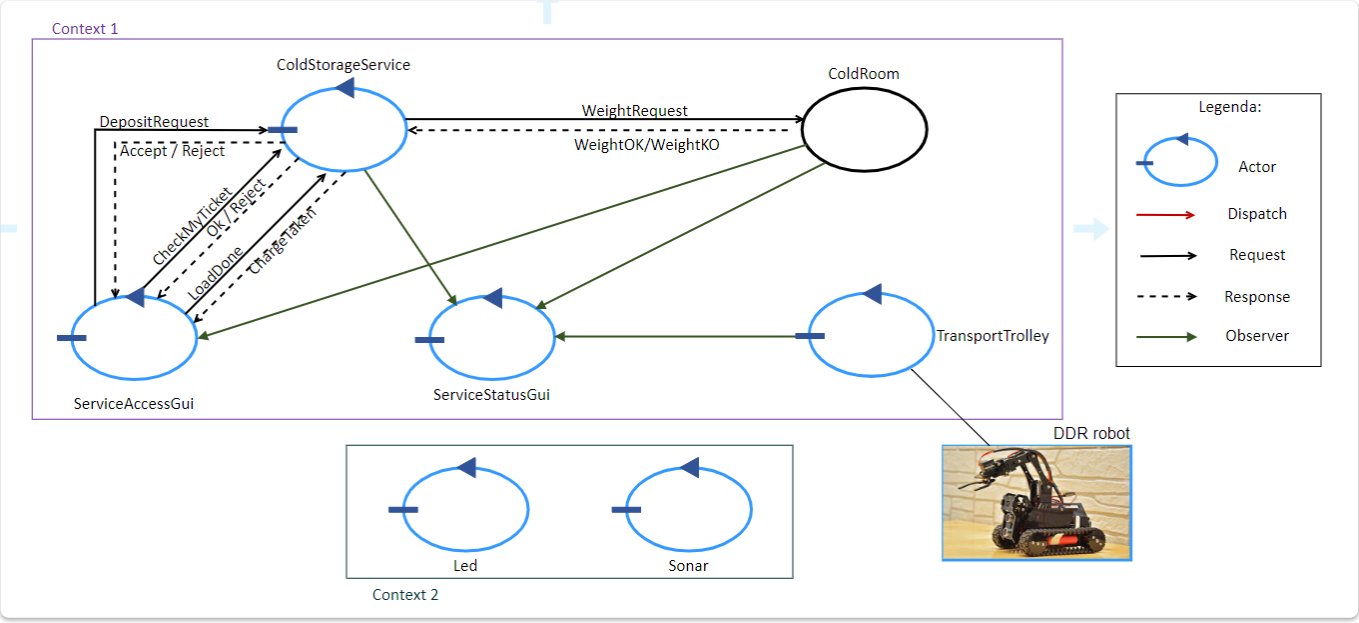
- [ ] Alcuni link fanno riferimento a doc solo sul mio pc, da cambiare...

Prodotto dello Sprint 0

È stata individuata un'architettura logica iniziale che definisca le macro-entità del sistema e le loro interazioni, link al modello precedente.



Goal Sprint 1

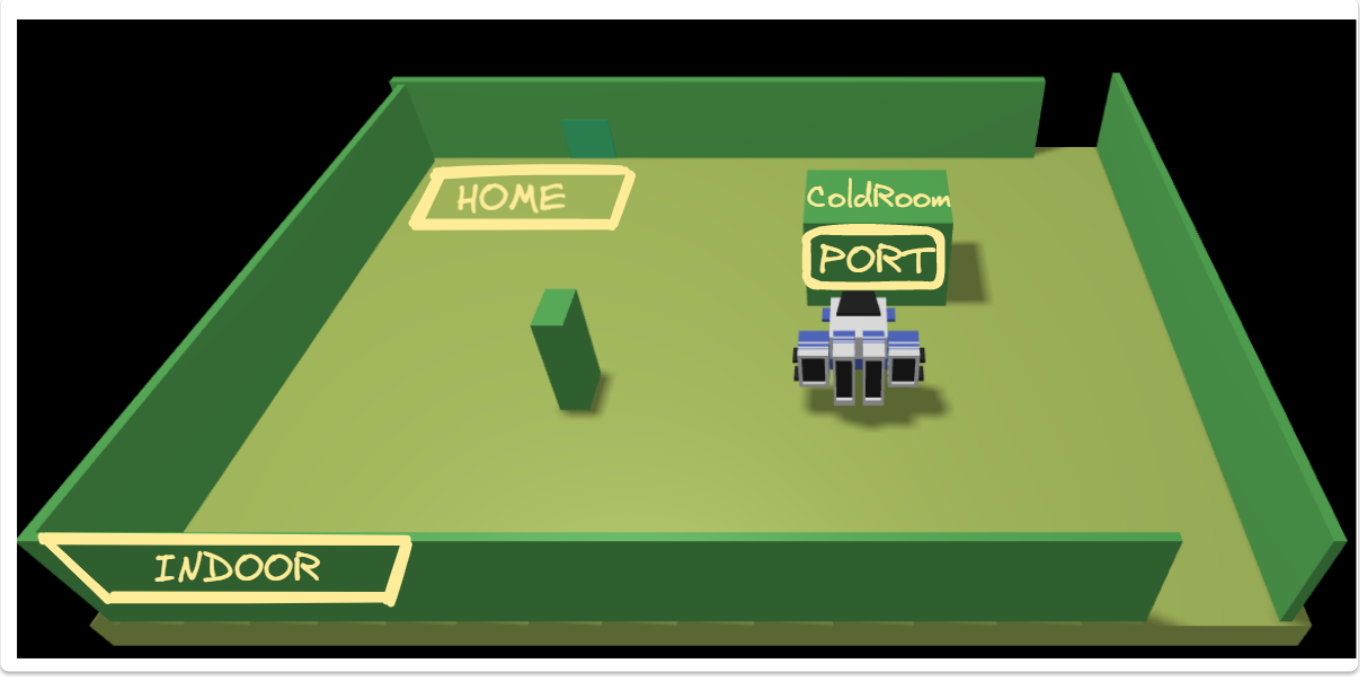
1. Transport Trolley + ColdStorageService

> **Descrizione** >
> Lo scopo del primo sprint è produrre una prima versione funzionante del core dell'applicazione. Questo comprende ColdStorageService con la logica di gestione dei Ticket e il TransportTrolley funzionante.
> A questa parte deve essere affiancata una mock version della ServiceAccessGUI per la fase di testing.

Requisiti relativi allo sprint corrente

A company intends to build a ColdStorageService, composed of a set of elements:

1. a service area (rectangular, flat) that includes:
   - an INDOOR port, to enter food (fruits, vegetables, etc.)
   - a ColdRoom container, devoted to store food, upto MAXW kg .
     The ColdRoom is positioned within the service area, as shown in the following picture:



2. a DDR robot working as a transport trolley, that is intially situated in its HOME location. The transport trolley has the form of a square of side length RD.
   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

> *Rinviato a Sprint successivo (see below)*
> *3. a ServiceAcessGUI that allows an human being to see the current weigth of the material stored in the ColdRoom and to send to the ColdStorageService a request to store new **FW** kg of food. If the request is accepted, the services return a ticket that expires after a prefixed amount of time (**TICKETTIME** secs) and provides a field to enter the ticket number when a Fridge truck is at the INDOOR of the service.*

Service users story

> *Rinviato a Sprint successivo (see below)*

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.
5. While the transport trolley is moving, the Alarm requirements should be satisfied. However, the transport trolley should not be stopped if some prefixed amount of time (**MINT** msecs) is not passed from the previous stop.
6. A *Service-manager* migtht use the ServiceStatusGUI to see:
   - the **current state** of the transport trolley and it **position** in the room;
   - the **current weigth** of the material stored in the ColdRoom;
   - the **number of store-requests rejected** since the start of the service.

Analisi del TF23

Nelle discussioni con il committente, sono emerse alcune problematiche:

- Il problema del load-time lungo.
- Il problema del driver distratto (non coerente, rispetto alle due fasi: scarico preceduto da prenotazione).
- Il problema del driver malevolo.
- Il problema di garantire che una risposta venga sempre inviata sempre solo a chi ha fatto la richiesta, anche quando la richiesta è inviata da un 'alieno' come una pagine HTML

Il problema del load-time lungo

> *Rinviato a Sprint successivo (see below)*

Il problema del load-time lungo è stato affrontato da Arnaudo/Munari con l'idea di inviare due messaggi di 'risposta' (una per dire al driver che il ticket inviato è valido e una per inviare `chargeTaken`). A questo fine hanno fatto uso diretto della connessione TCP stabilita da una versione prototipale dell'accessGui fatta come GUI JAVA.
Per consentire questa possibilità anche a livello di modellazione qak, in *ActorBasicFsm* è stato introdotto il metodo storeCurrentRequest() che permette di ricordare la richiesta corrente (cancellata da una **replyTo**). Questo però è un trucco/meccanismo che potrebbe risultare pericoloso.
Meglio affrontare il problema dal punto di vista logico, impostando una interazione a DUE-FASI tra driver e service (compito che può svolgere la *serviceAcessGui*).

- FASE1: il driver invia il ticket e attenda una risposta (immediata) come ad esempio `ticketaccepted/ticketrejected`
- FASE2: il driver invia la richiesta `loaddone` e attenda la risposta (`chargeTaken` o fallimento per cause legate al servizio)

Il problema del 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 al carico indicato nella prenotazione.
Ciò in quanto non vi sono sensori (bilance , etc) che possano fornire il valore del carico effettivo sul Truck.

Analisi dei Requisiti

requisiti sprint 0

> **domanda** >
>
> Possiamo limitarci a mettere il riferimento allo sprint 0 o dobbiamo riportare tutto?

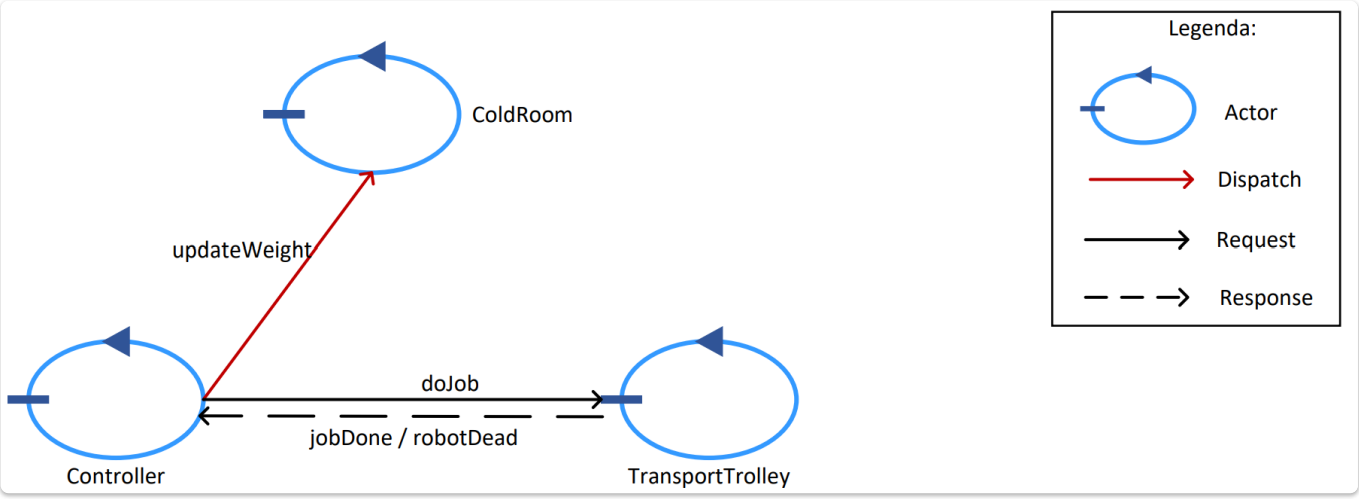
Analisi del Problema

Responsabilità di ColdStorageService

ColdStorageService è un componente caratterizzato da troppe responsabilità, abbiamo quindi deciso di sostituirlo con 2 attori:
- Controller: si occupa di gestire il robot ed aggiornare il peso di ColdRoom.
- TicketHandler: si occupa di gestire il ciclo di vita dei Ticket.

Nello sprint corrente ci occuperemo solo del Controller. La logica di gestione dei ticket è rimandata allo sprint successivo (Sprint 1.1 - V2)

Cerchiamo quindi di realizzare la seguente Architettura logica:



Segnale per Transport Trolley

Introduciamo un nuovo segnale "doJob" di tipo Req/Res inviato dal controller.

```
Request doJob : doJob(KG)
Reply jobdone : jobdone(NO_PARAM)
Reply robotDead : robotDead(NO_PARAM)
```

> **motivazioni** >
>
> Definiamo il seganle come un req/res poichè vogliamo sapere se il servizio richiesto è andato a buon fine oppure se il DDR robot ha avuto problematiche che lo hanno interrotto prima di proseguire con una seconda doJob.
>
> Limitiamo il controller ad un semplice comando di doJob, non è compito suo sapere quali operazioni deve compiere il robot per portare a termine il lavoro, è compito del robot stesso.

ATTENZIONE: la risposta deve essere inviata appena il carico è rilasciato nella ColdRoom e non quando il robot torna alla home per requisiti.

Aggiornamento peso ColdRoom

Se il servizio è andato a buon fine e viene restituita una "jobdone" allora il Controller aggiorna il peso della ColdRoom tramite Dispatch.

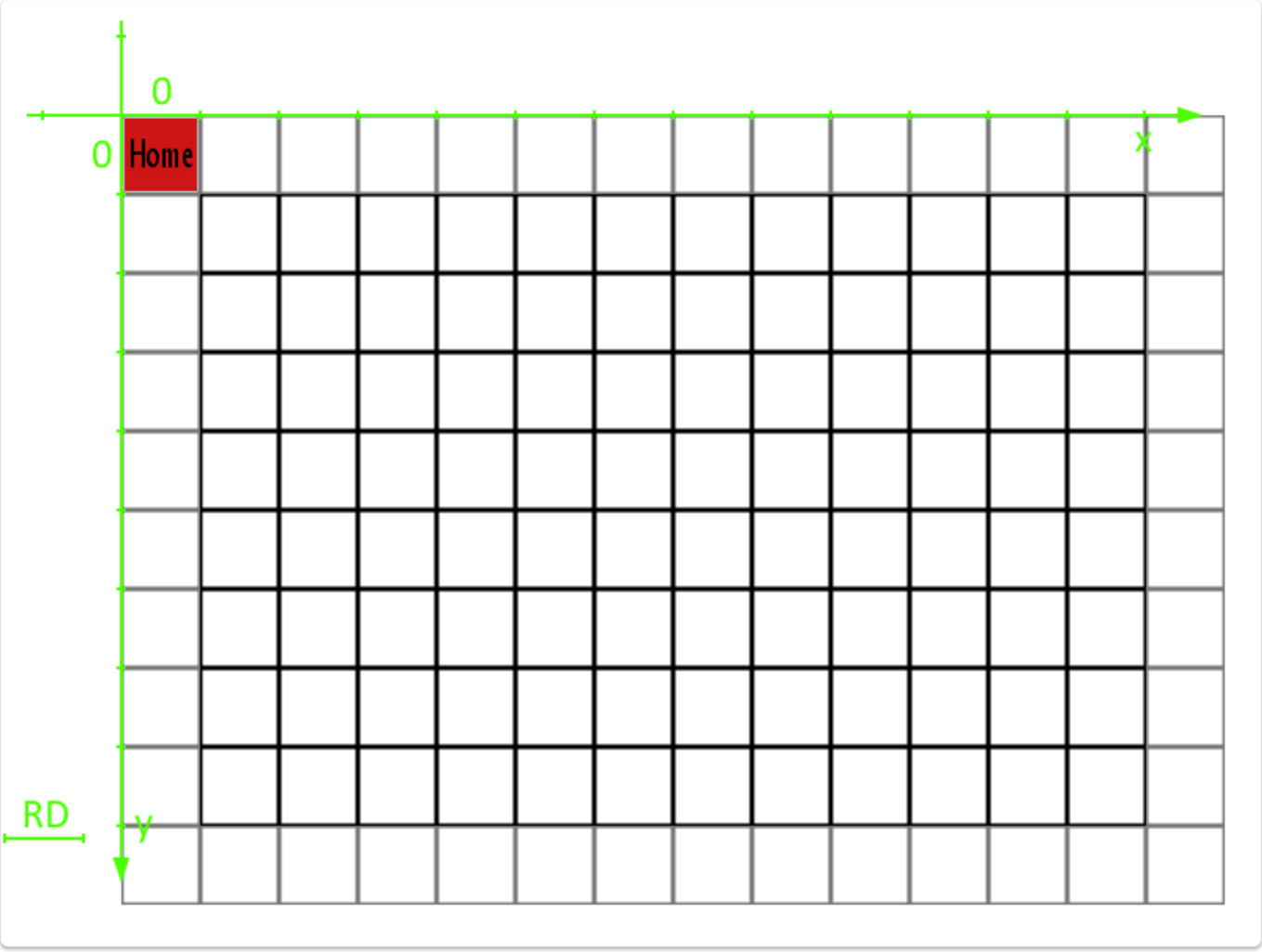
```
Dispatch updateWeight : updateWeight(PESO)
```

Da "doJob" a comandi per TransportTrolley

Dalla documentazione fornita è chiaro che non sia presente un comando che ci permetterebbe di limitarci ad un comando "doJob".
Se non fosse possibile implementarlo risulterebbe necessario aggiungere un componente intermedio che traduca in comandi comprensibili al TransportTrolley fornitoci.
Allo stesso modo è anche evidente la mancanza di un comando per caricare e scaricare i materiali trattati e quindi non risulta sufficiente.

Posizione nella Service Area

Per definire la posizione del TransportTrolley e permettere il movimento autonomo dividiamo la stanza in una griglia di quadrati di lato RD (lunghezza del DDR robot).
La Home corrisponderà all'origine (0, 0). Useremo coordinate crescenti verso il basso e verso destra.



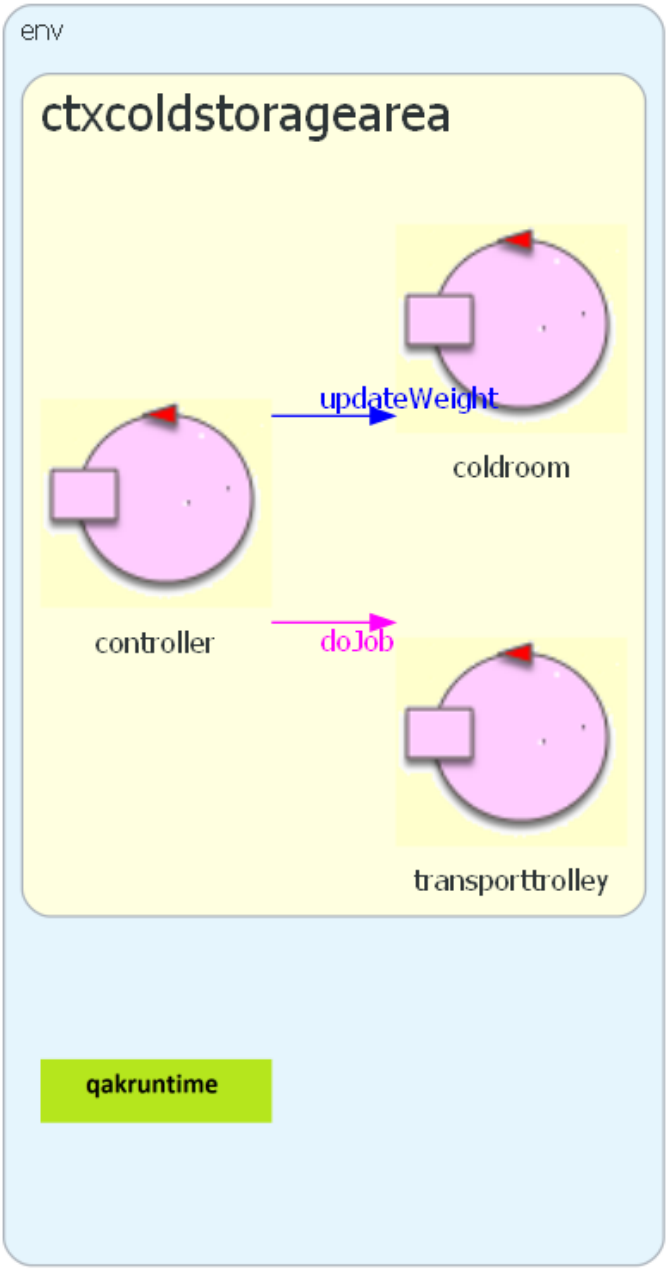
Date le dimensioni dell'area, Service Area sarà divisa in una griglia 4 x 6.
ColdRoom si troverà in posizione (5, 2).

Il TransportTrolley fornito possiede già il supporto a questo tipo di tecnologia. La mappatura della stanza deve essere fatta a priori e fornita tramite file all'avvio.

Peso massimo trasportabile

Dopo discussioni con il committente è stato decretato che il peso da scaricare non sarà mai maggiore del peso trasportabile del robot fisico.

Architettura logica dopo l'analisi del problema



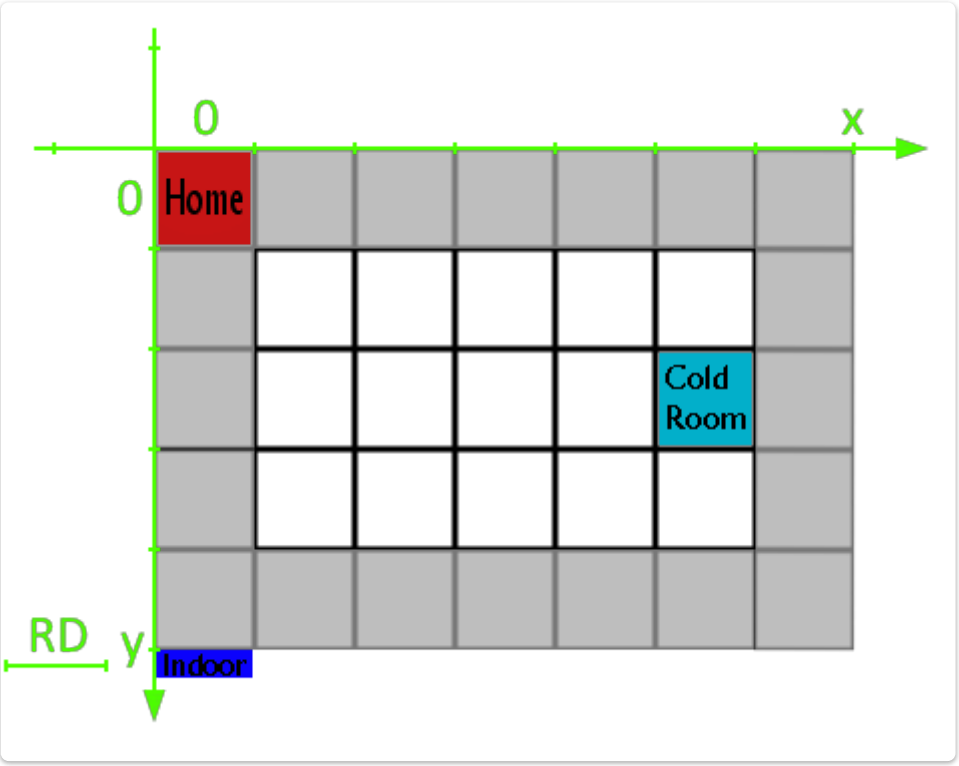
coldstorageArch

Progettazione

Sistema di coordinate

Sia RD l'unità di misura

```
Home = (0, 0)
Indoor = (0, 4)
ColdRoom = (5, 2)
ColdRoomPORT = (5, 3)      # posizione del robot per poter scaricare
Service Area = {
        height = 4              # asse x
        lenght = 6              # asse y
}
```



Definizione messaggi e contesti

```
System coldstorage

//-------------------------------------------------------------------

Request doJob : doJob(KG)
Reply jobdone : jobdone(NO_PARAM)
```

```
Reply robotDead : robotDead(NO_PARAM)

Dispatch updateWeight : updateWeight(PESO)

//-------------------------------------------------------------------

Context ctxcoldstoragearea ip [host="localhost" port=8040]

//-------------------------------------------------------------------
```

NOTA: in questo momento ColdRoom è definita nello stesso contesto di Controller, in futuro potrebbe non essere così (dipende dall'implementazione fisica della ColdRoom).

Controller

```
QActor controller context ctxcoldstoragearea {

        [# var KG = 0 #]

        State s0 initial { printCurrentMessage }
        Goto mockRequest

        # generiamo una richiesta casuale per il testing
        State mockRequest {
                [# KG = Math.random() #]
                request transporttrolley -m doJob : doJob($KG)
        } Transition endjob whenReply robotDead -> handlerobotdead
                                                   whenReply jobdone -> jobdone

        State jobdone{
                forward coldroom -m updateWeight : updateWeight($KG)
        } Transition repeat whenTime 15000 -> mockRequest

        State handlerobotdead{
                printCurrentMessage
        }
}
```

ColdRoom

```
QActor coldroom context ctxcoldstoragearea {
        [# var PesoEffettivo = 0 #]

        State s0 initial { printCurrentMessage }
        Transition update whenMsg updateWeight -> updateWeight
```

```
        State updateWeight {
                printCurrentMessage
                onMsg ( updateWeight : updateWeight(PESO) ) {
                        [# PesoEffettivo += payloadArg(0).toInt() #]
                }
        } Transition update whenMsg updateWeight -> updateWeight
}
```

TransportTrolley

```
QActor transporttrolley context ctxcoldstoragearea {
        [# var Peso = 0 #]

        State s0 initial{
                forward robotpos -m setrobotstate : setpos(0,0,down)         //set
Home pos
        } Transition ready whenMsg robotready -> work

        State work{
                println("robot waiting") color green
        } Transition startworking whenRequest doJob -> startjob          //wait for
doJob

        State startjob{
                onMsg(doJob : doJob( KG )){
                        [# Peso = payloadArg(0).toInt() #]
                        println("peso ricevuto: $Peso") color green
                }
        } Goto movingtoarrival

        State movingtoarrival{
                request robotpos -m moverobot : moverobot(0,4)
//move to indoor
        } Transition gofetch whenReply moverobotdone -> movingtocoldroom

        State movingtocoldroom{
                request robotpos -m moverobot : moverobot(5,3)
//move to coldroom
        } Transition godrop whenReply moverobotdone -> waitforjob

//alla fine di waitforjob mandiamo la risposta "jobdone" e attendiamo per
```

```
verificare //che non ci siano altre richieste "doJob" da portare avanti prima di
tornare alla Home
        State waitforjob {
                replyTo doJob with jobdone : jobdone( 1 )
                println("transporttrolley ! aspetto") color green
        } Transition gofetchagain
                        whenTime 3000 -> goinghome
//torna alla Home
                        whenRequest doJob -> startjob
//torna a scaricare

        State goinghome{
                request robotpos -m moverobot : moverobot(0,0)                //
Home pos
                forward robotpos -m setdirection : dir(down)
        }       Goto work
}
```

Deployment

1. Avviare il container itunibovirtualrobot23 su docker
   Viene lanciato l'ambiente virtuale con il robot all'indirizzo http://localhost:8090/
2. In intellij avviare il file MainCtxbasicrobot.kt del progetto BasicRobot
3. In intellij avviare il file MainCtxColdStorageArea.kt del progetto coldStorage