Sprint 0 V1

Indice

Obiettivi

In questo sprint0 i nostri obiettivi sono di formalizzare i requisiti forniti dal committente e definire il nostro problema, in questa formalizzazione saremo in grado di definire problemi che saranno poi gestiti e sviluppati in sprint da eseguire eventualmente anche in parallelo, improntare le componenti della nostra archiettura (macrocomponenti principali & interazioni tra loro sotto forma di messaggi).

Requisiti del commitente

Requisiti del commitente

Analisi dei requisiti

Hold

È la stiva della nave, cioè l'area di lavoro in cui si muove il nostro robot e si trovano i container. In questo progetto la modelliamo come una zona rettangolare(idealmente una matrice in cui ogni posizione raffigura una cella) con degli slot e una porta di ingresso/uscita (IOPort). Il modello di questa permetterà al cargoservice di eseguire le operazioni richieste dal committente.

DDR - Differential Drive

È il robot a guida differenziale (Differential Drive Robot) incaricato di spostare i container dentro la stiva e piazzarli nello slot assegnato. Dopo il lavoro torna sempre alla sua posizione "HOME". Cargorobot gestirà le azioni del DDR affinchè rispecchi i requisiti del committente.

Products

Sono i beni/merci che devono essere caricati sulla nave. Ogni prodotto viene messo in un container di dimensioni prefissate e registrato in un sistema.

Weight

Il peso del prodotto/container. Serve per verificare che non venga superato il limite massimo di carico della nave (`MaxLoad`).

PID

PID o Product Id è l'identificativo di prodotto restituito da productservice. E' un numero naturale maggiore di 0

CargoService

È il servizio software che si occupa di gestire l'hold, decidendo dove e quando muovere i prodotti e gestire gli errori del sonar.

Io-port

È la porta di ingresso/uscita della stiva. Davanti a questa porta c'è un sensore sonar che rileva se un container è presente. È il punto dove il prodotto viene consegnato prima che il robot lo carichi. Si trova sul perimetro della stiva.

Componenti fornite dal committente

Si elencano di seguito le componenti software fornite dal committente

BasicRobot

Componente che governa il movimento del DDR-robot all'interno della mappa. Non è a conoscenza della tecnologia con il quale il robot è stato implementato (potrebbe trattarsi di un robot fisico come di uno virtuale). Dato un punto di arrivo è capace far raggiungere il DDR quel determinato punto. Pertanto conosce l'Hold e i suoi vincoli. Comunica tramite messaggi.

SonarLed2025

Software per la misurazione della distanza dal sonar (componente hardware) e per accendere un LED

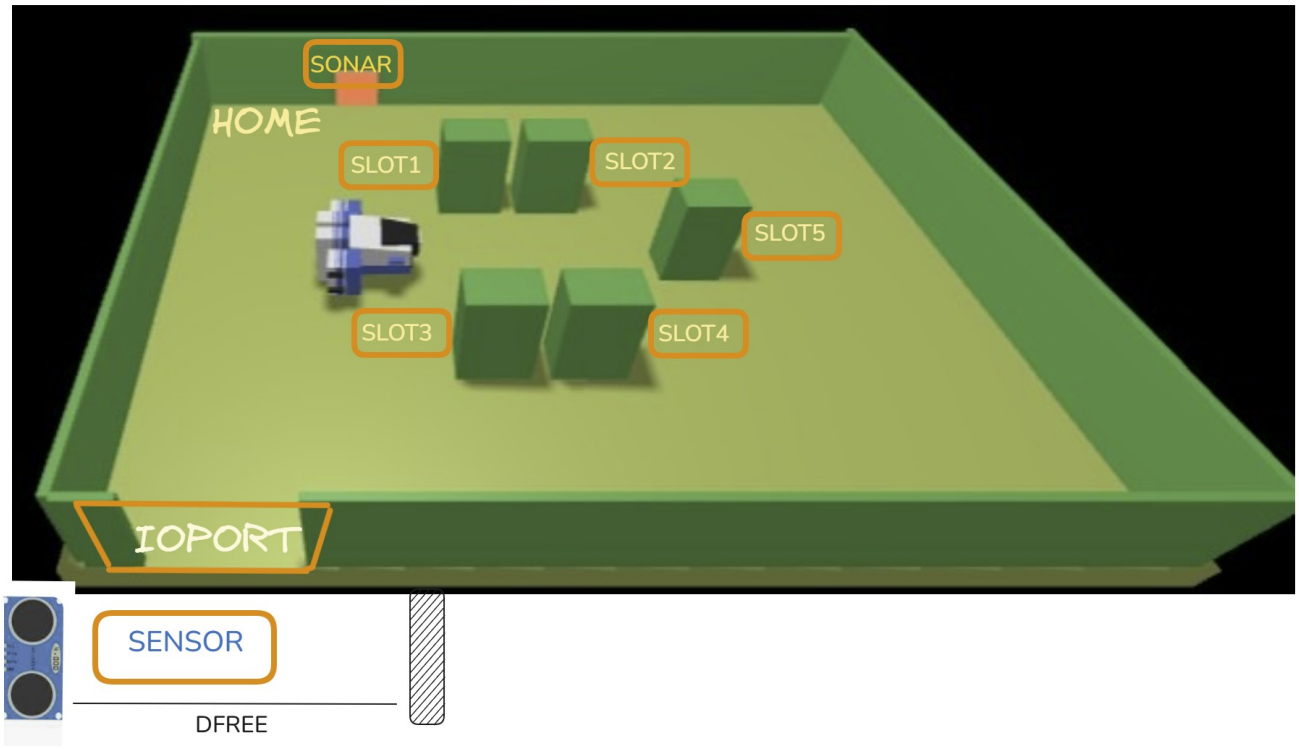
WENV

WENV è un ambiente di simulazione software ("Web Environment") usato per testare il sistema, mostrare la stiva e i movimenti del robot tramite un'interfaccia grafica web.

Web-gui

Interfaccia grafica web che consente la supervisione dello stato dell'hold,quindi dei singoli slot, del sonar e del led. Deve essere aggiornata dinamicamente (possibile utilizzo di eventi)

Area di Lavoro



Plain Old Java Objects (POJO)

I POJO sono essenziali per costruire il modello di dominio in DDD.
Sono usati per:
• Rappresentare entità con identità persistente • Definire value object immutabili • Formare aggregati, unità coerenti per la consistenza dei dati
• Incapsulare logica di business passiva e stateless

I POJO sono passivi, non gestiscono il loro stato in autonomia nel tempo in risposta a eventi esterni e non hanno code di messaggi proprie

Attori

Il modello ad Attori si basa su entità autonome che comunicano tramite messaggi. Un Attore Qak è un componente attivo con un proprio flusso di controllo autonomo e uno stato interno.

Le loro caratteristiche principali includono:

• **Comunicazione a messaggi**: Interagiscono esclusivamente inviando e ricevendo messaggi (Dispatch, Request, Reply, Event), promuovendo isolamento e resilienza
• **Coda di messaggi locale**: Ogni Attore ha una coda (msgQueue) per processare i messaggi sequenzialmente, gestendo naturalmente l'esigenza di non elaborare più richieste contemporaneamente
• **Comportamento come FSM**: Il loro comportamento può essere modellato come una macchina a stati finiti (FSM)
• **Adatti a sistemi distribuiti e microservizi**: Il modello Qak è specificamente pensato per la progettazione di prototipi di sistemi distribuiti, con attori che si comportano come FSM, strettamente correlati all'architettura a microservizi
• **Qak come DSL**: Il linguaggio Qak è un Domain Specific Language (DSL) che fornisce un alto livello di astrazione per definire modelli eseguibili di sistemi basati su attori, aiutando a colmare l'abstraction gap

• **Raggruppamento in contesti**: Gli attori sono raggruppati in contesti che gestiscono le interazioni di rete tramite protocolli come TCP, CoAP, MQTT

Macrocomponenti

Ora che sono stati definiti i requisiti ed i concetti fondamentali che utilizzeremo iniziamo a ragionare su come potremmo modellarne i macrocomponenti.

Cargorobot

Il cargorobot sarà un attore in quanto sarà incaricato di reagire ad eventi come l'arrivo di un container. Esso aggiungerà al basicrobot la possibilità di caricare e scaricare container negli appositi slot dell'Hold.

CargoService

Il CargoService rappresenta il nucleo della logica di business del sistema, allineandosi al concetto di "Gestione Carico" (Loading Management) identificato come Bounded Context

Le sue responsabilità principali sono:

• **Gestire le richieste di carico**: Riceve le richieste esterne per il carico di un prodotto e ne gestisce l'intero ciclo di vita.
• **Orchestrare il processo**: Decide se accettare o rifiutare una richiesta dopo aver eseguito le necessarie validazioni, come il controllo del peso massimo e la disponibilità di slot nella stiva
• **Coordinare gli altri componenti**: Interagisce con gli altri macro-componenti per portare a termine il processo. Richiede informazioni sul prodotto (come il peso) a productservice, verifica la disponibilità di slot nella stiva, e comanda al cargorobot di eseguire il caricamento.

Verrà modellato come un attore in quanto la comunicazione avverà con elementi eterogenei e pertanto riteniamo sia piu congruo lo scambio di messaggi come metodo comunicativo. Inoltre prevediamo l'utilizzo di eventi quindi il concetto di attore QAK ci potrebbe tornare utile.

Sonar

Il sonar si interfaccia con il sensore a ultrasuoni fisico, verrà montato in un nodo fisico separato (raspberry) e pertanto,facendo parte di un sistema distribuito, potremmo modellarlo come attore anche se il suo ruolo sarà di emettere messaggi in modo passivo quindi demandiamo questa decisione agli sprint futuri.

Led

Il led anche esso si trova su un nodo fisico separato e per lo stesso motivo del sonar demanderemo la decisione sulla tipologia di modellazione allo sprint futuro. Di seguito le sue responsabilità principali:

• **Fornire feedback visivo**: La sua funzione primaria è quella di segnalare visivamente la presenza di container.
• **Esporre un'interfaccia di controllo**: Deve offrire un'interfaccia semplice per essere comandato da altri componenti, accettando messaggi come `turnOn` e `turnOff`.

Web-gui

Interfaccia grafica web che consente la supervisione dello stato dell'hold,quindi dei singoli slot, del sonar e del led. Deve essere aggiornata dinamicamente (possibile utilizzo di eventi) per cui prevediamo la sua modellazione come attore. Inoltre potremmo così permettere a questa interfaccia di essere un componente attivo fornendogli dei bottoni con cui modificare lo stato dell'hold.

Architettura di Riferimento

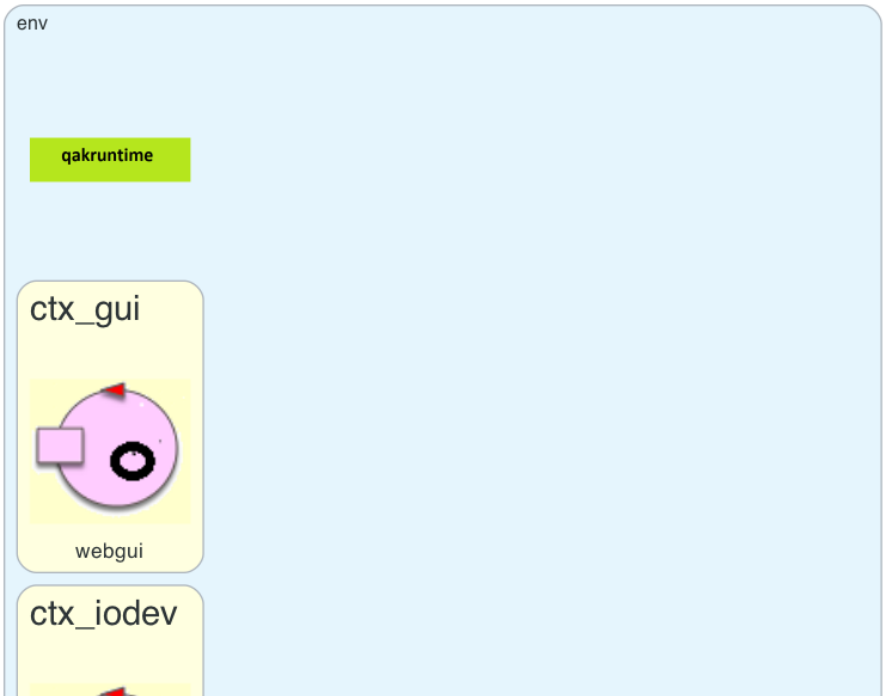
Modello di comunicazione a Messaggi

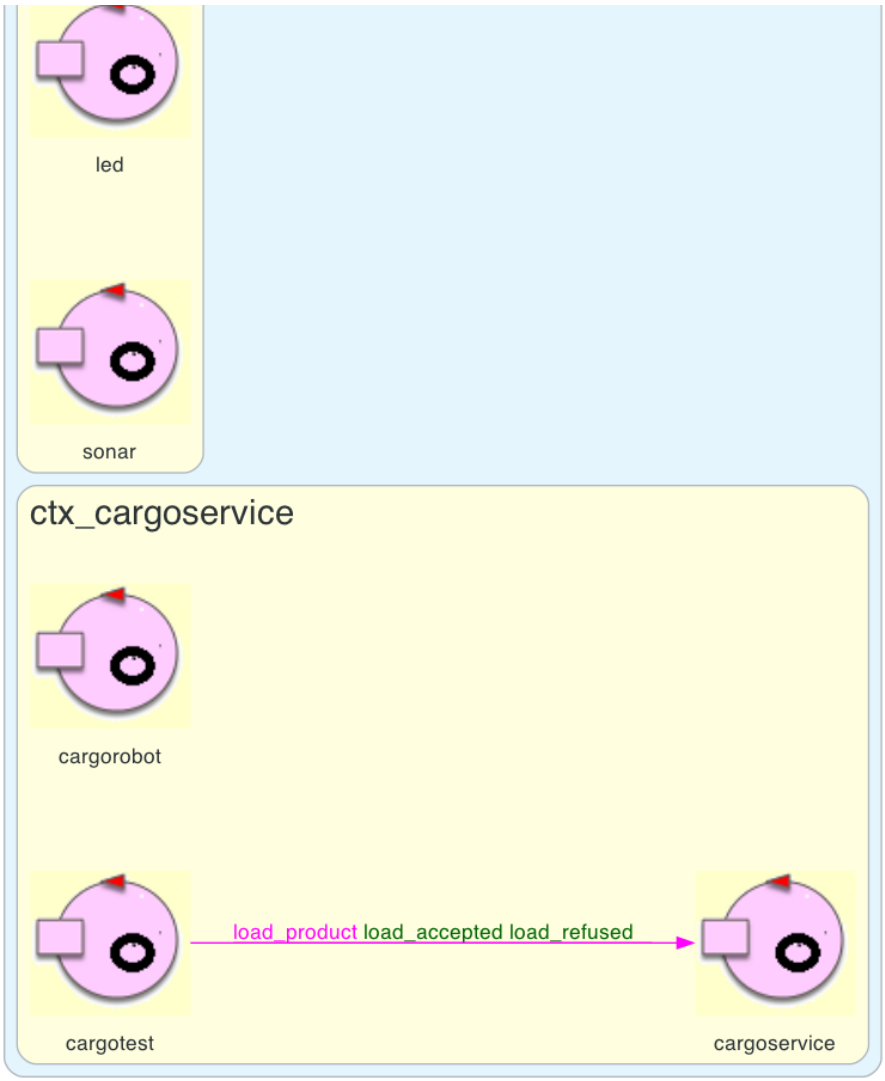
Il modello ad attori sfrutta la comunicazione tramite messaggi e dai requisiti forniti siamo in grado di comprendere alcune delle interazioni che avvengo tra gli attori. Si indicano di seguito i messaggi che sono in grado di scambiarsi tra di loro.

- `Request load_product : load_product(PID)` richiesta di carico di un prodotto con PID
- `Reply load_accepted : load_accepted(SLOT) for load_product` restituisce lo slot assegnato
- `Reply load_refused : load_refused(CAUSA) for load_product` ritorna la causa del mancato carico

Serviranno successive decisioni per la modellazione e l'implementazioni di messaggi tra attori per ulteriori funzionalità.

Schema dell'architettura





sprint0Arch

Piano di Test

In questa prima fase i test servono a controllare che i prototipi dei componenti interagiscano come richiesto dal committente.

- tentativo accettato di carico
- tentativo rifiutato di carico per troppo peso
- tentativo rifiutato per mancanza di slot

```
State richiesta {
    println("[cargotest] Invia una nuova richiesta") color yellow
    // Invio della richiesta
    request cargoservice -m load_product:load_product(1)
    request cargoservice -m load_product:load_product(1)
```

```
    request cargoservice -m load_product:load_product(1)
}
Goto waiting_for_response

State waiting_for_response {
}

Transition t0
    whenReply load_accepted -> loadAccepted
    whenReply load_refused  -> loadRefused

State loadAccepted {
    println("[cargotest] risposta arrivata") color blue

    onMsg(load_accepted : load_accepted(SLOT)) {
        [# val Msg = payloadArg(0).toInt() #]
        println("[cargotest] Richiesta accettata, slot n. $Msg ") color
yellow
    }
}
Goto waiting_for_response

State loadRefused {
    onMsg(load_refused : load_refused(CAUSA)) {
        [#
        var Msg = payloadArg(0)
        #]
        println("[cargotest] Richiesta rifiutata causa : $Msg ") color
yellow
    }
}
```

Piano di Lavoro

Successivi allo sprint0 si distinuguono i seguenti sprint operativi del nostro processo Scrum

1. Sprint 1 (30h)
    - CargoService (core buisness del sistema)
    - Cargorobot
2. Sprint 2(20h)
    - Sonar
    - Led
3. Sprint 3(10h)
    - Web Gui