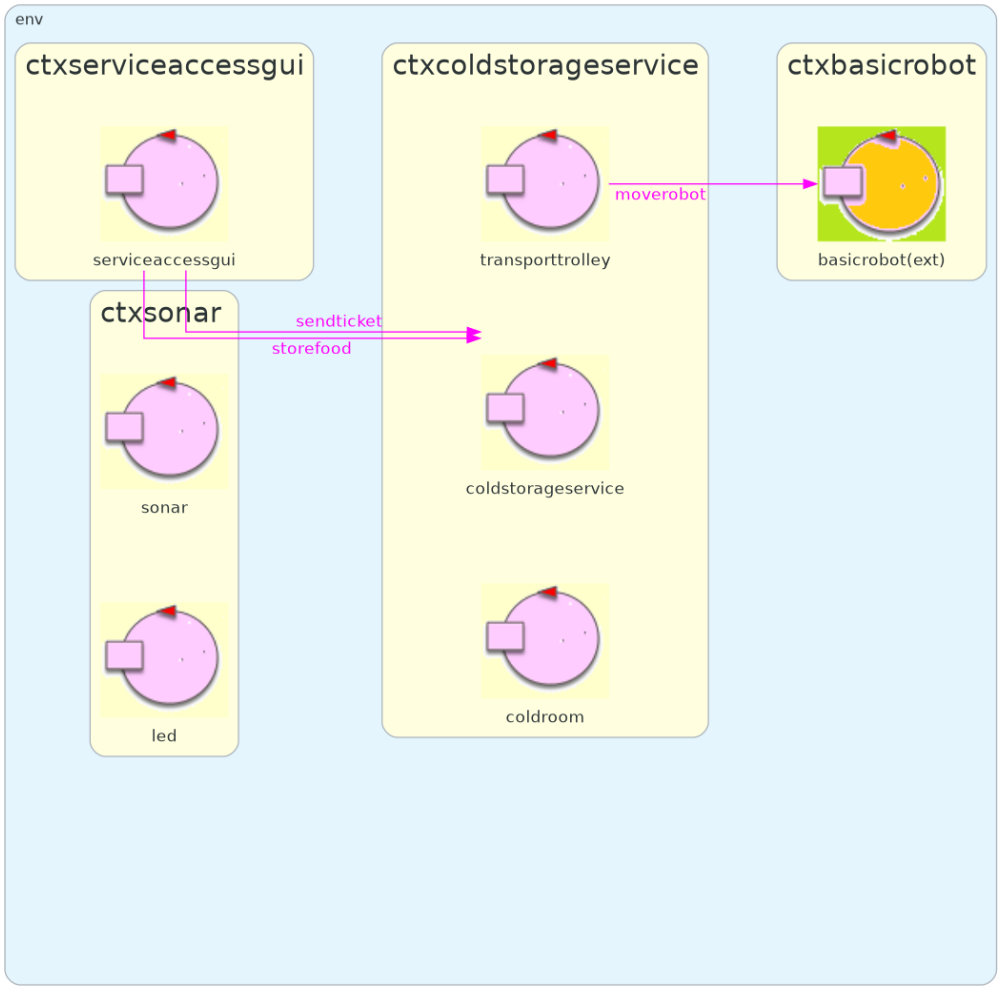
ColdStorageSprint1v3

Introduzione

Goal conseguiti nello sprint0

- individuare un architettura logica iniziale che definisca le macro-entità del sistema e le loro interazioni
- definire un piano di lavoro iniziale

architettura logica sprint0



coldstorageserviceArch

Goal dello sprint1

- prototipazione del core business ColdStorageService + TransportTrolley+ ColdRoom

Requirements

The transport trolley is used to perform a deposit action that consists in the following phases:

- pick up a food-load from a Fridge truck located on the INDOOR
- go from the INDOOR to the PORT of the ColdRoom

- deposit the food-load in the ColdRoom

SERVICE USER STORY

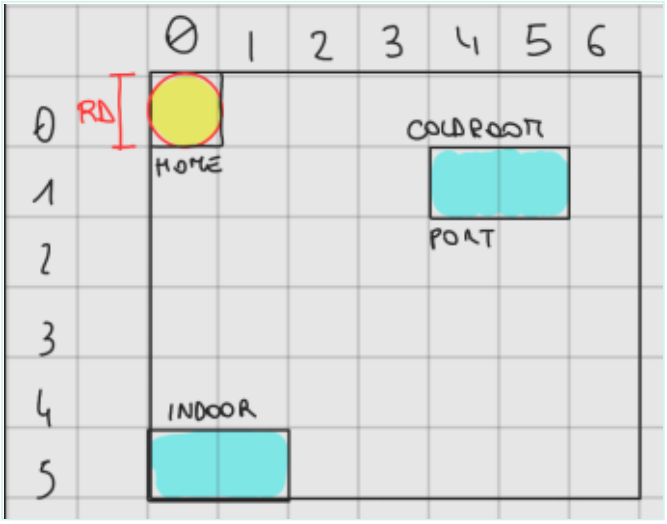
- A Fridge truck driver uses the ServiceAccessGUI to send a request to store its load of FW kg. If the request is accepted, the driver drives its truck to the INDOOR of the service, before the ticket expiration time TICKETTIME.
- When the truck is at the INDOOR of the service, the driver uses the ServiceAccessGUI to enter the ticket number and waits until the message charge taken (sent by the ColdStorageService) appears on the ServiceAccessGUI. At this point, the truck should leave the INDOOR.
- When the service accepts a ticket, the transport trolley reaches the INDOOR, picks up the food, sends the charge taken message and then goes to the ColdRoom to store the food.
- When the deposit action is terminated, the transport trolley accepts another ticket (if any) or returns to HOME.

Requirements analysis

Modello della service area

- dai requisiti si evince che la stanza sia modellabile tramite una mappa che rappresenta una suddivisione in celle. La dimensione di ogni cella è legata alla dimensione del transport trolley. La mappa quindi è modellata come una griglia di quadrati di lato RD, dato che nei requisiti è specificaato che "The transport trolley has the form of a square of side length **RD**"
- Il robot viene considerato un oggetto inscrivibile in un cerchio di raggio RD numero reale positivo
- Il concetto di INDOOR e di PORT vengono modellate come posizioni nella mappa, ovvero come coppie di coordinate. In particolare INDOOR è formalizzata con la cella di coordinate la coordinata **(5,0)** oppure **(5,1),** mentre PORT è formalizzata con la cella di coordinate **(2,4)**

oppure **(2,5)**



Transport Trolley

deposit action: termine con cui si descrive il seguente ciclo di operazioni del sistema, formalizzate dai messaggi presenti in questa documentazione: BasicRobot23.html

Ticket

struttura dati formulata secondo la seguente struttura

```
int TICKETTIME INTEGER
int TICKETNUMBER INTEGER
```

Cold Storage Service

- ColdStorageService è l'entità core business del sistema e siccome il sistema è distribuito, allora ColdStorageService è modellata come un attore

Dopo opportuni colloqui con il committente, possiano affermare che :

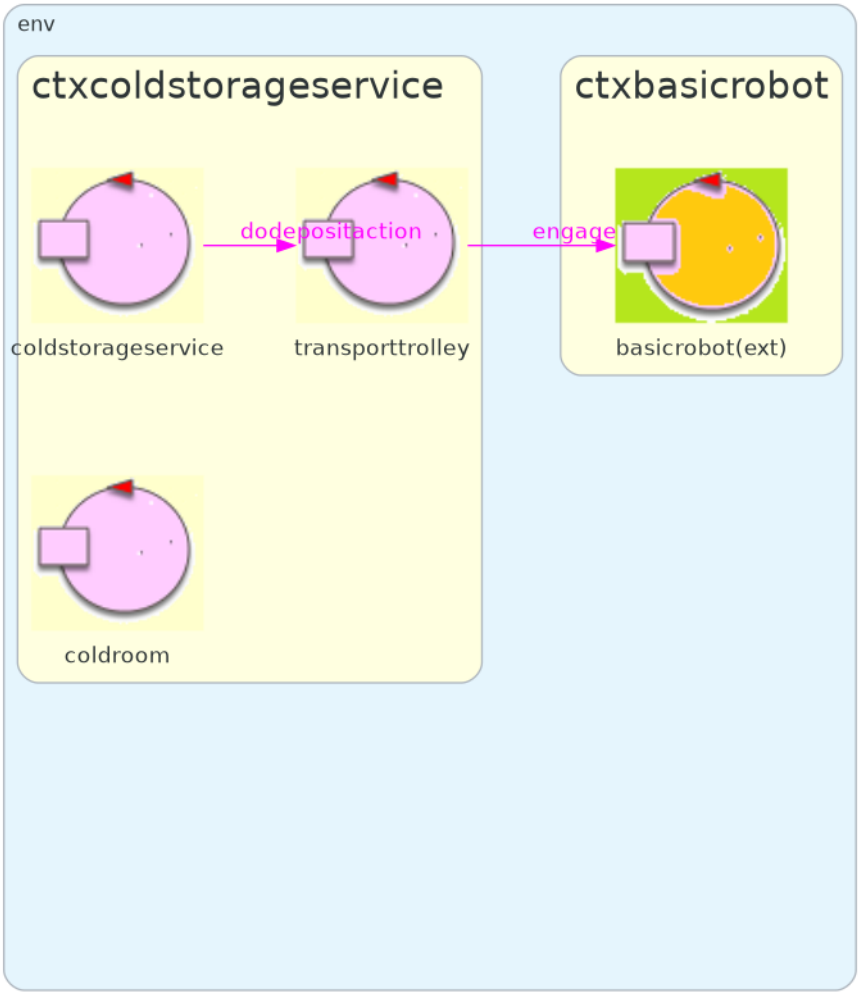
- Non deve succedere che un camion, ricevuto il proprio ticket si veda rifiutata l'operazione di scarico una volta arrivato in INDOOR.
- la richiesta di un ticket può avvenire mentre sono ancora in corso operazioni di scarico precedenti

Dopo opportuni colloqui con il committente, possiano affermare che :

- le operazioni di carico e di scarico della ColdRoom potrebbero essere effettuate in parallelo oppure in maniera sequenziale. Per semplicità di realizzazione, dato che il committente non

ha espresso riflessioni in materia, vengono effettuate in maniera sequenziale, ma nel caso realistico esse verrebbero fatte in parallelo.

Architettura logica requisiti



coldstorageserviceArch

Problem Analysis

Cold Room

- Cold Room viene modellata come un attore di modo da interagire con ColdStorageService.
- KEYPOINT: essa potrebbe essere modellata come un POJO all'interno di ColdStorageService stesso, ma riteniamo che modellando Cold Room come un attore porti ai seguenti vantaggi:
    - si alleggerisce ColdStorageService di un altro compito, dato che ColdStorageService è l'entità core business
    - si segue il principio di singola responsabilità
    - se verranno previste in futuro una entità di "Scarico della Cold Room", allora questa entità potrà interagire direttamente con la Cold Room, senza dover interagire con ColdStorageService

Il messaggio chargeTaken

dai requisiti si evince che il messaggio **chargetaken** deve essere inviato da coldstorageservice nel momento in cui il transporttrolley ha effettuato lo scarico della merce, questo presume che esista un qualche tipo di comunicazione fra i due nel momento in cui questo avviene
possibili soluzioni:

- **transporttrolley emette un evento** che, viene catturato da coldstorageservice, che invia il messaggio chargetaken, in seguito viene comunicato al transporttrolley di spostarsi verso la coldroom
- **transporttrolley emette un Dispatch** verso coldstorageservice, che, di conseguenza invia il messaggio chargetaken, in seguito viene comunicato al transporttrolley di spostarsi verso la coldroom
  Si decide di optare per emettere un evento e per limitare l'occupazione di banda si decide di sfruttare la proprietà del supporto qak di emettere eventi locali.

Carico del transport trolley

il transporttrolley potrebbe ammettere un carico massimo trasportabile, dopo opportune discussioni con il committente si è deciso di non prendere in carico questa eventualità

Il problema delle operazioni parallele

dato che i ticket possono essere erogati in maniera parallela è possibile che il sistema eroghi un ticket che non possa essere accettato in fase di scarico merce dato che la dimensione disponibile nella **COLDROOM** è stata ridotta da una operazione di scarico precedente
per ovviare a questo problema si predispone una lista che memorizza i ticket emessi che sono ancora in corso e ne tiene conto alla ricezione di `storefood`

Sicurezza del Ticket

Il ticket ha al suo interno un "ticket number", e per ragioni di sicurezza due o più Fridge truck driver non devono poter conoscere i ticket degli altri, e non devono poter inserire i ticket degli altri né in maniera malevola né incidentale.
per ovviare al problema si aggiunge alla struttura dati `TICKET` si prevede un campo generato randomicamente

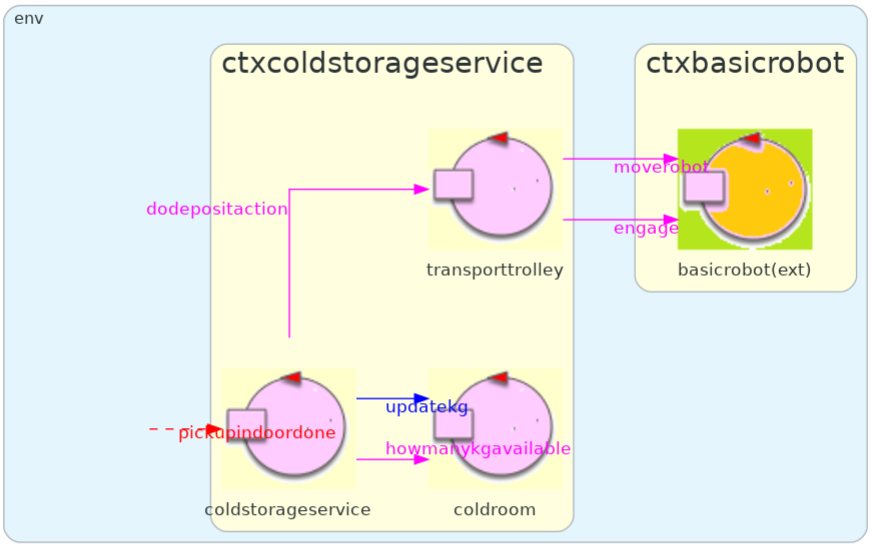
```
String TICKETSECRET
```

Architettura logica dopo analisi del problema

Il sistema è composto da:

- ***ColdStorageService***: prende in carico richieste di generazione ticket e richieste di invio di un camion; si interfaccia con la ColdRoom per saperne lo stato; fa partire le deposit action;
- ***Transport Trolley***: invia al Basic Robot la sequenza di comandi necessari per effettuare una deposit action
- ***Basic Robot***: esegue i comandi del Transport Trolley
- ***ServiceAccessGui***: si interfaccia con ColdStorageService per la richiesa di ticket
- ***Cold Room***: aggiorna lo stato della quantità di kg
- ***Scarico***: entità esterna che effettua una operazione di scarico della Cold Room, diminuendo i kg presenti in essa

per evitare di



coldstorageserviceArch

Test plan

si prevede di testare le seguenti funzionalità del sistema

- i movimenti del transport trolley all'interno della mappa secondo i requisiti
- la correttezza dello stato della **COLDROOM** durante le fasi di richiesta del ticket e di scarico merce

Test effettuati in Junit

creare un attore che funge da fake user e nel codice Kotlin inseriamo delle Assert

- fake user manda il messaggio **storefood**, mi aspetto di ricevere il ticket
- fake user inserisce il ticket e si aspetta un dispatch di chargetaken
- dopo un chargetaken, controllare la cold room che abbia diminuito lo storage

```
@Test
public void testMultipleLoad() throws InterruptedException {
```

```
    int numberOfThreads = 10;
    ...

    for (i = 0; i < numberOfThreads; i++) {
       int finalI = i;
       service.submit(() -> {
             String truckRequestStr = CommUtils.buildRequest("tester", "storefood",
"storefood(50)", "coldstorageservice").toString();
             IApplMessage reply = connTcp[finalI].request(new ApplMessage(truckRequestStr));
             assertTrue(reply.msgContent().contains("ticketaccepted") ||
reply.msgContent().contains("ticketdenied"));

             if(reply.msgContent().contains("ticketaccepted")){
                String ticket = reply.msgContent().split(",")[0].split("[\\\\(]")[1];
                String secret = reply.msgContent().split(",")[1];
                assertTrue(!ticket.isEmpty() || !ticket.isBlank());
                assertTrue(!secret.isEmpty() || !secret.isBlank());
             }
          latch.countDown();
       });
    }
    latch.await();
}
```

Il test prevede di creare molteplici "fake user", ognuno dei quali crea una connessione con ColdStorageService e invia una richiesta.

realizzazione mediante eventi

per implementare i test si prevede di sfruttare la generazione degli eventi da parte del transport trolley e della cooldroom in modo da essere il meno invasivi possibile sul sistema.

PROGETTAZIONE

Tutto il codice della parte di progettazione è consultabile al seguente link github

Il messaggio chargetaken

si prevede di implementare la comunicazione del messaggio chargetaken tramite un evento per le seguenti motivazioni

- rendere più facile il testing dell'applicazione tramite la cattura degli eventi generati
- rendere il transporttrolley in grado di svolgere la **deposit action** in maniera indipendente questa soluzione ci consente di separare in maniera netta le competenze di coldstorageservice transporttrolley

La gestione delle richieste parallele

da requisiti sappiamo che la accettazione di un ticket e le operazioni di carico e scarico merce devono essere eseguite in parallelo, per consentire ciò si prevede, all'interno di coldstorageservice una struttura dati come segue:

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

public TicketList(long expirationTime) {
tickets = new ArrayList<>();
lastNumber = 0;
this.expirationTime = expirationTime;
}
```

la struttura prevede una lista di oggetti POJO Ticket che rappresentano i ticket attivi nel sistema, alla creazione di un ticket coldstorageservice verifica che la quantità di merce da scaricare sia inferiore alla capacita residua meno il totale della somma delle quantita dei ticket ancora attivi nel sistema

di seguito l'implementazione dell'entità Ticket:

```
public class Ticket{
private int ticketNumber;
private String ticketSecret;
private long timestamp;
private int kgToStore;
}
```

Implementazioni di COLDROOM e COLDSTORAGESERVICE

si prevede una struttura del attore coldstorageservice come segue:
l'attore resta in uno stato di attesa in cui attende uno dei seguenti messaggi:

- `Request storefood : storefood( KG )`
- `Request sendticket: sendticket(TICKETCODE, TICKETSECRET)`
- `Event pickupindoordone:pickupindoordone(ARG)`
- `Event depositactionended:depositactionended(ARG)`
  ognuno di questi messaggi innesca un diverso servizio fornito dall'attore, che lo riporta in questo stato in attesa di altri messaggi.

L'implementazione dell'attore COLDROOM segue lo stesso principio con i seguenti messaggi

- `Request howmanykgavailable : howmanykgavailable(ARG)`
- `Dispatch updatekg : updatekg(KG)`

Implementazione TRANSPORTTROLLEY

il transport trolley viene concepito come entità in grado di eseguire le sue responsabilità in maniera indipentente, al termine degli spostamenti genera degli eventi per notificare il suo stato a coldstorageservice

Il problema della perdita di contesto

una problematica sorta in fase di progettazione, è stata fare in modo che alla cattura di un evento `depositactionended` , generato da transporttrolley nel momento in cui avviene lo scarico della merce presso la coldroom, coldstorageservice fosse in grado di risalire all'identità del ticket che la ha generata in modo da aggiornare la lista di ticket attivi, per fare ciò viene fornito al trolley l'identificativo del ticket come payload del messaggio `dodepositaction`

I ticket scaduti

durante lo sviluppo è sorta la problematica di gestire i ticket scaduti nel momento in cui viene effettuata la richiesta di un nuovo ticket, ovvero venivano esclusi dal calcolo dello spazio disponibile tutti i ticket che non superavano il check temporale della scadenza, anche se questi erano già stati reinseriti nel sistema ed era in corso l'operazione di trasporto presso la coldroom.
per ovviare a questo si è previsto di aggiungere al ticket il sequente campo:

```
//legend

//0=emitted but not reinserted

//1=emitted and reinserted

private int status;
```

il ticket può quindi trovarsi in due stati distinti

- **caso 0** il ticket è stato emesso da parte del sistema ma non è stato ancora reinserito da parte del truckdriver, il biglietto è quind ancorai soggetto al controllo temporale della scadenza
- **caso 1** il ticket è stato emesso da parte del sistema ed è stato reinserito da parte del truckdriver, il biglietto non è quindi soggetto al controllo temporale della scadenza

l'attore GUIMOK

si prevede di aggiungere all'architettura logica predisposta in analisi del problema un attore **GUIMOK** per simulare il comportamento di un utente che interagisce con la main logic di sistema

Lettura parametri di configurazione da file

Si è deciso di impostare i parametri del sistema tramite un file JSON.
A run-time, l'applicazione legge il file JSON e inizializza i parametri di conseguenza:

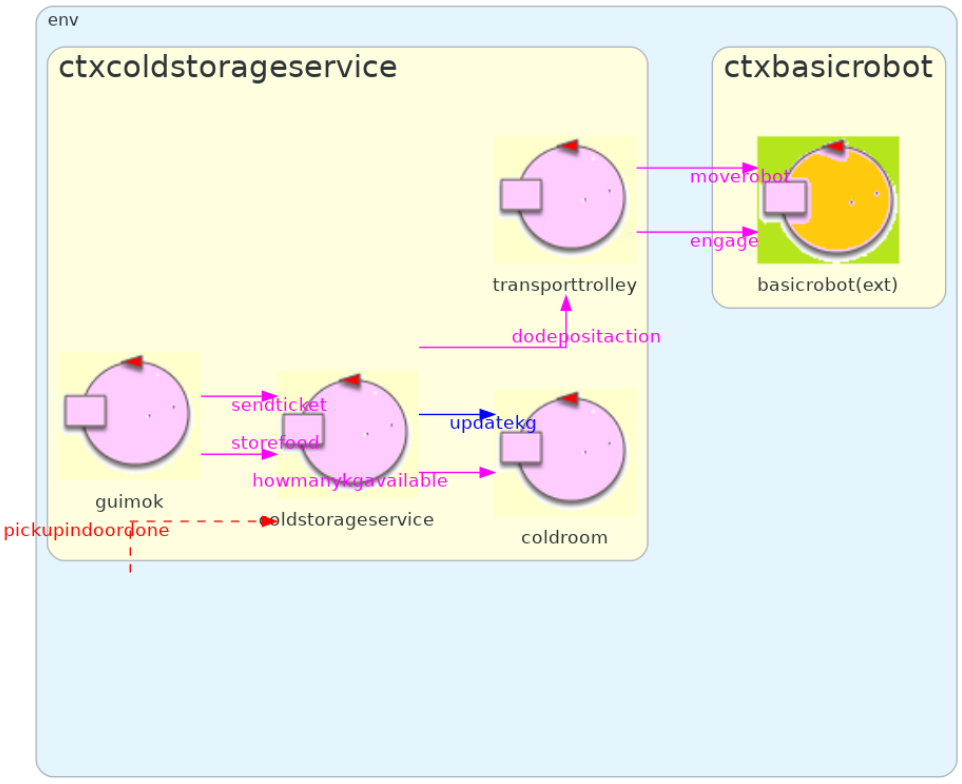
```
object DomainSystemConfig {

   private var Expirationtime : Long = 0;
    private var Maxweight : Long = 0;

    init {
            val config = File("AppConfig.json").readText(Charsets.UTF_8)
            val jsonObject   =  JSONObject( config );

            Expirationtime= jsonObject.getLong("Expirationtime")
            Maxweight= jsonObject.getLong("Maxweight")
        }
}
```

Architettura finale progettazione



coldstorageserviceArch

By Caterina Leonelli email: caterina.leonelli2@studio.unibo.it, GIT repo: https://github.com/RootLeo00/sw-eng.git

By Matteo Longhi email: matteo.longhi5@studio.unibo.it
GIT repo:
https://github.com/carnivuth/iss_2023_matteo_longhi.git