# SPRINT 0 – TemaFinale25

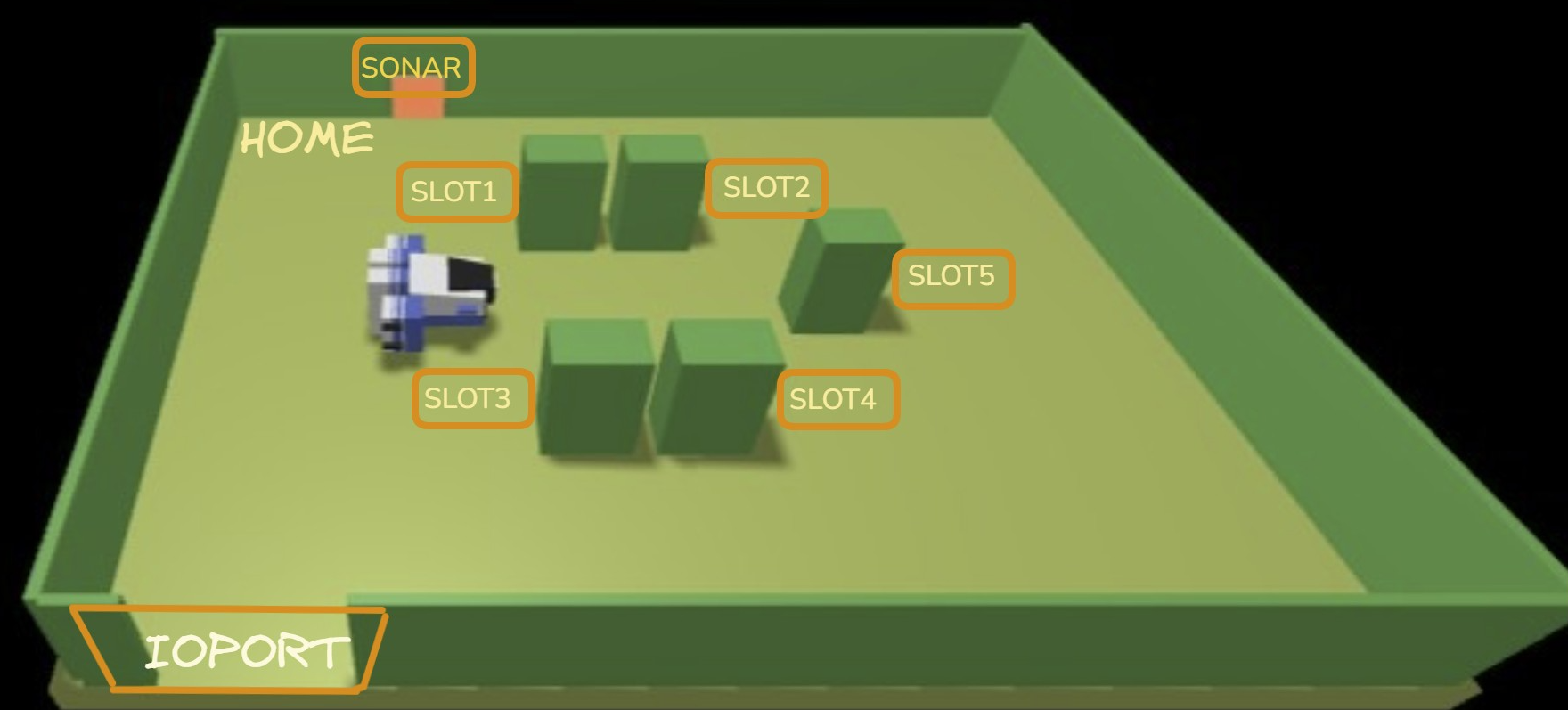
## Link ai requisiti del committente

Requisiti TemaFinale25: <https://github.com/anatali/issLab2025/blob/main/iss25Material/docs/_build/html/TemaFinale25.html>

A *Maritime Cargo shipping company* (fron now on, simply *company*) intends to automate the operations of load of freight in the ship’s cargo hold (or simply *hold*). To this end, the company plans to employ a *Differential Drive Robot* (from now, called *cargorobot*) for the loading of goods (named *products*) in the ship’s hold.

The products to be loaded must be placed in a container of predefined dimensions and registered, by specifying its *weight*, within a database, by using a proper service (*productservice*). After the registration, the *productservice* returns a **unique product identifier** as a natural number **PID,** **PID>0**.

The hold is a rectangular, flat area with an Input/Output port (*IOPort*). The area provides **4** **slots** for the product containers.

  
In the picture above:

* The **slots** depict the *hold storage areas*, when they are occupied by *product containes*
* The **slots5** area is permanentely occupied, while the other slots are initially empty
* The **sensor** put in front of the *IOPort* is a sonar used to detect the presence of a product container, when it measures a distance **D**, such that **D** **<** **DFREE/2**, during a reasonable time (e.g. **3** secs).

REQUISITI SPECIFICATI DAL COMMITTENTE:

The company asks us to build a software systems (named *cargoservice*) that:

1. is able to receive the **request to load** on the cargo a product container already registered in the *productservice*.

The request is rejected when:

* + the product-weight is evaluated too high, since the ship can carry a maximum load of **MaxLoad>0**  **kg**.
  + the hold is already full, i.e. the **4** **slots** are alrready occupied.

If the request is accepted, the *cargoservice* associates a slot to the product **PID** and returns the name of the reserved slot. Afttwerds, it waits that the product container is delivered to the *ioport*. In the meantime, other requests are not elaborated.

1. is able to detect (by means of the *sonar* **sensor**) the presence of the product container at the *ioport*
2. is able to ensure that the product container is placed by the *cargorobot* within its reserved slot. At the end of the work:
   * the *cargorobot* should returns to its HOME location.
   * the *cargoservice* can process another *load-request*
3. is able to show the current state of the *hold*, by means of a dynamically updated **web-gui**.
4. interrupts any activity and turns on a led if the *sonar sensor* measures a distance **D** **>** **DFREE** for at least **3** secs (perhaps a sonar failure). The service continues its activities as soon as the sonar measures a distance **D** **<=** **DFREE**.

## GOAL dello Sprint 0

Costruire un Sistema logico di riferimento.

Evidenziare su quanti nodi computazionali diversi deve essere distribuito.

Distinguere i Boundary Context.

Distinguere i macro-componenti(hardware/software) che occorre sviluppare.

Fornire un modello delle macro-parti del sistema, specificando quali componenti sono fornite dal committente e quelle da sviluppare.

Fornire un quadro architetturale complessivo dal quale dedurre un possibile piano di lavoro.

## Costruire un sistema logico di riferimento:

Il sistema logico è la base concettuale su cui costruire l’architettura del progetto.

Serve a identificare gli elementi chiave e a stabilire le relazioni tra loro in modo chiaro e strutturato.

Definiamo a tal scopo:

### Gli attori :

Essi sono i componenti attivi che influenzano il comportamento del sistema.

Possono essere componenti esterne al sistema , ovvero, esseri umani che interagiscono con esso:

* **worker :** colui che inserisce il prodotto nel contenitore

Attori software esterni:

* **company:** la compagnia che si occupa di inviare le richieste di carico.

Sistemi fisici o sensori:

* **cargorobot** : il robot che esegue il trasporto e il posizionamento dei prodotti.
* **sonar :** registra la presenza/assenza di un contenitore all’IOPort.

### Le interazioni tra gli attori:

* **registrazione prodotto**: il prodotto viene registrato tramite il product service che assegna un identificatore univoco (PID>0) e ne memorizza le caratteristiche (il peso). Le informazioni vengono caricate sul Database.

Si suppone che dopo aver registrato il prodotto, il worker ponga il prodotto sulla IO-Port.

* **ricezione / richiesta di carico** : il cargoservice riceve una richiesta di carico dalla company.
* **controllo richiesta di carico:**

il cargoservice rifiuta la richiesta di carico SE:

* il prodotto non è stato registrato dal productservice
* il peso supera la capacità della stiva
* la stiva è già piena.

Se viene accettata viene assegnato un PID (un product ID) al prodotto e viene comunicato al robot il nome dello slot in cui caricare il prodotto.

* **Attesa del prodotto**:dopo aver riservato uno slot, il cargoservice attende che il sonar registri la presenza del prodotto da caricare all’IO-Port.
* **Caricamento del prodotto**: una volta rilevata la presenza del prodotto, il cargoservice riceve il PID del prodotto e lo slot su cui caricarlo. Manda quindi al robot indicazioni affinchè lo prelevi dal’IO-Port e lo porti allo slot assegnato.
* **ritorno alla posizione HOME:** il robot torna al punto HOME se un’operazione viene annullata o ha terminato tutte le operazioni.
* **Visualizzazione stato della stiva:** il cargoservice aggiorna dinamicamente lo stato della stiva, visibie tramite la webgui.

### Vincoli interni :

* la richiesta di carico va rifiutata se il peso eccederebbe il massimo carico della stiva o se la stiva è già piena (i 4 slot sono tutti occupati).
* Non si possono richiedere prodotti non registrati nel DB.
* La risoluzione della richiesta non può eccedere un certo intervallo di tempo.
* il PID deve essere unico.
* I posti negli slot possono essere riutilizzati solo una volta liberati.
* Un elemento può essere inserito nella stiva solo se non supera i limiti di peso e di spazio.

### Vincoli esterni:

* dimensioni del contenitore
* peso massimo che il contenitore può sopportare
* dimensioni stiva
* numero massimo di posti negli slot

## Evidenziare su quanti nodi computazionali diversi deve essere distribuito:

Un nodo computazionale è un’unità di calcolo che esegue operazioni o gestisce dati all’interno del sistema distribuito. È quindi un componente autonomo e interagisce da sé con altri elementi.

|  |  |  |
| --- | --- | --- |
| Nodo | Tipo | Funzione |
| cargoservice | Microservizio | Gestione richieste di carico |
| productservice | Microservizio | Registrazione e gestione dei prodotti |
| sonarservice | Microservizio | Rilevamento presenza/assenza container |
| webgui | GUI | Visualizzazione stato stiva |

Boundary Context:

è un concetto fondamentale del Domain-Driven Design (DDD).  
Rappresenta una sezione specifica di un sistema software o dominio in cui il modello è valido, con un linguaggio univoco e regole ben definite.

**productservice**: si occupa della registrazione dei prodotti e della generazione del PID.

**cargoservice**: gestisce le richieste di carico, verifica i vincoli, assegna gli slot e coordina le operazioni del robot.

**sonarservice**: fornisce l’informazione sulla presenza presso l’IO-Port.

**webgui**: mostra dinamicamente lo stato attuale della stiva.

## Distinguere i macro-componenti(hardware/software) che occorre sviluppare:

### Componenti hardware forniti:

|  |  |  |
| --- | --- | --- |
| Componente | Tipo | Descrizione |
| DDRrobot | Microservizio | Robot mobile differenziale per il trasporto di container. È fornito sia l’harware che un software con cui controllarlo. |
| Sonar | Harware | Sensore di distanza per rilevare la presenza di container |
| IOPort | Hardware | Punto fisico di ingresso/uscita container |
| Slot (1,2,3,4,5) | Hardware | Slot che contengono i prodotti (il quinto non è disponibile, è sempre pieno). |
| ProductService | Microservizio | Si occupa di parlare con il database su cui sono stati registrati i prodotti di cui conosce il PID e il peso. |

### Componenti da sviluppare:

|  |  |  |
| --- | --- | --- |
| Componente | Tipo | Descrizione |
| webgui | GUI | È l’interfaccia attraverso cui si può osservare la stiva. |
| cargoservice | Microservizio | Gestisce le richieste di carico, di assegnazione di slot, coordina i robot. |
| sonarservice | Microservizio | segnala l’avvenuta consegna di un carico all’IOPort e di conseguenza informa il cargoservice. |

## Fornire un quadro architetturale complessivo:

L’architettura del sistema prevede quattro componenti software principali — **CargoService**, **ProductService**, **SonarService** e **WebGUI** — che collaborano con elementi hardware quali robot, sonar, IOPort e stiva, seguendo una logica distribuita ma modellabile.

Tutti i componenti sono implementati come microservizi autonomi che comunicano tra loro mediante messaggi ed eventi.

L’interfaccia **WebGUI** permette di visualizzare lo stato aggiornato della stiva in tempo reale.

La compagnia invia a **CargoService** una richiesta di carico, che viene inoltrata a **ProductService** per verificarne l’esistenza e ottenere il peso del prodotto. Successivamente, **CargoService** effettua ulteriori controlli sul carico, verificando che il peso complessivo sia entro i limiti supportati dal robot e che siano disponibili slot liberi nella stiva.

Se tutti i controlli hanno esito positivo e il **SonarService** conferma la presenza del prodotto all’IOPort, **CargoService** invia al robot il comando di trasporto per spostare il prodotto dall’IOPort allo slot assegnato.

Lo stato della stiva viene aggiornato periodicamente e mostrato tramite la **WebGUI**.

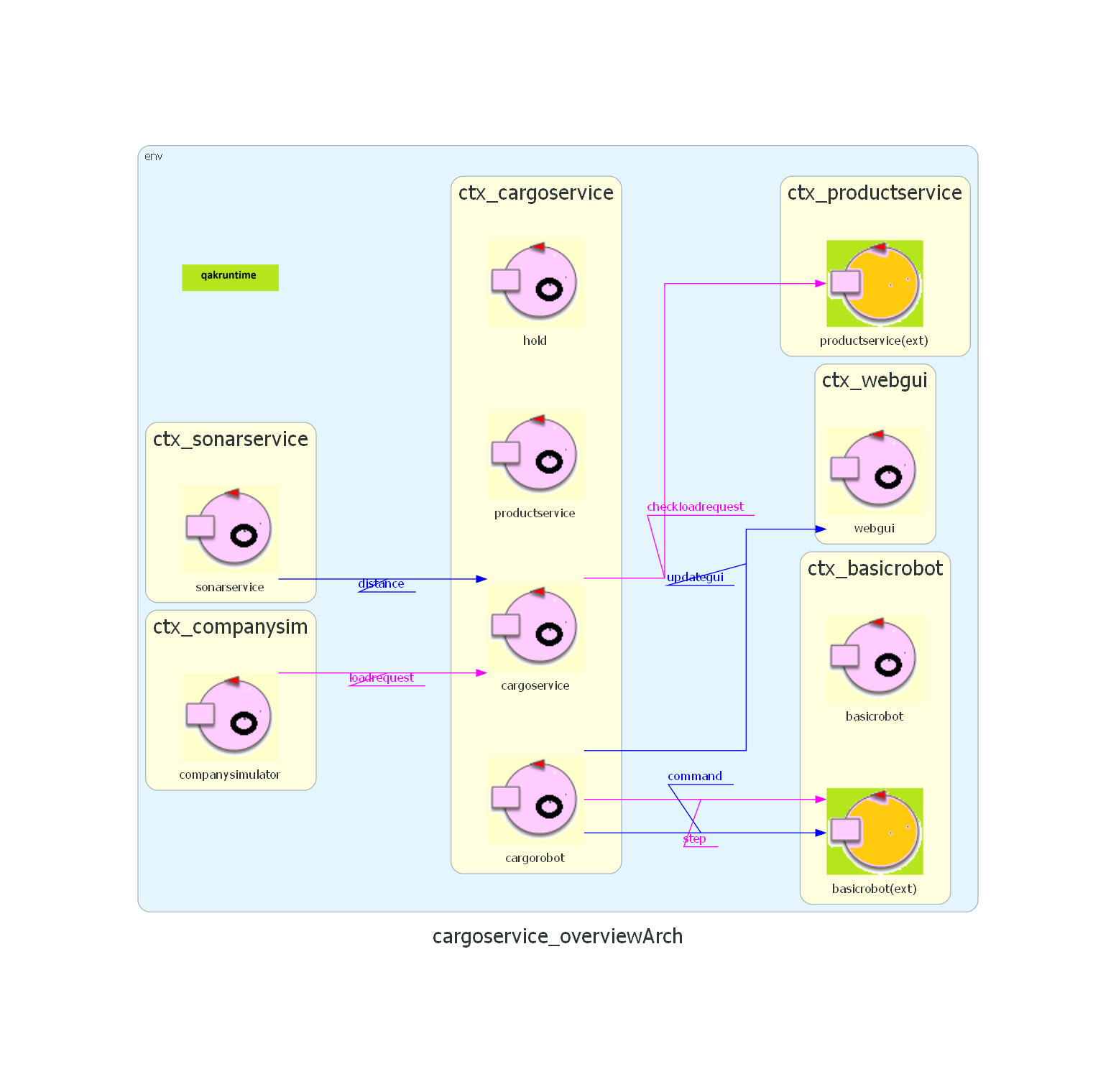
Al termine del trasporto, il robot ritorna automaticamente alla posizione di HOME di default.

**Modello dei messaggi** :

Si ritiene utile fornire un modello che mostri lo scambio di informazioni tramite messaggi che avviene tra i bounded contexts evidenziati precedentemente.

Per crearlo è stato utilizzato il linguaggio qak [(overview del linguaggio QAK )](../documentazione_qak/QakActors25Intro.html)

Usare qak è conveniente per modellare i messaggi perché offre un ambiente concettualmente adatto alla comunicazione tra attori, è pensato per il disegno e la verifica di sistemi distribuiti, e permette di simulare rapidamente il comportamento dei componenti attraverso automi a stati finiti, prima della codifica vera e propria.



## Possibile piano di lavoro :

ATTENZIONE: per ogni fase si intende presentare i risultati al committente.

## fase 0: analisi dei requisiti e modello concettuale.

* Analisi formale dei requisiti del committente (mantenendo la formulazione originale).
* Identificazione degli attori, bounded contexts, nodi computazionali.
* Distinzione tra componenti hardware e software.
* Descrizione delle interazioni tra componenti.
* Produzione del **quadro architetturale complessivo**.

**OUTPUT** : documento sprint0

**COMPLETATA.**

## fase 1: definizione tecnica e setup d’ambiente

* Studio tecnico dei requisiti
* Definizione dei bounded context
* Modellazione iniziale in QAK
* Mappa dei messaggi/eventi tra i servizi
* Scelta del formato di comunicazione (es. JSON, MQTT, HTTP)

**OUTPUT**: documento sprint1, modello iniziale

## fase 2: sviluppo dei singoli microservizi

## fase 2.1: sviluppo del microservizio sonarservice

* Definizione delle API/eventi
* Implementazione logica
* Test unitari e di integrazione
* Simulazione con robot virtuale

OUTPUT : documento sprint2.1, codice del microservizio sonarservice

## fase 2.2: sviluppo del microservizio cargoservice

* Definizione delle API/eventi
* Implementazione logica
* Test unitari e di integrazione
* Simulazione con robot virtuale

OUTPUT: documento sprint2.2, codice del microservizio cargoservice

## fase 2.3: sviluppo della webgui

* Definizione delle API/eventi
* Implementazione logica
* Test unitari e di integrazione
* Simulazione con robot virtuale

OUTPUT: documento sprint2.3, codice della webgui

## fase 3: integrazione e test del sistema

* Integrazione dei microservizi
* Simulazione completa del flusso di carico
* Test con casi limite (peso massimo, stiva piena, errore sonar)
* Validazione con scenari reali

## fase 4: deployment e documentazione

* Deployment su ambiente distribuito (es. Docker, Raspberry, VM)
* Documentazione tecnica e utente
* Manuale d’uso della webgui
* Presentazione finale

## Membri del team di sviluppo:

[Silvia Angela Sveva Carollo](https://github.com/SvevaNullBologna)

Il tempo previsto è di 140h/u