LABORATORIO DI INGEGNERIA DEI SISTEMI SOFTWARE

Introduction

Goal Sprint 0: analizzare e formalizzare i requisiti

Requirements

Requisiti dati dal committente

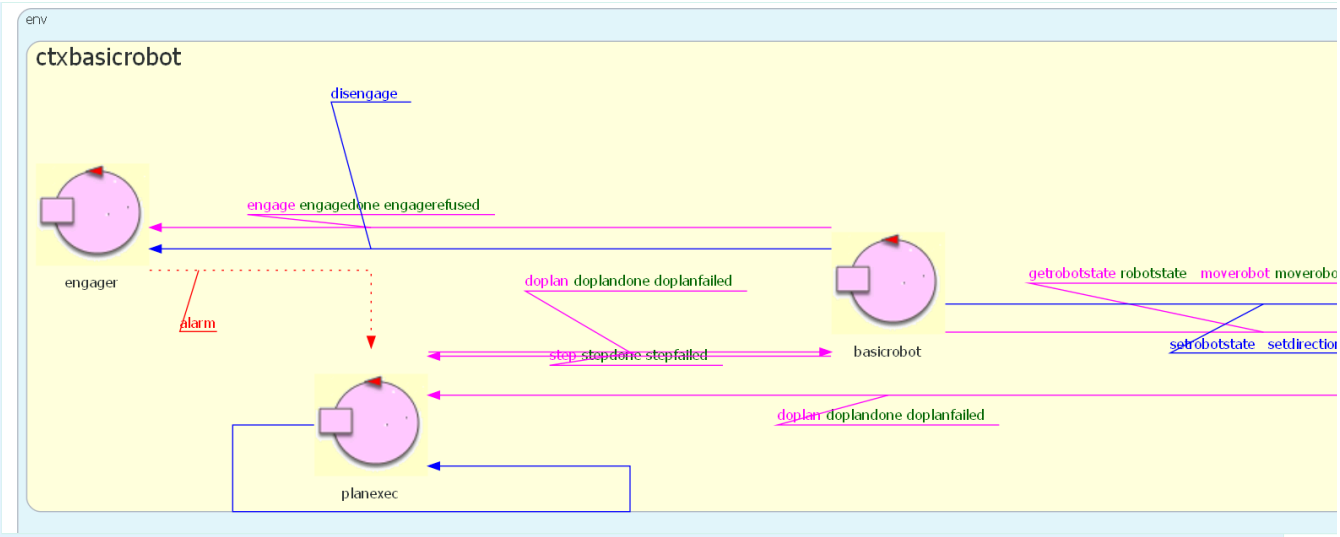
Requirement analysis

.

ColdStorageService	il servizio che si richiede di progettare
Service area	stanza piana e rettangolare che comprende INDOOR port e Cold Room
INDOOR port	area di servizio dove viene scaricato il carico
Cold Room	container di deposito del carico, con capacità massima di **MAXW** kg e una **PORT** di accesso
Transport trolley	interfaccia per l'utilizzo di un DDR robot, modellato come un quadrato con lato **RD**. Posizionato inizialmente in **HOME**
Service Access GUI	interfaccia utente che consente di visualizzare il peso dei materiali attualmente nella Cold Room e mandare la richiesta di depositare ulteriori **FW** kg. Se la richiesta viene accettata, l'utente ottiene un ticket valido per un tempo **TICKETTIME**
Service Status GUI	interfaccia utente che consente ad un *service manager* di visualizzare lo stato del servizio
Sonar	dispositivo connesso ad un Raspberry Pi. Misura la distanza: • quando è *minore* del limite dato **DLIMIT**, il transport trolley si ferma • riparte quando la distanza è maggiore di **DLIMIT**
Led	dispositivo connesso ad un Raspberry Pi. Il Led è: • **spento** quando il trolley è in **HOME** • **lampeggia** quando il trolley si sta muovendo • è **acceso** quando il trolley è fermo
Truck driver	l'utente che usa il servizio

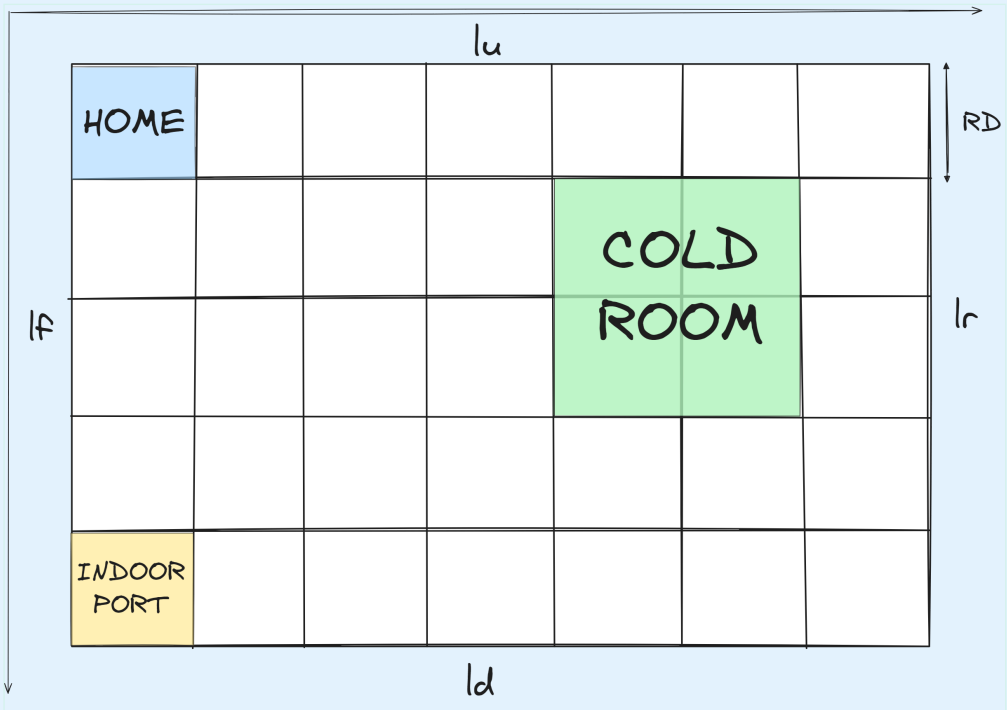
Il committente fornisce

- il metamodello QActor per la modellazione del sistema (si veda QakActors24 per maggiori informazioni)
- il servizio BasicRobot24: un componente software che esegue comandi di spostamento di un DDR robot in *modo indipendente dalla tecnologia* con cui questo è realizzato (virtuale o reale). Il servizio è realizzato ad **attori**:



La **SERVICE AREA** è rappresentabile come un rettangolo di lati lu, ld, lr e lf, con **ld == lu** e **lr == lf**. In riferimento alla modellazione del **DDR Robot** come quadrato di lato RD, possiamo:

- dividere l'area in **celle** quadrate di lato **RD**
- definendo un sistema di coordinate cartesiane, modellare INDOOR port e Cold Room come posizioni sulla mappa



Posizione: ogni cella della mappa è identificata da una coppia di coordinate cartesiane

```
   0  1  2  3  4  5  6  7  x
0 |r, 1, 1, 1, 1, 1, 1,
1 |1, 1, 1, 1, X, X, 1,
2 |1, 1, 1, 1, X, X, 1,
3 |1, 1, X, 1, 1, 1, 1,
4 |1, 1, 1, 1, 1, 1, 1,
5 |X, X, X, X, X, X, X,
y
```

- **r**: posizione corrente del robot
- **X**: cella occupata da un ostacolo

- **1**: cella libera

La **Service Access GUI** è l'interfaccia che consente l'interazione dell'utente con il sistema per:

- vedere il peso del carico attualmente nella **ColdRoom**
- inviare la richiesta di deposito di **FW** kg di cibo al *ColdStorageService*
- inserire il numero del ticket quando il *Fridge truck* raggiunge l'**INDOOR port**

La **Service Status GUI** è l'interfaccia che consente al *Service-manager* (un utente esterno) la visualizzazione di informazioni sul sistema.

Entrambe le interfacce possono essere inizialmente modellate anch'esse come attori grazie all'uso del metamodello QActor.

Alarm requirements

Il committente fornisce il software di supporto per l'uso di Sonar e Led. I due dispositivi fisici possono essere inizialmente modellati come attori esterni al sistema.

Use cases and scenarios

User story data dal committente

1. L'utente invia una richiesta tramite la *Service Access GUI* per depositare **FW** kg di carico. Se la richiesta è accettata, deve arrivare alla INDOOR port nel tempo **TICKETIME**, altrimenti la richiesta sarà rifiutata.
2. Una volta accettata la richiesta, il *ColdStorageService* risponde con un messaggio **charge taken**. Una volta ricevuto, l'utente lascia la INDOOR port.
3. Quando il *ColdStorageService* accetta una richiesta, viene inviato un messaggio al *trolley*, che deve raggiungere la INDOOR port e prendere il carico. In seguito, il *trolley* risponde con il messaggio **charge taken** e va alla *ColdRoom*.
4. Quando finisce un'azione di deposito, il trolley può accettare un'altra richiesta - se presente - o tornare in HOME.
5. Mentre il *trolley* è in movimento, i requisiti di allarme devono essere rispettati. Il trolley non si fermerà solo nel caso in cui un prefissato intervallo di **MINT** secondi non sia passato dallo stop precedente.
6. La *Service Status GUI* può consentire di monitorare lo **stato corrente** del trolley, il **peso** del carico nella *ColdRoom*, il numero di **richieste rifutate** dall'inizio del servizio.

I key points *5*, *6* saranno trattati in seguito, in quanto non parte significativa del core del servizio.

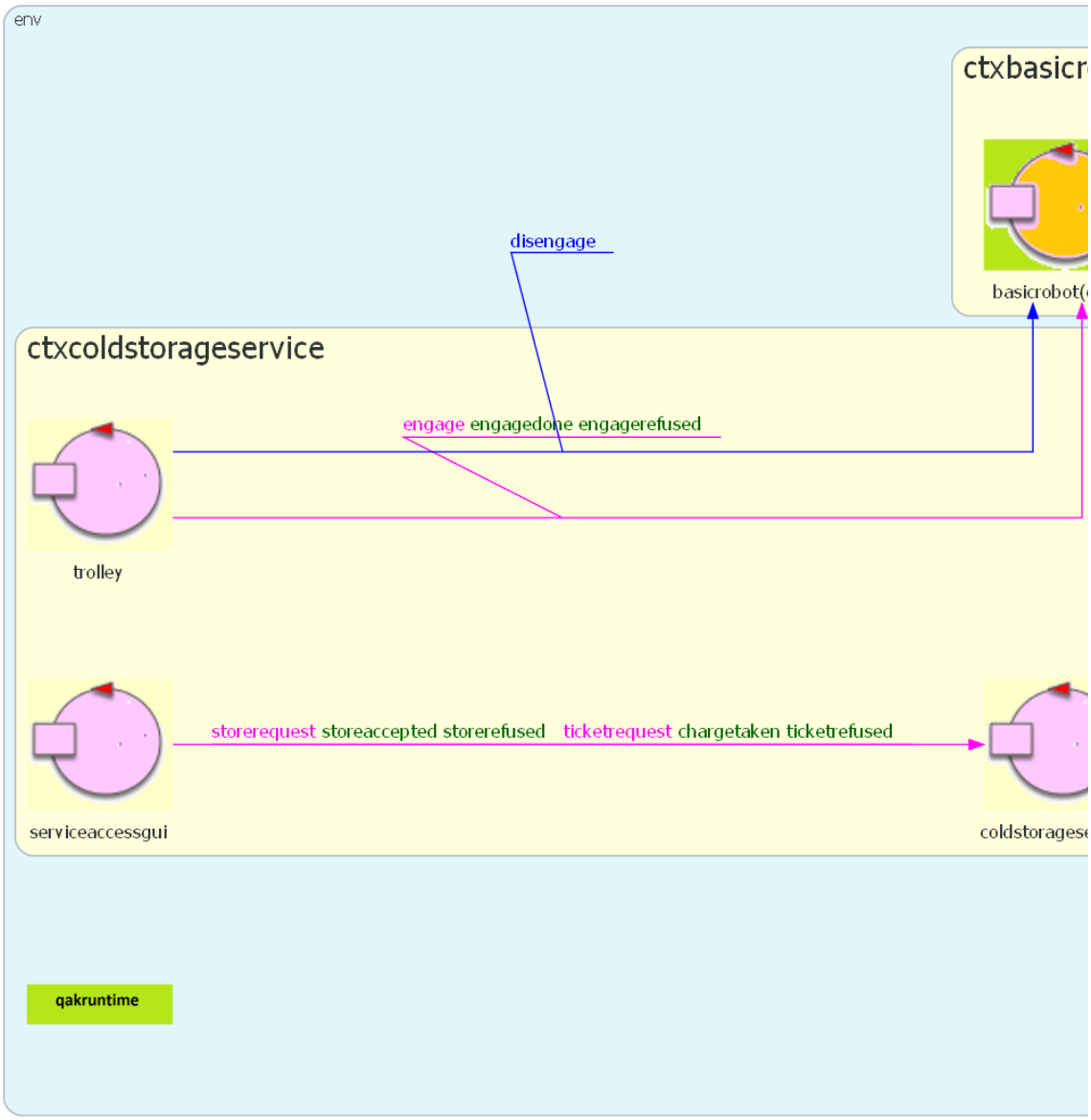
Problem analysis

Per realizzare un primo modello del sistema sulla base delle analisi, si sceglie di utilizzare il linguaggio di modellazione Qak fornito dal committente. Il metamodello

- consente di catturare gli aspetti essenziali del sistema
- offre l'astrazione **QActor** per rappresentare le entità come componenti autonomi ed indipendenti.

La *Software Factory* definita per il linguaggio crea automaticamente un modello eseguibile in **Kotlin**.

L'architettura del sistema è la seguente:



coldstorageserviceArch

Il sistema é costituito da due **contesti**:

- **ctxbasicrobot** per il **basicrobot** fornito dal committente (attore esterno)
- **ctxcoldstorageservice** per **serviceaccessgui**, **coldstorageservice** e **trolley**

Qui è possibile visualizzare il modello definito: link modello

SERVICE ACCESS GUI

```
QActor serviceaccessgui context ctxcoldstorageservice {
```

```
State s0 initial {
    printCurrentMessage
    println("$name START ") color blue
}
Goto sendstore

State sendstore {
```

- La **serviceaccessgui** è modellata come un QActor che simula le interazioni dell'utente con il sistema
- ***sendstore***: invia la richiesta di *storerequest* al **coldstorageservice** e attende una risposta che può essere
  - negativa: *storerefused* che porta allo stato ***endwork***
  - positiva: *storeaccepted* che porta allo stato ***sendticket***
- ***sendticket***: si simula lo spostamento dell'utente all'INDOOR e l'invio del numero del ticket al **coldstorageservice**, che può accettare o meno il carico: *chargetaken*, *ticketrefused*

COLD STORAGE SERVICE

```
QActor coldstorageservice context ctxcoldstorageservice {

    // model the cold room
    [#
        var MAXW = 200
        var TICKETTIME = 15
        var Current_load = 0f
        var TicketNumber = 1
    #]
```

- Il **coldstorageservice** definisce le variabili di sistema:
  - **MAXW**: carico massimo della coldroom
  - **TICKETTIME**: tempo di validità del ticket
  - **Current_load**: il carico attuale della coldroom
  - **TicketNumber**: per ottenere i numeri incrementali dei ticket
- L'attore gestisce due possibili richieste: *storerequest* e *ticketrequest*
- ***handlestore***: si verifica che nella coldroom ci sia abbastanza spazio per il carico:
  - *storeaccepted*: la richiesta viene accettata e viene inviato il ticket all'utente
  - *storerefused*: la richiesta viene rifiutata
- ***handleticket***: si calcola il tempo trascorso dall'emissione del ticket:
  - *chargetaken*: il tempo trascorso è minore di TICKETTIME e la richiesta è accettata
  - *ticketrefused*: il tempo trascorso è maggiore e la richiesta non è più valida

TROLLEY

```
QActor trolley context ctxcoldstorageservice {

    State s0 initial {
        printCurrentMessage
        println("$name START ") color magenta
        println("$name engage BASIC ROBOT ") color magenta
        request basicrobot -m engage: engage(trolley, 150)
    }
    Transition t0 whenReply engagedone -> waitrequest
```

- ***s0***: il **trolley** invia la richiesta di *engage* al **basicrobot** e attende l'esito positivo dell'operazione per passare allo stato ***waitrequest***
- ***waitrequest***: attende la ricezione di una richiesta da parte del servizio. Dato che l'interazione tra trolley e servizio non è direttamente ricavabile dai requisiti, al momento questa viene simulata con un *delay* per poi passare allo stato successivo
- ***gotoindoor***: simulazione dello spostamento del robot verso la INDOOR port
- ***takeload***: simula il caricamento del robot
- ***gotocoldroom***: simula il movimento del robot dalla INDOOR port alla porta della coldroom
- ***storeload***: il robot scarica il carico nella coldroom. Dopo due secondi passa nello stato successivo
- ***gohome***: il robot ritorna in HOME, in posizione **(0, 0)**
- ***trolleyathome***: stato finale in cui viene inviata la richiesta di *disengage* al **basicrobot**

Test plans

Per testare la corretta gestione delle richieste inviate al sistema, è stata realizzata una variante del modello in cui i messaggi non vengono inviati dalla **serviceaccessgui**, usando invece una specifica test class.

- Link modello **coldstorageservice** per test
- Link test unit

storeRequestTest()

```
public class ColdStorageServiceTest {

    public static final String ADDR         = "localhost";
    public static final int PORT            = 8015;
    public static final String MSG          = "msg( storerequest, request, serviceaccesgui,
    public static final String REPLY        = "storeaccepted";

    @Test
    public void storeRequestTest() {
```

Test sul funzionamento del sistema: viene inviata la richiesta di depositare un numero di KG inferiore a **MAXW** e si verifica che la richiesta venga accettata.

handleTicketTest()

```
@Test
public void handleTicketTest() {


    try (
        Socket client = new Socket(ADDR, PORT);
        BufferedWriter out = new BufferedWriter(new OutputStreamWriter(client.getOutputStrea
        BufferedReader in = new BufferedReader(new InputStreamReader(client.getInputStream()
    ) {
        // send store request
```

Test sulla gestione della seconda richiesta: dopo aver ricevuto il ticket, viene inviata una seconda richiesta al sistema prima dello scadere del **TICKETTIME** e il ticket risulta valido.

Project

Piano di lavoro

SPRINT1	• prototipo *coldstorageservice*: keypoints 1, 2, 3, 4 della **user story** • testing
SPRINT2	• estensione del sistema con introduzione degli alarm requirements • testing
SPRINT3	• realizzazione GUI di sistema • testing

- By Letizia Mancini
- email: letizia.mancini3@studio.unibo.it
- GIT repo: https://github.com/llevtizia/coldstorageservice-iss2023
- matricola: 0000926656