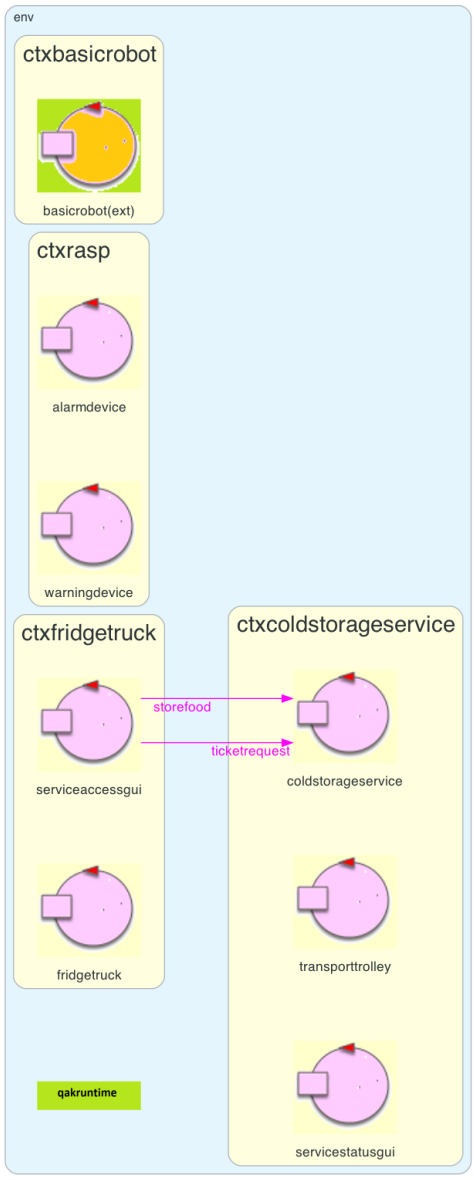
ColdStorageSprint1V1

Introduction

Lo sprint0 ha prodotto la seguente architettura:



coldstorageserviceArch

Requirements

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

1. pick up a food-load from a Fridge truck located on the INDOOR
2. go from the INDOOR to the PORT of the ColdRoom
3. deposit the food-load in the ColdRoom

The story of the ColdStorageService can be summarized as follows:

1. A Fridge truck driver uses the ServiceAcessGUI 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 exipration time TICKETTIME.
2. When the truck is at the INDOOR of the service, the driver uses the ServiceAcessGUI to enter the ticket number and waits until the message charge taken (sent by the ColdStorageService) appears on the ServiceAcessGUI. At this point, the truck should leave the INDOOR.
3. 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.
4. When the deposit action is terminated, the transport trolley accepts another ticket (if any) or returns to HOME.

Goal di questo sprint:

- sviluppo del core business del sistema: coldstorageservice, transporttrolley, coldroom
- testing dei componenti sviluppati

Requirement analysis

KEY POINT

Dai requisiti possiamo evincere che:

- Le richieste di emissione ticket devono poter essere elaborate in parallelo alle azioni di carico e scarico del TransportTrolley

Analisi del Problema

Problema load-time lungo

Quando il fridge truck invia il ticket come messaggio di showticket non riceve alcuna risposta fino a quando il robot completa l'operazione di carico. In questo modo il truck è in grado di capire se il suo ticket è valido solamente quando riceve un messaggio di chargeTaken, quindi nel caso in cui il suo ticket sia stato rifiutato non riceverebbe nessun messaggio e continuerebbe a rimanere in attesa. Quindi per affrontare questo problema è stato pensato di impostare un'interazione a DUE-FASI tra driver e ColdStorageService:

- **FASE 1**: il driver invia il ticket e attenda una risposta (immediata) come ad esempio ticketaccepted/ticketrejected
- **FASE 2**: il driver invia la richiesta loaddone e attenda la risposta (chargeTaken o fallimento per cause legate al servizio)

ColdRoom

È stato deciso di rappresentarla mediante due variabili e non come attore poichè non dovrà ricevere o inviare alcun tipo di messaggio. Le due variabili rappresenteranno il peso effettivo contenuto all'interno della ColdRoom (peso reale) e il peso effettivo della ColdRoom sommato al peso totale dato dai ticket emessi ma non ancora elaborati (peso virtuale).

Svuotamento ColdRoom

Per gestire lo svuotamento della ColdRoom abbiamo modellato un attore apposito (EmpytColdRoom) che ad intervalli di tempo regolari invia un richiesta al coldstorageservice. Abbiamo deciso di modellarlo come attore per rappresentare un operatore che si occupa dello svuotamento.

Struttura Ticket

Il Ticket viene modellato come un POJO e contiene al suo interno un identificativo univoco e casuale che per ragioni di sicurezza deve essere conosciuto solamente dal Fridge Truck driver che ha richiesto il Ticket.

TicketService

Viene modellato come attore per permettere all'applicazione di essere in grado di rispondere sempre alle richieste di nuovi ticket, per quindi consentire un'elaborazione in parallelo delle operazioni di emissione ticket e azioni di carico e scarico del TransportTrolley. Per fare ciò si è scelto di delegare a tale attore i messaggi di richiesta di un nuovo ticket

in ingresso. Inoltre si occupa di verificare la validità di un ticket presentato da un fridge truck, e verifica che il peso virtuale non ecceda il limite imposto dalla capacità della ColdRoom. Alla richiesta di un nuovo Ticket verifica che tutti i Ticket scaduti ma non ancora utilizzati siano eliminati e quindi esclusi dal calcolo del peso virtuale della ColdRoom.

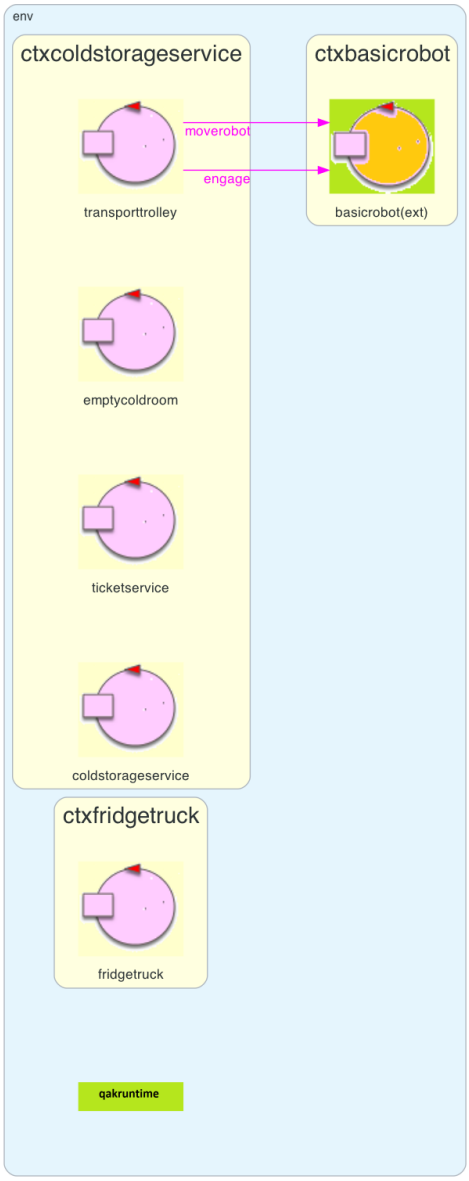
Driver distratto

Questo problema ha indotto il committente ad affermare che: quando un agente esterno (driver) invia il ticket per indurre il servizio a scaricare il truck, si SUPPPONE GARANTITO che il carico del truck sia UGUALE (o al più MINORE) del carico indicato nella prenotazione. Al momento della presentazione del ticket viene fornito dal truck driver il peso effettivo della consegna. Questo valore viene confrontato con il peso dichiarato nella fase di richiesta del ticket. Se tale valore è inferiore al peso dichiarato, il peso virtuale viene decrementato della differenza dei due valori.

Problema del driver malevolo

Questo problema ha indotto il committente ad affermare che si fa l'ipotesi che nessun driver cerchi di imbrogliare (rubando ticket, etc.)

Architettura logica



coldstorageserviceArch

Dall'architettua logica possiamo evidenziare i seguenti componenti:

- ColdStorageService: riceve dal fridge truck le richieste di verifica di validità dei ticket che vengono successivamente inoltrate al TicketService, riceve le richieste di svuotamento dall'EmptyColdRoom e invia i messaggi al TransportTrolley per iniziare la deposit action
- TransportTrolley: invia al BasicRobot i messaggi necessari per eseguire la deposit action
- TicketService: si occupa di gestire le richieste per l'emissione di nuovi ticket e di verificarne la validità nel momento in cui viene presentato dal fridge truck
- EmptyColdRoom: si occupa di svuotare periodicamente la coldroom inviato un apposito messaggio al ColdStorageService
- FridgeTruck: invia le richieste per l'emissione di un nuovo ticket e viene introdotto solamente per simulare un eventuale utente finale
- BasicRobot: esegue i comandi ricevuti dal TransportTrolley

Progettazione

Struttura ticket

L'entità ticket è stata modellata nel seguente modo:

```
data class Ticket(val id: String, val creationTime: Long, val fw: Int) {

    companion object{
        @JvmStatic
        fun getRandomId() : String {
            val length = 5
            val allowedChars = ('A'..'Z') + ('a'..'z')
            return (1..length)
                .map { allowedChars.random() }
                .joinToString("")
        }
    }
}
```

I parametri per la creazione di un Ticket sono i seguenti:

- ***Id***: consiste in una stringa, contenente caratteri alfabetici, generata casualmente che permette di identificare univocamente il ticket.
- ***creationTime***: equivale all'istante di tempo in cui viene creato il ticket. Questo campo verrà utilizzato per verificarne la validità
- ***fw***: riguarda l'informazione relativa al peso del carico, in modo da poter gestire il problema del driver distratto che verrà affrontato in seguito.

La funzione getRandomId() è una funzione statica che permette di generare casualmente l'id del ticket.

Problema load-time lungo

Tale problema, descritto in precedenza, viene risolto con le due seguenti request-response:

```
Request ticketrequest : ticketrequest( TICKET, FW )
Reply ticketaccepted  : ticketaccepted( MESSAGE )
Reply ticketrejected  : ticketrejected( REASON )

Request loaddone        : loaddone(FW)
Reply chargetaken       : chargetaken(MESSAGE)
Reply chargefailure : chargefailure(REASON)
```

- ticketrequest è la richiesta con cui si presenta il ticket per poter successivamente eseguire l'operazione di scarico. Ticket rappresenta l'id fornito in precedenza in risposta alla richiesta di newticket (tale messaggio verrà descritto più avanti nel documento); FW rappresenta il peso effettivo consegnato dal fridge truck.
- loaddone viene invato dal fridge truck appena si riceve la risposta ticketaccepted. Con questa richiesta l'utente si mette in attesa della terminazione dell'operazione di carico del transporttrolley che viene segnalata dalla ricezione del messaggio chargetaken.

Implementazione TicketService

L'attore resta in uno stato di attesa in cui attende uno di questi messaggi:

```
Request newticket               : newticket( FW )
Request ticketrequest : ticketrequest( TICKET, FW )
Dispatch updatevirtualweight : updatevirtualweight(FW)
```

- newticket è la richiesta con cui si richiede la creazione di un nuovo ticket. FW rappresenta la quantità di peso per cui si richiede il ticket.
- updatevirtualweight viene ricevuto quando si esegue un'operazione di svuotamento della ColdRoom. Esso è necessario per aggiornare il peso virtuale sottraendo il peso svuotato. FW rappresenta il peso svuotato dalla ColdRoom.

Implementazione EmptyColdRoom

L'attore rimane in uno stato nel quale periodicamente invia il seguente messaggio:

```
Request clearColdRoom : clearColdRoom(_)
```

Tale messaggio (senza parametri) viene inviato alla ColdStorageService.

Implementazione FridgeTruck

Questo attore simula l'utente finale. Tale attore, come l'utente finale, interagisce con il sistema inviando i seguenti messaggi:

```
Request newticket               : newticket( FW )
Request ticketrequest : ticketrequest( TICKET, FW )
Request loaddone        : loaddone(FW)
```

Implementazione ColdRoom

è stato deciso di rappresentarla mediante due variabili:

```
var CurrentWeightReal;
var CurrentWeightVirtual;
```

- CurrentWeightReal rappresenta il peso effettivo contenuto all'interno della ColdRoom, e viene gestito dalla ColdStorageService
- CurrentWeightVirtual rappresenta il peso effettivo contenuto all'interno della ColdRoom sommato al peso totale dato dai ticket emessi ma non ancora elaborati e viene gestito dal TicketService.

Implementazione ColdStorageService

L'attore gestisce i seguenti messaggi:

```
Request ticketrequest : ticketrequest( TICKET, FW )
Request clearColdRoom : clearColdRoom(_)
Dispatch goMoveToIndoor   : goMoveToIndoor(_)
Dispatch goMoveToHome     : goMoveToHome(_)
Dispatch deposit : deposit(_)
```

- goMoveToIndoor viene inviato al transporttrolley per iniziare la deposit action
- goMoveToHome viene inviato al transporttrolley quando non ci sono richieste di carico in attesa
- deposit viene invato dal transporttrolley per indicare che è terminata correttamente l'operazione di deposito e l'attore aggiorna il valore del CurrentWeightReal.

Problema driver distratto

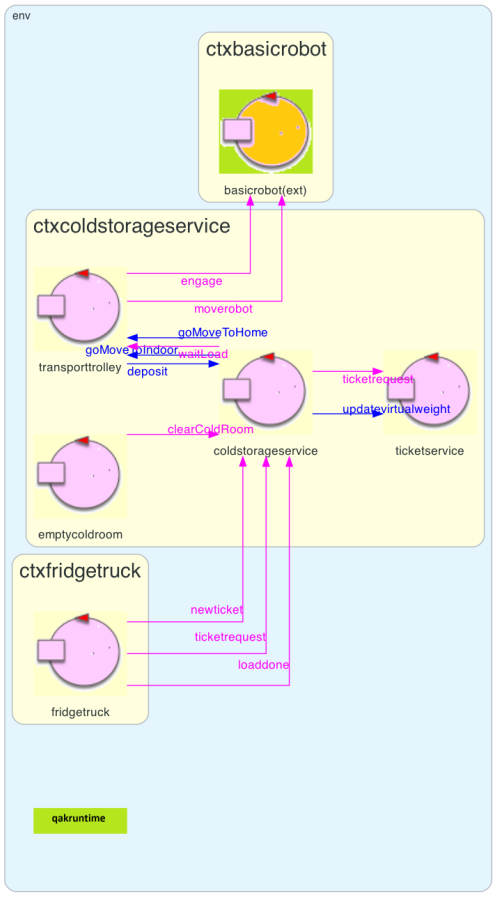
Tale problema, descritto in precedenza, viene risolto con il seguente controllo:

```
if(foodWeight < ticket!!.fw) {
    currentWeightVirtual -= (ticket!!.fw-foodWeight)
}
```

- foodWeight è il peso effettivo dall'utente
- ticket!!.fw è il peso dichiarato dall'utente nella fase di emissione del ticket

Architettura logica finale

Metamodello qak



coldstorageserviceArch

Test

I test sviluppati prevedono:

1. il controllo del flusso di esecuzione del fridge truck, dalla richiesta del ticket al deposito
2. svuotamento cold room

Fridge truck

```
@Test
public void mainUseCaseFridgeTruckTest() throws IOException, InterruptedException {

    String fw = "10";

    //invio messaggio
    out.write("msg(newticket,request,tester,coldstorageservice,newticket("+fw+"),12)\n");
    out.flush();

    //attesa risposta
    String response = in.readLine();
    assertTrue(response.contains("newticketaccepted"));
    String ticket= response.split(",")[4].split("\\(")[1].split("\\)")[0];
    //String secret= response.split(",")[5].split("\\)")[0];

    //invio elabTicketRequest
    out.write("msg(ticketrequest,request,tester,coldstorageservice,ticketrequest("+ticket+","+fw+"),13)\n");
```

```
    out.flush();

    //verifica ticket accepted
    response = in.readLine();
    assertTrue(response.contains("ticketaccepted"));

    //invio loadDone
    out.write("msg(loaddone,request,tester,coldstorageservice,loaddone("+fw+"),14)\n");
    out.flush();

    //risposta chargetaken
    response = in.readLine();
    assertTrue(response.contains("chargetaken"));

}
```

Svuotamento cold room

```
@Test
public void emptyColdRoom() throws IOException {
    //invio messaggio
    out.write("msg(clearColdRoom,request,tester,coldstorageservice,clearColdRoom(0),12)\n");
    out.flush();

    //attesa risposta
    String response = in.readLine();
    assertTrue(response.contains("coldRoomCleared"));
}
```

By Ziosi Lorenzo email: lorenzo.ziosi3@studio.unibo.it, GIT repo: https://github.com/role-nzo/ISSZiosiLorenzo/

By Dominici Luca email: luca.dominici3@studio.unibo.it, GIT repo: https://github.com/lucaDomo/Iss23DominiciLuca

By Zacchiroli Enrico email: enrico.zacchiroli@studio.unibo.it, GIT repo: https://github.com/zack-99/IssLab2023EnricoZacchiroli

Tema finale GIT repo: https://github.com/role-nzo/ISSTemaFinale/