**UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA**

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di laurea in Informatica

**Riprogettazione e ottimizzazione di software per l’interfaccia uomo-macchina nell’ambito dell’automazione industriale**

**Relatore:** **Candidato:**

Giacomo Cabri Enrico Marras

**Anno Accademico 2022-2023**

**Indice**

[**Introduzione** 3](#_Toc146732022)

[**Capitolo 1 – Contesto di sviluppo** 4](#_Toc146732023)

[**1.1** **Caratteristiche Hardware** 4](#_Toc146732024)

[**1.2** **Architettura di comunicazione** 5](#_Toc146732026)

[**1.3** **Caratteristiche Software** 6](#_Toc146732027)

[**1.4** **Protocollo di comunicazione** 6](#_Toc146732028)

[1.4.1 Modbus RTU 6](#_Toc146732029)

[**Capitolo 2 – Fase di analisi** 9](#_Toc146732030)

[**2.1** **Lentezza** 9](#_Toc146732031)

[**2.2** **Codice obsoleto** 10](#_Toc146732032)

[**2.3** **Sicurezza** 11](#_Toc146732033)

[**2.4** **Problematiche secondarie** 11](#_Toc146732034)

[**2.4** **Vincoli di sviluppo** 12](#_Toc146732035)

[**Capitolo 3 – Fase di progettazione** 13](#_Toc146732036)

[**3.1** **Tecnologie impiegate** 13](#_Toc146732037)

[**3.2** **Priorità comunicative** 14](#_Toc146732038)

[**3.3** **Task** 15](#_Toc146732039)

[**3.4** **Coda di priorità** 16](#_Toc146732040)

# **Introduzione**

In questo elaborato si vuole descrivere l’attività di sviluppo effettuata durante il tirocinio curriculare, mettendo in evidenza tutto il processo che ha portato alla produzione di una soluzione conforme alle necessità evidenziate.

Il tirocinio in questione si è focalizzato sullo sviluppo di una **Human Machine Interface**, in breve HMI, ovvero un software che permette a un utente di comunicare con una macchina, un programma o un sistema, attraverso un’interfaccia grafica[[1]](#footnote-1), comunemente utilizzato in ambito industriale.

L’automazione industriale si occupa dell’impiego coordinato di soluzioni tecnologiche allo scopo di ridurre la necessità dell’intervento umano[[2]](#footnote-2), specialmente per quanto riguarda operazioni ripetitive, complesse o pericolose.

Nonostante questo campo abbia avuto una significativa evoluzione grazie a metodologie come l’industria 4.0, durante il tirocinio si è constatato come alcuni di questi settori siano rimasti più legati a un paradigma di lavoro antiquato, che predilige una maggiore dipendenza dal lavoro manuale e una carenza di tracciabilità e/o sicurezze.  
Per questo motivo, tramite l’analisi delle problematiche e delle necessità del caso di studio, sono state adottate tecnologie e paradigmi moderni che hanno permesso il miglioramento dell’efficienza operativa anche in contesti precedentemente identificati come critici.

Nel primo capitolo verrà contestualizzata la situazione preesistente, fornendo informazioni sulle caratteristiche software, hardware e sul protocollo impiegati nel prodotto originale.

Nel secondo capitolo sarà finalizzato alla fase di analisi, ovvero la fase nella quale verranno evidenziati e analizzati i vari problemi riscontrati.

Nel terzo capitolo si passerà alla fase di progettazione, nella quale si mostreranno le principali soluzioni impiegate sia dal punto di vista delle tecnologie scelte, sia dall’effettiva riprogettazione della logica interna.

Infine, nel quarto capitolo verrà analizzata l’effettiva implementazione delle funzionalità presentate nel terzo capitolo.

# **Capitolo 1 – Contesto di sviluppo**

Il tirocinio è stato svolto presso la DOT S.n.c., una realtà del territorio emiliano che da quasi trent’anni si occupa principalmente di progettazione e produzione di prodotti a servizio dell’industria.

Durante il tirocinio è stato preso in analisi “MultiBench”, un prodotto consolidato per la gestione di una HMI attualmente utilizzata in più macchinari e sviluppato in Visual Basic 6.0.

È importante notare come questo software sia completamente funzionante secondo le necessità per le quali è stato originariamente progettato, e non mostri problematiche evidenti che ne comprometterebbero il conseguimento delle mansioni.

Siccome MultiBench presenta più versioni, la fase di analisi si è concentrata principalmente sulla versione installata sul macchinario a disposizione durante il periodo di tirocinio, ovvero la prima sviluppata.

* 1. **Caratteristiche Hardware**

Il macchinario oggetto di analisi è una stazione di assemblaggio dei componenti di motori, che permette di svolgere diversi compiti al fine di personalizzare la fase dell’assemblaggio in base ai requisiti tecnici, il tutto comandabile da un operatore.

Le funzionalità sopra citate vengono comandate da cinque controllori logici programmabili, in breve *PLC*. Essi hanno il compito di monitorare e salvare nei propri registri interni stati riportati dai sensori del macchinario e, all’evenienza, anche di pilotare le sue componenti.

Si noti come l’attività di riprogettazione si è focalizzata esclusivamente sulla parte software che si interfaccia direttamente con i *PLC*, e pertanto non sono state necessarie modifiche sulla loro programmazione in quanto completamente funzionanti.

Tra i principali componenti pilotabili dai controllori vi sono quattro assi, ovvero gli effettivi protagonisti dell’attività di assemblaggio dei motori.

Per consentire un collegamento tra i vari controllori al **livello fisico** dello *stack* ISO/OSI[[3]](#footnote-3), è stato usato lo standard per le comunicazioni seriali **RS-485** a **due fili**. Questo implica il suo utilizzo in modalità **half-duplex**, la quale prevede che le comunicazioni possano viaggiare indipendentemente dalla direzione, ma che solo un dispositivo alla volta possa trasmettere informazioni.

In accordanza a quanto consigliato dallo standard RS-485[[4]](#footnote-4), i vari controllori, referenziati con il nome di “*Drive*”, “*Inverter*” e “*Stepper*”, sono interconnessi attraverso una *daisy-chain* o più comunemente chiamata struttura a bus (figura 1).



Figura 1 – Struttura delle connessioni tra i vari dispositivi

* 1. **Architettura di comunicazione**

Per quanto riguarda la comunicazione, i vari controllori sono stati predisposti per lavorare in un’architettura di tipo **master-slave**, nella quale è sempre presente:

* Un **master.**
* Uno o più **slave**.

In questo tipo di architettura, “ogni scambio di informazioni è originato dal master, il quale invia […] sul bus una particolare richiesta”[[5]](#footnote-5).

Gli slave “sono normalmente in ricezione e ascoltano le richieste del master. Solo lo specifico slave interrogato cattura le informazioni inviate dal master […] e risponde inviando a sua volta le proprie informazioni sulla rete”[[6]](#footnote-6).

In questo tipo di predisposizione, il PC che esegue l’HMI prende il ruolo di *master*, mentre tutti gli altri controllori saranno gli *slave*.

**1.3 Caratteristiche Software**

Tra le principali funzionalità di MultiBench per questo macchinario sono presenti:

* Gestione dell’autenticazione su diversi profili utente in base alla tipologia di utilizzatore (ospite, operaio, manutentore, …).
* Continuo monitoraggio degli input/output del sistema e della comunicazione software-hardware.
* Visualizzazione del log eventi e feedback degli allarmi in tempo reale.
* Attuazione di movimenti manuali comandati da una figura umana.
* Configurazione dei parametri globali di riferimento.
* Creazione, salvataggio, caricamento ed esecuzione di programmi automatici di lavoro.

## **Protocollo di comunicazione**

Al fine di gestire tutte le comunicazioni tra i vari controllori è stato adottato lo standard comunicativo del protocollo **Modbus**. Seppur sia stato pubblicato originariamente nel 1979, rimane ad oggi uno dei protocolli di comunicazione più usati per connettere dispositivi elettronici industriali.

Il protocollo Modbus ha il compito di gestire le comunicazioni su più *layer* dello *stack* ISO/OSI.

Al livello ***Data Link****[[7]](#footnote-7)*, prevede tutte le specifiche relative allo scambio dei *frame* [[8]](#footnote-8) (sequenze di byte) tra un dispositivo e l’altro, le quali comprendono5:

* Invio dei dati sul bus.
* Controllo delle temporizzazioni.
* Controllo degli errori mediante *checksum*.

Al livello **Applicativo**, si occupa della codifica delle possibili richieste del *master* e le relative risposte degli slave all’interno dei *frame*.

Questo livello permette inoltre di interagire con le varie applicazioni in esecuzione sui dispositivi.

* + 1. Modbus RTU

Modbus è disponibile in tante varianti comunicative, tra cui *RTU*, *ASCII* e *TCP/IP*.

Per questo progetto è stata utilizzata la modalità **RTU** o *Remote Terminal Unit[[9]](#footnote-9)*.

Figura 2 – Struttra del frame Modbus in modalità RTU

Come si evince dalla figura 2, il messaggio Modbus è suddiviso in diversi campi[[10]](#footnote-10):

* *Address* indica l’indirizzo dello slave destinatario del messaggio e può variare da 1 a 247. Il valore 0 è riservato per messaggi in broadcast dove non verranno effettuate risposte.
* *Function* permette al destinatario di capire quale azione dovrà svolgere e al mittente di capire quale azione è stata svolta dallo *slave*. In seguito verranno approfondite nel contesto di questo caso di studio.
* *Data* contiene informazioni addizionali che lo slave deve sapere per portare a termine la richiesta indicata tramite il *function code*.
* *CRC*, o *Cyclic Redundancy Check* è un codice di controllo effettuato su tutto il frame per verificare la sua integrità dopo la trasmissione. Viene inizialmente calcolato dal master e successivamente dallo slave destinatario e, qualora il valore *CRC* calcolato dallo slave differisca da quello indicato dal master, il frame verrà scartato.
* Analogamente alla logica degli *Internal Frame Gap*, si ricorre a utilizzare i campi *start* ed *end* per cercare di fornire più garanzie sulla corretta ricezione del pacchetto prima dell’invio del successivo.
* Opzionalmente, è possibile anche aggiungere un bit di parità alla fine del messaggio che rappresenta un’ulteriore garanzia sullo stato dell’integrità del *frame*.

Per quanto concerne la codifica dei dati, Modbus utilizza una rappresentazione ‘*Big-Endian*’ per indirizzi e per l’effettivo contenuto informativo (ad eccezione del controllo CRC, che usa una codifica ‘*Little-Endian*’). Questo implica che all’invio di una quantità numerica superiore a un singolo byte, verrà inviato per primo il byte più significativo[[11]](#footnote-11).

È infine importante marcare che per garantire la corretta comunicazione tra i dispositivi è necessario assicurarsi che mittente e destinatario possano comunicare alla stessa velocità e che utilizzino la stessa struttura del messaggio Modbus.

# **Capitolo 2 – Fase di analisi**

Durante lo svolgimento del tirocinio, è stata posta molta importanza alla fase di progettazione in quanto è stato necessario individuare preventivamente le maggiori problematiche prima di passare all’effettiva pianificazione della soluzione da applicare.

Nel corso di questo capitolo, verranno esaminate in dettaglio le principali criticità riscontrate, le limitazioni progettuali e le tecnologie scelte.

## **2.1 Lentezza**

Tra le principali criticità d’uso identificate, una delle più significative ha riguardato la lentezza generale dell’interfaccia utente.

Questa ritardo si è presentato sia per le operazioni che richiedevano l’aggiornamento di elementi dell’interfaccia, ma anche, seppur in maniera minore, per tutte le interazioni tra l’utente e la macchina dove vi era la necessità di cambiare lo stato interno di registri attraverso l’HMI.

L’esempio più rilevante per questa tematica riguarda la parte superiore dell’interfaccia di MultiBench mostrata nella figura 3, la quale presenta una serie di indicatori che riportano misure inerenti a stati interni degli assi.



Figura 3 – Pannello superiore dell’interfaccia di MultiBench

Il ritardo dell’aggiornamento di queste misurazioni rappresenta un problema, vista la loro importanza all’interno del corretto funzionamento della macchina.

Una delle principali cause di questo ritardo di aggiornamento è dovuta all’inefficienza del *polling* per il controllo dei vari *widget*, ovvero “all’attività di campionamento attivo degli stati di un dispositivo esterno da parte di un programma client come un’attività sincrona”[[12]](#footnote-12).

Seppur rappresenti una soluzione adatta per software prevedibili e di piccole dimensioni, il *polling* non permette di gestire agilmente la potenza di calcolo[[13]](#footnote-13), talvolta sprecata non potendola fornire su richiesta delle attività che ne hanno bisogno.

## **2.2 Codice obsoleto**

Il codice sorgente di MultiBench si distanza notevolmente dai moderni paradigmi di programmazione in quanto, in primo luogo, non prende spunto da un *design pattern*, presentando una struttura talvolta monolitica e poco modulare.

Questa scarsa modularità la si può notare molto di frequente nelle parti di controllo del programma relative al *polling*, dove spesso molti stati di registri sono gestiti attraverso sequenze molto lunghe di *if-else* o di *switch* *case*. Seguendo questo paradigma di scrittura, nel caso dell’estensione di una *feature*, implicherebbe un aumento significativo della complessità di comprensione e di manutenzione del codice esistente, con la possibilità di introdurre errori difficili da individuare.

Per questo motivo, risulta evidente la necessità di avere una logica costruttiva che consenta agilmente l’aggiunta di *widget* o di controlli di stato senza l’appesantimento eccessivo del codice.

Seppur supportato già dalla versione 4.0[[14]](#footnote-14), al momento della scrittura del sorgente originale non è stato adottato un paradigma di programmazione a oggetti. Di conseguenza, MultiBench non presenta alcuna delle caratteristiche principali della *OOP[[15]](#footnote-15)*, tra cui si ricordano:

* Uso e divisione concettuale in classi.
* Polimorfismo.
* Ereditarietà.
* Incapsulamento.

Al posto di organizzare le funzionalità in classi con metodi e proprietà correlate, è stato originariamente utilizzato un approccio più procedurale che tendeva a complicare la comprensione tra le diverse parti del sistema.

Inoltre, l’assenza di ereditarietà e polimorfismo limitava la capacità di estendere e personalizzare le componenti grafiche per adattarle alle proprie esigenze.

In aggiunta a quelli che verranno trattati nel capitolo 2.3, la mancanza di incapsulamento poteva comportare problemi di sicurezza e la possibilità di accessi non autorizzati ad attributi sensibili.

## **2.3 Sicurezza**

Dal punto di vista meccanico, il macchinario preso in oggetto presenta molteplici accorgimenti sul punto di vista della sicurezza, come l’uso di una barriera protettiva, numerevoli interruttori di abilitazione, e vari sistemi secondari che forniscono garanzie anche in caso di fallimento dei primari.

Evitando di focalizzarsi in questa sede sulle caratteristiche fisiche o sulle vulnerabilità delle più vecchie versioni di visual basic, sarebbe il caso di porre l’attenzione sull’assenza di occultazione dei dati sensibili degli utenti da parte di MultiBench.

Poiché è stato necessario avere un sistema che garantisse l’utilizzo di specifiche funzionalità a precisi utenti, era stato originariamente creato un file contenente le loro informazioni necessarie.

Il file in questione era completamente visibile in chiaro, esponendo i dati degli utenti a potenziali rischi di accesso non autorizzato e in maggior modo inadatto a un paradigma di progetto sicuro per future interazioni con l’industria 4.0.

## **2.4 Problematiche secondarie**

Oltre a quelle precedentemente elencate, sono state identificate altre problematiche, sicuramente minori, ma che potrebbero comunque fornire spazio di miglioramento. Tra queste:

* Carenza di una documentazione esaustiva e talvolta precisa, che ha rappresentato una problematica per la manutenzione del codice, nonché una complessità maggiore nella riprogettazione.
* Programma originale unicamente legato all’ambiente Windows, in quanto utilizzatore del *runtime environment* necessario all’esecuzione di codice visual basic.
* Assenza di un vero paradigma di programmazione per *multithreading* che avrebbe permesso di sfruttare più efficientemente l’hardware messo a disposizione.

## **2.4 Vincoli di sviluppo**

Nel corso del progetto, sono stati imposti determinati vincoli che hanno influenzato le scelte e le direzioni prese nello sviluppo del sistema.

Anche se non si trattava di un obbligo progettuale formale, è stato necessario rispettare una limitazione temporale ben definita per l'intero sviluppo del software. La durata del progetto doveva essere rigorosamente contenuta entro il periodo di tirocinio, che corrispondeva a un totale di 375 ore.

Nonostante in fase progettuale si fosse discusso dell’idea di svincolare il prodotto da un preciso sistema operativo, l’azienda ha ritenuto più opportuno continuare lo sviluppo mirato alla distribuzione su ambiente Windows, in quanto più familiare e più adatto alle loro necessità. Ciononostante, è rimasto un obiettivo importante da tenere in considerazione nella scelta delle tecnologie impiegate, per non limitare future adozioni di altri sistemi operativi.

In aggiunta all’ambiente Windows, il prodotto finale avrebbe dovuto disporre anche di un *installer* che semplificasse l’installazione del software.

Infine, è stato vincolante evitare l’utilizzo di *framework* o librerie che richiedessero l’acquisto di licenze per uso professionale, ed è anche stato ritenuto fondamentale utilizzare tecnologie consolidate, per minimizzare la possibilità che il loro supporto cessasse nell’immediato futuro.

# **Capitolo 3 – Fase di progettazione**

In questo capitolo verranno discusse in primo luogo le tecnologie scelte e successivamente, seppur a un livello più astratto, le soluzioni metodologiche usate per risolvere i problemi precedentemente discussi.

## **3.1 Tecnologie impiegate**

Visto il contesto di sviluppo in parte di basso livello, si era inizialmente valutato il linguaggio C++, in quanto avrebbe garantito un’ottima gestione specifica della comunicazione, delle risorse e avrebbe permesso lo sviluppo non vincolato a un sistema operativo. Tuttavia, durante le prime prove, è stato notato come l’organizzazione e l’utilizzo di diverse librerie grafiche di C++ nell’ambiente Windows fosse complesso e poco agevole. Per questo motivo l’attenzione si è spostata sul linguaggio Python.

Seppur più lento per via della sua natura, Python avrebbe consentito una gestione molto più agevole delle librerie grafiche e, grazie alla sua maggiore astrazione, avrebbe permesso di completare con più garanzie le funzionalità precedentemente discusse entro il periodo di tirocinio.

Nativamente, Python non fornisce alcun supporto diretto per la comunicazione con il protocollo Modbus, tuttavia sono presenti molti moduli esterni alla libreria standard.

La scelta per questo ambito è ricaduta su MinimalModbus*[[16]](#footnote-16)*, un modulo *open-source* che implementa il protocollo Modbus e permette la comunicazione tra un computer (*master*) e gli strumenti (*slaves*)[[17]](#footnote-17).

Questo modulo fornisce diverse *API[[18]](#footnote-18)* di basso livello che consentono una vasta gamma di operazioni di comunicazione, incluso il controllo dei parametri ad essi associati. Infine, MinimalModbus soddisfa i vincoli di sviluppo posti dall’azienda poiché pubblicato la licenza Apache 2.0che permette, nel rispetto delle sue linee guida, l’uso commerciale e la distribuzione*[[19]](#footnote-19)*.

Per quanto riguarda la libreria grafica è stata scelta Tkinter, un *framework* attualmente incluso nella libreria standard di Python di relativamente facile utilizzo.

La scelta è ricaduta su Tkinter principalmente per via della sua stabilità, in quanto è una libreria grafica ampiamente utilizzata, leggera, documentata e già presente da diversi anni.

Benché Tkinter possa apparire datato dal punto di vista dell'aspetto grafico, ha comunque soddisfatto i requisiti del progetto, poiché non erano imposti vincoli riguardanti l'aspetto estetico dell'interfaccia utente.

In linea con MinimalModbus, Tkinter offre la possibilità d’utilizzo commerciale e la distribuzione dei prodotti che ne fanno uso come dipendenza in quanto rilasciato con la licenza Tcl/Tk[[20]](#footnote-20):

## **3.2 Priorità comunicative**

Come menzionato nel capitolo 2.1, durante la fase di analisi era stata evidenziata la necessità di migliorare la responsività generale del programma con particolare riferimento alle misurazioni indicate nella figura 3. Prendendo spunto dalla pagina “DIAGNOSI I/O #1” in figura 4, una delle interfacce di controllo più esigente dal punto di vista delle trasmissioni, è stata scelta una logica comunicazione con priorità.

A screenshot of a computer

Description automatically generated

Figura 4 – Interfaccia “DIAGNOSI I/O #1 di Multibench

La scelta è stata possibile per via di una minore necessità di aggiornamento di tutti i *widget* contenuti nella pagina che, in contrapposizione alle misurazioni presenti nel pannello superiore della figura 3, non avrebbero richiesto necessariamente un continuo aggiornamento tempestivo.

Questo significa che al posto di effettuare continue misurazioni su tutti i registri interessati dai *widget*, si sarebbero potuti effettuare più spesso campionamenti su quelli più “importanti” e più di rado su quelli secondari.

Conseguentemente a questa decisione, sono state stabilite due tipologie di priorità comunicative:

* **Alta priorità**: per tutte le comunicazioni inerenti a *widget* che necessitano di essere aggiornati tempestivamente e per tutti i cambi di stato che l’utente decide di apportare ai registri interni degli slave per mezzo di scritture.
* **Bassa priorità**: per tutte le comunicazioni inerenti a *widget* che non necessitano di un aggiornamento tempestivo e possono essere aggiornati in un secondo momento rispetto alle comunicazioni di alta priorità, senza pregiudicare il funzionamento del macchinario.

## **3.3 Task**

Al fine di organizzare al meglio ogni singola comunicazione, e quindi anche le priorità definite nel capitolo 3.2, è stata creata un’unità informativa composta da tutti i parametri riguardanti una singola trasmissione, il Task.

Internamente al Task, saranno quindi contenuti:

* **Priorità**: alta o bassa (3.2).
* **Porta**: identificatore della porta seriale (ad esempio “COM1”, “COM2”, … , “COM256” su Windows).
* **Slave-id**: Identificatore univoco dello slave.
* **Codice** **funzione**: codice dell’operazione Modbus da eseguire.
* **Indirizzo** **del registro**: indirizzo iniziale dell’operazione del Task.
* **Bytesize**: lunghezza del messaggio.
* **Baud-rate**: determina la velocità della comunicazione sul canale trasmissivo.
* **Bit di stop**: numero di bit di stop alla fine di ogni messaggio Modbus.
* **Bit di parità**: usato per determinare se i dati trasmessi sono stati ricevuti correttamente.
* **Timeout**: secondi dopo i quali il destinatario della comunicazione è considerato irraggiungibile.
* **Periodicità**: indicatore booleano per differenziare i Task da eseguire una singola volta da quelli periodici.

Questo tipo di organizzazione per le comunicazioni consente di avere oggetti più facili da maneggiare, in quanto una volta costruiti contengono già tutte le informazioni necessarie a un determinato compito, e pertanto non devono essere istanziati a ogni uso. Inoltre, agevolano il processo di ottimizzazione in quanto diventa più facile costruire e gestire dei Task fatti appositamente per un dispositivo rispetto a un altro.

Si prenda come esempio la velocità trasmissiva. È noto che in molti contesti applicativi si tende a sfruttare al massimo la velocità di trasmissione disponibile per ridurre al minimo il tempo necessario alle comunicazioni. In tal senso, ci si aspetta che tutti i dispositivi operino al massimo delle loro capacità di trasmissione, tuttavia ciò non implica che si possa sempre comunicare allo stesso modo. È questo il caso dei due *Inverter*, che possono trasmettere al più con un *baud-rate* di 38400 bit/secondo[[21]](#footnote-21), ben lontano dai 115200 bit/secondo[[22]](#footnote-22) di tutti gli altri dispositivi.

## **3.4 Coda di priorità**

Ora che si è definita un’unità informativa contenente i compiti da eseguire, è necessario definire una struttura dati che permetta la loro gestione. A questo scopo è stata scelta una coda di priorità, la quale ha i seguenti compiti:

* Permettere l’inserimento dei vari Task discriminando tra:
  + Task **periodici** ad **alta priorità**
  + Task **periodici** con **priorità normale**
  + Task **non periodici** ad **alta priorità**
* Gestire l’accodamento dei vari task periodici che sono stati eseguiti o alternativamente, la rimozione dalla coda dei task non periodici eseguiti.

Si noti come ai fini del progetto, non è stato ritenuto fondamentale discriminare anche tra i Task non periodici a bassa priorità, in quanto non rappresentano una tipologia di compito necessario alle necessità progettuali.

Al fine di permettere a Task diversi di eseguire più o meno frequentemente si definiscono i seguenti puntatori della coda:

* Un **puntatore all’ultimo elemento della coda,** o fondo, per gli inserimenti dei Task periodici con priorità normale.
* Un **puntatore al primo elemento della coda** o cima.
* Un **puntatore al secondo elemento con priorità normale a partire dalla cima della coda**. Nel caso in cui non ce ne siano o ce ne sia solo uno, punterà al fondo della coda. Quest’ultimo puntatore è utilizzato per gli inserimenti dei Task periodici ad alta priorità.

La priorità di esecuzione dei singoli Task è determinata dalla loro posizione all’interno della coda. In particolare, i Task più vicini alla cima nella coda saranno eseguiti prima di quelli più lontani.

Inoltre, l’effettiva priorità di un Task non garantisce alcun vantaggio di esecuzione tutt’ora che il compito è stato inserito nella coda, bensì viene utilizzato per **fornire una posizione più o meno vantaggiosa al momento dell’inserimento**.

Nella figura 4 si può osservare un generico stato della coda e dei suoi puntatori.



Figura 5 - Generico stato della coda e dei suoi riferimenti

Nei prossimi capitoli 3.4.1, 3.4.2 e 3.4.3, verranno analizzate le metodologie di inserimento e di vita di ogni singola tipologia di Task, utilizzando per brevità la notazione , con e .

* + 1. Vita di un Task periodico ad alta priorità

PUNTATORI: FONDO DELLA CODA? + indice di accodamento per task ad alta priorità

TASK SINGOLI NON PERIODICI CHE DEVONO ESSERE ESEGUITI UNO DIETRO L’ALTRO PER EVITARE PROBLEMI DI AGGIORNAMENTO

GESTIONE DEGLI ERRORI: MANCATA RISPOSTA DI UNO SLAVE (SKIP TASK) -> MAGARI DA METTERE NELL’IMPLEMENTAZIONE

VELOCITà TRASMISSIVE DIVERSE + REGISTRI NON INTERROGABILI

1. What is an HMI, Copadata, consultato il 10 Settembre 2023, <https://www.copadata.com/en/product/zenon-software-platform-for-industrial-automation-energy-automation/visualization-control/what-is-hmi/> [↑](#footnote-ref-1)
2. Automazione Industriale, Treccani, consultato il 9 Settembre 2023, <https://www.treccani.it/enciclopedia/automazione-industriale_%28Enciclopedia-Italiana%29/> [↑](#footnote-ref-2)
3. ISO/IEC 7498-1:1994, Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model, 7.7 Physical Layer [↑](#footnote-ref-3)
4. The RS-485 Design Guide, TI, consultato il 10 Settembre 2023, pagina 1-2, <https://www.ti.com/lit/an/slla272d/slla272d.pdf> [↑](#footnote-ref-4)
5. Protocollo Modbus su RS485 – Introduzione, Overdigit, consultato il 9 settembre 2023, pagina 3, <https://www.overdigit.com/data/Blog/RS485-Modbus/Protocollo%20Modbus%20su%20RS485.pdf> [↑](#footnote-ref-5)
6. What is the OSI Model?, Forcepoint, consultato il 9 Settembre 2023,   
   <https://www.forcepoint.com/cyber-edu/osi-model> [↑](#footnote-ref-6)
7. Introduction to Modbus Serial and Modbus TCP, Ccontrols, consultato il 10 Settembre 2023, pagina 1, <https://www.ccontrols.com/pdf/Extv9n5.pdf> [↑](#footnote-ref-7)
8. Unità informativa del livello 2 dello stack ISO/OSI. [↑](#footnote-ref-8)
9. Modbus RTU communication guide, Virtual-serial-port, consultato il 10 settembre 2023, <https://www.virtual-serial-port.org/articles/modbus-rtu-guide/> [↑](#footnote-ref-9)
10. Modbus Networking Guide, libelium, consultato il 10 settembre 2023, <https://development.libelium.com/modbus_networking_guide/introduction> [↑](#footnote-ref-10)
11. Unidrive M700 / M701 / M702 Guida dell'utente al controllo Versione numero: 2, 9.1.5 Codifica dei dati, pagina 122, consultato il 12 settembre 2023 [↑](#footnote-ref-11)
12. Polling (computer science), Wikipedia, consultato il 16 settembre 2023, <https://en.wikipedia.org/wiki/Polling_(computer_science)> [↑](#footnote-ref-12)
13. Polling, Teach-ict, consultato il 16 settembre 2023,

    <https://teach-ict.com/2016/A_Level_Computing/OCR_H446/1_2_software/121_operating_systems/interrupts/miniweb/pg2.php> [↑](#footnote-ref-13)
14. Programming Microsoft Visual Basic 6, Chapter 6 – Classes and Objects, Visualbasicbooks, consultato il 17 settembre 2023, <https://www.visualbasicbooks.com/progVB6samplepg1.html> [↑](#footnote-ref-14)
15. Object-oriented programming [↑](#footnote-ref-15)
16. MinimalModbus, Github, consultato il 25 settembre 2023, <https://github.com/pyhys/minimalmodbus> [↑](#footnote-ref-16)
17. Features, MinimalModbus, consultato il 25 settembre 2023, <https://minimalmodbus.readthedocs.io/en/stable/readme.html> [↑](#footnote-ref-17)
18. Application Program Interface [↑](#footnote-ref-18)
19. Apache License, Version 2.0, Apache, consultato il 25 settembre 2023, <https://www.apache.org/licenses/LICENSE-2.0> [↑](#footnote-ref-19)
20. Tcl/Tk License Terms, Tcl.tk, consultato il 26 settembre 2023,

    <https://www.tcl.tk/software/tcltk/license.html> [↑](#footnote-ref-20)
21. Commander SK Advanced User Guide – Issue Number: 8, consultato il 27 settembre 2023, <https://www.nidec-netherlands.nl/media/2125-frequentieregelaars-commander-sk-advanced-user-guide-en-iss10-0472-0001-10.pdf> [↑](#footnote-ref-21)
22. Unidrive M700 / M701 / M702 Guida dell'utente al controllo Versione numero: 2, 9.1.5 Codifica dei dati, pagina 121, consultato il 27 settembre 2023 [↑](#footnote-ref-22)