstorageserviceKtTest.kt per inviare alla suddetta porta i messaggi di interesse.