chronicles_sprint0_v4

II team: Diego Bruno, Marco Crisafulli, Sebastiano Giannitti
Link repository Github

Introduction

Lo scopo di questo sprint, e di questo documento, è formalizzare i singoli termini del testo fornito da company e anche quello di fornire una prima visione di insieme del sistema da realizzare.
Come goal di questo sprint ci poniamo:

- Fornire una panoramica dei componenti già forniti dal committente e dei componenti da sviluppare.
- Produrre un piano di test preliminare da proporre e validare insieme al committente
- Chiarire eventuali dubbi relativi ai requisiti

Requirements

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 already occupied.*
2. *If the request is accepted, the cargoservice associates a slot to the product **PID** and returns the name of the reserved slot. Afterwards, it waits that the product container is delivered to the ioport. In the meantime, other requests are not elaborated.*
3. *is able to detect (by means of the sonar **sensor**) the presence of the product container at the ioport*
4. *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*
5. *is able to show the current state of the hold, by means of a dynamically updated **web-gui**.*
6. *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**.*

Requirement analysis

Abstraction Gap

I linguaggi di programmazione esistenti mostrano una limitazione: non sono progettati per modellare in modo nativo la comunicazione tra servizi tramite **request-reply** e certi costrutti come ad esempio il **servizio** non esistono nei linguaggi tipo java. Questo rende complessa la traduzione diretta dei requisiti in un'architettura che sfrutti appieno le interazioni.

Per superare questa restrizione la nostra *software house* si avvale di **QAK**, un linguaggio di modellazione dedicato che fornisce gli strumenti necessari a colmare l'***abstraction gap*** tra la specifica dei requisiti e la loro implementazione.

Un **attore QAK** è un componente attivo che funge da unità fondamentale per la modellazione dei nostri sistemi. Le sue caratteristiche chiave sono:

- **Contesto:** Nasce, vive e muore in un contesto che può essere condiviso con molti altri attori.
- **Identità:** Possiede un **nome univoco** all'interno dell'intero sistema.
- **Autonomia:** È logicamente **attivo**, dotato di un flusso di controllo autonomo.
- **Comunicazione:** È capace di inviare messaggi a qualsiasi altro attore (incluso sé stesso), di cui conosce il nome.
- **Elaborazione:** È capace di eseguire elaborazioni autonome e/o in risposta ai messaggi ricevuti.
- **Archivio:** È dotato di una sua **coda locale (msgQueue)** in cui sono depositati i messaggi a lui inviati, garantendo un'elaborazione ordinata e asincrona.

Da una prima analisi dei requisiti possiamo identificare la tipologia dei componenti necessari:

Componente	**Tipologia**	
cargoservice	Attore	Deve essere in grado di ricevere una **richiesta** e di valutarla. Deve rispondere con esito positivo o negativo.
productservice	Attore	Necessaria comunicazione con il cargoservice, per verificare la presenza del prodotto in database. Fornito dal committente.
product	Classe	Ogni prodotto ha un nome, un peso e un PID associati. Responsabilita’ di **productservice**
hold	Necessaria analisi del problema	Composta da 4 slot disponibili, ioport ed HOME, ma dai requisiti non è esplicito come formalizzarla.
slot	Necessaria analisi del	Spazio dedicato al carico dei prodotti.

	problema	
ioport	Necessaria analisi del problema	Spazio dedicato alla consegna dei prodotti.
HOME	Necessaria analisi del problema	Spazio dedicato al *cargorobot* in attesa di richieste.
cargorobot	Attore	Utilizza il *basicrobot24* fornito dal committente per effettuare i movimenti di carico.
sonar	Necessaria analisi del problema	In grado di rilevare la presenza di un prodotto entro una certa distanza in ioport.
web-gui	Necessaria analisi del problema	Rappresenta dinamicamente lo stato della hold.
led	Necessaria analisi del problema	Viene acceso quando il sonar misura una distanza D > DFREE per almeno 3 sec.

Tabella riassuntiva

Componente	**ID requisito**	**Requisito (sintesi)**
productservice	PRD-1	Registra prodotti e restituisce PID > 0
products	P-1	Ogni prodotto ha un peso
hold	H-1	Area rettangolare con IOPort e 4 slot
hold / slots	H-2	Uno slot è permanentemente occupato, gli altri inizialmente vuoti
hold / slots	H-3	Slot identificati da coordinate / nome univoco
hold / IOPort	H-4	Punto di arrivo del container prima del posizionamento

sensor (sonar)	SNS-1	Rileva presenza container: D < DFREE/2 per ~3s ⇒ presente
sensor (sonar)	SNS-2	Guasto: D > DFREE per ≥3s ⇒ failure
LED	LED-1	Indicatore di failure sonar
cargorobot	ROB-1	Posiziona il container nello slot assegnato
cargorobot	ROB-2	Ritorna alla posizione HOME a fine lavoro
cargoservice	CRS-1	Riceve richiesta di carico per PID registrato
cargoservice	CRS-2	Rifiuta la richiesta se peso > MaxLoad
cargoservice	CRS-3	Rifiuta la richiesta se stiva piena (4 slot occupati)
cargoservice	CRS-4	Se accettata la richiesta, associa uno slot al PID e ritorna il numero dello slot
cargoservice	CRS-5	Attende il container all'IOPort; nel frattempo altre richieste non sono elaborate
cargoservice	CRS-6	Assicura il posizionamento del container nello slot risrvato tramite cargorobot
cargoservice	CRS-7	Interrompe attività e accende LED se SNS-2
web-GUI	GUI-1	Mostra dinamicamente lo stato corrente della hold

Test Plans

Dall'analisi dei requisiti possiamo stilare un primo funzionamento del servizio:

- il *cargoservice* riceve una *richiesta di carico* di un prodotto registrato
- se il peso del prodotto supera la soglia *MaxLoad* la richiesta viene negata
- in caso venga accettata la richiesta, *cargoservice* associa uno slot (libero) al *PID* del prodotto corrispondente

La richiesta viene accettata solo se:
- il PID del prodotto è registrato nel database
- il peso del prodotto non eccede MaxLoad
- c'è almeno uno slot libero

Project

Implementiamo il funzionamento evidenziato dai Test Plans

- Il ***cargoservice*** è nello stato `wait` fino all'arrivo di una richiesta di carico che formalizziamo sfruttando il modello request/reply fornito dal QAK:

  ```
  Request requesttoload : requesttoload (PID)
  Reply replyrequesttoload : replyrequesttoload (X) for
  requesttoload
  ```

- Ricevuto il PID, si interfaccia con il ***productservice***:

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

  che restituisce i dati in formato:
  `String JSON '{"productId":31,"name":"p31","weight":311}'`

- Effettua il controllo sul peso del prodotto e la soglia MaxLoad, per poi valutare la richiesta

In questa prima istanza di test assumiamo slot sempre disponibili.

Testing

Nello sprint 1 approfondiremo l'aspetto di testing. Per il momento definiamo degli edge cases che portano al fallimento della requestToLoad:

1. Peso container in stiva + Peso container da caricare > MaxLoad

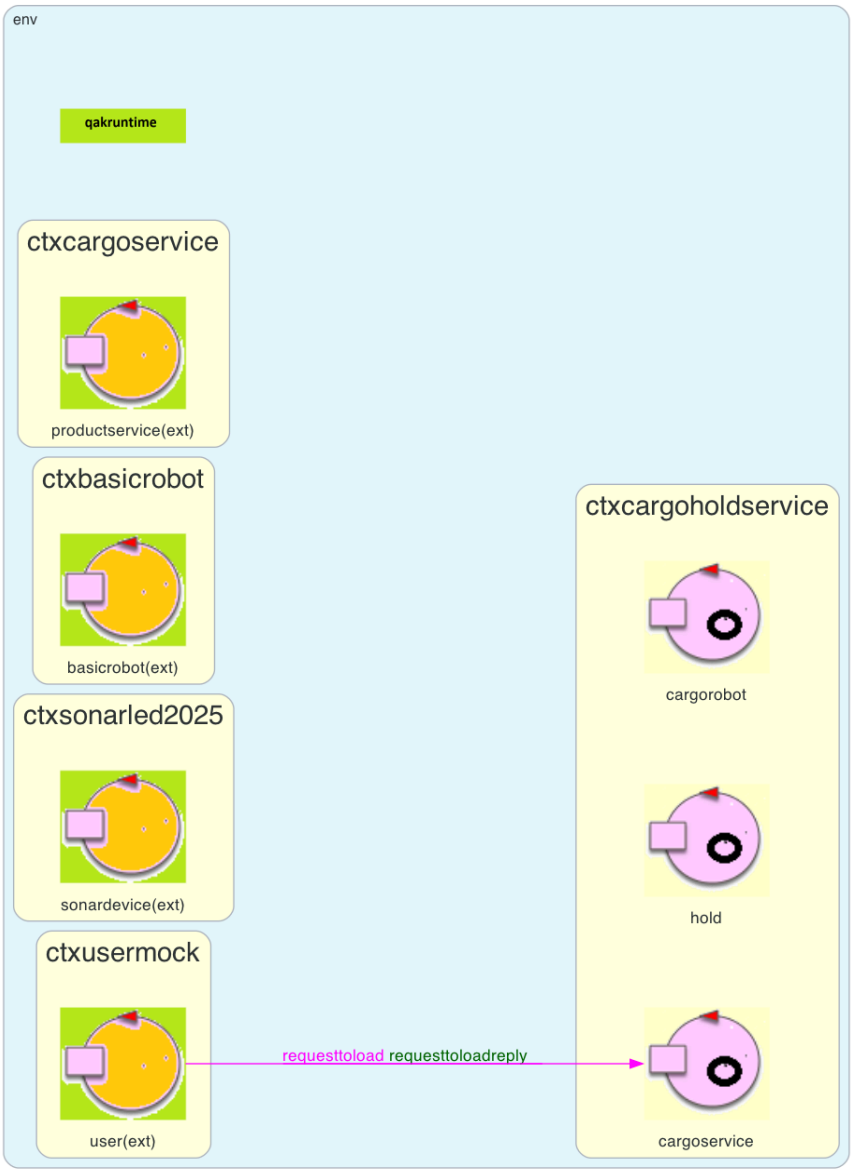
2. 4 Slot Occupati
3. Container non presente/non rilevato all'IOPort
4. Il productservice non risponde correttamente con i dettagli del container
5. Altra requestToLoad in corso
6. Il cargorobot non risponde/non si muove

Deployment

Date le osservazioni e le preliminari analisi svolte finora risulta possibile formulare un modello, o meglio, una prima iterazione di esso.
Notiamo a primo sguardo che l'attore cargoservice si assume molte responsabilità, come se fosse un macro attore e ciò non è una buona pratica.
Il linguaggio qak fornisce dei meccanismi di delega la cui utilità verrà valutata nei successivi sprint.
Per il resto osserviamo diverse richieste (compatibili con i componenti forniti dal committente) emesse verso contesti esterni e la presenza di un componente hold nello stesso contesto del cargoservice.
Contiamo perciò dei bounded context esterni con dei componenti forniti dal committente:

- contesto ctxbasicrobot con cargorobot
- contesto ctxcargoservice con productservice
- contesto ctxsonarled2025 con sonardevice

Abbiamo poi inserito un contesto esterno ctxusermock per mimare un utente o un operatore che usa il sistema, non abbiamo inserito un contesto per la webgui in quanto potrebbe essere lo stesso contesto ctxusermock ad essere sfruttato come interfaccia utente / webgui compatibilmente con i requisiti.
Inoltre al fine di avere una webgui che rappresenti correttamente il sistema, il sistema deve funzionare correttamente, quindi ci occuperemo di sviluppare questo componente in seguito. Il componente hold tiene traccia dello stato degli slot ed è pertinente nel contesto del nostro componente cargoservice che è oggetto di progetto.



cargoserviceArch

Maintenance

L’obiettivo è quello di dividere la progettazione in 3 sprint, con la finalità di ottenere 3 prodotti separati dove il successivo incapsula il precedente (immaginiamo il tutto come l’armatura di Ironman). La divisione sarà la seguente:

1. gestione della richiesta di carico, la registrazione dei prodotti e l'avviamento del robot (40h/uomo).
2. IOManager, in questa fase si svilupperà la parte di gestione del sonar, il rilevamento del product container nella porta di IO e l'interfacciamento hardware dei dispositivi fisici di sonar e led con il sistema.(30h/uomo)
3. Realizzazione della GUI (20h/uomo)