

**Università degli Studi di Modena e Reggio Emilia**

---

Dipartimento di Ingegneria Enzo Ferrari

Corso di Laurea in Ingegneria Informatica - sede di Mantova

**Progettazione e Realizzazione di un  
Sistema per l'Interazione Uomo-Macchina  
basato sui Digital Twin**

Relatore:

**Prof. Marco Picone**

Candidato:

Correlatore:

**Prof. Valeria Villani**

**Giorgio Menini**

A. A. 2021/2022



# Indice

<b>Introduzione</b>	<b>iii</b>
<b>1 Stato dell'arte</b>	<b>1</b>
1.1 Human-to-Machine Interaction . . . . .	1
1.1.1 Human-Robot Collaboration e CoBot . . . . .	2
1.1.2 Open challenges e Industria 4.0 . . . . .	4
1.2 Digital Twin . . . . .	6
1.2.1 Tecnologie abilitanti . . . . .	7
1.2.2 Proprietà . . . . .	8
1.2.3 Scopi e scenari applicativi . . . . .	10
1.3 Tecnologie e protocolli . . . . .	10
1.3.1 MQTT . . . . .	10
1.3.2 ROS . . . . .	13
1.3.3 WebSocket . . . . .	15
<b>2 Design e architettura del sistema</b>	<b>17</b>
2.1 Obiettivi e motivazioni . . . . .	17
2.2 Architettura . . . . .	19
2.3 Operator Digital Twin . . . . .	21
2.4 Robot Digital Twin . . . . .	26

2.5	Plant Manager . . . . .	32
2.5.1	Caricamento missione . . . . .	33
2.5.2	Gestione delle policy . . . . .	33
2.5.3	Esempio del flusso comunicativo . . . . .	34
<b>3</b>	<b>Implementazione del sistema</b>	<b>37</b>
3.1	Implementazione dell'operatore . . . . .	38
3.2	Implementazione dell'Operator Digital Twin . . . . .	39
3.3	Implementazione del robot . . . . .	42
3.4	Implementazione del Robot Digital Twin . . . . .	45
3.5	Implementazione del Plant Manager . . . . .	51
3.6	Validazione sperimentale . . . . .	57
3.6.1	Scenario applicativo . . . . .	57
3.6.2	Validazione e discussione . . . . .	61
<b>4</b>	<b>Conclusioni e sviluppi futuri</b>	<b>63</b>
4.1	Conclusioni . . . . .	63
4.2	Sviluppi futuri . . . . .	64
	<b>Bibliografia</b>	<b>68</b>

# Introduzione

Fin dalla sua nascita negli anni Cinquanta, l'informatica ha sempre impattato fortemente il mondo dell'industria.

Le prime tecnologie, infatti, diedero il via alla terza rivoluzione industriale che fu caratterizzata da un primo aumento dell'automazione e della velocità dei processi. Vennero introdotti, ad esempio, i primi bracci robotici e dei primi PLC. Grazie alle tecnologie informatiche (dai primi computer fino ad arrivare a internet) ed elettroniche, ci sono stati miglioramenti non solo nell'ambito produttivo ma anche e soprattutto dal punto di vista organizzativo. In questo periodo storico, però, l'integrazione delle diverse parti all'interno della fabbrica è minima, ogni macchinario è indipendente dagli altri e difficilmente vi è cooperazione durante la produzione.

Sempre grazie all'innovazione si è arrivati alla quarta rivoluzione industriale. Con quarta rivoluzione industriale o Industria 4.0 si intende la graduale fusione del mondo fisico (robot, PLC, ecc.) con il mondo digitale (internet, cloud, ecc.). Questa fusione è realizzata introducendo, all'interno delle imprese, nuove tecnologie, come il cosiddetto "Internet of Things". Tali tecnologie permettono di avere prodotti e processi interconnessi e queste interconnessioni generano all'interno di ogni azienda una grande quantità di dati che può essere sfruttata per rendere veramente efficiente la produzione. Tali dati possono essere archiviati in cloud per poi essere analizzati e pro-

cessati attraverso algoritmi di Machine Learning. Il loro utilizzo può essere fondamentale per riuscire ad apprendere dalle esperienze passate e incrementare la resa dei processi oltre che aumentare la conoscenza dei prodotti.

Inoltre, attraverso queste tecnologie dette “abilitanti”, i macchinari sono in grado, ad esempio, di comunicare eventuali problemi e necessità di manutenzione oppure aggiornare in tempo reale il proprio stato di lavorazione.

Un altro tema molto importante su cui si concentra l’Industria 4.0 è il miglioramento dell’automazione all’interno delle fabbriche. Robot e macchinari automatizzati sono presenti in ambito industriale già da molti anni ma con un limite importante: dover lavorare autonomamente e seguendo esattamente la propria programmazione separandosi dal resto dell’ambiente, anche dagli operatori. L’Industria 4.0 cerca di risolvere questo limite e realizzare una collaborazione sempre più stretta tra robot e uomo al fine di efficientare e migliorare sempre di più la produzione. Tale interazione può essere realizzata solo garantendo elevati livelli di sicurezza e una buona capacità di coordinamento delle attività.

Il lavoro svolto in questa tesi si colloca proprio in questo contesto. Il progetto consiste nella prototipazione di un sistema dove ogni entità, operatore o robot che sia, è rappresentata da un Digital Twin (DT). Tale DT è responsabile della comunicazione con il manager centrale che si occupa del gestire le diverse entità e di assegnare i carichi di lavoro.

Come vedremo, l’utilizzo di una rappresentazione virtuale per ogni elemento del sistema dà la possibilità di fornire al manager una interfaccia unica comune per interagire. Questo facilita il suo lavoro perché esso potrà gestire tutti gli elementi presenti attraverso un’unica modalità di comunicazione e quindi potrà parlare facilmente anche con robot o dispositivi di tipologie diverse che hanno metodologie e formati di messaggi completamente opposti.

# Capitolo 1

## Stato dell'arte

### 1.1 Human-to-Machine Interaction

Con l'avvento dell'Industria 4.0 sempre più macchinari e sistemi automatizzati sono stati inseriti all'interno delle fabbriche, dando vita a sistemi cyber-fisici. Questo ha portato ad una necessità crescente di studiare quelle che sono le Human-to-Machine Interaction (HMI), ovvero le possibili interazioni tra le persone fisiche (gli operatori) e le macchine, in modo da poter progettare e realizzare nel modo migliore i sistemi produttivi.

Le HMI devono essere pensate in modo tale da garantire:

- Funzionalità: intesa come il numero di azioni che possono essere eseguite da un sistema, il suo vero valore dipende da quante di queste azioni e servizi possono essere effettivamente sfruttate dagli operatori.
- Usabilità: dato un sistema con una determinata funzionalità, la sua usabilità è espressa tramite il grado di utilizzo del sistema, ovvero quanto può essere utilizzato in modo efficiente e adeguato il sistema per raggiungere determinati obiettivi per determinati utenti.

### 1.1.1 Human-Robot Collaboration e CoBot

All'interno della macro area dell'Human-Machine Interaction vi è compresa anche la più specifica Human-Robot Interaction (HRI) e la sua naturale evoluzione Human-Robot Collaboration (HRC).

La collaborazione uomo-robot è una tipologia di lavoro che combina le capacità umane con l'efficienza e la precisione delle macchine. Questa collaborazione si sta affermando sempre di più con la rivoluzione 4.0 che sta progredendo in questi anni all'interno del mondo industriale. In questi casi i robot che vengono usati sono i cosiddetti "robot collaborativi" o "CoBot". Essi condividono lo spazio di lavoro con l'operatore, il quale sfrutta la macchina per realizzare operazioni complesse, svolgendo principalmente un ruolo di supervisione e di intervento in caso di bisogno.

Quando si considerano situazioni di HRC bisogna ricordare che esiste sempre una porzione di workspace, detta collaborative workspace (CWS), che può essere occupata sia dal robot che dall'operatore. L'abolizione delle barriere di sicurezza e il passaggio a una maggiore collaborazione ha certamente aumentato la flessibilità e la produttività ma ha anche introdotto dei problemi per quanto riguarda la sicurezza all'interno delle CWS. Per affrontare questo tema, sono stati introdotti dalla *International Organization for Standardization* (ISO) gli standard ISO 10218-1/2:2011 [1][2] che impongono delle specifiche precise ai vari robot e macchinari.

All'interno di questi standard si possono individuare, come mostrato in Figura 1.1, quattro tipologie di modalità collaborative:

- Safety-rated monitored stop: quando l'operatore è nel CWS, al robot non è concesso muoversi; il robot deve quindi bloccarsi nella posizione in cui si trova.

- Hand guiding: in questa modalità, quando si ha la presenza dell'operatore nell'area, il robot non si muove se non guidato dall'operatore stesso.
- Speed and separation monitoring: questa modalità prevede di monitorare la presenza di un operatore e la sua distanza dal robot; in questo modo il robot non dovrà fermarsi immediatamente, come nella prima modalità, ma potrà gradualmente adattare la propria velocità e posizione sulla base della distanza dell'operatore, fino a fermarsi quando l'operatore è a lui vicino.
- Power and force limiting: l'ambiente e il robot sono dotati di sensori che permettono al robot di continuare a lavorare anche in presenza dell'operatore, limitando però la forza con cui il robot potrebbe entrare in contatto con la persona.

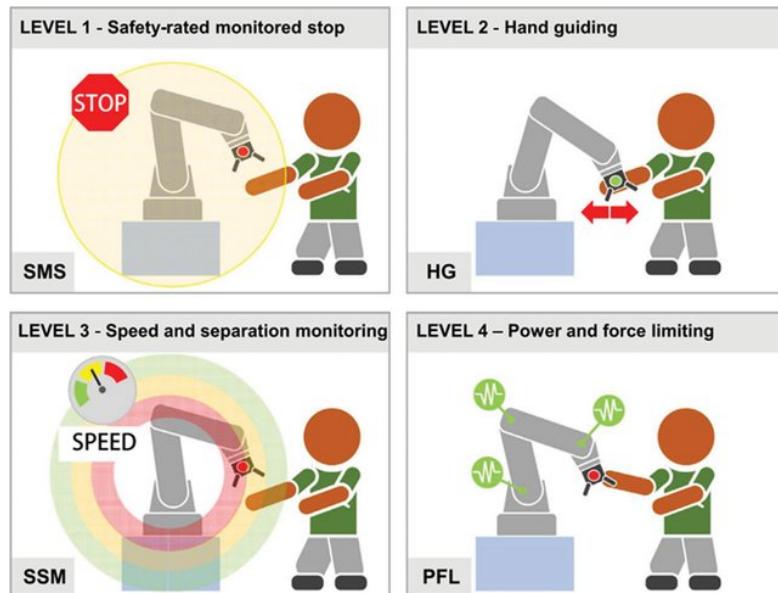


Figura 1.1: Le quattro modalità collaborative dello standard 10218-1/2:2011

I CoBot sono un esempio di robot che opera senza alcuna barriera, come illustrato nella Figura 1.2. Applicando gli standard imposti, i CoBot sono progettati appositamente per rilevare la presenza dell'operatore attraverso dei sensori e decidere autonomamente se è necessario fermare l'esecuzione, rallentare il proprio movimento, deviare la traiettoria oppure se è possibile proseguire l'azione in sicurezza. I CoBot sono perciò programmati fin dall'inizio per essere allo stesso tempo e il più possibile flessibili e sicuri.



Figura 1.2: Esempio di un robot collaborativo industriale

### 1.1.2 Open challenges e Industria 4.0

Con il crescente numero di elementi presenti all'interno di una azienda, come macchinari, operatori e tecnologie coinvolte, la complessità del sistema è sempre più alta. Avere un elevato numero di tipologie di figure che interagiscono all'interno del sistema richiede di sviluppare numerosi protocolli e formati, da utilizzare all'interno della rete di comunicazione dell'azienda per favorire la cooperazione e collaborazione.

Date le particolarità e le caratteristiche di ogni elemento, l'Industria 4.0 ha

avuto un ruolo fondamentale perché ha portato un aiuto anche in questo senso, permettendo alle aziende di affrontare tale complessità.

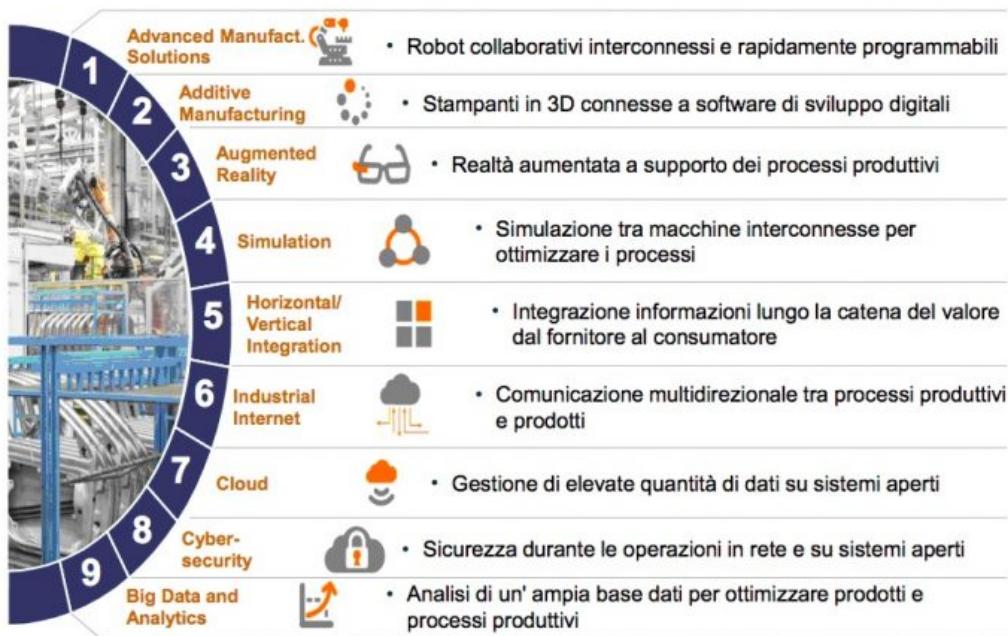


Figura 1.3: Tecnologie abilitanti dell'Industria 4.0

Oltre a introdurre all'interno delle aziende i robot collaborativi, infatti, l'Industria 4.0 ha dato una forte spinta anche ad altre innovazioni tecnologiche, come mostrato in Figura 1.3.

Queste tecnologie, chiamate "abilitanti", permettono grossi miglioramenti in tutto il processo produttivo e non solo.

In particolare, alcune innovazioni, come la simulazione e l'uso del cloud, permettono anche di gestire in maniera più facile e flessibile le comunicazioni interne tra i vari elementi della fabbrica, dato che queste ed altre tecnologie sono alla base dei Digital Twin, le rappresentazioni virtuali che approfondiremo nella sezione successiva.

## 1.2 Digital Twin

Per Digital Twin si intende la rappresentazione virtuale di una risorsa fisica, come ad esempio un oggetto, un processo, una persona, un'infrastruttura o un qualsiasi altro sistema più o meno complesso [4].

Questo termine venne introdotto la prima volta nel 2002 dal dr. Grieves durante una presentazione sul Product Lifecycle Management (PLM). All'interno della presentazione egli utilizzò l'immagine che si può vedere in Figura 1.4, la quale, seppur semplificata, descrive un Digital Twin attraverso i suoi principali componenti: uno spazio reale, uno spazio virtuale e i collegamenti tra i due per lo scambio di dati e informazioni.

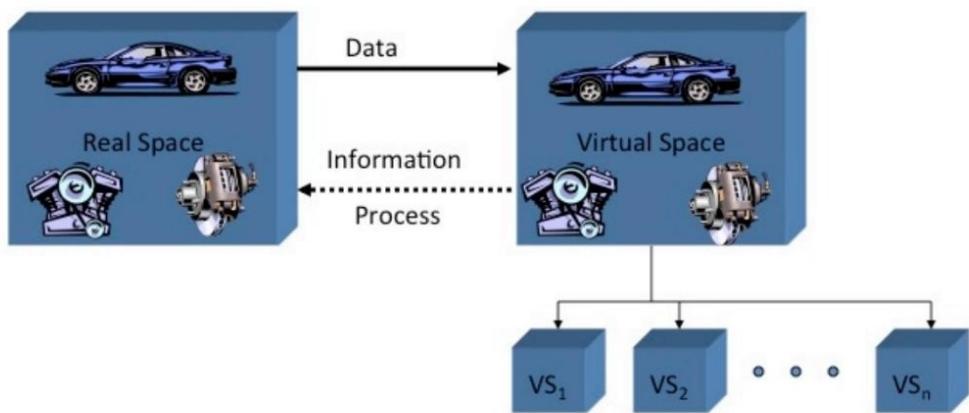


Figura 1.4: Idea concettuale del Digital Twin per PLM

Nonostante ciò, fu la National Aeronautics and Space Administration (NASA), nel 2010, ad utilizzare per prima tale termine in una pubblicazione ufficiale [8], dando la seguente definizione di Digital Twin:

*“Una simulazione ultra-realistica ad alta scalabilità, che utilizza i migliori modelli fisici disponibili, i dati dei sensori e quelli storici per il mirroring di uno o più sistemi reali”.*

Inizialmente questa tecnologia non prese piede in modo diffuso, soprattutto per le difficoltà di implementazione che si incontravano. È solo grazie all'Internet of Things (IoT) che è diventato conveniente implementarlo: con tale tecnologia, infatti, si è reso più facile dialogare con gli oggetti e quindi anche più facile costruire i Digital Twin.

Un DT, come detto, rappresenta un oggetto qualsiasi e di tale oggetto esso simula tutto, sia lo stato fisico che i comportamenti. Il Digital Twin è constantemente connesso all'oggetto e si aggiorna in tempo reale in risposta a sue modifiche di stato, comportamento o contesto.

Come si può immaginare, ad ogni DT può essere associato un solo oggetto fisico per necessità di realizzazione. Questo, però, non vale al contrario: a un oggetto fisico possono corrispondere più DT, ognuno con un identificatore univoco per poterli distinguere.

### 1.2.1 Tecnologie abilitanti

Per poter definire e utilizzare correttamente un DT è necessario che lavorino insieme le seguenti tecnologie:

- API e Standard aperti: per la realizzazione dei DT servono dati da più sistemi che possono essere raccolti solo attraverso questi sistemi.
- Internet of Things: l'uso di sensori e strumenti ad alta precisione consentono la raccolta continua dei dati da inviare sullo stato e sulle condizioni dell'asset fisico.
- Intelligenza artificiale: questa tecnologia è fondamentale per poter sfruttare i dati raccolti, storici e in tempo reale, e poter fare previsioni su scenari o eventi futuri.

- Realtà aumentata e virtuale: dà la possibilità di rendere il modello spaziale e poter visualizzare il Digital Twin così da poter osservarlo anche da remoto e interagire con esso a distanza.
- Cloud ed Edge Computing: il cloud consente l'archiviazione di grandi quantità di dati e, attraverso di esso, è possibile accedere ai DT e ai suoi servizi da remoto; l'edge computing, invece, è fondamentale per garantire bassa latenza nella comunicazione e avere quindi aggiornamenti il più possibile real-time.

### 1.2.2 Proprietà

Ogni Digital Twin deve avere un serie di proprietà fondamentali per essere considerato tale:

- Rappresentatività e contestualizzazione: il DT deve rappresentare l'oggetto fisico in un determinato contesto nel modo più preciso possibile. Per farlo il DT deve rappresentare almeno quelle proprietà, comportamenti e relazioni che lo qualificano nel caso preso in esame.
- Reflection: è la capacità che il DT deve avere per essere sempre perfettamente sincronizzato con l'oggetto fisico sui valori di un qualsiasi suo campo.
- Replication: un DT può essere replicato più volte per essere utilizzato in diverse applicazioni, ognuna con delle necessità differenti. Queste repliche potrebbero essere associate ad una tecnica di sincronizzazione One-to-One dove ogni copia è direttamente sincronizzata con l'entità fisica oppure ad una tecnica Master-Slave dove invece vi è un DT (master) collegato all'entità fisica e gli altri DT (slave) sincronizzati con il master.

- Composability: il DT deve avere la capacità di astrarre la complessità dei sistemi fisici, i quali solitamente sono aggregazioni di sottosistemi e componenti. Un DT, inoltre, deve poter supportare la correlazione di diversi DT elementari in organizzazioni complesse e riuscire a fornire informazioni sul DT aggregato e sui singoli componenti.
- Augmentation: un DT può estendere le funzioni dell'oggetto fisico che rappresenta. Solitamente le funzionalità offerte dall'oggetto sono limitate a causa delle scarse capacità di elaborazione e quindi il DT può andare ad aggiornarle, modificarle e migliorarle attraverso delle API che il DT fornisce.

Come è possibile notare in Figura 1.5, i Digital Twin hanno anche ulteriori importanti proprietà, come la memorizzazione, la persistenza ed altre qui non riportate, ma quelle descritte permettono facilmente di capire la grande importanza di questi strumenti.

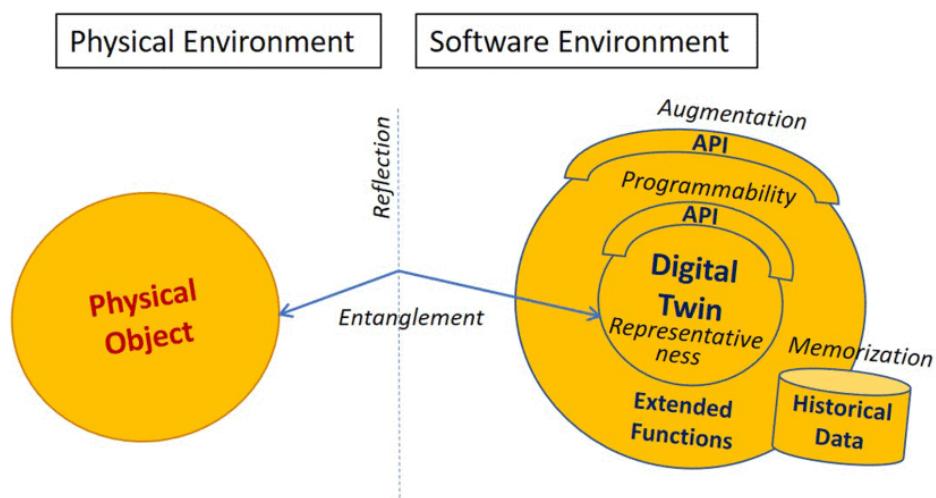


Figura 1.5: Rappresentazione Digital Twin e sue proprietà

### **1.2.3 Scopi e scenari applicativi**

I Digital Twin possono essere utilizzati per diversi scopi:

- Creare e testare sistemi più o meno complessi in ambienti virtuali.  
Questo permette di ridurre i costi di progettazione e produzione aumentando l'efficienza del processo.
- Porre un intermediario tra l'oggetto fisico e il mondo esterno. In questo modo, oltre a garantire maggiore sicurezza facendo passare le comunicazioni attraverso il DT, è possibile anche fornire all'esterno nuove funzioni e servizi rispetto a ciò che dà l'oggetto fisico.
- Tenere traccia della storia dell'oggetto rappresentato lungo il proprio ciclo di vita. Questa funzione dà la possibilità di analizzare, simulare e prevedere il comportamento di ciò che il DT rappresenta e poter quindi agire di conseguenza.

## **1.3 Tecnologie e protocolli**

All'interno del progetto sviluppato in questa tesi vengono utilizzate tre tecnologie e protocolli, nello specifico MQTT, ROS e WebSocket.

In questa sezione ognuno di questi elementi verrà descritto in modo completo.

### **1.3.1 MQTT**

Uno dei metodi di comunicazione adottati è l'utilizzo del protocollo MQTT (Message Queue Telemetry Transport). MQTT è un protocollo di messaggistica estremamente veloce e leggero di tipologia publish/subscribe basato su TCP [5].

E' il protocollo ideale per dispositivi a basso consumo, con poca capacità computazionale e con banda limitata. Grazie a queste caratteristiche è diventato, nel tempo, uno standard per il mondo IoT.

Essendo di tipo pub/sub, sono previste tre figure principali:

- Publisher: l'entità che pubblica le informazioni e dati.
- Subscriber: l'entità che si registra per ricevere informazioni e dati.
- Broker: entità responsabile di indirizzare i messaggi pubblicati verso i subscriber interessati.

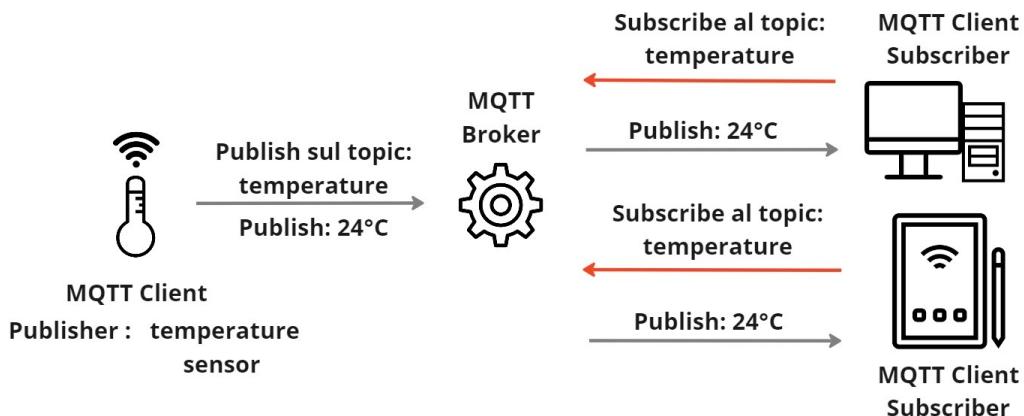


Figura 1.6: Comunicazione attraverso MQTT

Come si può notare nella Figura 1.6, il suo funzionamento si basa sull'utilizzo dei topic. I topic sono degli indirizzi e un publisher può scrivere su uno o più indirizzi in base a quello che vuole comunicare. I subscriber, invece, possono registrarsi a uno o più indirizzi di loro interesse e ricevere i relativi messaggi. La comunicazione però non è diretta publisher-subscriber ma vi è un intermediario, il broker. Egli si occupa di prendere i messaggi pubblicati sui vari topic e distribuirli ai vari subscriber che sono interessati ai messaggi.

in base alle sottoscrizioni.

E' importante notare che un publisher per certi topic può anche essere subscriber su altri e viceversa.

Una particolare tipologia di messaggi che possono essere scambiati sui topic sono i cosiddetti messaggi "retained".

Ogni messaggio inviato al broker da un client MQTT possiede un flag di *retain* che può essere valorizzato "vero" o "falso". Quando un messaggio ha tale flag con valore "vero" allora questo messaggio verrà salvato dal broker e tenuto in memoria fino a quando non arriva, sullo stesso topic, un nuovo messaggio retained.

Questa modalità è utile perché in questo modo il broker può inviare il messaggio salvato a qualunque client si connetta sul topic del messaggio retained, anche in un secondo momento. Il problema di MQTT, infatti, è che i subscriber ricevono i messaggi che il broker riceve a sua volta solo da quando si sottoscrivono, perdendo invece quelli precedenti. In questo modo un subscriber dopo essersi sottoscritto potrebbe anche aspettare diverso tempo prima di ricevere il primo messaggio.

Un'altra caratteristica importante di MQTT è la Quality of Service (QoS). La QoS è un accordo tra chi invia e riceve i messaggi su come devono essere scambiati i messaggi stessi. In MQTT esistono tre livelli di QoS:

- Al massimo uno: prevede che il mittente mandi un solo messaggio, senza preoccuparsi che questo sia arrivato a destinazione ("fire and forget").
- Almeno uno: garantisce che un messaggio venga consegnato almeno una volta al destinatario. Il mittente dopo aver inviato il messaggio attende un ACK (Acknowledgment) di conferma dal destinatario e se questo non arriva allora rimanda il messaggio. Potrebbe avvenire che,

durante la comunicazione, il primo messaggio arrivi mentre si perde o si corrompe l'ACK di conferma: in questo caso il mittente rinvierà il messaggio, anche se non necessario.

- Esattamente uno: garantisce che un messaggio venga consegnato una sola volta. È il livello più sicuro ma anche più lento perché prevede un handshake a quattro parti per fare in modo di consegnare un solo messaggio corretto a destinazione.

I vantaggi dati da un protocollo di questo tipo sono i seguenti:

- La messaggistica è asincrona, non bloccante. I produttori e i consumatori dei messaggi non devono essere collegati nello stesso momento per comunicare.
- È una struttura altamente scalabile poiché i broker devono solo instradare i messaggi e quindi possono essere replicati facilmente per supportare maggiori volumi di dati trasferiti.
- Fornisce la possibilità di instaurare una comunicazione affidabile attraverso la QoS.

### 1.3.2 ROS

ROS è l'acronimo di Robot Operating System ed è un meta-sistema operativo che gira principalmente su piattaforme basate su Unix [7]. Fornisce non solo tutti i servizi che normalmente dà un sistema operativo (astrazione dell'hardware, controllo a basso livello dei dispositivi, comunicazione tra processi, ecc.) ma anche librerie e strumenti che consentono all'utente di realizzare intere applicazioni robotiche.

ROS basa il proprio funzionamento sui processi o nodi. Ogni nodo può essere sviluppato in modo indipendente dagli altri e poi interagire con gli altri solo al momento dell'esecuzione. Un insieme di nodi possono essere organizzati in un package e messi a disposizione degli utilizzatori di ROS.

Tutto il software è organizzato in package, che possono essere distribuiti dalla comunità dietro ROS oppure creati dai singoli sviluppatori.

I package forniti direttamente dalla comunità di sviluppatori implementano una serie molto completa di funzionalità robotiche come la pianificazione della traiettoria, la percezione, la visione, il controllo e la manipolazione.

ROS è un sistema multilingue: i vari package possono essere sviluppati in C++, Python e altri linguaggi di programmazione.

Quando vengono avviati più nodi questi creano una rete tra loro e, grazie all'infrastruttura realizzata da ROS, possono comunicare in due modi diversi, mostrati in Figura 1.7 e Figura 1.8.

- Attraverso topic: come avviene in MQTT, si può utilizzare una comunicazione asincrona per scambiare informazioni tra i nodi. La tipologia è sempre publish/subscribe in modo da rendere chi scrive e chi legge indipendenti tra di loro. Lo svantaggio dei topic è il fatto di essere unidirezionali e tipizzati: quando si va a definire un publisher su un topic bisogna anche specificare la tipologia di messaggio e quindi i campi contenuti che verranno mandati.
- Attraverso servizi: in questo caso si ha una comunicazione sincrona. La tipologia non è più pub/sub ma client/server e di conseguenza vi sarà un nodo ROS server che potrà offrire un servizio ai nodi ROS client che lo richiedono attraverso l'apposita request.

Oltre ai nodi, vi è sempre in esecuzione il ROS Master. Il suo scopo è quello

di orchestrare i nodi e gestire le loro comunicazioni. Ogni volta che un nodo viene avviato gli dovrà essere assegnato un id univoco per essere identificabile e questo, ad esempio, è un compito del Master.

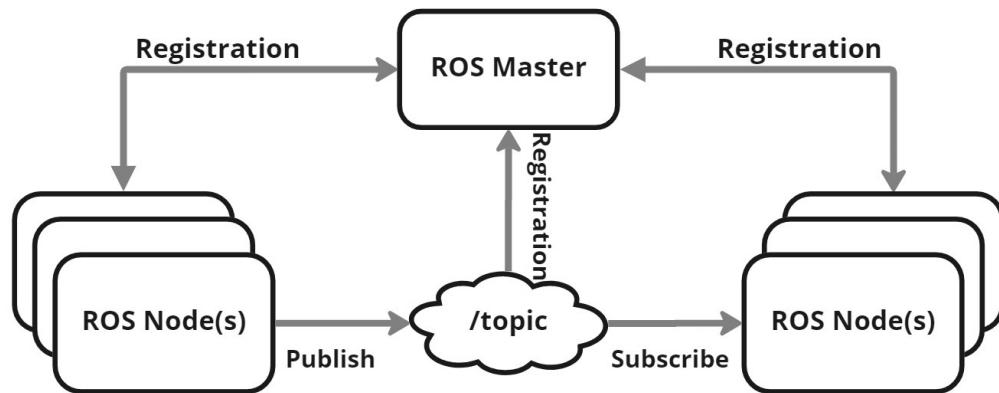


Figura 1.7: Comunicazione in ROS tramite topic

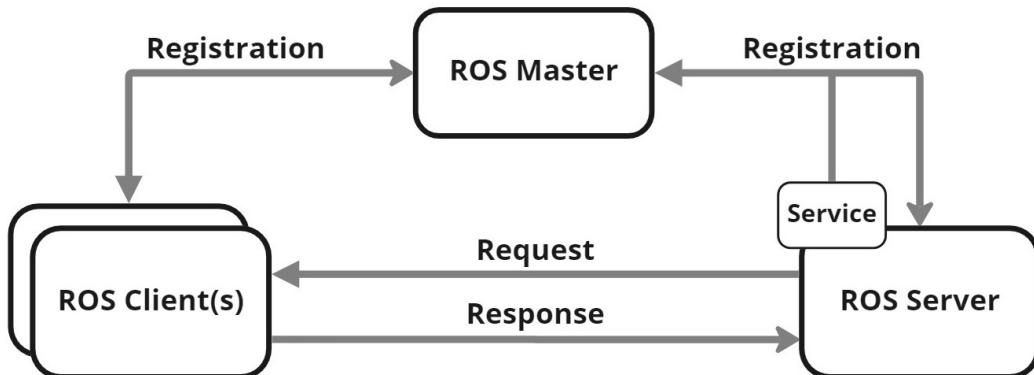


Figura 1.8: Comunicazione in ROS tramite servizi

### 1.3.3 WebSocket

WebSocket è un protocollo di comunicazione full-duplex basato su socket TCP [3].

WebSocket è stato progettato per essere implementato sia sui browser che

sui server ma può essere utilizzato in qualsiasi applicazione client-server. Questo protocollo facilita l'interazione tra i partecipanti alla comunicazione, facilitando così quelle applicazioni, come chat di assistenza o giochi online, che richiedono continui aggiornamenti.

WebSocket ha la fase di handshake simile a quella di HTTP ma da quel punto i due protocolli si differenziano.

Con WebSocket, dopo l'handshake, il canale di comunicazione rimane sempre aperto. Questo consente sia al client che al server di poter comunicare con la controparte senza attendere alcuna richiesta. Quando il server ha un aggiornamento diventa attivo autonomamente e lo invia sul canale, come è rappresentato in Figura 1.9. Questo protocollo viene utilizzato tutte le volte che è necessaria una comunicazione rapida e quindi non è possibile implementare un sistema di richiesta/risposta. Siti di assistenza tramite chat, siti che riportano gli indici delle borse ma anche i social media stessi sono esempi di implementazioni dove è presente questo protocollo.

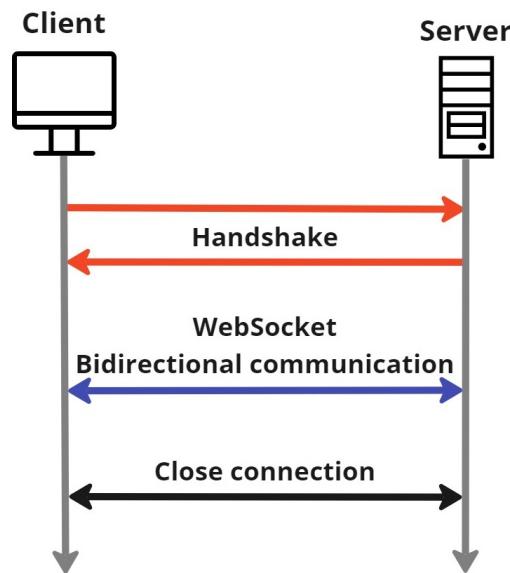


Figura 1.9: Funzionamento comunicazione WebSocket

# Capitolo 2

## Design e architettura del sistema

### 2.1 Obiettivi e motivazioni

La collaborazione uomo-macchina e, in particolare, uomo-robot in ambito industriale è un argomento in continua evoluzione.

In generale, nella progettazione di sistemi che prevedono l'interazione di macchine e apparecchiature con l'uomo si dà sempre la priorità alla facilità di gestione e alla sicurezza degli esseri umani che interagiscono con esse.

Non sempre però questa collaborazione è facile da realizzare, in quanto numerosi sono gli aspetti da considerare.

Il problema di questi sistemi si palesa nel momento in cui ogni macchinario, ogni robot e ogni device utilizzato dagli operatori interagisce e comunica in modo indipendente dagli altri. Come mostrato in modo esemplificativo in Figura 2.1, ogni entità può utilizzare protocolli, tipologie di messaggi e contenuti diversi tra loro, rendendo impossibile una comunicazione lineare e fluida tra robot, operatori e manager.

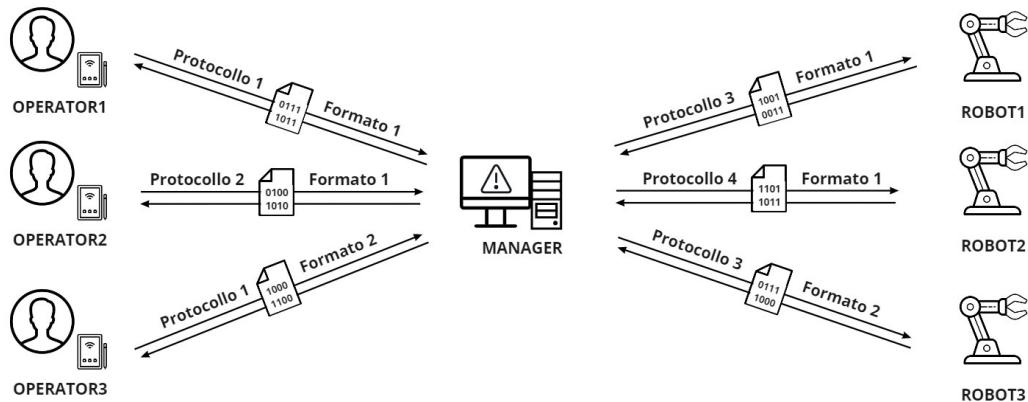


Figura 2.1: Architettura del sistema senza DT

Questo progetto nasce con l'obiettivo di migliorare la gestione delle interazioni uomo-robot e, allo stesso tempo, favorire una maggiore sicurezza durante il lavoro.

L'architettura stessa del progetto è stata pensata per realizzare un sistema che semplifichi le modalità di connessione e comunicazione.

L'idea alla base è quella di portare l'utilizzo dei Digital Twin all'interno della fabbrica. Si è pensato che attraverso la potenza dei DT sia possibile semplificare quelle operazioni, anche elementari, che altrimenti richiederebbero conoscenze approfondite dei macchinari e delle loro impostazioni. Grazie alla loro presenza, infatti, si riesce ad ottenere un completo disaccoppiamento tra le varie figure presenti nell'architettura.

Questa separazione e disaccoppiamento del sistema da due vantaggio:

- Non si è legati ad una singola tecnologie, di conseguenza ogni componente può essere implementato sfruttando quella migliore per il suo caso.
- Grazie ai DT è possibile definire delle policy di gestione del lavoro semplici ma che garantiscono la massima efficienza.

## 2.2 Architettura

Come anticipato in precedenza, il fulcro e la novità di questo lavoro di tesi è l'utilizzo dei Digital Twin all'interno di uno scenario industriale dove operatore e robot collaborano insieme.

I principali componenti del progetto sono mostrati di seguito in Figura 2.2.

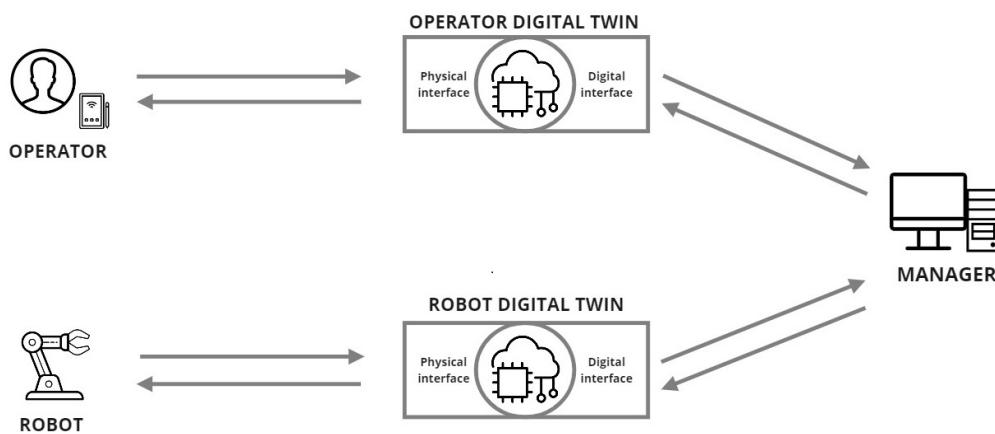


Figura 2.2: Architettura del sistema

Nell'architettura proposta, gli elementi cardine sono cinque e ognuno di essi ha delle proprie specificità. Gli operatori e i robot sono gli unici elementi fisici mentre i rimanenti tre sono entità digitali.

Ogni operatore all'interno dello stabilimento è dotato di un proprio device connesso a internet. Attraverso tale device esso può indicare quando inizia e finisce il proprio turno ma è utile in particolar modo per ricevere i task da eseguire e altri avvisi dal manager centrale. Inoltre, ad ogni operatore è fornito un device IoT, il quale è in grado di rilevare lo stato di salute dell'operatore in tempo reale.

Questa rilevazione continua è fondamentale per garantire la sicurezza dell'o-

peratore in ogni momento, soprattutto mentre opera in collaborazione con dei macchinari.

Oltre agli operatori, come già detto, i robot sono l’altro elemento fisico rappresentato. Questi macchinari esistono di diversi tipi: robot manipolatori, robot mobili ma non solo. Tali robot sono connessi ad internet e attraverso questa connessione essi vengono gestiti. Ogni macchina è in grado di eseguire uno o più tipologie di azioni che possono variare in base a come ciascuna viene programmata. L’esecuzione delle varie missioni può essere compiuta dal robot a diverse velocità, in base alle configurazioni fornite.

Gli altri tre elementi dell’architettura sono l’Operator Digital Twin (ODT), il Robot Digital Twin (RDT) e il Plant Manager (PM).

Il ruolo dei DT è astrarre e virtualizzare il rispettivo oggetto fisico in una replica digitale, mentre il PM ha il compito di gestire i vari elementi presenti e distribuire i task ad essi.

Inoltre, sebbene nello schema appaiano solo cinque “individui”, è importante sottolineare che in questo contesto possono sempre essere presenti più entità fisiche contemporaneamente, anche di diverso tipo. Ad esempio, possono esserci più operatori di diverse specializzazioni oppure più macchine di tipologie diverse o anche più operatori e più macchine. In ogni caso, le varie entità potranno sempre continuare ad interagire tra loro e scambiarsi informazioni perché ognuna di esse sarà caratterizzata da un proprio gemello digitale che semplifica l’interazione con il manager.

È importante sottolineare che in questo capitolo si approfondirà in particolar modo l’architettura alla base della comunicazione tra DT e Plant Manager, la quale è omogenea per tutti.

Un esempio dell’architettura progettata per comunicare con la parte fisica verrà mostrato nel capitolo successivo.

## 2.3 Operator Digital Twin

L'Operator Digital Twin è la rappresentazione virtuale dell'operatore.

Ogni operatore ha un proprio ODT e ogni Digital Twin è associato ad una singola e specifica istanza dell'oggetto rappresentato.

Ogni ODT è composto al suo interno da una interfaccia fisica e una digitale.

L'interfaccia fisica collega la rappresentazione con l'operatore vero e proprio.

Attraverso questa interfaccia l'ODT può conoscere tutte le caratteristiche dell'operatore ad esso associato, come illustrato in Figura 2.3.

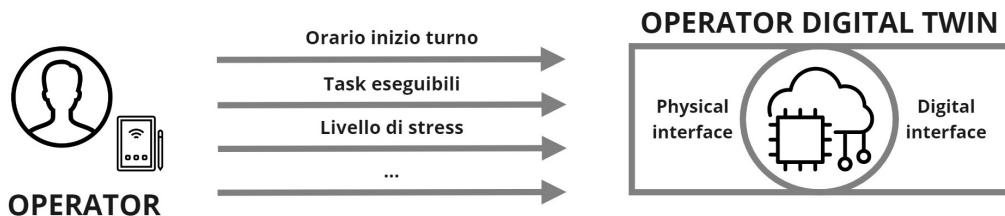


Figura 2.3: Messaggi scambiati tra operatore e ODT

In particolare, verranno mandate le seguenti informazioni:

- Orario di inizio e fine turno: tali valori indicheranno l'orario in cui l'operatore è arrivato in fabbrica ed è pronto e l'orario in cui invece termina il proprio turno. Tali valori, oltre a indicare al manager quando può assegnare del lavoro a quel determinato operatore, permettono anche di segnalare ritardi o uscite anticipate non previste.
- Livello di esperienza: questo valore può essere tenuto in considerazione dal manager in fase di scelta dell'operatore quando sono da eseguire operazioni più o meno complesse.

- Task che è in grado di eseguire: anche questa è una informazione fondamentale per il manager per sapere a quali task ogni operatore può essere assegnato.
- Stato di salute: al fine di garantire la maggiore sicurezza possibile, ogni operatore è dotato, come già anticipato, di un dispositivo IoT che rileva il livello di stress e altri segnali vitali e li comunica al proprio DT.

Queste informazioni vengono mandate quando l'operatore si collega la prima volta accendendo il proprio tablet, ma poi saranno re-inviate ogni volta che vi sarà un aggiornamento di un qualsiasi campo.

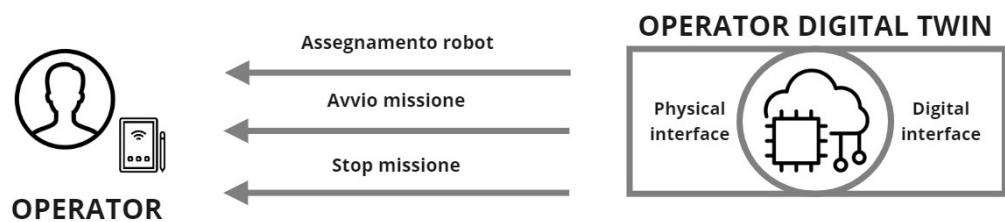


Figura 2.4: Messaggi scambiati tra ODT e operatore

Come si può notare dalla Figura 2.4, questo collegamento è fondamentale anche per l'operatore stesso, in quanto egli riceve dall'ODT le seguenti informazioni:

- Assegnamento del robot: il DT indica all'operatore il robot o altro macchinario a cui è assegnato.
- Segnale di avvio missione: indica l'inizio del lavoro in collaborazione alla macchina assegnata.
- Segnale di stop: indica all'operatore di fermarsi e lo libera dall'ultimo assegnamento fatto. Questo segnale può essere mandato o quando la

lavorazione è terminata oppure quando si hanno avuti dei problemi durante l'esecuzione.

Questi messaggi vengono formulati dall'ODT appositamente per il proprio operatore sulla base di ciò che viene deciso dal PM. La necessità che ogni DT formuli dei propri messaggi nasce dal fatto che tra i vari DT degli operatori l'interfaccia fisica può variare. Tale interfaccia, infatti, è costruita appositamente sulla base delle caratteristiche dei diversi operatori. Se un operatore utilizza un dispositivo con un particolare protocollo di comunicazione e, soprattutto, un certo formato dei messaggi, allora il relativo DT sarà costruito in modo tale che esso possa capire il protocollo e il formato utilizzati e riuscire così a comunicare.

L'interfaccia digitale, invece, è fondamentale per comunicare con le altre entità digitali presenti come il PM. Le informazioni che l'ODT riceve dall'operatore vengono usate per registrare l'operatore stesso presso il Plant Manager centrale e mantenerlo costantemente aggiornato. Il PM, in questo modo, viene a conoscenza della presenza dell'operatore e può assegnarlo nei lavori richiesti.

L'interfaccia digitale dell'ODT, a differenza di quella fisica, è comune per tutti i Digital Twin degli operatori.

Questa interfaccia comunicherà, per tutti i DT, sfruttando il protocollo MQTT ed è strutturata come rappresentato nella Figura 2.5:

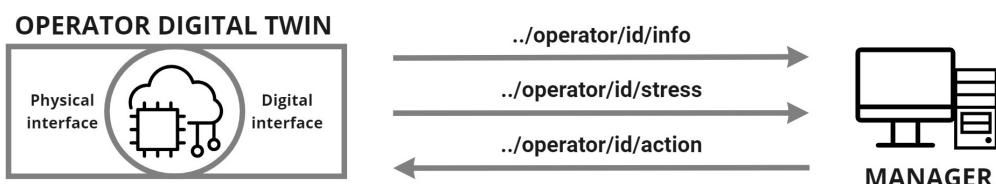


Figura 2.5: Messaggi scambiati tra ODT e PM

Ogni ODT scambia messaggi su tre topic principali:

- *operator/id/info*: viene usato dall'ODT per registrare il proprio operatore la prima volta e mandare le informazioni aggiornate ogni volta che vi è un cambiamento.
- *operator/id/stress*: viene usato dall'ODT quando deve segnalare al manager una situazione di stress per l'operatore.
- *operator/id/action*: viene usato dal manager per comunicare all'ODT e di conseguenza all'operatore quale task eseguire o se stoppare una azione in corso.

Ad ogni ODT corrispondono tre topic sui quali può scrivere o leggere e si possono distinguere attraverso l'id contenuto al suo interno. Tale id corrisponde all'id dell'operatore che l'ODT rappresenta.

Anche il PM per mandare un messaggio ad un operatore dovrà sapere il suo id per scrivere sul topic corretto.

Per ogni topic i Digital Twin utilizzano un tipo di messaggio predefinito ognuno con una precisa struttura.

Sul topic */info* il messaggio che verrà mandato avrà i seguenti campi:

- *operator\_id*: indica l'id dell'operatore, è un campo ridondante perché già presente nel topic ma può essere utile al manager in particolari casi.
- *operator\_level*: indica il livello di esperienza dell'operatore.
- *operator\_action*: è una lista dei task che l'operatore può eseguire.
- *operator\_current\_action*: di default questo campo ha valore *null*; quando questo ha un valore diverso allora indica l'operazione in cui l'operatore è impegnato.

- operator\_robot\_id: questo campo di default è *null*; assume valore quando l'operatore viene assegnato ad un robot, salvandone il relativo id.
- operator\_status: indica lo stato dell'operatore ovvero se è in turno oppure no.
- operator\_stress: campo inizialmente negativo che diventa positivo quando viene inviato un segnale di stress; è utile per mantenere aggiornate tutte le informazioni sullo stato di un operatore.

Sul topic */stress* il messaggio è così composto:

- operator\_id: indica l'id dell'operatore di cui si vuole segnalare una situazione di stress.
- operator\_stress\_alert: campo che viene posto a True quando si rileva stress mentre verrà posto a False quando la situazione tornerà alla normalità.

Sul topic */action* invece i campi contenuti nei messaggi sono:

- operator\_id: indica l'id dell'operatore a cui il manager vuole inviare il messaggio.
- robot\_id: indica l'id del robot a cui l'operatore viene assegnato.
- action\_type: indica la tipologia di messaggio che il manager sta mandando. Può essere di tipo task se si vuole inviare una nuova missione o stop se si vuole terminare quella corrente.
- task\_name: nel caso di messaggio di tipo task esso assume il nome del task da compiere altrimenti è *null*.

Questa uniformità dà la possibilità a chi vuole parlare agli operatori, di non dover conoscere tutti i diversi protocolli adottati ma esclusivamente l'unico protocollo utilizzato da tutte le interfacce digitali dei diversi ODT. Saranno poi gli ODT stessi ad occuparsi di tradurre le informazioni ricevute dalle altre entità sulla base delle caratteristiche del proprio operatore associato.

## 2.4 Robot Digital Twin

Come è facile intuire, il Robot Digital Twin è la rappresentazione digitale del robot che collabora con l'operatore. Anche in questo caso se si hanno più robot sarà necessario avere un DT per ciascun robot poiché l'associazione è sempre 1-1.

Al proprio interno si distinguono le due interfacce come per il Digital Twin dell'operatore.

L'interfaccia fisica è la parte del RDT che si mette in comunicazione con il robot fisico. Questa parte dovrà essere sviluppata appositamente per il robot che viene rappresentato utilizzando i suoi protocolli, i suoi topic e i suoi formati messaggio ma la struttura in generale è composta come in Figura 2.6, se si suppone un robot manipolatore.

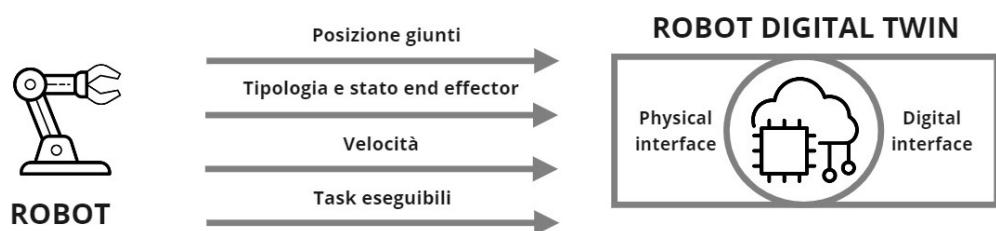


Figura 2.6: Messaggi scambiati tra robot e RDT

Quello che solitamente viene mandato dal robot al DT è:

- Posizione dei giunti: indicano come il robot è posto nello spazio in tempo reale.
- Tipologia e stato dell'end effector: indica quale strumento il robot utilizza come parte finale del braccio e in quale stato esso si trova (es. aperto/chiuso).
- Velocità: indica la velocità con cui il robot e, quindi, i giunti si muovono nello spazio.
- Task eseguibili: come per l'operatore, ciascun robot è programmato per eseguire una serie limitata di lavori che il manager deve sapere.

Anche in questo caso le varie informazioni vengono mandate quando il robot si collega la prima volta, ma poi saranno re-inviate ogni volta che vi sarà un aggiornamento di un qualsiasi campo.

Viceversa, ogni Robot Digital Twin, dopo aver ricevuto i segnali dal Plant Manager, deve tradurli in modo che il robot a lui associato possa capirli e poi inoltrarli. Questa traduzione è necessaria perché ogni robot, come ogni operatore, può usare un protocollo e un formato dei messaggi diverso dagli altri e quindi per farsi capire ma anche per capire il robot stesso dovrà adattarsi alle sue impostazioni.

Come riassunto in Figura 2.7, le informazioni che possono essere contenute nei messaggi dal RDT verso il robot sono le seguenti:

- Indicazione dell'operatore a lui assegnato.
- La lista delle coordinate dove il robot dovrà andare a posizionarsi per prelevare pezzi o per fare altri tipi di lavorazioni.

- Le coordinate in cui il robot si dovrà posizionare ogni volta che o è in attesa o ha finito una lavorazione.
- Un segnale per avviare una nuova missione o stoppare quella corrente.
- Un segnale per modificare la velocità del robot con all'interno la nuova velocità da adottare.

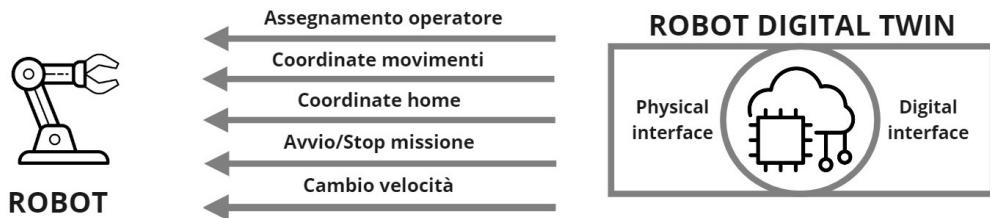


Figura 2.7: Messaggi scambiati tra RDT e robot

L'interfaccia digitale del RDT, invece, si occupa come sempre della comunicazione verso il Plant Manager.

Il RDT, riuscendo a capire il robot in che situazione si trova grazie agli aggiornamenti continui, può gestire i messaggi in arrivo dal PM ma soprattutto quelli in uscita. Esso riesce a inviare soltanto le informazioni essenziali di cui la gestione centrale necessita senza sovraccaricare la comunicazione.

Anche per questa tipologia di Digital Twin l'interfaccia digitale è univoca e questo facilita ulteriormente il lavoro del Plant Manager. Il protocollo scelto per la comunicazione è anche in questo caso MQTT per rimanere coerenti con le scelte fatte finora, ma soprattutto per avere un basso consumo e un'alta velocità.

Gli scambi tenuti tra Robot Digital Twin e il PM sono rappresentati in Figura 2.8

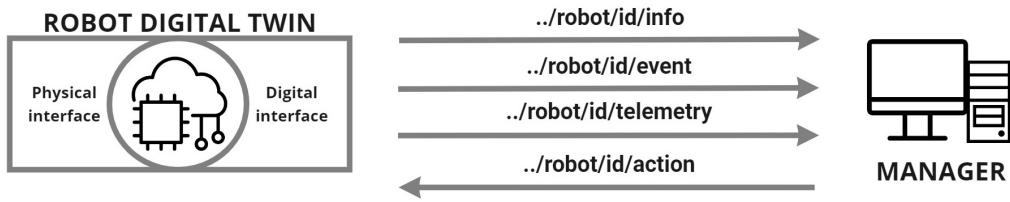


Figura 2.8: Messaggi scambiati tra RDT e PM

I topic utilizzati dall’interfaccia digitale sono quattro e le loro funzionalità possono essere descritte come segue:

- *robot/id/info*: serve, come per l’ODT, per segnalare la presenza del robot al manager e mantenere aggiornato il suo stato.
- *robot/id/event*: viene usato per segnalare particolari azioni o eventi del robot al manager. Su questo topic vengono, ad esempio, mandati dei messaggi durante l’esecuzione dei task per dare aggiornamenti sull’operazione, ma anche messaggi di errore quando qualcosa non funziona correttamente.
- *robot/id/telemetry*: viene utilizzato per mandare la telemetria del robot. Attraverso questo topic ogni minima variazione della posizione dei giunti e dello stato dell’end effector del robot viene segnalata dal relativo DT in tempo reale cosicché il manager o chi è in ascolto sappia esattamente come è posto nello spazio.
- *robot/id/action*: ha una funzione simile al topic action dell’operatore ma ha anche uno scopo aggiuntivo: su questo canale, infatti, il manager può mandare messaggi di avvio missione, con anche le relative coordinate, di stop della missione, e viene utilizzato anche per segnalare al robot di ridurre la velocità.

Per ogni RDT sono definiti quattro topic, ognuno dei quali è identificato dall'id del robot che esso rappresenta.

Sul topic `/info` il Robot Digital Twin invierà un messaggio composto da più campi:

- `robot_id`: contiene l'id del robot che il DT rappresenta. Campo ridondante come nel caso dell'ODT.
- `robot_speed`: valore della velocità a cui si sta muovendo il robot
- `robot_action`: lista dei task che il robot può compiere.
- `robot_current_action`: se il robot è in funzione questo campo ha il valore del task che sta eseguendo altrimenti è `null`.
- `robot_operator_id`: anche questo campo è di default `null` ed assume valore quando il manager assegna un operatore al robot che il DT rappresenta.

Per il topic `/event`, invece, i campi contenuti nel messaggio sono:

- `event_type`: campo che identifica il tipo di evento che si sta comunicando con tale messaggio. Può ad esempio essere un avviso di avvio o di stop ma anche un errore di esecuzione del task richiesto.
- `metadata`: in base alla tipologia di evento questo campo serve per dare maggiori informazioni su quanto accaduto.
- `timestamp`: contiene l'indicazione temporale di quando è stato inviato il messaggio per facilitare il manager.

Sul topic `/telemetry` il messaggio è composto da:

- `robot_id`: campo con l'identificativo del robot.

- robot\_speed: campo che indica la velocità attuale del robot.
- robot\_joints: campo che contiene tutte le posizioni dei giunti del robot.
- robot\_state\_end\_effector: questo campo indica in che stato si trova l'end effector.

Infine sul topic */action* il manager invia un messaggio che può avere diverse funzioni in base ai valori che assumono i campi contenuti che sono:

- robot\_id: identificativo del robot a cui si riferisce.
- action\_type: indica la tipologia di messaggio che si sta mandando. Può essere di tre tipi: "task" che indica una nuova missione da eseguire, "stop" che indica, invece, la volontà di fermare tale robot e infine "speed" che indica la volontà di modificare la velocità del robot a cui ci si riferisce.
- task\_name: questo campo è diverso da *null* solo in caso di nuova missione e assume il nome del task che il robot dovrà eseguire.
- speed: questo campo imposta la velocità di movimento del robot. È 100 in caso di nuova missione, 0 in caso di stop e variabile nel caso di messaggio di modifica.

I successivi quattro campi hanno di default valore *null* ma assumono significato nel caso si stia mandando un messaggio per avviare una nuova missione. In quel caso ogni campo avrà un significato preciso:

- movement\_type: questo valore definisce il tipo di movimento che il robot dovrà adottare.
- home\_pose: contiene le coordinate a cui il robot andrà prima dell'esecuzione e una volta finito il task.

- place\_pose: contiene le coordinate a cui il robot andrà a porre i pezzi lavorati o semplicemente selezionati.
- pick\_pose\_list: contiene la lista delle coordinate a cui il robot andrà per prelevare i pezzi da trasportare oppure per lavorarli sul posto.

Come già sottolineato questa omogeneità dell’interfaccia digitale sarà fondamentale per garantire efficienza nella gestione dei robot e delle loro lavorazioni perché, in questo modo, saranno gli intermediari, ovvero i Digital Twin, ad occuparsi delle differenze fisiche dei vari macchinari e non chi li coordina.

## 2.5 Plant Manager

Il Plant Manager all’interno di un sistema produttivo è l’entità che gestisce tutte le operazioni necessarie a garantire la massima funzionalità ed efficienza dei processi e delle infrastrutture presenti.

Si occupa dell’organizzazione del lavoro, assegnando task e stabilendo le tempestive opportune per ogni area produttiva. Inoltre, vigila sulla sicurezza degli operatori, agendo tempestivamente in caso di bisogno. I Digital Twin sono, per il Plant Manager, di grande aiuto e supporto nel raggiungere i suoi scopi.

Essi forniscono grandi quantità di dati facilmente accessibili che permettono non solo di capire come sta procedendo la lavorazione, ma anche come tendenzialmente potrebbe andare in futuro, dando quindi modo di anticipare possibili problematiche o errori.

L’altro vantaggio fornito dai DT è che essi possono essere sfruttati anche da remoto, permettendo al PM di non dover essere necessariamente nello stesso loro stabilimento per poter essere collegato con loro.

### **2.5.1 Caricamento missione**

Quando si vuole effettuare una nuova lavorazione è necessario caricare i dati di tale lavoro sul Plant Manager in modo tale che esso possa gestirlo nel migliore dei modi.

Questo caricamento viene sempre eseguito dall'esterno. Il PM, da solo, non ha conoscenza delle missioni da eseguire e di conseguenza egli di sua iniziativa non può avviare alcun macchinario. Il Manager può, ad esempio, recuperare le missioni da inviare ai robot da un database esterno oppure attraverso il caricamento di file.

Le informazioni da caricare sul PM sono tutte quelle informazioni che lui stesso utilizzerà per comandare i diversi robot attraverso il topic */action*, come ad esempio l'id del robot da comandare, l'azione da eseguire, le diverse posizioni in cui andare a porsi, ecc...

### **2.5.2 Gestione delle policy**

Quando il PM ha la necessità di assegnare un task, esso innanzitutto guarda se il macchinario indicato per fare tale lavoro è disponibile, ovvero se non sta eseguendo nessuna azione in quel momento. Questa azione è possibile da parte del PM poiché lo stato di ciascuna macchina viene aggiornato in tempo reale dal relativo DT, come visto in precedenza. La raccolta e l'analisi dei dati di produzione incide in modo importante su come il Plant Manager decide di gestire i carichi di lavoro.

Se tale macchina risulta essere disponibile allora si passa all'assegnamento dell'operatore altrimenti se non disponibile, si aspetterà un certo tempo prima di riprovare la richiesta.

La scelta dell'operatore non viene fatta dall'esterno, come per il robot, ma

se ne occuperà il PM stesso. Per effettuare questa scelta egli osserva quali operatori sono registrati presso di esso e, tra questi, sceglie il primo libero che sia in grado di realizzare il task richiesto. L'assegnamento del task potrebbe essere rimandato anche nel caso in cui non vi siano operatori disponibili oppure nel caso in cui quelli disponibili non siano in grado di compiere il task richiesto.

Dall'esterno il Plant Manager può ricevere anche dei segnali di stop per un determinato robot. In tal caso egli dovrà identificare il robot indicato e inviare ad esso un messaggio di tipo "stop" e contemporaneamente inviare lo stesso messaggio anche all'operatore assegnato in quel momento a tale robot.

Infine, l'ultima tipologia di azione che può mandare il Plant Manager è quella del cambio velocità. Questa tipologia di messaggio viene mandata ad uno specifico robot quando l'operatore associato al robot stesso segnala una situazione di stress. In questo caso, la velocità viene impostata alla metà della velocità originale per permettere all'operatore di continuare a lavorare in maggiore sicurezza.

### 2.5.3 Esempio del flusso comunicativo

Si può osservare un esempio completo del flusso della comunicazione nella Figura 2.9. Tale esempio riguarda una configurazione semplice, dove si ha un solo operatore e un solo robot, sebbene il manager sarebbe in grado di gestirne anche più contemporaneamente. Si possono notare le seguenti fasi:

- Registrazione di nuovi operatori e di nuovi robot presso il manager.
- Comunicazione del manager al robot per l'esecuzione di una nuova missione e all'operatore per assegnarlo al robot che dovrà eseguire la lavorazione.

- Durante l'esecuzione operatore e robot comunicano in tempo reale al manager le proprie informazioni aggiornate.
- Se durante l'esecuzione l'operatore dovesse entrare in una situazione di stress questa viene segnalata al manager.
- Il manager, recepito il segnale, invia al robot un comando per rallentare i movimenti.
- Quando il robot termina invia un segnale di stop al manager.
- Ricevuto il segnale di stop il manager libera l'operatore occupato sul robot appena terminato.

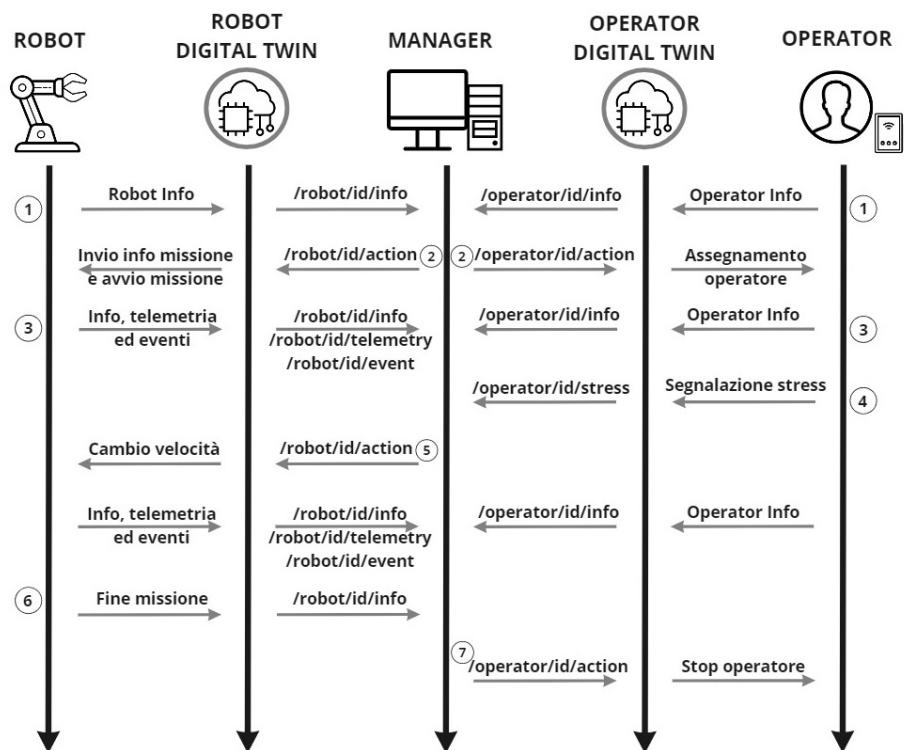


Figura 2.9: Esempio di flusso di comunicazione completo



# Capitolo 3

## Implementazione del sistema

Lo scopo di questo capitolo è identificare e descrivere il modello implementato sulla base dell'architettura progettata per questo caso d'uso.

Un primo schema che rappresenta l'implementazione è visibile in Figura 3.1

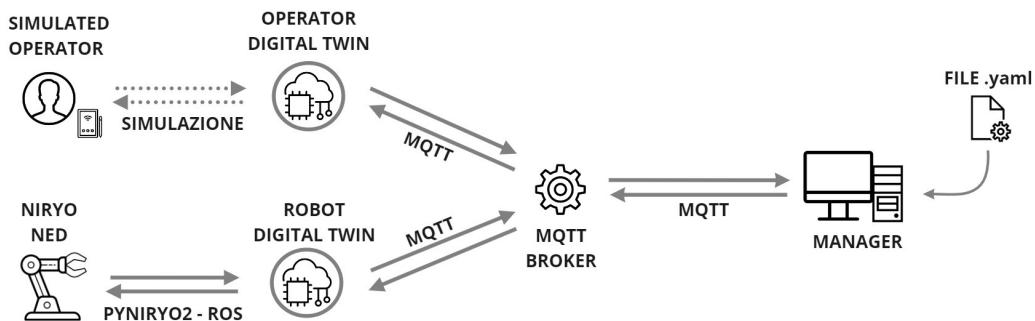


Figura 3.1: Implementazione architettura del sistema

Si può notare dalla figura che, oltre agli elementi introdotti nel capitolo precedente, è presente anche una figura in più: il broker MQTT. Come è stato anticipato, il protocollo di comunicazione utilizzato tra i Digital Twin e il Plant Manager è MQTT, comune per tutti i DT. Al fine di poter utilizzare tale protocollo correttamente è necessario avere la presenza di un broker che permetta lo scambio tra publisher e subscriber.

Si è deciso di mantenere momentaneamente il broker in locale in modo tale che esso possa essere raggiunto in modo veloce e in ogni momento, senza la necessità di una connessione a internet. Per la sua realizzazione si è utilizzato Eclipse Mosquitto che è uno dei broker MQTT open source più utilizzati. Rimane comunque la possibilità in ogni momento di passare a un broker non in locale, semplicemente cambiando l'indirizzo e la porta del broker al quale ci si vuole collegare.

Un'altra caratteristica che è valida per tutti gli elementi presenti, oltre al protocollo di comunicazione, è il formato dei messaggi. Il formato adottato da tutti nello scambio dei messaggi è JSON (JavaScript Object Notation). Dal momento in cui JSON permette di svincolarsi dal tipo di architettura e linguaggio di programmazione utilizzato, questo formato permette di avere messaggi compatti e leggeri ma, allo stesso tempo, leggibili e di facile utilizzo.

### 3.1 Implementazione dell'operatore

Come si può vedere dallo schema, un altro elemento differente rispetto al modello di architettura presentato in precedenza è il fatto di avere un operatore simulato.

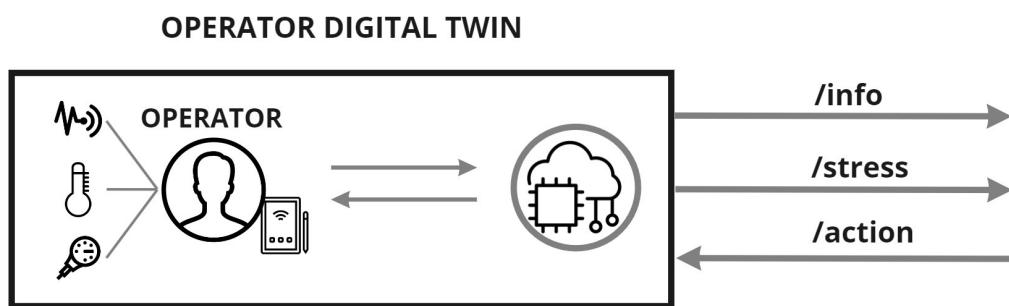


Figura 3.2: Simulazione dell'operatore

Questa differenza è data dal fatto che in fase di realizzazione si è deciso di non utilizzare una persona reale, ma simulare tale componente all'interno del relativo Digital Twin, come mostrato dalla Figura 3.2.

Questa scelta è stata fatta perché il progetto realizzato ha il proprio focus principale non sullo sviluppo dell'operatore e della sua comunicazione, ma sullo sviluppo dei DT delle risorse fisiche e sulla loro capacità di comunicazione e di mediazione; per semplicità di progettazione si è deciso, perciò, di rimandare l'implementazione reale di questa parte.

Quello che l'ODT simula sono, innanzitutto, i dati degli operatori che poi invia sul topic */info*, ma anche quei dati come temperatura, pressione e battito che portano alla segnalazione dello stress sul relativo topic. Inoltre, l'ODT si occupa di simulare anche lo stato dell'operatore quando riceve messaggi sul topic */action* per interagire correttamente con il manager.

## 3.2 Implementazione dell'Operator Digital Twin

Come già anticipato, l'implementazione dell'Operator Digital Twin ha una doppia funzione ovvero quella di simulare l'operatore e realizzare un DT che possa comunicare efficacemente con il Plant Manager.

Per raggiungere il primo scopo è stata definita la classe *OperatorDescriptor*. Tale classe contiene tutti i campi che un operatore, attraverso il proprio dispositivo, comunicherebbe al DT.

```
1 class OperatorDescriptor:
2     def __init__(self, operator_id, operator_level,
3                  operator_action, operator_current_action=None,
4                  operator_robot_id=None, operator_status=True):
5         self.operator_id = operator_id
6         self.operator_level = operator_level
```

```

5     self.operator_action = operator_action
6     self.operator_current_action =
7         operator_current_action
8     self.operator_robot_id = operator_robot_id
9     self.operator_status = operator_status
10    self.operator_stress = False

```

Listing 3.1: Classe operatore

Ogni volta che viene lanciato un nuovo Operator Digital Twin, questo al suo interno crea una istanza della classe OperatorDescriptor con informazioni sempre diverse, simulando un nuovo operatore disponibile che comunica le proprie informazioni al DT.

```

1 operator_descriptor = OperatorDescriptor("Operator_0001", "junior", ["Duck", "Tree"])

```

Listing 3.2: Istanziamento operatore

Se la parte di interfaccia fisica è stata semplificata, simulando i dati all'interno del DT, la parte digitale è invece articolata in più parti.

Come primo passaggio, è necessario istanziare un client MQTT che permetta all'ODT di comunicare. Per poter sfruttare il protocollo MQTT e poter istanziare dei client funzionanti in Python, il linguaggio utilizzato in questa implementazione, è necessario importare la libreria paho-mqtt.

E' importante che ad ogni DT corrisponda un client MQTT diverso con un id che sia univoco.

Quando il client si connette al broker allora esso esegue due azioni attraverso il codice riportato qui di seguito:

```

1 def on_connect(client, userdata, flags, rc):
2     publish_operator_info()
3
4     operator_action_topic = "{0}/{1}/{2}/{3}".format(

```

```

5     MQTTConfigurationParameters.MQTT_BASIC_TOPIC,
6     MQTTConfigurationParameters.OPERATOR_TOPIC,
7     operator_descriptor.operator_id,
8     MQTTConfigurationParameters.OPERATOR_ACTION_TOPIC)
9
10    mqtt_client.subscribe(operator_action_topic)

```

Listing 3.3: Azioni eseguite alla connessione del Client

Nella riga 2 del codice 3.3, il DT pubblica le prime informazioni dell'operatore che esso rappresenta sul proprio topic `/info`. Il messaggio che viene mandato è, come già detto, in formato JSON, ma la particolarità dei messaggi mandati su questo topic dal DT è che sono messaggi di tipo retained. Questa modalità è utile perché, in questo modo, il broker potrà inviare il messaggio salvato al PM che, per un qualsiasi motivo, si dovesse iscrivere appena dopo la registrazione di un operatore o robot al broker. Se non ci fosse questo tipo di messaggi, il PM, per sapere della loro presenza, dovrebbe aspettare l'aggiornamento successivo che potrebbe anche non arrivare. Il messaggio retained elimina l'attesa dell'aggiornamento successivo.

Alla linea 10 del codice 3.3, invece, il DT si sta sottoscrivendo al topic `/action` su cui il Plant Manager poi invierà i comandi da eseguire.

Il messaggio che riceve il DT è di una unica tipologia ma quello che cambia è il valore del campo `action_type`. Esso può essere:

- "Task": il manager vuole assegnare il proprio operatore al robot e al task indicato nel messaggio che si è appena ricevuto.
- "Stop": il manager vuole fermare il lavoro dell'operatore su un macchinario e renderlo disponibile per altri lavori.

Come si diceva prima, tutta la parte fisica è simulata, così come il rilevamento dello stress dell'operatore.

La segnalazione viene lanciata sulla base di un timer, il quale si avvia quando l'operatore viene assegnato ad un robot per l'esecuzione di un lavoro. Se il lavoro ha una durata maggiore del timer, la segnalazione verrà inviata al Plant Manager, altrimenti il timer verrà resettato attendendo la prossima attivazione. Tale segnalazione avviene tramite la pubblicazione sul topic `/stress` di un messaggio contenente un campo in particolare che è il campo `operator_stress_alert`, il quale è un booleano che viene posto a True nei casi di stress.

### 3.3 Implementazione del robot

All'interno di questo progetto, per la realizzazione della parte fisica relativa al robot, si è deciso di utilizzare il Niryo Ned [6], mostrato in Figura 3.3.



Figura 3.3: Niryo Ned

Il Niryo Ned è un CoBot a 6 assi ed è principalmente usato per scopi educativi o di ricerca, dal momento che ha capacità ridotte. Esso, infatti, può portare fino a 300 grammi e può raggiungere una distanza massima di 440 mm. Tali valori sono piuttosto limitati per un uso industriale, ma utili per effettuare simulazione di casi reali.

Il robot utilizzato è dotato, come end effector, di un gripper attraverso il quale il robot potrà effettuare e simulare facilmente operazioni di "pick and place".

Il Ned è basato su Ubuntu 18.04 e ROS (Robotic operating system). Sono entrambi sistemi open source che hanno permesso lo svilupparsi di diverse modalità di connessione e controllo come mostrato in Figura 3.4.

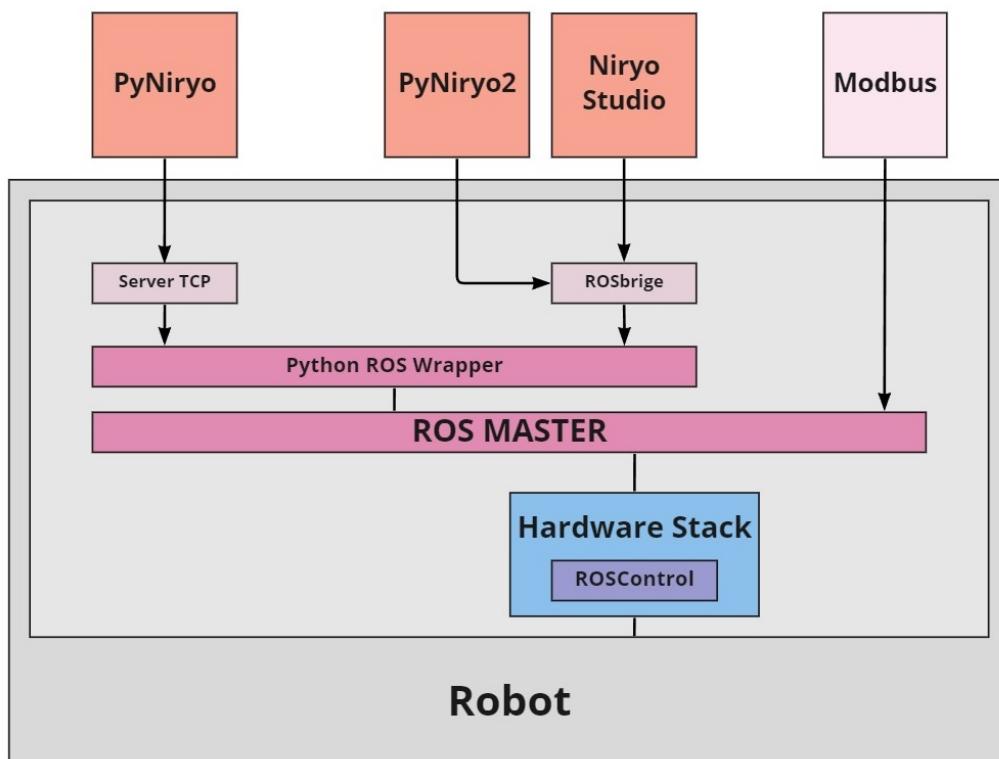


Figura 3.4: Overview globale del software di Niryo Ned

Si possono, ad esempio, utilizzare le seguenti soluzioni:

- ROS: è sicuramente la soluzione che permette un controllo maggiore perché non si ha intermediari con il sistema centrale, ma richiede una conoscenza approfondita del robot e di come funziona ROS e la sua comunicazione.
- Modbus: è una soluzione più specifica che permette di far parlare il robot con altri strumenti che parlano questo linguaggio, come i PLC.
- Niryo Studio: è una applicazione software che permette di interagire con il robot attraverso una interfaccia semplice ma completa. Attraverso di essa si possono impostare parametri, muovere il robot oppure programmarlo per una determinata missione tramite la programmazione a blocchi.
- PyNiryo / PyNiryo2: sono due librerie Python che forniscono delle API attraverso le quali ci si può connettere al robot e controllarlo facilmente.

Per il progetto in esame la scelta è stata PyNiryo2, come da Figura 3.5. Tale scelta è stata fatta perché, essendo la libreria basata al proprio interno su `roslibpy`, essa non richiede né una connessione attraverso terminale al robot né che si abbia installato un ambiente ROS sul dispositivo con il quale ci si vuole connettere al Ned, cosa che invece succedeva con le altre soluzioni. La connessione in questo caso avviene tramite WebSockets. Questo è possibile perché sul Niryo Ned è presente il pacchetto `rosbridge` di ROS, il quale è sempre in esecuzione e permette di far comunicare il sistema ROS con gli altri sistemi attraverso questa tecnologia.

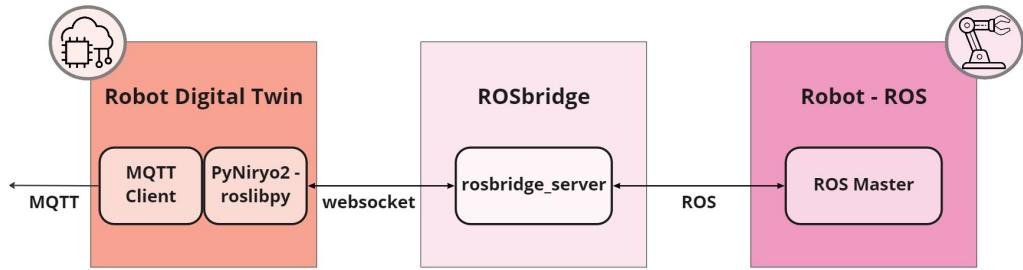


Figura 3.5: Implementazione comunicazione RDT - Robot

### 3.4 Implementazione del Robot Digital Twin

Dopo aver analizzato come la parte fisica relativa al robot è stata implementata all'interno del progetto, in questo paragrafo tratteremo del suo Digital Twin e di come le due entità sono collegate.

All'avvio, l'RDT esegue due azioni principali:

- Creazione di un client MQTT e della sua connessione al broker locale.
- Connessione al robot fisico e il suo l'avviamento.

La prima azione riproduce quanto già descritto per l'Operator Digital Twin, dovendo costruire anche per il DT del robot una comunicazione con il manager. Per quanto riguarda la seconda azione, invece, essa è necessaria per poter inviare i comandi e ricevere gli aggiornamenti dal robot.

```

1 ##### Connessione
2 robot = NiryoRobot(robot_descriptor.robot_ip)
3 publish_robot_info()
4 publish_robot_event("Start", "Robot connected")
5
6 ##### Calibrazione
7 robot.arm.calibrate_auto()
8 robot.tool.update_tool()
9 publish_robot_event("Calibration", "Robot calibrated")

```

```
10  
11 robot_descriptor.joints = robot.arm.get_joints()
```

Listing 3.4: Connessione e avviamento del robot

In queste righe di codice, quello che viene mostrato è la connessione e la calibrazione del robot sempre sfruttando la libreria PyNiryo2.

Per effettuare la connessione è necessario conoscere l'indirizzo IP del robot. Tale indirizzo dipende dal caso d'uso: si può collegare il robot in rete e quindi utilizzare l'indirizzo a lui assegnato, ci si può collegare al robot tramite cavo Ethernet oppure sfruttando la rete Wi-Fi che il robot crea. Ovviamente solo la prima soluzione è realmente efficiente perché non obbliga la presenza fisica affianco al robot ma per praticità d'uso in fase sperimentale si è preferito sfruttare la rete che il robot creava quando acceso. Nel codice qua riportato l'indirizzo è salvato nel campo `robot_id` all'interno di `robot_descriptor` che è la rappresentazione virtuale del robot all'interno del DT.

Una volta effettuata la connessione, se il robot ne ha necessità viene eseguita la calibrazione con il comando a riga 6 mentre a riga 7 il comando serve per far aggiornare al robot le informazioni sull'end effector montato.

Si può notare già da queste prime righe come ogni azione che il RDT fa nei confronti del robot è subito seguita da una comunicazione con il broker MQTT e quindi con il Plant Manager. Questa sincronia è importante per mantenere aggiornate tutte le informazioni su tutte le entità.

Alla riga 3 viene utilizzata la funzione `publish_robot_info` la quale, come nel caso dell'ODT, pubblica sul topic `/info` relativo al robot un messaggio retained con tutte le informazioni aggiornate sul robot.

Alla riga 4 e in altre righe successive, invece, la funzione `publish_robot_event` serve per pubblicare sul topic `/event` i passaggi importanti del robot durante la sua attività, come in questo caso, sono la connessione e la calibrazione.

Infine, nell'ultima riga riportata, si può vedere come il DT, dopo essersi connesso al robot, vada a leggere attraverso il comando `get_joints()` la posizione dei giunti del robot per salvarla all'interno della propria rappresentazione e rimanere così consistente.

Dopo aver eseguito queste operazioni iniziali di avviamento il Robot Digital Twin è pronto a fare da tramite e traduttore tra il Niryo Ned e il Plant Manager. Come l'ODT esso si mette in ascolto sul topic `/action` in attesa dei messaggi del manager i quali, come anche anticipato nel capitolo precedente, potranno essere di tre tipi:

- Task: per avviare una nuova missione.
- Stop: per fermare il robot.
- Speed: per modificare la velocità del robot.

Innanzitutto, quando arriva un nuovo messaggio sul topic viene controllato lo stato del robot se è in movimento o meno. Questo è importante perché se il messaggio dovesse contenere un nuovo task con il robot ancora in movimento allora il task deve essere rifiutato e segnalato tramite un evento. La stessa cosa vale se si ha il robot fermo e un nuovo messaggio "Stop" o "Speed".

Una volta fatto questo se si ha un messaggio di tipo "Task" si fa un ulteriore controllo ovvero si guarda se il robot è in grado di fare la tipologia di task richiesta.

Controllato quest'ultimo dettaglio allora il RDT si salva le informazioni riguardanti la missione come il nome del task da eseguire, l'operatore assegnato e la velocità aggiornando i propri dati e segnalando sul topic `/info` tale modifica e infine il RDT lancia il task scelto dal manager sul robot attraverso le seguenti istruzioni:

```
1 pick_pose_thread = Thread(target=pick_pose_task)
```

```
2 pick_pose_thread.start()
```

Listing 3.5: Lancio di un nuovo task

Con queste linee di codice si vuole andare a creare un thread parallelo a quello attualmente in esecuzione, il quale si occuperà di eseguire la funzione "pick\_pose\_task", la quale si occupa di mandare i comandi da eseguire al robot per portare a compimento il task. La necessità di creare un thread parallelo nasce dal fatto che, così facendo, non viene bloccata tutta la parte di comunicazione MQTT, la quale altrimenti rimarrebbe in attesa della conclusione del task.

Questa funzione chiamata ha due passaggi particolarmente importanti riportati nel Codice 3.6 e 3.7.

```
1 publish_telemetry_thread = Thread(target=
2   publish_robot_telemetry)
2 publish_telemetry_thread.start()
```

Listing 3.6: Lancio del Thread per telemetria

Il primo riguarda la parte di codice che avvia un ulteriore thread dedicato alla telemetria. Questo thread esegue una funzione in loop che scrive sul topic */telemetry* ogni volta che rileva una differenza di posizione dei giunti del robot fino a quando la missione non è completata o stoppata.

```
1 for pick_pose in action_robot_message.pick_pose_list:
2   if not stop_movement:
3     pick_n_place(pick_pose, action_robot_message.place_pose)
4     print("Pick and place " + str(pick_pose_number) + "
5       complete")
6     publish_robot_event("Pick and place complete", str(
7       pick_pose_number) + " complete")
8     pick_pose_number += 1
```

```
7     time.sleep(5)
```

Listing 3.7: Esecuzione pick and place

Il secondo passaggio invece è la parte di codice necessaria per eseguire il pick and place. In queste righe viene avviato un for che scorre tutte le posizioni di pick indicate all'interno della missione e, una alla volta, vengono passate alla funzione pick\_n\_place la quale è così realizzata:

```
1 # Going Over Object
2 robot.arm.move_pose(pick_pose_high)
3 # Opening Gripper
4 robot.tool.release_with_tool()
5 robot_descriptor.robot_end_effector = False
6 # Going to picking place and closing gripper
7 robot.arm.move_pose(pick_pose)
8 robot.tool.grasp_with_tool()
9 robot_descriptor.robot_end_effector = True
10 # Raising
11 robot.arm.move_pose(pick_pose_high)
12
13 # Going Over Place pose
14 robot.arm.move_pose(place_pose_high)
15 # Going to Place pose
16 robot.arm.move_pose(place_pose)
17 # Opening Gripper
18 robot.tool.release_with_tool()
19 robot_descriptor.robot_end_effector = False
20 # Raising
21 robot.arm.move_pose(place_pose_high)
```

Listing 3.8: Funzione pick\_n\_place

Tale funzione utilizza al suo interno le API messe a disposizione da PyNiryo2 per agire sul robot. Quello che viene fatto per ogni posizione passata sono le

seguenti azioni:

- Posizionamento sopra il punto desiderato con il comando move\_pose
- Apertura del gripper con release\_with\_tool
- Discesa sul punto desiderato sempre con la funzione move\_pose e chiusura del gripper con il comando grasp\_with\_tool
- Risalita
- Spostamento sopra il punto di rilascio
- Discesa sul punto di rilascio e rilascio del pezzo con il comando release\_with\_tool
- Risalita

Come si può notare ad ogni ciclo, prima di avviare un pick and place, viene controllato che il flag stop\_movement sia falso. Questo controllo è necessario perché questo flag indica se è arrivato un segnale di stop o meno. Come politica di gestione, infatti, si è deciso che quando il manager segnala ad un robot di fermarsi, questo non si blocchi istantaneamente ma termini il movimento di pick and place che stava compiendo e solo allora si fermi in attesa di altri comandi (per questo il controllo è ad ogni ciclo). Questo flag viene gestito sempre dal RDT sulla base dei messaggi di tipo "Stop" che esso riceve sul topic */action*. Se riceve un messaggio di stop egli definisce stop\_movement a True così da segnalarlo immediatamente al thread gestore del pick and place.

Dopo aver analizzato i messaggi di tipo "Task" e "Stop", il rimanente è il messaggio di tipo "Speed".

Come già anticipato il messaggio di cambio velocità viene inviato al RDT dal

manager in caso di rilevazioni di situazioni di stress e, di conseguenza, queste modifiche devono essere il più veloci possibili. Nel caso arrivi un messaggio su questo topic, il RDT analizza il messaggio, estrae la nuova velocità desiderata e infine, attraverso le API python, imposta la nuova velocità massima del robot associato. Una volta fatta questa modifica, le informazioni del robot verranno aggiornate e segnalate sul topic opportuno.

```

1 elif second_action_robot_message.action_type == 'speed':
2     print(second_action_robot_message.speed)
3     robot_descriptor.robot_speed =
4         second_action_robot_message.speed
5     robot.arm.set_arm_max_velocity(robot_descriptor.speed)
6     publish_robot_info()
    publish_robot_event("Change speed", "New speed set")

```

Listing 3.9: Cambio velocità robot

### 3.5 Implementazione del Plant Manager

L'ultimo componente rimasto da trattare è quello principale all'interno dell'architettura. Tale componente, come già detto, si occupa della gestione delle entità registrate e della distribuzione del lavoro da svolgere.

La prima azione del Plant Manager è istanziare un client MQTT con cui potersi poi sottoscrivere ai diversi topic, dove pubblicheranno sia i robot che gli operatori.

```

1 operator_info_topic = "{0}/{1}/+/{2}".format(
2     MqttConfigurationParameters.MQTT_BASIC_TOPIC,
3     MqttConfigurationParameters.OPERATOR_TOPIC,
4     MqttConfigurationParameters.OPERATOR_INFO_TOPIC)
5 operator_stress_topic = "{0}/{1}/+/{2}".format(
6     MqttConfigurationParameters.MQTT_BASIC_TOPIC,

```

```

7     MQTTConfigurationParameters.OPERATOR_TOPIC,
8     MQTTConfigurationParameters.OPERATOR_STRESS_TOPIC)
9     robot_info_topic = "{0}/{1}/+/{2}".format(
10        MQTTConfigurationParameters.MQTT_BASIC_TOPIC,
11        MQTTConfigurationParameters.ROBOT_TOPIC,
12        MQTTConfigurationParameters.ROBOT_INFO_TOPIC)
13
14     mqtt_client.subscribe(operator_info_topic)
15     mqtt_client.subscribe(robot_info_topic)
16     mqtt_client.subscribe(operator_stress_topic)

```

Listing 3.10: Sottoscrizione ai topic da parte del Plant Manager

Come si può vedere, il manager non si sottoscrive ai topic di un particolare operatore o robot ma effettua le sottoscrizioni attraverso l'utilizzo di wildcards ('+'). Tale elemento permette di iscriversi a tutti i topic che hanno quelle determinate strutture a meno della wildcard, la quale può essere invece sostituita da qualsiasi carattere o parola.

Una volta effettuata l'iscrizione, il manager si mette in ascolto e agisce in diversi modi in base ai messaggi che riceve.

```

1 if "/operator/" in message.topic:
2     if "/stress" in message.topic:
3         stress = StressOperatorMessage(**json.loads(
4             message_payload))
5         modify_speed_robot(operator_dict[stress.operator_id].
6             operator_robot_id)
7     else:
8         operator = OperatorInfo(**json.loads(message_payload))
9
10    operator_dict[operator.operator_id] = operator

```

Listing 3.11: Ricezione messaggi sul topic /operatore

Quando il manager riceve un messaggio sul topic */operator*, viene controllato se è un messaggio di info o stress.

Se il messaggio arriva sul topic */operator/id/info*, il manager si salva le informazioni aggiornate relative all'operatore. Queste informazioni vengono salvate dal manager all'interno di un dizionario chiave-valore, dove la chiave è l'id di ciascun operatore mentre il valore è l'istanza stessa appena creata con le nuove informazioni. In questo modo, se l'operatore non era ancora registrato, viene salvato, altrimenti viene sovrascritto con le nuove informazioni.

Se il messaggio arriva invece sul topic */operator/id/stress* allora il manager legge il messaggio ricevuto e manda il segnale di modifica della velocità del robot associato sul topic apposito.

```
1 if "/robot/" in message.topic:  
2     robot = RobotInfo(**json.loads(message_payload))  
3     if robot.robot_id in robot_dict and robot.  
        robot_current_action is None and operator_dict[robot_dict[  
            robot.robot_id].robot_operator_id].operator_current_action  
            is not None:  
                send_stop_operator(robot_dict[robot.robot_id].  
                    robot_operator_id)  
5     robot_dict[robot.robot_id] = robot
```

Listing 3.12: Ricezione messaggi sul topic */robot*

Quando invece il messaggio arriva sul topic */robot*, il manager effettua un controllo iniziale per capire se il messaggio arrivato è di un robot che si è appena registrato, uno che ha aggiornato le proprie informazioni oppure di un robot che ha terminato il proprio lavoro e lo sta segnalando. Nei primi due casi, come avviene anche per l'operatore, le informazioni vengono salvate all'interno del dizionario dei robot appositamente definito, mentre nel terzo

caso, se il robot segnala che ha terminato il proprio task, il manager invia un segnale di stop all'operatore che era associato a quel determinato robot fino a quel momento.

```

1 name_file_yaml = input("\nEnter configuration file name:")
2
3 with open(name_file_yaml) as f:
4     try:
5         dict_yaml = yaml.load(f, Loader=yaml.FullLoader)
6     except yaml.YAMLError as e:
7         print(e)
8
9 action_robot_message = ActionRobotMessage()
10 action_robot_message.get_from_yaml(dict_yaml)
```

Listing 3.13: Apertura file yaml e salvataggio dati missione

Oltre ai messaggi di stop e di cambio velocità di cui abbiamo già parlato, un'ulteriore tipologia di messaggio è quella di inizio nuova missione. Nella implementazione realizzata, la decisione di avviare una nuova missione avviene attraverso linea di comando. Al manager viene passato il nome di un file di configurazione, il quale poi viene letto e le informazioni vengono caricate all'interno di una istanza di ActionRobotMessage. Qui di seguito si può vedere un esempio di file utilizzato come configurazione di una nuova missione.

```

1 id: NiryoNed
2 action_type: task
3 task: Duck
4 speed: 100
5 movement_type: default
6 home_pose:
7   x: 0.0
8   y: 0
```

```

9   z: 0.0
10  roll: 0.0
11  pitch: 0.0
12  yaw: 0.00
13 place_pose:
14    x: 0.224
15    y: -0.015
16    z: 0.203
17    roll: -1.721
18    pitch: 1.508
19    yaw: -1.712
20 pick_list:
21  pick1:
22    x: -0.089
23    y: 0.339
24    z: 0.096
25    roll: -1.073
26    pitch: 1.481
27    yaw: 0.511
28 pick2:
29    x: -0.089
30    y: 0.339
31    z: 0.096
32    roll: -1.073
33    pitch: 1.481
34    yaw: 0.511

```

Listing 3.14: Esempio di file yaml passato al manager

Una volta ottenute tutte le informazioni sulla missione da eseguire questa deve essere assegnata al robot indicato e al primo operatore capace disponibile.

```

1 if (action_robot_message.robot_id in robot_dict.keys())

```

```

2     and robot_dict[action_robot_message.robot_id].
3         robot_current_action is None
4     and action_robot_message.task_name in robot_dict[
5         action_robot_message.robot_id].robot_action):
6         for op_dict in operator_dict.values():
7             if (action_robot_message.task_name in op_dict.
8                 operator_action and op_dict.operator_robot_id is None and
9                 op_dict.operator_status is True):
10                 action_robot_message.operator_id = op_dict.
11                 operator_id
12
13                 action_operator_message =
14                     ActionOperatorMessage(op_dict.operator_id,
15                         action_robot_message.robot_id, action_robot_message.
16                         action_type, action_robot_message.task_name)

```

Listing 3.15: Controllo e assegnamento missione

Come mostrato nel codice 3.15 quello che viene fatto è, innanzitutto, controllare se il robot richiesto per la missione è registrato, se è disponibile e se è in grado di eseguire quanto richiesto. Una volta controllato ciò a riga 4 viene eseguito un for per cercare il primo operatore disponibile che può essere assegnato. Anche per decidere l'operatore il manager deve controllare che sia libero e che il task richiesto sia da lui eseguibile.

Il primo operatore trovato il manager lo assegna al robot e invia a entrambi il messaggio di tipo "Task" sulla base di quanto indicato nel file caricato.

Non sempre l'esecuzione di una missione viene assegnata, può avvenire che il robot che viene richiesto nella configurazione non sia disponibile oppure che non vi siano operatori disponibili. In questi casi il manager rifiuterà la missione e si potrà caricare un altro task da eseguire.

Nella Figura 3.6 si può vedere alla sinistra del manager i due dizionari dove il PM salva operatori e robot. A destra, invece, è quello che il manager

realizza quando deve assegnare una nuova missione, ovvero l'assegnamento operatore-robot sulla base di chi è disponibile e in grado.

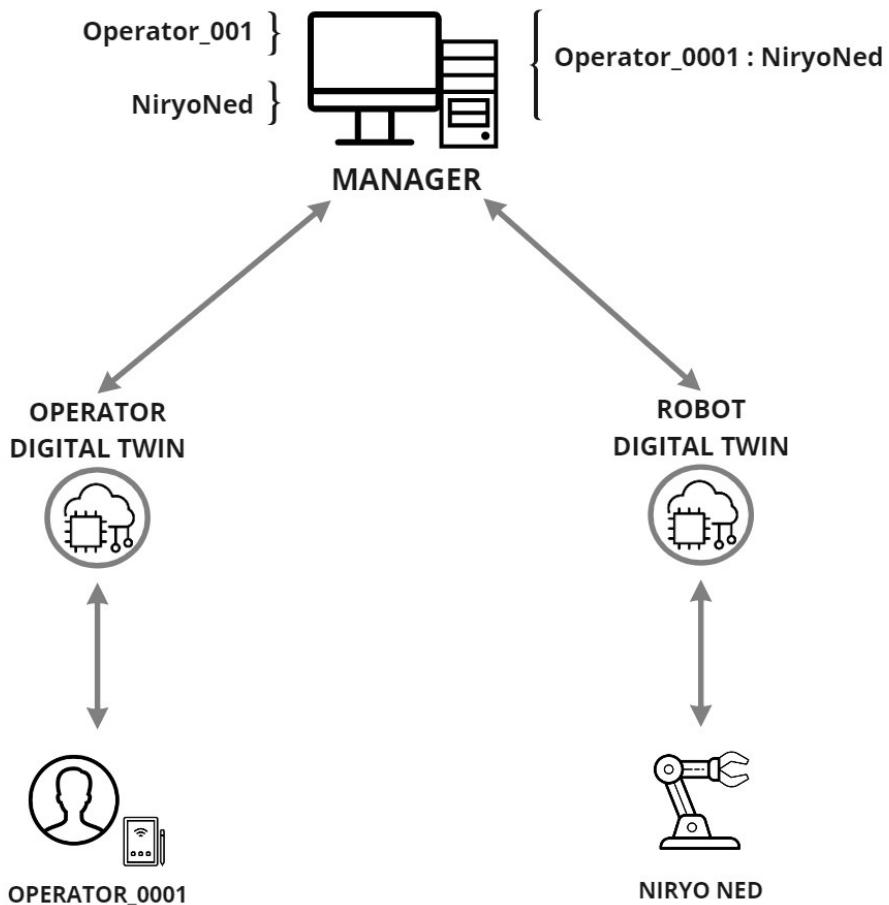


Figura 3.6: Implementazione Plant Manager

## 3.6 Validazione sperimentale

### 3.6.1 Scenario applicativo

Lo scenario descritto finora è stato progettato e in seguito implementato in due modalità diverse, le quali sono, in un certo modo, legate tra loro.

## Scenario virtuale

Una prima versione di questa architettura è stata implementata in uno scenario virtuale.



Figura 3.7: Scenario virtuale

Questa scena è stata realizzata in modo semplificato rispetto alla versione finale reale perché era una fase preparatoria a quella che sarebbe stata sviluppata dopo. Non sono presenti l'operatore e il relativo DT, ma solo un semplice manager esterno alla scena che gestiva le operazioni.

Si è scelto di partire ad implementare questo progetto all'interno di un ambiente virtuale per diversi motivi:

- Possibilità di interagire con il robot in sicurezza;
- Possibilità di effettuare più prove con diverse configurazioni dei task senza rischi di danneggiamento;
- Possibilità di lavorare e fare esperimenti sul robot anche senza la necessità di essere fisicamente accanto al robot;

- Ampie possibilità di migliorie future.

Come ambiente per la simulazione si è utilizzato Unity.

Unity è un motore grafico per lo sviluppo per lo più di applicazioni e videogame. In questo ambiente è possibile creare o inserire oggetti tridimensionali, impostare luci e materiali allo scopo di costruire la scena desiderata che, per noi, era il piano di lavoro con il robot per il pick and place.

Nel nostro caso la scena è stata costruita attraverso l'utilizzo di componenti prefabbricati e importando l'immagine del robot Niryo Ned da GitHub [10]. Una volta realizzata la scena si è deciso di controllare il robot attraverso l'utilizzo di ROS. Nativamente Unity non può parlare verso un ambiente ROS, ma è necessario un pacchetto aggiuntivo che si chiama ROS# [9].

ROS# è un insieme di librerie software e strumenti in C# per comunicare con ROS da applicazioni .NET, in particolare Unity. Questa libreria al suo interno ha un componente in particolare, il RosConnector, il quale permette di stabilire una connessione con il ROS master e comunicare quindi con i vari nodi ROS. Le modalità di comunicazione sono simili a quelle viste in precedenza con PyNiryo2, infatti anche ROS# si basa sull'utilizzo di WebSocket sfruttando la rosbridge\_suite presente lato ROS (vedi Figura 3.5). In questo modo, una volta indicato l'indirizzo IP su cui Unity e in particolare il RosConnector si deve collegare poi sarà il rosbridge ad occuparsi della traduzione. Data questa necessità di tradurre i messaggi nell'ambiente ROS è necessario definire i messaggi per la comunicazione con Unity anche per ROS e definire i topic corretti per l'invio e la ricezione dei messaggi.

Una volta fatti tutti questi passaggi, la comunicazione tra i due ambienti è molto semplice, basandosi su una messaggistica principalmente di tipo pub/sub e con una logica alla base simile a quella già spiegata ampiamente nel capitolo dedicato alla progettazione.

## Scenario reale

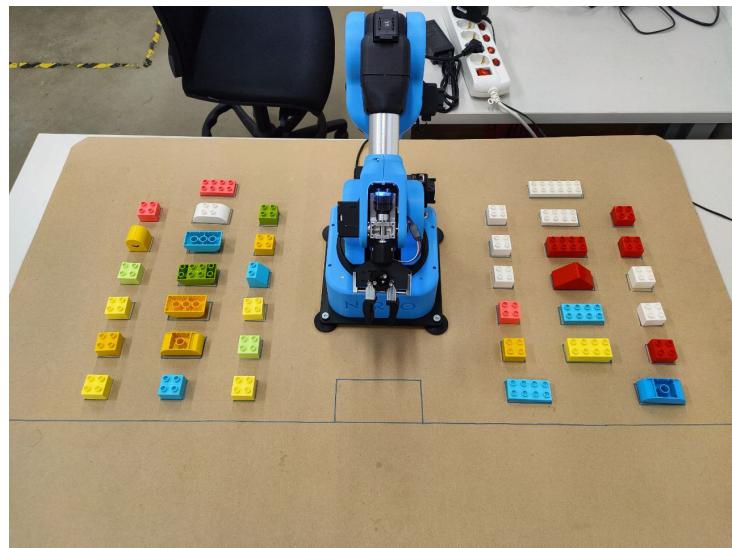


Figura 3.8: Postazione pronta

A differenza dello scenario virtuale, quello reale rispecchia maggiormente l'architettura progettata in precedenza.

Come già anticipato si ha un robot, il Niryo Ned, che compie delle operazioni di pick and place. In questa realizzazione le operazioni del robot serviranno a prelevare e depositare mattoncini "Duplo", in modo tale che l'operatore possa comporre una determinata figura.

La figura che dovrà realizzare l'operatore dipende dalla missione data, ma esistono diverse possibilità come si può vedere in Figura 3.9.

Quando il manager invia la missione al robot, esso, dopo tutti i controlli opportuni, inizia le operazioni di prelievo sulla base delle posizioni dei pezzi indicate all'interno della missione.

Anche l'operatore sa qual è la figura da realizzare e, supportato dal poster con tutte le costruzioni mostrato in Figura 3.10, riesce ad assemblare i pezzi nell'ordine corretto.



Figura 3.9: Figure realizzabili



Figura 3.10: Poster di aiuto

Come detto in precedenza, per la simulazione di un operatore stressato si utilizzerà un timer interno al processo dell'Operator Digital Twin per inviare il segnale. Quando questo segnale arriva al manager egli comanda al robot di rallentare. Questa modifica della velocità nello scenario realizzato è fatta facendo attendere il robot per più tempo rispetto al normale tra un prelievo e l'altro. Si è preferito fare questa modifica invece che ridurre la velocità di movimento perché questa nel Niryo è già piuttosto bassa ed avrebbe portato ad un rallentamento troppo consistente compromettendo la buona riuscita della prova.

Quando invece l'operatore passa la situazione di stress allora viene segnalato nuovamente e il manager ripristinerà nel robot le velocità originali.

### 3.6.2 Validazione e discussione

La realizzazione del progetto in due scenari differenti ha permesso di venire in contatto con diverse tipologie di tecnologie e di strumenti di lavoro e ha messo in mostra i punti forti e deboli dell'architettura che era stata progettata inizialmente.

Durante l'implementazione, ad esempio, ci si è accorti della mancanza di informazioni importanti per il funzionamento delle diverse entità o, al contrario, della presenza di informazioni superflue che appesantivano la comunicazione. In questi casi, si è andati a modificare la struttura in base a quanto visto arrivando così allo stadio attuale.

Un altro vantaggio che ha portato la realizzazione prima della scena virtuale e poi di quella fisica è che grazie alla simulazione si è potuto evitare inconvenienti e problemi nei primi approcci al robot dovuti all'inesperienza.

Un possibile sviluppo futuro potrebbe prevedere di integrare questi due scenari. L'idea è quella di fare in modo che mondo reale e virtuale si possano parlare, potendo così ricreare, all'interno della scena virtuale, quello che sta avvenendo nella realtà. Inoltre, se le due scene si riuscissero a parlare in tempo reale, potrebbe essere possibile interagire con la scena virtuale da remoto e, con una latenza minima, far sì che le modifiche fatte vengano applicate anche nel mondo reale.

Queste sono solo alcune delle potenziali implementazioni che potrebbero essere realizzate, ma si tratterà meglio dei possibili sviluppi futuri nel capitolo successivo.

# Capitolo 4

## Conclusioni e sviluppi futuri

### 4.1 Conclusioni

Questo lavoro di tesi aveva l'intenzione di mettere in evidenza ciò che la rivoluzione 4.0 e in particolare i Digital Twin stanno portando all'interno del mondo industriale.

Cercando di perseguire questo scopo, si è realizzato un progetto il quale, nella versione attuale, è costituito da un Plant Manager centrale e due entità, operatore e robot, con i relativi Digital Twin.

Già all'interno di questo semplice progetto si può vedere il potenziale dei DT. Durante l'implementazione, infatti, è stato possibile sviluppare i singoli componenti (manager, robot e operatore) quasi in modo indipendente, senza dover tenere in considerazione le altre parti. Questo è stato possibile perché tra di essi vi sarà sempre un intermediario, il Digital Twin, che controlla ciò che viene scambiato e si occupa di tradurlo, nel caso sia in formato diverso da ciò che può capire l'entità che rappresenta.

Questo esempio fa capire le potenzialità di questo strumento. In ogni momento, infatti, il progetto sviluppato finora può essere aggiornato e ingrandito

ma, grazie alla mediazione dei Digital Twin, non sarà necessario cambiare nulla di ciò che è già presente perché non saranno le diverse entità presenti a doversi adattare ma saranno i nuovi DT degli elementi aggiunti che si occuperanno di essere compatibili con il manager e gli altri strumenti.

## 4.2 Sviluppi futuri

Questo lavoro è stato un primo esempio di utilizzo dei Digital Twin all'interno di un contesto industriale, ma è comunque una versione primitiva e diversi sono gli aspetti migliorabili come anche le implementazioni aggiuntive che possono essere fatte.

Esempi di sviluppi futuri potrebbero essere:

- Implementare l'operatore utilizzando un dispositivo esterno (tablet, smartwatch o altro), che interagisca con il proprio DT al fine di avere una scena completa da cui partire per ulteriori modifiche.
- Implementare la segnalazione dello stress attraverso dei sistemi di intelligenza artificiale.
- Creare un contesto in cui si hanno più operatori e più robot fisici. Questa variazione finora è stata testata solo attraverso delle simulazioni, ma sarebbe opportuno provarla nella realtà.
- Realizzare un manager con maggiori capacità di elaborazione che possa prendere le informazioni necessarie non solo da semplici file yaml ma anche, ad esempio, da database esterni.
- Integrare altre tecnologie allo scenario, come ad esempio la realtà aumentata. Tale tecnologia permetterebbe all'operatore di essere ancora

più interconnesso con il manager e i robot presenti. Si potrebbero ad esempio utilizzare degli occhiali appositi dove compaiono le informazioni più importanti sulla base di ciò che sta guardando e dove l'operatore riceve direttamente cosa deve eseguire e su quale robot.

Queste elencate sono solo alcune delle prime funzionalità che si possono integrare, ma essendo l'ambito industriale estremamente ampio e vario, il progetto può essere declinato e adattato in diversi modi in base alle esigenze.



# Bibliografia

- [1] ISO. *ISO 10218-1:2011 – Robots And robotic devices – Safety requirements for industrial robots – Part 1: robots.* 2011.
- [2] ISO. *ISO 10218-2:2011 – Robots And robotic devices – Safety requirements for industrial robots – Part 2: robot systems and integration.* 2011.
- [3] Alexey Melnikov and Ian Fette. The WebSocket Protocol. RFC 6455, December 2011.
- [4] Roberto Minerva and Noël Crespi. Digital twins: Properties, software frameworks, and application scenarios. *IT Professional*, 23(1):51–55, 2021.
- [5] MQTT. MQTT website. <https://mqtt.org/>. Ultimo accesso: 28 Settembre 2022.
- [6] Niryo. Niryo Ned. <https://niryo.com/product/ned-education-research-cobot/>. Ultimo accesso: 28 Settembre 2022.
- [7] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Ng. *ROS: an open-source Robot Operating System*, volume 3. 01 2009.

- [8] M. Shafto, M. Conroy, R. Doyle, H. G. E., C. Kemp, J. LeMoigne, and L. Wang. *DRAFT Modeling, Simulation, Information Technology Processing Roadmap*. NASA, 2010.
- [9] Siemens. Github ROS# folders. <https://github.com/siemens/ros-sharp>. Ultimo accesso: 28 Settembre 2022.
- [10] Unity. Niryo Ned Unity. [https://github.com/Unity-Technologies/Unity-Robotics-Hub/tree/main/tutorials/pick\\_and\\_place](https://github.com/Unity-Technologies/Unity-Robotics-Hub/tree/main/tutorials/pick_and_place). Ultimo accesso: 28 Settembre 2022.

# Ringraziamenti

Vorrei dedicare le ultime righe a tutti coloro che mi sono stati vicini in questo percorso.

Un ringraziamento particolare va al mio relatore il prof. Marco Picone che mi ha seguito, con grande disponibilità, in ogni step della realizzazione dell'elaborato, fin dalla scelta dell'argomento.

Grazie anche alla mia correlatrice la prof. Valeria Villani per i suoi preziosi consigli e per avermi suggerito puntualmente le giuste modifiche da apportare alla mia tesi.

Grazie agli "Hackerz" e a tutti gli amici per aver potuto condividere con loro le difficoltà e le gioie di questi tre anni insieme.

Grazie ai miei genitori e alla mia famiglia per il loro grande sostegno nella scelta di questo percorso e per il loro accompagnamento fino a questo traguardo.

Ed infine grazie ad Agnese per essermi stata affianco in questi tre anni e non solo. Grazie per avermi capito, sostenuto nei momenti difficili e aver portato tanta pazienza.

Un grande grazie, ancora, a tutti.