Sprint 0

Indice

- Obiettivi
- Requisiti
- Vocabolario
- Macrocomponenti
- Architettura di Riferimento
- Piano di Test
- Piano di Lavoro
- Team di Lavoro e Attività Specifiche

Obiettivi

L'obiettivo prinicipale dello sprint 0 è analizzare i requisiti forniti dal committente riguardo al progetto **TemaFinale25** eliminando eventuali ambiguità relative ai termini utilizzati e formalizzando quest'ultimi con dei componenti software (che possono essere già sviluppati o ancora da sviluppare).

Una volta formalizzati i requisiti si procederà con la definizione di una prima **architettura logica generale del sistema** che sarà il riferimento iniziale per lo sprint 1.

Questa architettura iniziale sarà costituita dai **macrocomponenti principali** (bounded context) deducibili dal documento dei requisiti, e dalle **interazioni** tra quest'ultimi sotto forma di messaggi (mappa di contesto).

L'architettura logica sarà anche corredata di un relativo piano di test iniziale.

Terzo e ultimo obiettivo dello sprint 0 sarà quello di definire un **piano di lavoro** in cui si definirà il numero e gli obiettivi degli sprint necessari al completamento del sistema.

Inoltre, in questa sezione si deciderà come suddividere il lavoro tra i membri del gruppo, ed eventualmente, quali sprint potranno essere realizzati in parallelo.

Premessa | metamodello QAK

L'analisi dei requisiti riportata in questo sprint iniziale verrà eseguita basandosi sul concetto di attore e sfruttando il metamodello QAK e la relativa **software factory**.

L'utilizzo del concetto di attore durante l'analisi dei requisiti permette di modellare le entità del sistema che hanno **un proprio flusso di controllo** (attive) e che **interagiscono con altre entità mediante scambio di messaggi**. Questa tipologia di entità è molto presente nel dominio e nei requisiti del sistema presentato dal commmittente; per questo motivo è stato ritenuto opportuno la modellazione tramite attori. Se si fossero utilizzati dei semplici POJO (vedi metamodello UML) l'abstraction gap tra le entità da modellare e gli strumenti di modellazione sarebbe stato troppo alto.

La particolare tipologia di attori utilizzata è quella del **metamodello QAK**. Quest'ultimo fornisce un insieme di concetti fondamentale per la modellazione di sistemi distribuiti a microservizi come quello richiesto dal

committente. Tra questi concetti, quelli di particolare interesse sono:

- le varie tipologie di **messaggi** scambiabili tra gli attori del sistema. Ognuno con la propria sintassi e semantica.
- il concetto di **contesto** come nodo logico all'interno di un sistema distribuito

Infine, l'adozione del metamodello QAK è anche motivata dalla presenza di una **software factory** che è in grado di generare automaticamente codice Kotlin a partire dai modelli scritti in linguaggio QAK. Questo permette di ottenere dei **modelli eseguibili**, riducendo notevolmente la fasi di progettazione e implementazione negli sprint successivi.

Per i motivi sopra elencati, il metamodello QAK continuerà ad essere usato anche negli sprint.

Requisiti

I requisiti sono descritti dal committente nel documento: documento dei requisiti.

Vocabolario

E' stato ritenuto più opportuno presentare i termini del dominio in un formato discorsivo, prolisso e ridondante, per garantire non solo che i termini non siano ambigui, ma anche che per il lettore sia difficile malinterpretare.

Questo si contrappone al più comune approccio, in cui il vocabolario è un mero elenco puntato, nel quale è possibile che una mancata disambiguazione, possa evolvere in errori considerevoli durante lo sviluppo.

Ci riserviamo la possibilità di ricondurci al classico vocabolario-elenco nel caso dovessero sorgere complicazioni inaspettate, sicuri del fatto che l'eventuale conversione sarà semplice.

Differential Drive Robot (DDR)

Un **Differential Drive Robot (DDR)** è un tipo di robot fisico e mobile che si muove grazie a due ruote motrici indipendenti, solitamente affiancate sullo stesso asse, che gli consentono di avanzare, arretrare o ruotare su sé stesso.
All'interno del sistema il DDR è il supporto fisico che viene comandato dal cargorobot per implementarne le azioni nel mondo reale.

Stiva ed Elementi della Stiva

Il **deposito/stiva (Hold)** rappresenta il workspace del cargorobot: lo scenario entro cui è confinato ed in cui svolge i suoi **interventi di carico**.
È di forma rettangolare e sono presenti alcune aree notevoli e fisse che è bene segnalare con un nome univoco:

- La **home** è la postazione da cui il cargorobot parte e a cui il cargorobot fa ritorno al termine di ogni intervento.
- Gli **slot** sono le cinque postazioni in cui i container vengono depositati.
  Notare che queste postazioni sono distinte dalle **laydown-positions**; siccome il cargorobot scarica il container che sta trasportando davanti a se (i requisiti specificano che lo slot numero 5 è sempre occupato e quindi si riduce ad un ostacolo).

- Le **laydown-position** sono le postazioni in cui il cargorobot si posiziona per effettuare l'operazione di deposito.
  Ogni laydown-position è associata ad uno slot e richiede che il cargorobot sia orientato verso quest'ultimo per effettuare la sua operazione di deposito.
- La **IO-Port** indica la posizione dove verranno depositati i container in arrivo.
  Similmente agli slot, alla IO-port è associata una pickup-position che ha una posizione distinta.
- La **pickup-position** è la postazione in cui il cargorobot si posiziona per recuperare il container arrivato alla IO-Port. Il cargorobot deve essere rivolto verso l'IO-port per caricare il container.

Le aree appena elencate sono **attributi** dell'entità deposito e possono essere modellate come POJO in quanto entità passive.
Inoltre, il deposito ha come ulteriore attributo **MaxLoad** che definisce il peso massimo trasportabile dalla nave.
Non sarebbe irragionevole immaginare di rappresentare il deposito come un POJO che aggrega gli attributi appena descritti, ma si è ritenuto più opportuno modellarlo come un attore per **separare la logica dei dati dalla logica di buisness** (responsabilità di cargoservice) seguendo il Single Responsability Principle.

Si è detto che il deposito è l'area di lavoro del cargorobot: si noti, a tal proposito che il tentativo di descrivere il deposit, e gli elementi che lo compongono, inevitabilmente, conduce a una prima vaga descrizione dei processi che vi si svolgono (**interventi di carico**).
Questo dipende dalla scelta, deliberata, di utilizzare gli elementi del deposito per descriverlo e questo crea nel deposito una dipendenza forte dai suoi elementi, che riteniamo ragionevole, perchè non è particolarmente realistico, che una delle estensioni future preveda un deposito senza questi elementi notevoli al suo interno.

Container e Prodotti

Un **container** è un recipiente di dimensioni predefinite che contiene un determinato prodotto da trasportare.
Ogni **richiesta di carico** è sempre associata ad un container, e quest'ultimi vengono sempre recuperati dalla IO-Port per poi essere trasportati dal cargorobot dentro a uno degli slot liberi.

I **prodotti** sono merci di qualsiasi tipo sempre trasportate all'interno di un container. Ogni prodotto, viene sempre **registrato** dentro a productservice prima di essere caricato.

productservice è un servizio utilizzato per registrare i prodotti che devono essere caricati all'interno del deposito dentro ad un database.
L'operazione di registrazione consiste nello specificare il peso del prodotto da registrare, successivamente productservice restituirà l'identificatore unico (PID) del prodotto appena registrato.

Abbiamo quindi che un prodotto è una entità passiva modellabile come un POJO che ha come attributi:

- un peso: valore reale,
- ed un PID: valore intero > 0.

Il committente fornisce già tutto il software necessario per l'implementazione di productservice e dei prodotti.
In particolare il componente che implementa le funzionalità di productservice è un **attore** che si chiama (sfortunatamente) cargoservice.

Questo è un nome infelice in quanto coincide con quello di un altro macrocomponente del sistema; da ora in poi ci si riferirà al software che il committente ha fornito con productservice.

I messaggi con cui si può interagire con productservice sono i seguenti.

```
Request createProduct : product(String)
Reply   createdProduct: productid(ID) for createProduct

Request deleteProduct  : product( ID )
Reply   deletedProduct : product(String) for deleteProduct

Request getProduct : product( ID )
Reply   getProductAnswer: product( JSonString ) for getProduct

Request getAllProducts : dummy( ID )
Reply   getAllProductsAnswer: products(  String ) for getAllProducts
```

Cargorobot e Basicrobot

Un **cargorobot** è un robot **logico** capace, **sotto richiesta**, di:

- muoversi liberamente all'interno del deposito.
- recuperare un container dall'IO-port,
- posizionare un container precendentemente recuperato in uno degli slot liberi all'interno del deposito.

È opportuno modellare il cargorobot come **attore** in quanto è un componente attivo che ha un proprio flusso di controllo.

Il committente fornisce del software per la modellazione del cargorobot. In particolare:

- l'ambiente virtuale WEnv che simula un DDR e la stiva della nave in cui il cargorobot dovrà operare
- un componente software chiamato basicrobot che permette di governare un DDR virtuale all'interno di WEnv

L'ambiente virtuale **WEnv** modella la tipica stiva della nave in cui dovrà andare ad operare il cargorobot e include un simulatore di DDR.

Il DDR all'interno di WEnv è un **robot inscrivibile in un cerchio di raggio R** che può compiere solamente 4 mosse:

- andare avanti per un certo periodo di tempo
- andare indietro per un certo periodo di tempo
- ruotare a destra di 90°
- ruotare a sinistra di 90°

Il committente fornisce il tempo necessario al DDR per andare avanti o indietro di **una distanza pari alla sua dimensione**.
Si ha quindi a disposizione anche una quinta mossa elementare chiamata **step** che corrisponde ad un passo del DDR lungo 2R.

Il committente ha anche specificato che lo **step fornisce un unità di misura per le dimensioni del deposito**.
Il deposito pertanto può essere modellato come una **griglia rettangolare composto da HHxHW (Hold Height per Hold Width) celle**.
Ogni cella è un quadrato con lato pari all'unità di misura robotica, ovvero 2R.

Basandosi sull'unità di misura robotica, il committente fornisce anche una mappa del deposito (utilizzata dal basicrobot) e le coordinate di tutte le aree notevoli al suo interno.
L'origine del sistema di riferimento è l'angolo in alto a sinistra che coincide con la home.

In seguito la mappa fornita dal committente e le coordinate delle posizioni notevoli.
Legenda:

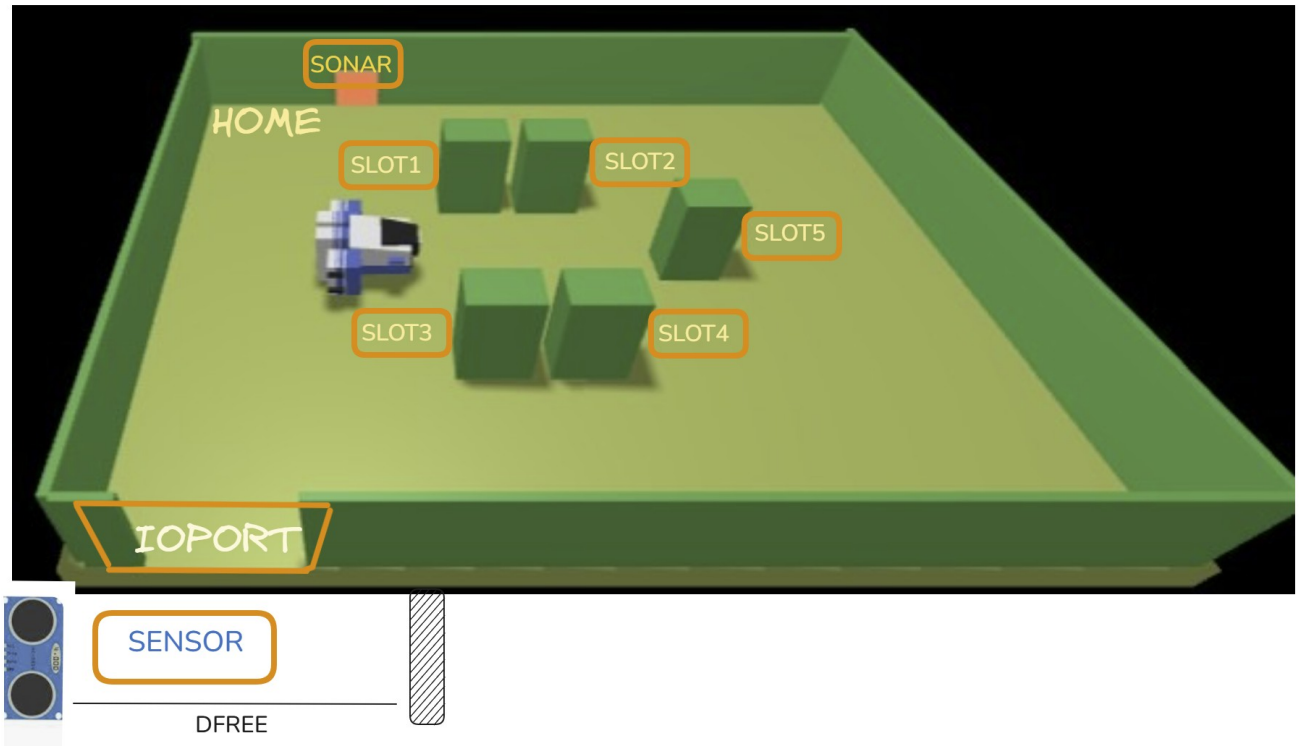
- '1': indicano una cella libera
- 'X': indicano un ostacolo
- 'r': indica la posizione del robot

```
r,  1,  1,  1,  1,  1,  1,
1,  1,  X,  X,  1,  1,  1,
1,  1,  1,  1,  X,  1,  1,
1,  1,  X,  X,  1,  1,  1,
1,  1,  1,  1,  1,  1,  1,
X,  X,  X,  X,  X,  X,  X,
```

- Home: Hold.Home = (0,0)
- Slots: Hold.Slots = {(2,1), (3,1), (2,3), (3,3)}
- Laydown-positions: Hold.LaydownPositions = {(1,1), (1,3), (4,1), (4,3)}
- IO-port: Hold.IoPort = (0,5)
- Pickup-position: Hold.PickupPosition = (0,4)

Indicando nella mappa fornita le coordinate specificate otteniamo la seguente mappa logica:

```
H,  1,  1,  1,  1,  1,  1,
r,  L1, S1, S2, L2, 1,  1,
1,  1,  1,  1,  X,  1,  1,
1,  L3, S3, S4, L4, 1,  1,
P,  1,  1,  1,  1,  1,  1,
IO, X,  X,  X,  X,  X,  X,
```



Il **basicrobot** fornito dal committente è un **attore** esecutore di comandi con cui è possibile governare il DDR all'interno di WEnv.
Il basicrobot modella quindi un DDR e permette di effettuare singole mosse o **sequenze di mosse**, a seguito di messaggi di richiesta.
Più nel dettaglio, il basicrobot incorpora un **planner** grazie al quale è in grado di produrre la sequenza di mosse necessarie per posizionare il DDR a una determinata coordinata, con un preciso direzionamento.

In seguito, i messaggi con cui è possibile interagire con il basicrobot.

```
Dispatch cmd         : cmd(MOVE)
Dispatch end         : end(ARG)

Request step        : step(TIME)
Reply stepdone      : stepdone(V)                 for step
Reply stepfailed    : stepfailed(DURATION, CAUSE) for step

Event  sonardata    : sonar( DISTANCE )
Event obstacle      : obstacle(X)
Event info          : info(X)

Request  doplan      : doplan( PATH, STEPTIME )
Reply doplandone     : doplandone( ARG )    for doplan
Reply doplanfailed   : doplanfailed( ARG )  for doplan

Dispatch setrobotstate: setpos(X,Y,D) //D =up|down!left|right

Request engage         : engage(OWNER, STEPTIME)
Reply   engagedone     : engagedone(ARG)    for engage
Reply   engagerefused  : engagerefused(ARG) for engage

Dispatch disengage     : disengage(ARG)
```

```
    Request checkowner    : checkowner(CALLER)
    Reply checkownerok    : checkownerok(ARG)      for checkowner
    Reply checkownerfailed: checkownerfailed(ARG)  for checkowner

    Event alarm           : alarm(X)
    Dispatch nextmove     : nextmove(M)
    Dispatch nomoremove   : nomoremove(M)

    Dispatch setdirection : dir( D )  //D =up|down!left|right

    //Inglobamento endosimbitico di robotpos
    Request moverobot    :  moverobot(TARGETX, TARGETY)
    Reply moverobotdone  :  moverobotok(ARG)                   for
moverobot
    Reply moverobotfailed:  moverobotfailed(PLANDONE, PLANTODO) for
moverobot

    //Richieste di info
    Request getrobotstate : getrobotstate(ARG)
    Reply robotstate      : robotstate(POS,DIR)  for getrobotstate

    Request getenvmap     : getenvmap(X)
    Reply   envmap        : envmap(MAP)  for getenvmap
```

Vi è un **abstraction gap** tra i requisiti del cargorobot e quelli soddisfatti dal basicrobot; quest'ultimo **non è in grado caricare/scaricare container**.
Il committente ha però specificato che nel sistema da sviluppare sarà sufficiente essere in grado di muovere il DDR virtuale dentro a WEnv, **le operazioni di carico e scarico dei container di cargorobot possono essere ridotte a delle stampe o noop**.
Il basicrobot fornito è quindi sufficiente nonostante non sia in grado di caricare/scaricare container per limitazione dell'ambiente virtuale WEnv.

Si noti a questo proposito la relazione tra i componenti cargorobot e basicrobot.
Cargorobot risulta essere un componente **interfaccia** che permette al resto del sistema di pilotare un DDR robot.
Il basicrobot risulta essere una **implementazione** di questa interfaccia con cui si pilota la specifica tipologia di DDR simulata da WEnv.
In una futura estensione del sistema, se si desidererà sostituire WEnv con un DDR reale, sarà sufficiente progettare un nuovo componente in grado di comandare gli attuatori di questo DDR rispettando l'interfaccia imposta da cargorobot.
Successivamente, bisognerà modificare cargorobot in modo tale che utilizzi questo nuovo componente come implementazione della sua interfaccia al posto di basicrobot.
Il resto del sistema rimarrà invariato ma in questo modo funzionerà anche nel mondo fisico e non solo in quello virtuale.

Per questi motivi, sara fondamentale prevedere nel cargorobot anche le operazioni di carico/scarico dei container anche se non sarà necessario implementarle.

Sonar e Led

Il **Sonar** è un componente attivo che, tenendo traccia delle misurazioni periodiche che effettua, è in grado di rilevare la presenza o l'assenza di un container presso l'IO-port.
In quanto componente attivo è opportuno modellare il sonar con un **attore**.
Il sonar è infine caratterizzato da una costante chiamata **DFREE** che definisce:

- la distanza massima con cui una misurazione può essere interpretata: "come assenza di container" (DFREE/2);
- la distanza massima con cui una misurazione può essere considerata valida (DFREE).

DFREE rappresenta una distanza ed è opportuno modellarla come valore reale maggiore di zero.
Un componente strettamente associato al sonar è il **Led**.
In caso di misurazioni del sonar riconducibili a malfunzionamenti di quest'ultimo, **il sistema deve accendere il led** in modo da notificare la presenza del malfunzionamento.
Questo è un comportamento **passivo** e pertanto modellabile tramite un POJO.
Tuttavia, siccome è presumibile che il led fisico da accendere risieda in un nodo fisico remoto rispetto al resto del sistema, è più opportuno modellare led come un **attore** parte di un sistema distribuito.

Cargoservice

Il **Cargoservice** è il macrocomponente principale che implementa il core-buisness del sistema. Si occupa di orchestrare gli altri componenti del sistema per decidere:

- se accettare o rifiutare una richiesta di carico,
- quale slot libero associare alla richiesta che si sta servendo,
- dove e quando spostare il cargorobot,
- di interrompere tutte le attività quando si accorge di un malfunzionamento del sonar.

Modellare il cargoservice come **attore** è una scelta naturale in quanto oltre ad essere un componente attivo, è anche un componente **reattivo** rispetto agli eventi del sonar.

La **dynamically updated web GUI** è la pagina web che mostra graficamente e in tempo reale, lo stato del deposito.
Si noti che non è previsto di poter visualizzare i container, ne le informazioni relative ai prodotti al loro inerno.
Le informazioni da visualizzare sono dunque:

- io-port libera/occupata
- slot liberi/occupati
- peso raggiunto

Dopo una interazione con il committente si è scoperto che le **richieste di carico** a cargoservice arrivano da un **attore esterno al sistema** non necessariamente umano.
Non è richiesto implementarlo nel nostro sistema.

Macrocomponenti

La scelta di modellare gran parte del sistema attraverso attori consente una notevole flessibilità architetturale, permettendo di distribuire o concentrare facilmente i componenti a seconda delle esigenze.
Decidere il grado di distribuzione del sistema durante lo sprint 0 risulta quindi prematuro, per questo motivo si rimanda questa decisione dopo aver effettuato le analisi del problema nei prossimi sprint.

Fanno eccezione i componenti che modellano un'entità reale, il deployment di quest'ultimi va per forza effettuato sul nodo fisico in cui sono presenti i sensori/attuatori da comandare.
Per questo motivo si può già affermare dallo sprint 0 che sonar e led risiederanno su un nodo fisico distinto rispetto al resto del sistema.
In futuro, anche il componente che comanderà il DDR fisico (sostituendo l'attuale basicrobot) dovrà per gli stessi motivi essere distribuito su un nodo fisico a se stante.

Segue l'elenco dei macrocomponenti software del sistema che non sono da sviluppare:

- productservice,
- basicrobot,
- WEnv.

Segue l'elenco dei macrocomponenti software del sistema da sviluppare:

- cargoservice (orchestratore che comunica con praticamente tutti),
- cargorobot,
- sonar,
- led,
- hold,
- web-gui.

Architettura di Riferimento

Messaggi

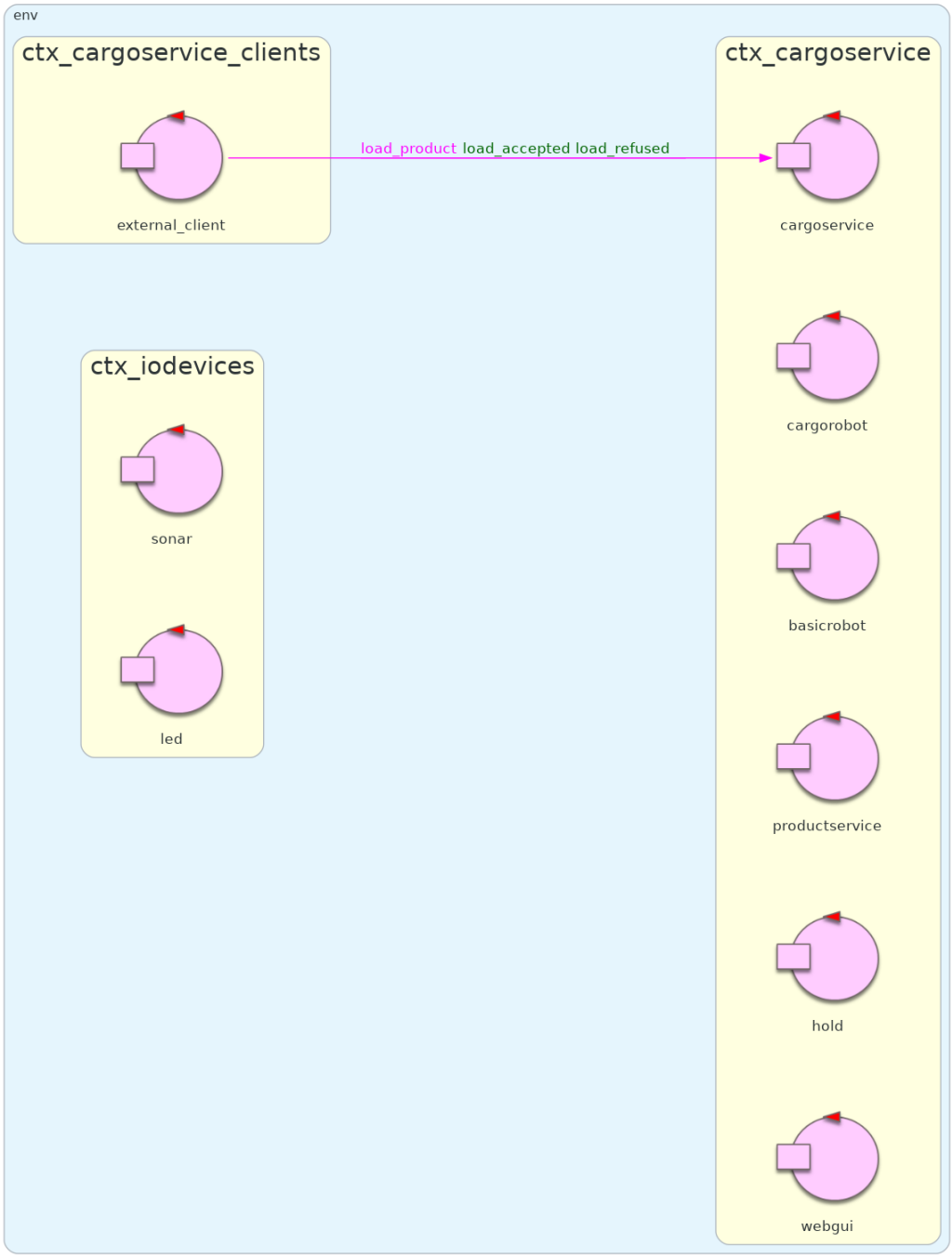
I messaggi costituiscono la base fondamentale della comunicazione nel modello distribuito ad attori adottato.

I requisiti non specificano la maggior parte delle interazioni tra i componenti del sistema, di conseguenza ci si limita definire solamente i messaggi che modellano le interazioni espresse esplicitamente.
Le interazioni rimanenti verranno discusse e modellate durante le fasi di analisi del problema nei successivi sprint.

```
Request load_product  : load_product(PID)
Reply   load_accepted : load_accepted(SLOT) for load_product
Reply   load_refused  : load_refused(CAUSA) for load_product
```

Diagramma dell'Architettura

Il seguente diagramma rappresenta l'architettura iniziale di riferimento per lo sprint 1.


env
ctx_cargoservice_clients
external_client
load_product load_accepted load_refused
ctx_cargoservice
cargoservice
cargorobot
basicrobot
productservice
hold
webgui
ctx_iodevices
sonar
led

arch0Arch

Piano di Test

E' disponibile il riferimento al codice dei test.

Test di Sistema

Il test di sistema è un collaudo interno che in questa prima fase ha il preciso compito di confermare il corretto funzionamento della rete e delle interazioni via messaggi attraverso di essa dei vari componenti (mock in questa prima fase).
Questo obbliga, inevitabilmente, a riflettere sulle interazioni tra i componenti del sistema, prima di realizzarli.

Scenario di test 1: Richiesta di intervento di carico accettata da cargoservice

```
    @Test
    public void testLoadRequestAccepted() throws Exception {
        //Costruzione di richiesta con PID valido.

        String requestStr = CommUtils.buildRequest("tester",
                "load_product", "load_product(10)",
                "cargoservice").toString();

        System.out.println("Richiesta: " + requestStr);

        //Risposta accettata perchè peso legato al PID inferiore di MaxLoad
        String response = conn.request(requestStr);

        System.out.println("Risposta: " + response); // Risposta contenente
lo slot libero dove posizionare il container

        //Verifica che sia stata accettata
        assertTrue("TEST: richiesta accettata",
                 response.contains("load_accepted"));
    }
```

Scenario di test 2: Doppia richiesta di intervento di carico accettata

```
    @Test
    public void testDoubleLoadRequest() throws Exception {
    // Costruzione della prima richiesta con PID valido.
    String request1 = CommUtils.buildRequest("tester",
            "load_product", "load_product(9)",
            "cargoservice").toString();

    //Risposta accettata perchè il peso legato al PID è inferiore di
MaxLoad

    String response1 = conn.request(request1);
    System.out.println("Risposta: " + response1); // Risposta contenente lo
slot libero dove posizionare il container
```

```
    assertTrue("TEST: Prima richiesta accettata",
             response1.contains("load_accepted"));

   // Costruzione della seconda richiesta con PID valido.
    String request2 = CommUtils.buildRequest("tester",
            "load_product", "load_product(10)",
            "cargoservice").toString();

    //Risposta positiva perchè il peso legato al PID è inferiore di MaxLoad

    String response2 = conn.request(request2);
    System.out.println("Risposta: " + response2); // Risposta contenente lo
slot libero dove posizionare il container

    assertTrue("TEST: Seconda richiesta accettata",
             response2.contains("load_accepted"));

    }
```

Scenario di test 3: Richiesta di intervento di carico rifiutata a causa della mancanza di slot libero

```
@Test
    public void testLoadRequestDeniedNoAvailableSlots() throws Exception {
    //Costruisci la richiesta con un PID valido.
    String requestStr = CommUtils.buildRequest("tester",
            "load_product", "load_product(20)",
            "cargoservice").toString();

    System.out.println("Richiesta: " + requestStr);

    //Risposta negativa a causa della non presenza di slot liberi.
    String response = conn.request(requestStr);

    System.out.println("Risposta: " + response); // Risposta contenente la
causa del rifiuto dell'intervento di carico

    //Verifica che sia rifiutata per mancanza di slot
    assertTrue("TEST: richiesta rifiutata per slot pieni",
            response.contains("load_refused") &&
            response.contains("no_slot_liberi"));
    }
```

Scenario di test 4: Richiesta di intervento di carico rifiutata a causa del peso eccessivo del container

```
 @Test
    public void testLoadRequestDeniedByWeight() throws Exception {
        //Costruisci la richiesta con PID valido.
        String requestStr = CommUtils.buildRequest("tester",
```

```
                "load_product", "load_product(11)",
                "cargoservice").toString();

        System.out.println("Richiesta: " + requestStr);

        //Risposta negativa perchè il peso legato al PID è superiore a
MaxLoad
        String response = conn.request(requestStr);

        System.out.println("Risposta: " + response); //Risposta contenente
la causa del rifiuto dell'intervento di carico

        // 3. Verifica che sia stata rifiutata per il peso eccessivo
        assertTrue("TEST: richiesta rifiutata",
                response.contains("load_refused") &&
                response.contains("overweight"));
    }
```

Piano di Lavoro

Oltre a questo sprint 0 iniziale, dedicato all'impostazione del progetto, nel nostro processo Scrum abbiamo previsto tre sprint operativi:

1. Sprint 1
    - cargoservice (core business del sistema)
    - cargorobot
2. Sprint 2
    - sonar
    - hold
3. Sprint 3
    - led
    - web-gui

Numero sprint	Data inizio (indicativa)	Data fine (indicativa)	Lavoro Stimato Totale (h)
Sprint 1	07/07/2025	11/07/2025	30
Sprint 2	14/07/2025	18/07/2025	20
Sprint 3	21/07/2025	25/08/2025	20

La pianificazione temporale costituisce un riferimento per il team.
Sono comunque contemplate variazioni, nel limite del ragionevole, purché non compromettano il ritmo generale del lavoro.
Eventuali modifiche significative potranno essere apportate solo in presenza di esigenze straordinarie, cambiamenti rilevanti durante il percorso progettuale, o situazioni tali da non poter essere ignorate.

La presentazione finale è prevista, indicativamente, in data 31 luglio 2025.

Team di Lavoro e Attività Specifiche



Bertozzi Pietro	Koltraka Kevin	La Rocca Andrea

Tutti e 3 i componenti del team hanno partecipato attivamente a tutte le fasi del progresso dello sprint, dando il loro contributo sia in forma di conoscenze, che di ore di lavoro.