

UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

# Progettazione e Sviluppo di una Soluzione SCADA per Isole Robotizzate

## Relatore

Prof. Menegatti Emanuele

## Laureando

Quartucci Davide

## Correlatore

Pavarin Laura

ANNO ACCADEMICO 2024-2025

Data di laurea GG/MM/AAAA



*Alla mia famiglia e Erika*



# Sommario

La progettazione di un sistema SCADA completo è stata articolata in quattro fasi principali. In primo luogo, si è da subito condotta un'accurata analisi dei requisiti, con la partecipazione attiva di ingegneri, tecnici e programmatore PLC per poter definire con precisione le scelte progettuali. Successivamente, lo step fondamentale è stato strutturare l'HMI (Human Machine Interface), progettata utilizzando il software FactoryTalk Optix Studio della Rockwell Automation per garantire un'interfaccia intuitiva e funzionale. L'interfaccia è stata organizzata in sezioni, tra cui la sezione Home, che consente il monitoraggio dello stato del robot, e altre aree come Produzione per la gestione degli ordini e dei comandi, Anagrafica per la configurazione dei prodotti, Comandi Manuali per il controllo diretto dell'impianto, Allarmi per la gestione dei segnali del sistema e Impostazioni per la configurazione dei parametri operativi. La terza fase ha riguardato la gestione di tutti i dati, per la quale sono stati usati 2 database attraverso SQLite e SQLExpress. La scelta di utilizzare sia un database locale che uno connesso ha permesso di rendere il sistema adatto sia per le macchine più semplici che per gli impianti più complessi, in linea con gli attuali principi dell'Industria 4.0; di fatto il sistema di gestione dei dati è stato pensato per garantire riferimenti sempre aggiornati alle tabelle, che potessero comunicare con l'interfaccia. L'ultima fase della progettazione ha riguardato lo sviluppo della parte di back-end e l'integrazione con il PLC. In questa fase, l'intera logica del sistema è stata implementata utilizzando C# su vari script, che consentono la comunicazione tra HMI, database e PLC. Infine, è stato deciso di applicare lo SCADA a una cella DEMO di piegatura, dimostrando l'efficacia del sistema nella gestione automatizzata dei processi produttivi industriali.

*Il codice sorgente citato e utilizzato per questa tesi fa riferimento alla seguente repository:*  
[https://github.com/Quuartu/SCADA\\_v2.git](https://github.com/Quuartu/SCADA_v2.git)



# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Obiettivo di progetto . . . . .	1
1.2	L’azienda: REA Robotics Srl . . . . .	2
1.2.1	La Mission . . . . .	2
1.2.2	Settori e applicazioni . . . . .	2
1.2.3	Robot utilizzati . . . . .	3
<b>2</b>	<b>Fondamenti teorici sugli impianti di automazione e robotica industriale</b>	<b>5</b>
2.1	Contesto di Automazione industriale . . . . .	5
2.2	Composizione di uno SCADA . . . . .	6
2.3	Funzioni principali dello SCADA . . . . .	7
2.4	Piattaforma SCADA e Strumenti Software per la progettazione . . . . .	8
2.4.1	Lista Software utilizzati . . . . .	9
2.4.2	Struttura Generale di FactoryTalk Optix™ . . . . .	10
<b>3</b>	<b>Progettazione del sistema SCADA</b>	<b>11</b>
3.1	Analisi dei requisiti del cliente . . . . .	11
3.2	Strutturazione HMI (Human Machine Interface) . . . . .	12
3.2.1	Considerazioni tecniche sulla risoluzione . . . . .	15
3.3	Gestione Database per la produzione . . . . .	15
3.3.1	Anagrafica Articoli . . . . .	15
3.3.2	Ordini di Produzione . . . . .	20
3.3.3	Storici Produzione/Allarmi . . . . .	24
3.3.4	Configurazione Database aziendale . . . . .	26
3.4	Sviluppo back-end e integrazione con il PLC . . . . .	28
3.4.1	Indicizzazione tag variabili . . . . .	28
3.4.2	Sviluppo Macchina a Stati . . . . .	30

<b>4 Caso Studio: Isole robotizzate di piegatura</b>	<b>43</b>
4.1 Importazione Tag . . . . .	43
4.2 Configurazione variabili . . . . .	44
4.2.1 Gestione FTP Robot, FTP Pressa e passi 50, 55 e 60 . . . . .	46
<b>5 Conclusioni</b>	<b>49</b>
<b>Bibliografia e Sitografia</b>	<b>51</b>
<b>Ringraziamenti</b>	<b>53</b>

# Elenco delle figure

1.1	Logo REA Robotics. Fonte: [1]	2
2.1	Esempio sistema SCADA con componenti principali. Fonte: [5]	7
2.2	Gestione impianto tramite SCADA	8
2.3	Esempi software per progettazione HMI	8
2.4	Illustrazione dashboard di progetto	10
3.1	Suddivisione interfaccia HMI	13
3.2	4 illustrazioni dell'HMI	14
3.3	Illustrazioni sezione <i>Articoli</i>	16
3.4	Illustrazioni sezione <i>Produzione</i> e sottosezione <i>Nuova Produzione</i>	21
3.5	Esempio richiesta di <i>Extra Produzione</i>	23
3.6	Illustrazione sezione <i>Storico Produzione</i>	24
3.7	Illustrazione pannelli presenti in FT Optix™	26
3.8	Illustrazione sezione <i>Allarmi</i>	27
3.9	esempio di salvataggio percorso variabile di modello	29
4.1	Illustrazione Tag importati	44
4.2	Illustrazioni Gestione FTP Robot ed FTP Pressa	46
4.3	Prova messa in funzione dello SCADA su pannello prova ASEM	47



# Capitolo 1

## Introduzione

### 1.1 Obiettivo di progetto

La presente tesi è stata sviluppata a partire da una task affidatami durante la mia esperienza di tirocinio presso l'azienda REA Robotics nel periodo compreso tra il 2 settembre 2024 e l'11 ottobre 2024. Nello specifico, il team mi ha chiesto di creare una piattaforma SCADA per impianti custom che potesse rispettare linee guida moderne, attraverso l'utilizzo di un nuovo software di sviluppo e gestione. La principale necessità dell'azienda era di favorire l'integrazione tra più impianti di diversa tipologia per poter creare un ecosistema modulare, che funzionasse su tutte le macchine, indipendentemente da:

- Tipologia di impianto
- Hardware utilizzato
- Sistema operativo installato
- Modalità d'uso del sistema

Questo approccio mira a standardizzare le interazioni tra i diversi impianti, semplificando gestione e migliorando l'efficienza per gli aggiornamenti. Sin da subito il tirocinio era incentrato sull'affiancamento al team di sviluppo software di REA, per poter conoscere le metodologie di sviluppo interconnesso tra ingegneri SCADA e tecnici PLC, con l'obiettivo di imparare a collaborare in modo efficiente in team ed acquisire una prospettiva concreta del contesto lavorativo. Inoltre, un altro obiettivo è stato di ampliare le mie competenze tecniche nel campo dell'automazione industriale, con particolare attenzione alla progettazione di sistemi SCADA. Grazie al progetto, ho avuto l'opportunità di approfondire l'uso di strumenti specifici, come il software FactoryTalk Optix™, e di consolidare conoscenze già acquisite durante il percorso di

studi, come la gestione di database e la progettazione di interfacce HMI. Un altro aspetto cruciale è stato sviluppare competenze trasversali, tra cui comunicazione efficace e risolvere problemi complessi adottando un approccio analitico in linea con le esigenze aziendali.

## 1.2 L’azienda: REA Robotics Srl



Figura 1.1: Logo REA Robotics. Fonte: [1]

Il gruppo REA opera da oltre 30 anni nel contesto industriale nazionale. Agli albori partner tecnologico dei primi costruttori di robot europei, nei primi anni si è focalizzata sul settore tradizionale della saldatura per poi estendere la propria esperienza alla General Industry: dall’automotive alle fonderie, fino ai giorni nostri includendo plastica e vetro. Oggi REA conta un organico in grado di coprire le principali aree di un’azienda moderna di automazione robotizzata, con particolare risalto alla Ricerca&Sviluppo, all’avanprogettazione, oltre a disporre di tecnici con esperienza consolidata nell’ambito della manutenzione degli impianti e dell’assistenza post-vendita.<sup>[2]</sup>

### 1.2.1 La Mission

*Ci assumiamo la responsabilità di generare risultati garantendo a ogni cliente le soluzioni strategiche e i sistemi produttivi più efficienti, guidati dal nostro know-how trasversale nel mondo dell’industria. Sfruttando le nuove tecnologie in ottica 4.0, vogliamo diffondere l’automazione industriale attraverso sistemi integrati solidi e flessibili, disegnati a misura di ogni realtà.*<sup>[3]</sup>

### 1.2.2 Settori e applicazioni

La saldatura rappresenta per REA l’applicazione più storica e più conosciuta, essendo il settore in cui l’azienda sviluppa e anticipa le migliori soluzioni in termini di isole robotizzate da più di 30 anni. Le principali tecnologie usate dagli impianti robotizzati di saldatura sono: MIG–MAG (saldatura a filo), TIG con e senza metallo di apporto, plasma con e senza metallo di apporto, laser, saldobrasatura MIG, brasatura induzione, resistenza, riporti anti-usura MIG-MAG. Un’altra applicazione riguarda il taglio termico, campo in cui l’automazione può e potrà dare i maggiori

benefici rispetto all'attuale esecuzione del taglio manuale. REA Robotics, nella sua pluriennale esperienza, realizza inoltre impianti di manipolazione per asservimento macchine e per assemblaggi industriali perfettamente adatte al processo produttivo del cliente e allo stesso tempo per ridurre i costi. In collaborazione con una delle più importanti aziende costruttrici di impianti per il taglio e la lavorazione della lamiera, REA ha sviluppato un sistema robotizzato di asservimento a presse piegatrici. La novità, e la sua caratteristica principale, è di avere un sistema meccanico dotato di encoder applicabile a qualsiasi presa che permette al robot di inseguire la presa durante il suo movimento senza mai rilasciare la lamiera. Infine REA realizza celle robotizzate di pallettizzazione specificatamente studiate per inserimento nei fine linea dei processi produttivi, con soluzioni complete pensate per ogni specifico prodotto del cliente, dalla pallettizzazione di inscatolati, ruote fino alla pallettizzazione di profili d'acciaio.<sup>[2]</sup>

### 1.2.3 Robot utilizzati

REA propone robot industriali di marchi come **ABB**, **Kuka** e **OTC Daihen** che comprendono le diverse applicazioni citate precedentemente.



# **Capitolo 2**

## **Fondamenti teorici sugli impianti di automazione e robotica industriale**

### **2.1 Contesto di Automazione industriale**

I sistemi autonomi in ambito industriale hanno trasformato e ridefinito il modo in cui le aziende gestiscono tutta la filiera, dalla produzione al controllo dei processi. Grazie all’automazione è possibile eseguire operazioni altamente complesse in modo continuo, con alta precisione, affidabilità, e soprattutto riducendo al minimo l’intervento umano e il rischio di errore. Da qui nasce la possibilità di controllare impianti industriali di qualsiasi dimensione, oppure intere linee di produzione senza qualsivoglia tipo di interruzione. Questo rappresenta un enorme vantaggio strategico per le aziende, perché permette di migliorare l’efficienza, la sicurezza e la qualità dei prodotti finiti. Tuttavia, automatizzare in modo corretto richiede un controllo in tempo reale e un sistema di monitoraggio che garantisca la consistenza dei parametri operativi, l’ottimizzazione delle risorse e soprattutto la possibilità di intervenire in modo immediato in caso di anomalie durante i processi.

In questo contesto nascono i sistemi SCADA (acronimo dall’inglese “Supervisory Control And Data Acquisition”, cioè “controllo di supervisione e acquisizione dati”), i quali hanno avuto un impatto determinante nella gestione e supervisione dei processi industriali. Ciò che li ha resi importanti è la loro capacità di raccogliere e analizzare dati in tempo reale, idealmente da un’ampia gamma di sensori e dispositivi presenti negli impianti. Questo permette di mantenere una visione d’insieme centralizzata delle operazioni, fondamentale per risolvere problematiche e ottimizzare i processi. Inoltre, l’esponenziale complessità delle operazioni industriali e soprattutto l’esigenza da parte dei clienti di massimizzare efficienza e sicurezza hanno reso i sistemi SCADA indispensabili: non solo permettono di monitorare e allo stesso tempo di controllare ogni singolo componente del sistema, ma supportano anche l’integrazione tra: impianti robo-

tizzati, linee di produzione e altri sistemi autonomi, permettendo la creazione di un ecosistema interconnesso.<sup>[4]</sup>

In breve, i sistemi SCADA sono diventati il cuore dell'automazione industriale moderna, garantendo gestione completa e affidabile delle operazioni, dall'acquisizione dei dati fino al controllo e intervento.

## 2.2 Composizione di uno SCADA

Un sistema SCADA è un sistema informatico distribuito (insieme di calcolatori interconnessi tra loro in un'architettura di tipo Client-Server, preposti a una o diverse funzionalità) che viene usato per il monitoraggio e la supervisione di sistemi fisici.

Un sistema SCADA di solito deve essere accompagnato da:

1. Unità di acquisizione dati, dette anche sensori ed attuatori che si occupano di misurare grandezze fisiche del sistema da controllare. Rappresentano il livello più basso della gerarchia SCADA.
2. PLC (dall'inglese "Programmable Logic Controller"): è il componente più usato nelle reti industriali, si occupa di elaborare i dati provenienti dai sensori ed attuatori per poi eseguire il programma indicato, regolando le logiche di automazione dell'impianto. Comunica con tutto il sistema e monitora le grandezze interessate, mantenendo nei registri lo stato di tutte le variabili cui il sistema ha accesso.
3. Sistema di telecomunicazione: è l'infrastruttura che consente di trasferire dati tra i vari dispositivi e il sistema centrale. È implementato attraverso protocolli di comunicazione come Modbus, Ethernet/IP e Profibus e si basa su reti cablate, wireless oppure satellitari, a seconda delle esigenze e della distanza dei dispositivi. Negli ultimi anni si predilige l'utilizzo di reti Ethernet e protocolli TCP/IP, poiché le grandi industrie necessitano di comunicare a grandi distanze.
4. Un supervisore o HMI(dall'inglese "Human-Machine Interface"): un pannello o computer visibile agli operatori. Raccoglie dati dai microcontrollori tramite il sistema di telecomunicazione e li elabora, con l'obiettivo di fornire pieno controllo all'impianto (come ad esempio: far suonare allarmi se un insieme di parametri dovesse superare un valore limite). Attraverso l'HMI, l'operatore può monitorare, esaminare lo stato, analizzare gli storici ed interagire con il sistema in modo intuitivo.

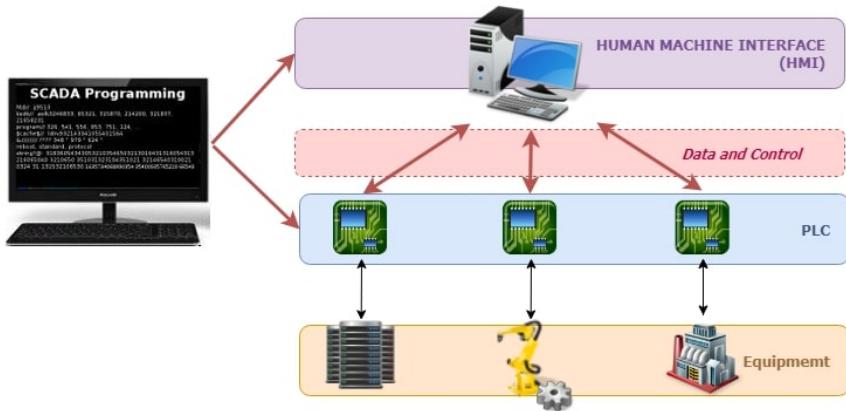


Figura 2.1: Esempio sistema SCADA con componenti principali. Fonte: [5]

## 2.3 Funzioni principali dello SCADA

Un sistema SCADA svolge diverse funzioni chiave per assicurare una gestione efficace e sicura dei processi industriali, che differiscono da impianto ad impianto. Le fasi principali che generalmente uno SCADA dovrebbe fornire sono:

- Acquisizione e raccolta dati: i sistemi SCADA acquisiscono dati dai dispositivi correlati di metadati e li trasferiscono al sistema centrale. Questo avviene in tempo reale per fornire continuamente consistenza sui vari parametri di processo.
- Visualizzazione e monitoraggio dei dati in tempo reale: i dati raccolti vengono rappresentati grazie all'HMI in formati intuitivi per favorire la lettura da parte degli operatori, attraverso grafici, tabelle interattive, led per allarmi e diagrammi di flusso.
- Archiviazione e storizziazione del dato: vengono registrati i dati storici sia per analisi successive, sia per tenere traccia delle performance dell'impianto. Risultano essenziali per ottimizzare processi e per la manutenzione predittiva; inoltre i dati possono essere salvati su archivi locali o database relazionali, così da poter essere usufruibili sia tramite i formati sopra citati oppure esportati direttamente su sistemi esterni.
- Gestione eventi: un sistema SCADA genera allarmi e warning per segnalare criticità/ anomalie nei parametri monitorati, è una delle funzionalità cardine e richiede sempre l'integrazione con un operatore per evitare guasti o problemi di sicurezza. Inoltre tutti gli eventi vengono registrati e documentati per eventuali verifiche future.
- Controllo e manutenzione manuale: i sistemi SCADA oltre che a interfacciarsi con gli operatori, si interfacciano anche con l'impianto. Tale funzionalità è sempre più utilizzata e richiesta dai clienti per i sistemi SCADA, favorendo più flessibilità al sistema stesso, ma anche più complessità a livello progettuale.

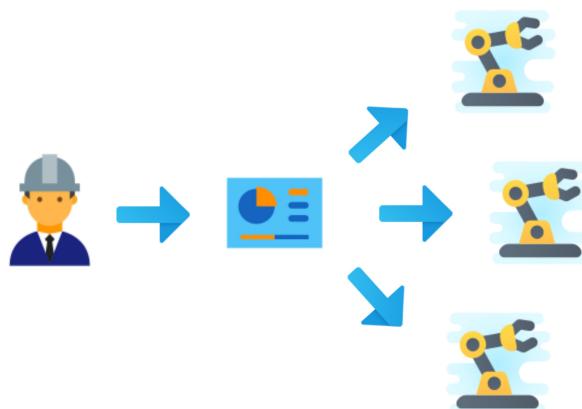


Figura 2.2: Gestione impianto tramite SCADA

## 2.4 Piattaforma SCADA e Strumenti Software per la progettazione

Con il termine piattaforma SCADA indichiamo solo lo strato software che svolge il lavoro definito in precedenza. Esistono diverse piattaforme che permettono di progettare un'interfaccia HMI per uno SCADA, e ognuna è affiancata dai propri linguaggi di programmazione proprietari, oppure i più famosi per poter progettare interfacce più avanzate (es: Jython, VBScript, C#, ...)



Figura 2.3: Esempi software per progettazione HMI

Ognuna di queste piattaforme possiede logiche diverse per poter progettare uno SCADA; nel mio caso è stata scelta come piattaforma FactoryTalk® Optix™, neo-piattaforma sviluppata da ASEM by Rockwell-Automation:

*È l'innovativa piattaforma software perfetta per realizzare HMI moderne e responsive, con un'esperienza utente di alto livello, Gateway IIoT, applicazioni Edge Computing e in generale soluzioni legate alle esigenze dell'Industry 4.0.*

*Grazie ad un'architettura completamente modulare ed estremamente flessibile sviluppata con tecnologie cross-platform, FT Optix™ permette di realizzare applicazioni compatibili con piattaforme ARM e x86 con sistemi operativi Windows e Linux, garantendo la massima flessibilità ai progettisti, che possono scegliere la piattaforma che meglio si adatta all'applicazione.*

*FT Optix™ fa parte della suite cloud-based FactoryTalk® Design Hub™, che permette al team di progettazione di collaborare allo sviluppo dell'applicazione HMI tramite un sistema di controllo di versione distribuito, semplificando la gestione del software, aumentando la produttività ed accelerando il time to market.<sup>[6]</sup>*

La scelta di approcciare questa piattaforma di sviluppo è nata dal fatto che in REA l'innovazione è alla base della politica aziendale e il passaggio ad un nuovo software rappresenta la scelta perfetta per rimanere al passo con i tempi. Il software possiede un'interfaccia simile alle altre controparti ma con l'aggiunta che per SCADA più avanzati è possibile sfruttare l'uso di C# in ambienti di sviluppo basati su .NET. Oltre a questo ho dovuto interfacciarmi con diversi software e librerie per poter far funzionare lo SCADA. Sul fronte C# e ambiente di sviluppo, la scelta è ricaduta su Visual Studio Community per l'ampia customizzazione e il supporto continuo da parte di Microsoft. Inoltre per quanto riguarda la gestione di tutti i dati interni allo SCADA, due sono state le scelte su cui lavorare:

1. Database locale di Optix™: con la creazione e la gestione del database in SQLite e l'ausilio di DB4S, è stato possibile interfacciarsi anche con le macchine più datate.
2. Microsoft SQL Server Management Studio (SSMS): applicativo utilizzato già in REA per gestire vari database lato Server per gli impianti più complessi.

Infine per il controllo delle versioni del progetto, data l'integrazione tra FT Optix™ e Git, la scelta è ricaduta su Git per tenere traccia di tutte le varie versioni dello SCADA durante lo sviluppo, per eventualmente collaborare con altri sviluppatori e per mantenere uno storico delle revisioni.

#### 2.4.1 Lista Software utilizzati

Ecco la lista dei software utilizzati durante la progettazione del sistema SCADA nel corso del tirocinio:

- **FT Optix™:** versione 1.4.2.3
- **Microsoft Visual Studio 2022:** versione 17.11
- **Microsoft SQL Server Management Studio (SSMS):** versione 20

- **DB Browser for SQLite (DB4S)**: versione 3.13.0
- **Git**: versione 2.47.0, direttamente da **GitHub Desktop**

## 2.4.2 Struttura Generale di FactoryTalk Optix™

FT Optix™ è un software strutturato per fornire modularità ai componenti utilizzati affinché i progetti possano essere suddivisi in componenti riutilizzabili, ciascuno con logica indipendente, e che possono essere richiamati in più parti dell'applicazione. Inoltre, come già anticipato, permette la gestione dei progetti multi-utente: con supporto per lo sviluppo collaborativo grazie al versionamento e controllo delle modifiche, oltre al supporto per protocolli industriali, come OPC UA, Modbus TCP/RTU, Ethernet/IP e altri standard di automazione. Altra caratteristica del software è la struttura a incapsulamento dei sinottici: come nella programmazione ad oggetti, viene sfruttato un tipo di design gerarchico, dove sono presenti oggetti grafici compositi, definiti Sinottici. Questi includono componenti nidificate con proprietà specifiche, logiche ed eventi. Inoltre, è possibile creare dei template parametrizzati per simulare macchinari, sensori o attuatori. Tutti questi oggetti grafici possono essere salvati in librerie condivise e riutilizzati in progetti diversi, riducendo tempi di sviluppo e migliorando la consistenza, oltre al fatto che è possibile utilizzare controlli avanzati come grafici, griglie e dashboard dinamiche per avere maggior controllo durante la progettazione. Infine, è possibile sfruttare sinottici reattivi - permettendo l'adattamento a diverse risoluzioni e formati dei pannelli HMI - e associare eventi o trigger ad ogni oggetto per eseguire azioni come aggiornamenti in tempo reale, invio di allarmi o gestione degli errori. Lato usabilità, fornisce simulazione attraverso ambiente di sviluppo, ossia è in grado di simulare il progetto completo con dati fintizi prima della messa in servizio, oltre a strumenti integrati per il debugging e la risoluzione di problemi, con visualizzazione di log e delle variabili in tempo reale.

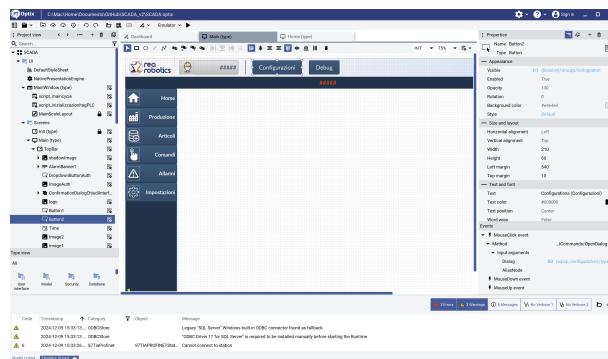


Figura 2.4: Illustrazione dashboard di progetto

# **Capitolo 3**

## **Progettazione del sistema SCADA**

La progettazione di un sistema SCADA è un processo articolato e strategico, che richiede attenta pianificazione e un ampio impiego di risorse. Senza una struttura ben definita, infatti, si possono generare configurazioni di prodotto che necessitano di essere successivamente ripensate, portando inevitabilmente a un incremento di costi, non solo in termini di tempo, ma anche di risorse umane. Per garantire un corretto avanzamento e rispondere in modo efficiente alle aspettative del cliente, è buona norma suddividere il lavoro in fasi predeterminate. In tal modo, si può procedere con maggiore chiarezza, sapendo già dal principio quali aspetti richiederanno attenzione specifica nelle varie fasi del progetto. Nel mio caso, il lavoro è stato suddiviso in quattro fasi principali:

1. Analisi dei requisiti
2. Strutturazione dell'HMI
3. Gestione Database
4. Sviluppo back-end con integrazione al PLC

Ognuna di queste fasi ha contribuito in modo sostanziale a preservare la coerenza e l'integrità del prodotto finito, consentendo di ottenere un risultato che rispecchiasse fedelmente le richieste.

### **3.1 Analisi dei requisiti del cliente**

La prima fase - definita anche come fase di design - prevede un momento di dialogo approfondito con il cliente per comprendere e prendere nota di tutte le necessità, in modo da tradurle in specifiche dettagliate, misurabili e quindi concretamente realizzabili dal team. L'analisi dei requisiti prevede diverse attività, la cui applicazione varia in base alla natura del prodotto finale.

Oltre alla raccolta, è fondamentale l'organizzazione del loro sviluppo, soprattutto nei progetti più complessi in cui è necessario stratificare il lavoro tra più individui.

Per quanto riguarda il mio progetto, dato l'obiettivo di creare un sistema che potesse essere altamente modulare e in grado di costituire una base solida per futuri progetti, mi è stato richiesto di soffermarmi su alcuni aspetti di fondamentale rilevanza:

1. Creare un'interfaccia quanto più user-friendly possibile, per facilitare l'utilizzo e l'accessibilità da parte degli operatori di macchina.
2. Definire un "ricettario" di base, ovvero una struttura configurabile e facilmente aggiornabile, sotto il nome di "anagrafica articoli".
3. Definire una sezione dedicata agli ordini di produzione che includa sia i comandi di produzione per la gestione operativa, sia gli storici.
4. Implementare un sistema efficiente per la gestione e l'accesso agli allarmi e warning, essenziale per il monitoraggio e la diagnostica.
5. Creare scorciatoie che permettono allo sviluppatore di poter testare il prodotto sia in fase di sviluppo, che a prodotto finito, anche tramite tele-assistenza.

## 3.2 Strutturazione HMI (Human Machine Interface)

Per la realizzazione dell'HMI, l'azienda mi ha mostrato altri prodotti fornendomi le basi per comprendere il funzionamento delle interfacce. Dopo aver ricevuto piena autonomia decisionale da parte dell'azienda, ho sviluppato un approccio originale, scegliendo di procedere nel seguente modo. All'interno di FT Optix™, per prima cosa l'interfaccia è stata organizzata in una finestra principale denominata *MainWindow*<sup>[7]</sup> (Figura 3.1), a sua volta suddivisa in 3 sotto-pannelli:

- Pannello di stato
- Pannello di funzione
- Pannello di menù

Il **Pannello di stato** è un elemento presente in tutta l'interfaccia e di conseguenza visibile in ogni menù e sotto-menù. La sua presenza è fondamentale poiché contiene al suo interno informazioni per il funzionamento dell'interfaccia stessa. Procedendo da sinistra troviamo il logo dell'azienda produttrice dell'impianto, che in questo caso è Rea Robotics, per dare riconoscibilità e identità a tutto il sistema. Subito dopo è presente un indicatore che contiene il nome

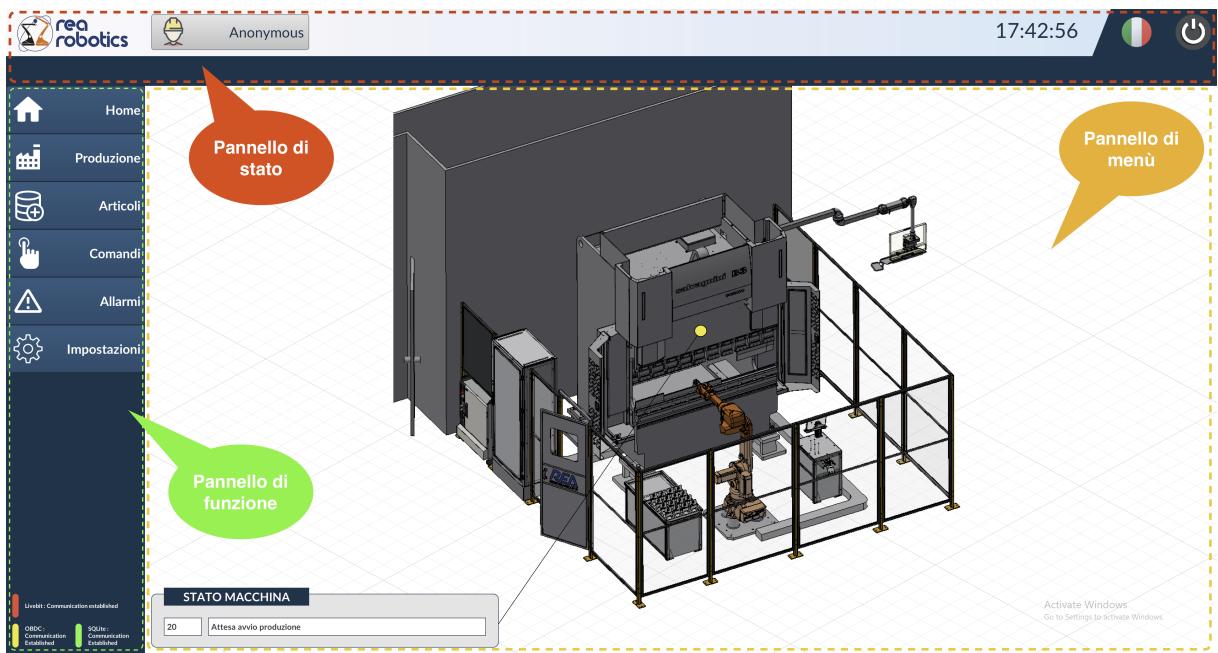


Figura 3.1: Suddivisione interfaccia HMI

dell’operatore che sta utilizzando la macchina (di default impostato ad anonimo). Se viene selezionato, apre un menù a tendina da cui è possibile poter eseguire l’accesso, sia come operatore, sia come sviluppatore. Tale distinzione è importante perché alcune funzioni dell’interfaccia sono accessibili solo a figure autorizzate come gli sviluppatori, per garantire sicurezza e integrità delle operazioni. Continuando verso destra è presente l’orario, utile per avere sempre sotto controllo il tempo durante le operazioni, e il pulsante per la selezione della lingua, che mostra l’icona della lingua attualmente attiva: l’interfaccia è stata progettata per poter funzionare sia in lingua italiana sia in lingua inglese, garantendo maggiore adattabilità anche in ambienti internazionali. Infine è presente il pulsante di spegnimento, che permette di chiudere l’interfaccia in modo sicuro. Per prevenire chiusure accidentali, alla pressione appare un popup che richiede un’ulteriore conferma di spegnimento della macchina, evitando che un tocco involontario possa interrompere il lavoro dell’operatore. Nella parte inferiore di questo pannello è presente una sezione blu che si occupa di monitorare lo stato degli allarmi e avvisi. In caso di problemi, tale sezione mostra chiaramente l’allarme e la descrizione rilevata, assicurando così pronta visibilità delle condizioni operative del sistema.

Il **Pannello di funzione**, come il pannello di stato, è un elemento fisso che permette la navigazione attraverso i vari menù. Questo pannello è costituito da un layout verticale con una serie di pulsanti che consentono all’operatore di accedere a tutte le principali sezioni dell’interfaccia. Ogni pulsante identifica un menù specifico e, una volta selezionato, consente di passare rapidamente alla schermata desiderata, interagendo dinamicamente con il pannello di menu posizionato alla sua destra. Il primo bottone è denominato *Home* e offre una panoramica generale

dello stato della macchina. In macchine più complesse permette anche la visualizzazione di indicatori di stato delle componenti. Il secondo invece permette di accedere alla *Produzione*, dove l'operatore può decidere di azionare la produzione. Il terzo bottone apre la sezione *Anagrafica articoli*, in cui è possibile inserire e visualizzare tutti i parametri di riferimento degli articoli dell'impianto. A seguire, troviamo la sezione *Comandi manuali*, dove è possibile impartire comandi specifici direttamente al robot di lavoro. Ancora più in basso, sono presenti i bottoni della sezione: *Allarmi* che forniscono una panoramica dettagliata di tutti gli allarmi e avvisi attivi. Infine, *Impostazioni* dove sono presenti i vari settaggi relativi sia allo SCADA sia ai parametri della macchina. Nella parte inferiore del pannello di funzione sono presenti tre LED, ognuno dei quali svolge una funzione di monitoraggio importante. Il primo LED, partendo dall'alto, segnala lo stato di collegamento con la Macchina a stati, permettendo di verificare se il programma dello SCADA funziona correttamente. Il secondo e il terzo LED, invece, mostrano rispettivamente lo stato di collegamento con l'ODBC, ovvero il server su cui gira il database, e lo stato di collegamento al database locale in SQLite, che assicurano che l'interfaccia sia connessa correttamente alle risorse necessarie per il suo funzionamento.

Per ultimo abbiamo il **Pannello di menù**, che a differenza degli altri due, è interamente dinamico. Questo pannello occupa circa l'80% di tutta l'interfaccia e ha il compito di visualizzare le varie schermate dei menu selezionate tramite il pannello di funzione.

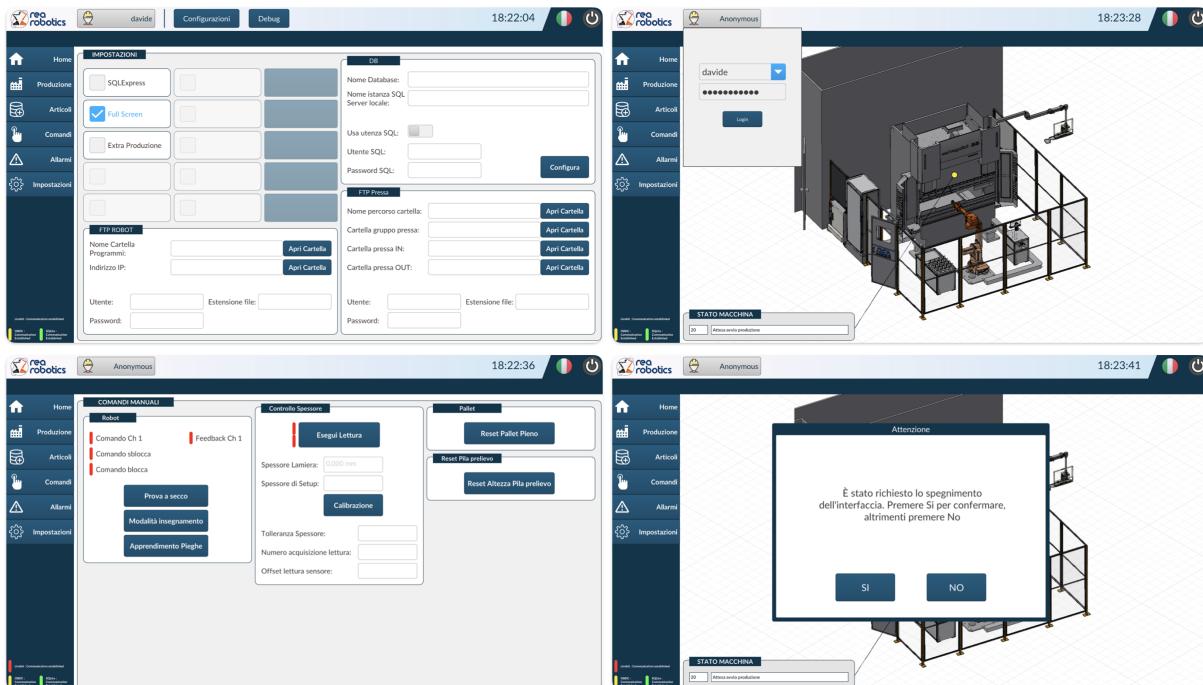


Figura 3.2: 4 illustrazioni dell'HMI

### **3.2.1 Considerazioni tecniche sulla risoluzione**

Grazie alla tecnologia responsive presente nativamente su FT Optix™, l'intero HMI è adattabile e scalabile su qualsiasi display, garantendo una corretta visualizzazione indipendentemente dalle dimensioni dello schermo. Tuttavia, poiché nel nostro caso lo SCADA è progettato per essere utilizzato principalmente su pannelli con risoluzione Full HD (1920x1080), è di fondamentale importanza che tutta l'interfaccia sia leggibile e fruibile dall'operatore senza barre di scorrimento laterali. Considerando a maggior ragione che lo SCADA sarà utilizzato su dispositivi dotati di tecnologia touch in ambienti industriali, è cruciale che ogni elemento risulti chiaro, affidabile e di facile accesso. La leggibilità e la precisione dei comandi sono fondamentali per garantire la sicurezza e l'efficacia delle operazioni svolte dall'operatore in tali contesti.

## **3.3 Gestione Database per la produzione**

Per quanto riguarda il database ho deciso di adottare diverse metodologie per la gestione dati. La prima esigenza dell'azienda è stata creare un ricettario per mantenere una serie di record con attributi ben specifici, e che permetesse l'integrazione e la comunicazione verso la scheda di produzione. La mia proposta di soluzione ideale è stata quella di suddividere il database in due tabelle: una completamente dedicata alla parte di *Anagrafica articoli* presenti in macchina, e l'altra che riguardasse la *Produzione*, in cui vengono registrati gli ordini. Questa rappresenta solo la prima fase di progettazione, poiché nello SCADA è necessario anche avere accesso ad altri dati, tra cui: lo *Storico Produzioni*, che archivia tutte le produzioni completate, sia con esito positivo che negativo; tutte le variabili che potrebbero influire sull'integrità dello stato di un ordine (ad es: l'extra produzione); e tutti gli interventi effettuati tramite input all'HMI, come l'importazione e l'esportazione delle ricette. Infine, ho deciso di integrare all'HMI non solo il database locale creato da me, ma anche il SSMS fornito dall'azienda, per garantire maggiore flessibilità al sistema.

### **3.3.1 Anagrafica Articoli**

Questa sezione si occupa della gestione dell'inserimento di nuovi articoli da produrre nell'impianto. La progettazione ha richiesto due tipi di interventi: il primo ha riguardato la gestione del front-end dell'anagrafica; il secondo ha coinvolto la parte di back-end, con particolare attenzione a bottoni e gestione database.

Il lavoro è iniziato con la creazione del menù, in particolare di un'interfaccia dedicata che possa integrare tutti i comandi necessari. Grazie a FT Optix™, è possibile implementare una sezione di inserimento dei singoli articoli composta dagli attributi richiesti dal sistema. In ag-

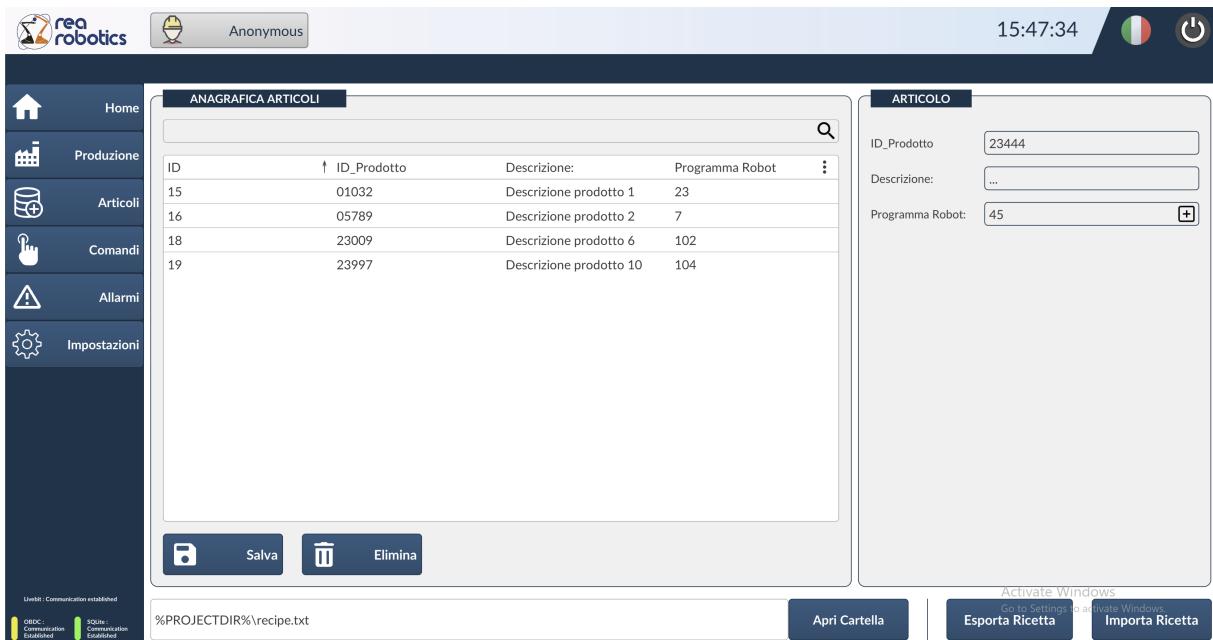


Figura 3.3: Illustrazioni sezione *Articoli*

giunta, è stata predisposta una sezione di visualizzazione, che mostra tutti i dati del database, compresi gli attributi richiesti in precedenza. Per facilitare la gestione dei dati, è stato sfruttato un oggetto di tipo *DataGridView*<sup>[8]</sup>, che emula il funzionamento di una tabella, rendendo chiara e accessibile la consultazione dei dati. Dato che uno SCADA industriale deve gestire qualche centinaio di articoli, sono stati implementati sia un sistema di filtraggio dei prodotti in base a parametri specifici (selezionando gli attributi d’interesse), sia una barra di ricerca che permette di individuare rapidamente un prodotto già presente nel sistema. Inoltre, su richiesta, è stata aggiunta una sezione in basso (Figura: 3.3) per l’esportazione e l’importazione delle ricette, consentendo al sistema di caricare serie di articoli preconfigurati dal cliente, velocizzando e semplificando il processo di inserimento articoli.

Per quanto riguarda la parte di back-end, il primo passo consiste nel configurare il database di riferimento. All’interno della Project View nella cartella DataStores deve essere creato un oggetto di tipo *Embedded database*, che permette di importare un database .sqlite su cui lavorare<sup>[9]</sup>. Successivamente, è necessario eseguire una serie di passaggi per poter impostare correttamente la struttura delle tabelle:

1. Importare/impostare in *Configuration* il nome del file del database da cui l’eseguibile dovrà estrarre i dati.
2. Creare, tramite FT Optix™, le tabelle sotto il nome di *RecipeSchema*, includendo tutti gli attributi di interesse necessari per la gestione delle ricette.

3. Aprire il file creato, sotto il nome di `filename.sqlite` tramite un software come DB4S, presente all'interno della cartella `ApplicationFiles` del programma, per impostare eventuali vincoli del database. Questo ultimo passaggio è facoltativo e può essere omesso qualora non siano richiesti vincoli specifici per i dati da raccogliere.

Prima di introdurre la sezione degli script in C#, è importante chiarire due scelte di stile che ho adottato durante la progettazione del seguente database. Nel nostro caso ogni articolo deve essere caratterizzato da attributi ben definiti, che devono essere sempre presenti, indipendentemente dalla macchina su cui lo SCADA dovrà essere installato. In particolare:

- ***ID***: un identificatore universale per il database, configurato come un intero, chiave primaria e auto-incrementale (`INTEGER PRIMARY KEY AUTOINCREMENT`). Questo attributo permette ad ogni articolo di avere un identificativo distinto, senza mai ripetersi, nemmeno in caso di eliminazione o modifica di un prodotto. Questa scelta viene fatta per motivi di sicurezza, prevenendo eventuali conflitti di tuple in situazioni di sincronizzazione non corretta.
- ***Product\_ID***: un identificatore specifico per il prodotto con vincoli meno restrittivi rispetto al primo. Questo campo non deve essere forzatamente un intero, bensì può essere anche un codice alfa-numerico, a discrezione del cliente. Come unici vincoli aggiunti, per non incorrere in errori di inserimento, ci sono l'unicità e l'obbligo di non essere nullo.
- ***Robot\_Program***: indica il programma che il robot dovrà utilizzare. Di norma, questi programmi sono identificati da numeri. Anche in questo caso, deve essere specificato il vincolo di obbligatorietà: non nullo, garantendo l'inserimento di un valore.

Partendo da questa base, è possibile inserire a piacimento ulteriori attributi in base alla necessità del cliente oppure dell'impianto su cui si sta lavorando, ripetendo i passaggi descritti precedentemente. L'altra considerazione da fare riguarda l'uso dei *RecipeSchema* in FT Optix™, per cui esistono alcuni vincoli dati dal software: ad esempio, l'obbligo di anteporre il carattere / a tutti gli attributi creati sia nel database locale sia in interfaccia. Inoltre, FT Optix™ supporta nativamente, attraverso le API, solo operazioni di base per le ricette, quali: `INSERT`, `UPDATE`, `DELETE` e `SELECT`<sup>[10]</sup>. Sebbene inizialmente questo possa sembrare un vantaggio, dato che permette un accesso semplificato tramite i tool integrati, in alcuni casi può rappresentare una limitazione se si desidera interagire attraverso funzioni più complesse con il database. Per questo motivo, FT Optix™ fornisce la possibilità di creare in prima persona script in C#, dando piena flessibilità e controllo diretto sul database al programmatore. Questa scelta si è dimostrata vincente, perché mi ha permesso di creare script unici per la successiva gestione dei dati.

Per quanto riguarda invece la parte di back-end, iniziamo con l'inserimento degli articoli nel database. Utilizzando i *RecipeSchema*<sup>[11]</sup>, FT Optix™ mette a disposizione un oggetto *Run-TimeNetLogic*<sup>[12]</sup> di tipo *RecipeController*, che include già una serie di funzioni di base. Nel nostro caso, è necessario implementare all'interno del file *RecipeController.cs* uno script che permetta l'inserimento di un prodotto in Anagrafica correttamente. In Visual Studio, creiamo quindi una funzione che verrà richiamata dal tasto SALVA presente nell'interfaccia. Per prima cosa, scriviamo il comando [ExportMethod] (riga 2) che consente di rendere accessibile la funzione direttamente in FT Optix™ (questa pratica verrà ripetuta anche per futuri script). Successivamente, definiamo la funzione specificando gli attributi in input che ci interessa salvare. All'interno del codice, viene richiamato il database locale (oppure SSMS) tramite *myStore* e la tabella di interesse *myTable* (rispettivamente righe 8-9). Da qui possiamo utilizzare la funzione *.Insert* di FT Optix™, con i valori precedentemente richiamati, per popolare il database (riga 18).

```

1 ...
2     [ExportMethod]
3     public void Insert_Product_in_Anagrafica(string Product_ID, string Descr, int Robot_Program)
4     {
5         try
6         {
7             // Inserimento nella tabella
8             var myStore = Project.Current.Get<Store>("DataStores/EmbeddedDatabase1");
9             var myTable = myStore.Tables.Get<Table>("RecipeSchema1");
10            string[] columns = { "Name", "/Descr", "/Robot_Program", "/Product_ID" };
11            var values = new object[1, 4];
12            values[0, 0] = Product_ID;
13            values[0, 1] = Descr;
14            values[0, 2] = Robot_Program;
15            values[0, 3] = Product_ID;
16
17            // Eseguire la query di inserimento
18            myTable.Insert(columns, values);
19
20            // Se l'operazione è andata a buon fine
21            Log.Info("RecipeSchema1", "Inserimento riuscito: " + values[0, 1]);
22        }
23        catch (Exception ex)
24        {
25            // In caso di errore, viene catturata l'eccezione e viene stampato il messaggio d'errore
26            Log.Error("RecipeSchema1", "Errore durante l'inserimento: " + ex.Message);
27            popup.OpenPopUp(ex.Message, 0);
28        }
29    }
30 ...

```

Due aspetti importanti meritano attenzione in questa fase. In primo luogo, si osserva la presenza del carattere / negli attributi, come richiesto dalle specifiche di FT Optix™ per le ricette. In secondo luogo, vi è un attributo Name, il quale rappresenta un vincolo essenziale nella costruzione delle ricette in FT Optix™; nel nostro caso, assegniamo a tale campo un valore a scelta tra quelli disponibili. Inoltre, la funzione è stata inserita in un blocco *try-catch* per garantire maggiore controllo su eventuali errori di inserimento in FT Optix™ (altra pratica ampliata anche ad altri script, come regola di buona programmazione). Ora passiamo ad un'ulteriore richiesta che riguarda le importazioni di articoli esterni. In questo caso, è stato necessario avvalerci di

un ulteriore oggetto, *RunTimeNetLogic*, direttamente associato alla tabella, al fine di implementare una funzione capace di controllare un file esterno definito come *dati.txt*. Di seguito è riportato il codice:

```

1 ...
2     // Metodo per controllare e leggere il file
3     private static void ControllaFile(Object source, ElapsedEventArgs e)
4     {
5         // Specifica la cartella e il nome del file
6         string cartella = @"C:\Users\davide.quartucci\Desktop\InserimentoAnagraficaSuDB";
7         string nomeFile = "dati.txt";
8
9         // Cerca il file nella cartella specificata
10        string percorsoFile = Path.Combine(cartella, nomeFile);
11
12        if (File.Exists(percorsoFile))
13        {
14            try
15            {
16                // Leggi il contenuto del file
17                string[] righe = File.ReadAllLines(percorsoFile);
18
19                foreach (string riga in righe)
20                {
21                    try
22                    {
23                        // Dividi la riga in tre parti usando il separatore ';'
24                        string[] campi = riga.Split(';');
25
26                        if (campi.Length != 3)
27                        {
28                            throw new FormatException("La riga non contiene esattamente 3 campi.");
29                        }
30
31                        // Assegna i campi a variabili locali
32                        string Product_ID = campi[0];
33                        string Descr = campi[1];
34                        int Robot_Program;
35
36                        // Prova a convertire il terzo campo in un intero
37                        if (!int.TryParse(campi[2], out Robot_Program))
38                        {
39                            throw new FormatException("Il campo Robot_Program non è un intero valido.");
40                        }
41
42                        //Esegue l'inserimento nel DB di una nuova anagrafica
43                        var myStore = Project.Current.Get<Store>("DataStores/EmbeddedDatabase1");
44                        var myTable = myStore.Tables.Get<Table>("RecipeSchema1");
45                        string[] columns = { "Name", "/Descr", "/Robot_Program", "/Product_ID" };
46                        var values = new object[1, 4];
47                        values[0, 0] = Product_ID;
48                        values[0, 1] = Descr;
49                        values[0, 2] = Robot_Program;
50                        values[0, 3] = Product_ID;
51                        myTable.Insert(columns, values);
52                    }
53                    ... // Gestione eccezioni ed eliminazione file dopo la lettura

```

Nel codice appena illustrato viene richiamata la seguente funzione: *dati.txt* situato in un percorso preimpostato (righe 6-7). La scelta di definire un percorso ben specifico è stata fatta per evitare problemi imprevisti durante l'inserimento dei dati. La funzione prosegue analizzando le righe del file, secondo le specifiche *Product\_ID*; *Descrizione*; *Robot\_Program* (righe da 17 a 37). A partire da riga 42, il codice procede con l'inserimento all'interno del database di tutti i dati del file, e infine, lo elimina come richiesto. Ovviamente, questa funzione deve essere ri-

chiamata ciclicamente per garantire un aggiornamento continuo del database, facilitando così, ad esempio, la trasmissione di nuovi articoli dagli uffici di produzione direttamente alla macchina. Per automatizzare l'intero processo, nel metodo Start (creato nativamente da FT Optix™) ho deciso di configurare un timer che ogni 10 secondi ripete l'operazione di lettura ed importazione degli articoli nell'anagrafica, assicurando aggiornamento costante e sincronizzazione dei dati:

```

1 ...
2     private static Timer timer; // Timer per eseguire l'operazione ogni 10 secondi
3
4     public override void Start()
5     {
6         // Insert code to be executed when the user-defined logic is started
7
8         // Imposta e avvia il timer
9         timer = new Timer(10000); // 10000 millisecondi = 10 secondi
10        timer.Elapsed += ControllaFile; // Associa l'evento Elapsed all'handler
11        timer.AutoReset = true; // Ripeti l'operazione ogni 10 secondi
12        timer.Enabled = true; // Abilita il timer
13    }
14 ...

```

Per tutte le altre funzionalità, come l'eliminazione di articoli, ricerche ed importazioni/esportazioni da HMI, sono state sfruttate funzionalità di base già presenti nel programma. Il lavoro si è concentrato sull'ampliamento e personalizzazione di questi strumenti affinché si integrassero correttamente con il sistema.

### 3.3.2 Ordini di Produzione

Questa sezione si focalizza sulla fase di produzione degli articoli, con focus sull'inserimento, l'avvio e la gestione di una nuova produzione. La progettazione ha seguito il medesimo approccio adottato per la sezione dell'anagrafica. Per quanto riguarda l'interfaccia, ho preferito dividere la creazione di un nuovo ordine di produzione dalla gestione della produzione stessa, poiché nel monitoraggio è necessario tenere in primo piano molti più attributi rispetto alla sezione di anagrafica. Dal punto di vista progettuale, troviamo ancora un oggetto di tipo *DataGridView*, che anche in questo caso serve per tenere sotto controllo lo stato degli ordini. Infatti, è stato anche configurato in modo che cambi colore in base allo stato dell'ordine: verde per gli articoli che sono attualmente in produzione, giallo per quelli in attesa e rosso per eventuali interruzioni impreviste. Anche in questa sezione è stato implementato un sistema di filtraggio, mentre nella parte inferiore della schermata troviamo un pannello che tiene sotto controllo lo stato/passo in cui si trova la macchina (il suo funzionamento verrà illustrato nella sezione 3.4). Sempre nella parte inferiore della schermata, inoltre, sono presenti i pulsanti che permettono la gestione degli ordini di macchina. Prima di procedere alla fase implementativa, è stato progettato anche un popup per *Nuova Produzione* (Figura:3.4), che permette di inserire gli ordini in database. Questo popup è una finestra che richiede le specifiche di produzione, ovvero:

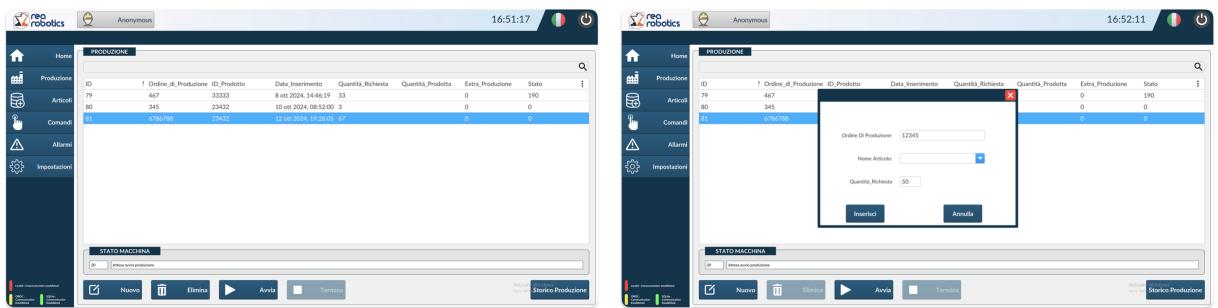


Figura 3.4: Illustrazioni sezione *Produzione* e sottosezione *Nuova Produzione*

- **Ordine di Produzione:** identificativo dell'ordine, configurato come campo testuale.
- **Nome Articolo:** mostra a schermo un menù a tendina che consente di selezionare il *Product\_ID* dell'articolo già presente in database, ereditando tutte le caratteristiche definite nella sezione Anagrafica.
- **Quantità Richiesta:** è la quantità dell'ordine da produrre inizialmente.

Fatto questo, ulteriori attributi vengono impostati durante l'inserimento dell'ordine. Infatti, dopo aver specificato i dati precedenti e premuto il tasto "inserisci", viene richiamata una funzione all'interno di *RecipeController.cs*, sviluppata seguendo le stesse linee guida di quella già analizzata in Anagrafica. In questo caso, poiché la produzione necessita di parametri aggiuntivi, come ad esempio: *Date\_Insert*, *Status*, *Extra\_Production*, ..., questi attributi vengono inizialmente impostati su valori predefiniti. Un attributo che necessita di analisi è la Data di Inserimento (gestita a riga 16). Qui viene utilizzato *DateTime.Now* per ottenere il momento esatto del salvataggio, con successivo parsing in *.ToString*, come richiesto per il corretto inserimento dei dati nel database.

```

1 ...
2 [ExportMethod]
3 public void Insert_newProduction(string Odp, string NomeArticolo, int quantità)
4 {
5     try
6     {
7         // Inserimento nella tabella
8         var myStore = Project.Current.Get<Store>("DataStores/EmbeddedDatabase1");
9         var myTable = myStore.Tables.Get<Table>("RecipeSchema2");
10        string[] columns = { "Name", "/Odp", "/NomeArticolo", "/Date_Insert", "/Quantità", "/Extra_Production", "/Total_Reject", "/Status" };
11        var values = new object[1, 8];
12
13        values[0, 0] = Odp;
14        values[0, 1] = Odp;
15        values[0, 2] = NomeArticolo;
16        values[0, 3] = DateTime.Now.ToString("yyyy-MM-ddTHH:mm:ss.fffffffK");
17        values[0, 4] = quantità;
18        values[0, 5] = 0;
19        values[0, 6] = 0;
20        values[0, 7] = 0;
21
22        // Eseguire la query di inserimento
23        myTable.Insert(columns, values);

```

```

24
25     // Se l'operazione è andata a buon fine
26     Log.Info("RecipeSchema2", "Inserimento riuscito: " + values[0, 1]);
27 }
28 catch (Exception ex)
29 {
30     // In caso di errore, viene catturata l'eccezione e viene stampato il messaggio d'errore
31     Log.Error("RecipeSchema2", "Errore durante l'inserimento: " + ex.Message);
32     popup.OpenPopUp(ex.Message, 0);
33 }
34 }
35 ...

```

Prima di passare all'implementazione della funzione *Avvia Produzione*, è necessario soffermarci su un'ulteriore richiesta da parte dell'azienda: la gestione di un'eventuale *Extra Produzione* per rispondere a esigenze particolari di produzione. Per questo motivo, sono stati attuati 2 passaggi:

1. Assicurarsi che l'*extra produzione* sia sempre disponibile e attivabile su richiesta tramite un'opzione accessibile solamente dal programmatore.
2. Nel caso in cui sia attiva, permettere alla macchina di poterla gestire lato back-end (che verrà approfondito nella sezione 3.4).

Per il primo punto la soluzione adottata consiste nell'inserire all'interno della sezione *Impostazioni* un interruttore *Extra Produzione*, attivabile esclusivamente in modalità sviluppatore o con le opportune autorizzazioni<sup>[13]</sup>. Fatto questo è stato progettato un popup che appare al termine di ogni ciclo di produzione e, tramite un menù, chiede se si vogliono produrre ulteriori componenti, permettendo così alla macchina di riavviare il ciclo produttivo (Figura:3.5).

Passiamo ora a due iterazioni principali nello SCADA: *Avvia Produzione* e *Termina Produzione*. Nel caso di *Avvia Produzione*, dopo aver dato conferma tramite apposito popup di verifica, viene richiamata la funzione *pr\_AvviaLocale*, che si occupa di configurare in autonomia le variabili fondamentali della produzione, permettendo alla macchina a stati di poterne usufruire:

```

1 ...
2     /// <summary>
3     /// Avvia produzione
4     /// </summary>
5     [ExportMethod]
6     public void pr_AvviaLocale(long id)
7     {
8         Project.Current.GetVariable(VariablePaths.PathOdlStart).Value = id;
9         Project.Current.GetVariable(VariablePaths.Pathap_start).Value = true;
10    }
11 ...

```

Per quanto riguarda *Termina Produzione* invece, viene semplicemente impostata la variabile *pr\_ButtonTerminaSelected = TRUE*, invocando la macchina a stati. Quest'ultima, nel passo corrispondente, richiama la funzione *pr\_TerminateAllRunningLocale*, che si occupa per

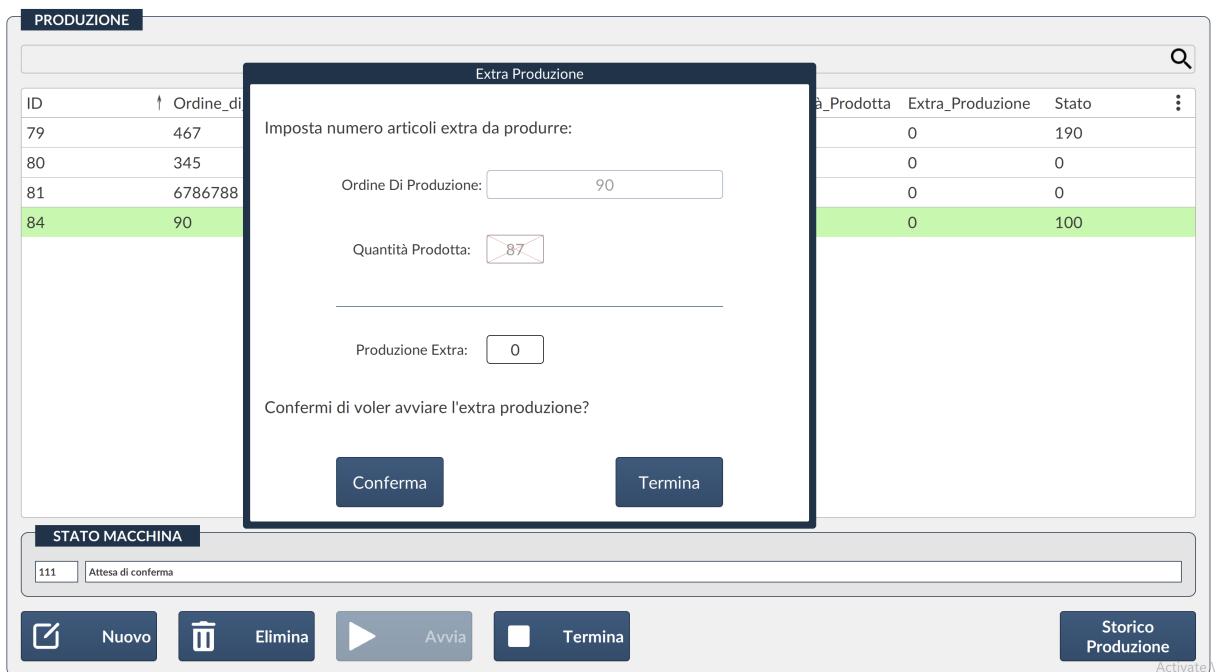


Figura 3.5: Esempio richiesta di *Extra Produzione*

spostare l'ordine nello storico della produzione. Nella funzione indicata e successivamente illustrata, il sistema procede dapprima ad individuare l'ordine con lo stato 160 "di aggiornamento tabelle e spostamento nello storico" (righe 8-9). Poi, una volta trovato l'ordine, viene creato un nuovo oggetto temporaneo in cui vengono inseriti tutti gli attributi dell'ordine, con l'aggiunta della data di terminazione della produzione (da riga 17 a riga 25). Infine, l'oggetto risultante viene trasferito nella tabella di *Storico Produzione* (riga 28).

```

1 ...
2     /// <summary>
3     /// Recupera i dati delle produzioni terminate e li sposta nello storico
4     /// </summary>
5     public void pr_TerminateAllRunningLocale()
6     {
7         object[,] result;
8         _store = Project.Current.Get<Store>(DATASTORE_DATABASE);
9         _store.Query($"SELECT * FROM {TABLE_NAME} WHERE \"/Status\" = 160", out _, out result);
10
11        if (result.GetLength(0) == 0 || result.GetLength(1) == 0)
12        {
13            Log.Warning("pr_TerminateAllRunning no record found");
14            return;
15        }
16
17        ReaToClienteLocale prod = new ReaToClienteLocale();
18        prod.Production_Command = "" + result[0, 0];
19        prod.Product_ID = (string)result[0, 3];
20        prod.dt_start = (DateTime?)result[0, 4];
21        prod.dt_end = DateTime.Now;
22        prod.Quantity_Requested = (long)result[0, 8];
23        prod.Quantity_Produced = (long)result[0, 9];
24        prod.Total_Reject = (long)result[0, 11];
25        prod.Extra_Production = (long)result[0, 10];
26
27        _storicoprod = new RuntimeNetLogicReaToClienteStoricoLocale();

```

Figura 3.6: Illustrazione sezione *Storico Produzione*

```

28     _storicoprod.sp_InsertLocale(prod);
29
30 }
31 ...

```

### 3.3.3 Storici Produzione/Allarmi

Lo storico di produzione risulta accessibile tramite pulsante apposito, inserito nell’interfaccia di *Produzione*. Al suo interno è stata progettata un’interfaccia simile a quella di *Produzione*, ma con alcune differenze. Nello specifico, non è presente nessun pulsante per l’interazione diretta con la macchina, ma solo un *DataGridView* che mostra tutte le informazioni sugli ordini prodotti. Difatti, le uniche iterazioni disponibili avvengono tramite un filtro di ricerca simile a quelli già trattati in precedenza, con l’aggiunta della possibilità di effettuare ricerche basate su data, o su un determinato periodo di tempo. La popolazione di questa tabella avviene solamente tramite la funzione analizzata nella sezione precedente, seguendo pressoché lo stesso approccio utilizzato per l’interfaccia di *produzione*. Per quanto riguarda la parte back-end, è stato dapprima creato un *RecipeSchema* contenente gli attributi necessari e, successivamente, sulla base del *RuntimeNetLogic* fornito, è stata implementata la funzione di inserimento nel database *sp\_InsertLocale*. Questa funzione, come già descritto nelle sezioni precedenti, si occupa di inserire l’oggetto *prod*, completo di tutti i suoi attributi nella tabella dello storico.

```

1 ...
2 public class RuntimeNetLogicReaToClienteStoricoLocale : BaseNetLogic
3 {
4     ... //Altro codice fornito da FT Optix
5
6     /// <summary>
7     /// Inserisce un record nello storico
8     /// </summary>
9     public void sp_InsertLocale(ReaToClienteLocale prod)
10    {
11        try
12        {
13            string[] columns = { "/Production_Command", "/Product_ID", "/dt_start", "/dt_end", "/Quantity_Requested", "/Quantity_Produced",
14                           "/Total_Reject", "/Extra_Production" };
15
16            var values = new object[1, 8];
17            values[0, 0] = prod.Production_Command;
18            values[0, 1] = prod.Product_ID;
19            values[0, 2] = prod.dt_start;
20            values[0, 3] = prod.dt_end;
21            values[0, 4] = prod.Quantity_Requested;
22            values[0, 5] = prod.Quantity_Produced;
23            values[0, 6] = prod.Total_Reject;
24            values[0, 7] = prod.Extra_Production;
25
26            _store = Project.Current.Get<Store>(DATASTORE_DATABASE);
27            _table = _store.Tables.Get<Table>(TABLE_NAME);
28            _table.Insert(columns, values);
29
30        }
31        catch (Exception ex)
32        {
33            Log.Warning($"ERROR: sp_InsertLocale {ex.Message}");
34        }
35    }
36}
37 ...

```

Per quanto riguarda la gestione degli allarmi<sup>[14]</sup>, ci sono diversi aspetti da analizzare. Innanzitutto, dal punto di vista progettuale sono stati creati due pannelli dedicati: uno per la visualizzazione della lista di allarmi e un altro per i warning attivi. Di seguito, la schermata rimane essenziale, includendo solamente i pulsanti per la gestione di alcuni allarmi e warning, oltre alla sezione di *Storico Allarmi*. A differenza di altre sezioni, gli allarmi non vengono gestiti tramite database ma direttamente da FT Optix™. Infatti, all'interno della cartella di progetto è presente una sotto-cartella dedicata a contenere solamente allarmi e warning (3.7a) dell'impianto.

Attraverso l'interfaccia di programma è, inoltre, possibile impostare diverse proprietà per ogni singolo allarme, come ad esempio il messaggio da visualizzare nello SCADA, la priorità e i tempi di visualizzazione del segnale (3.7b). Tutta la parte di gestione degli allarmi viene poi effettuata da FT Optix™. Infine, se è necessario visualizzare i messaggi di allarme in altre sezioni, come ad esempio sul Pannello di Stato, è sufficiente richiamarli tramite funzioni specifiche di FT Optix™. In particolare, nel nostro caso, grazie alla libreria `AllarmBannerlogic` il processo di visualizzazione degli allarmi viene automatizzato. Questo approccio permette di semplificare il lavoro, poiché è necessario solo inserire gli allarmi nella cartella di progetto. Naturalmente in sistemi SCADA più complessi è possibile lavorare manualmente tramite script sulla libreria, permettendo di personalizzare ulteriormente la gestione degli allarmi.

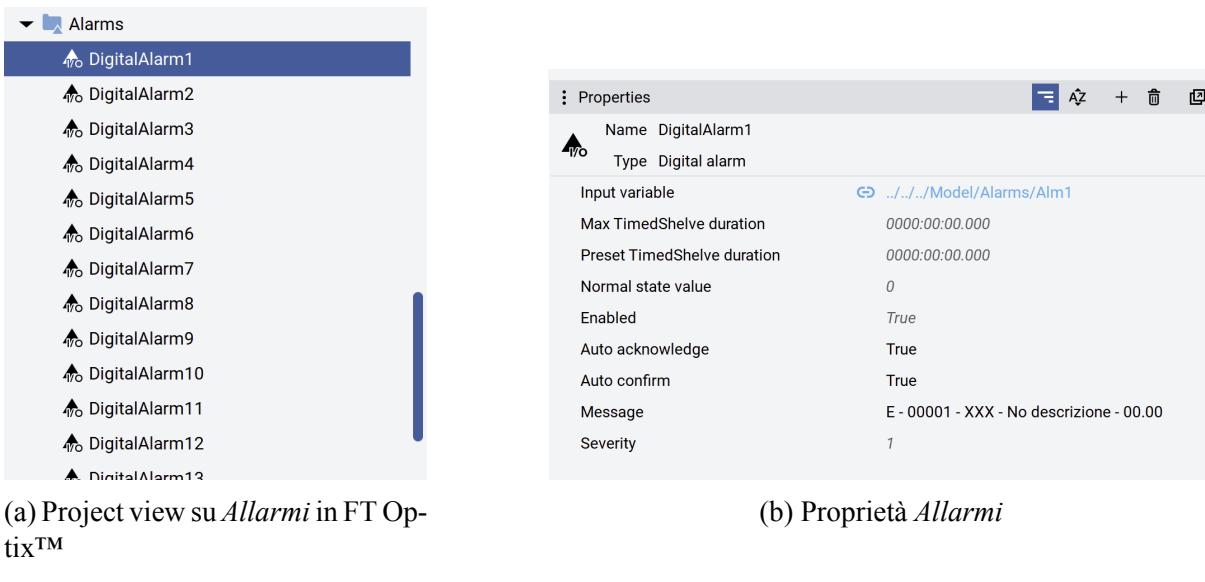


Figura 3.7: Illustrazione pannelli presenti in FT Optix™

### 3.3.4 Configurazione Database aziendale

Per quanto riguarda la parte di database aziendale, in questa sottosezione ci occuperemo della configurazione base di un database di tipo SQL Server, come già visto per SQLite. Inizialmente andiamo ad aggiungere all'interno della Project View, nella cartella DataStores, un oggetto di tipo *ODBC database*<sup>[15]</sup>. A questo punto dobbiamo impostare sulle proprietà del database appena creato:

1. Il DBMS type uguale a SQL Server.
2. La struttura compresa dei vari campi delle tabelle, come già visto per SQLite.
3. Compilare i campi con le informazioni di accesso a SSMS
4. Un *DesignTimeNetLogic*, chiamato DesignTimeConfiguraDB in cui andremo ad inserire informazioni fondamentali per l'utilizzo del database esterno.

Successivamente non resta altro che replicare tutto ciò che è stato fatto per SQLite con le diverse variabili di modello, ricette e script specifici per far funzionare il database. Prima di passare al NetLogic di configurazione, è bene sottolineare che la scelta di utilizzare un database rispetto all'altro si trova all'interno della sezione Impostazioni, con la prima flag presente nel pannello. Grazie a questa flag viene cambiato il valore di una variabile booleana DB, impostando l'utilizzo delle tabelle corrette. Inoltre, questa variabile agisce sull'interfaccia di *Produzione* e *Articoli* visualizzata dall'operatore, che ha a disposizione due sezioni di facile utilizzo e comprensione per chi opera direttamente sul campo. Per quanto riguarda DesignTimeConfiguraDB, è stata innanzitutto creata la classe con le costanti di riferimento del database:

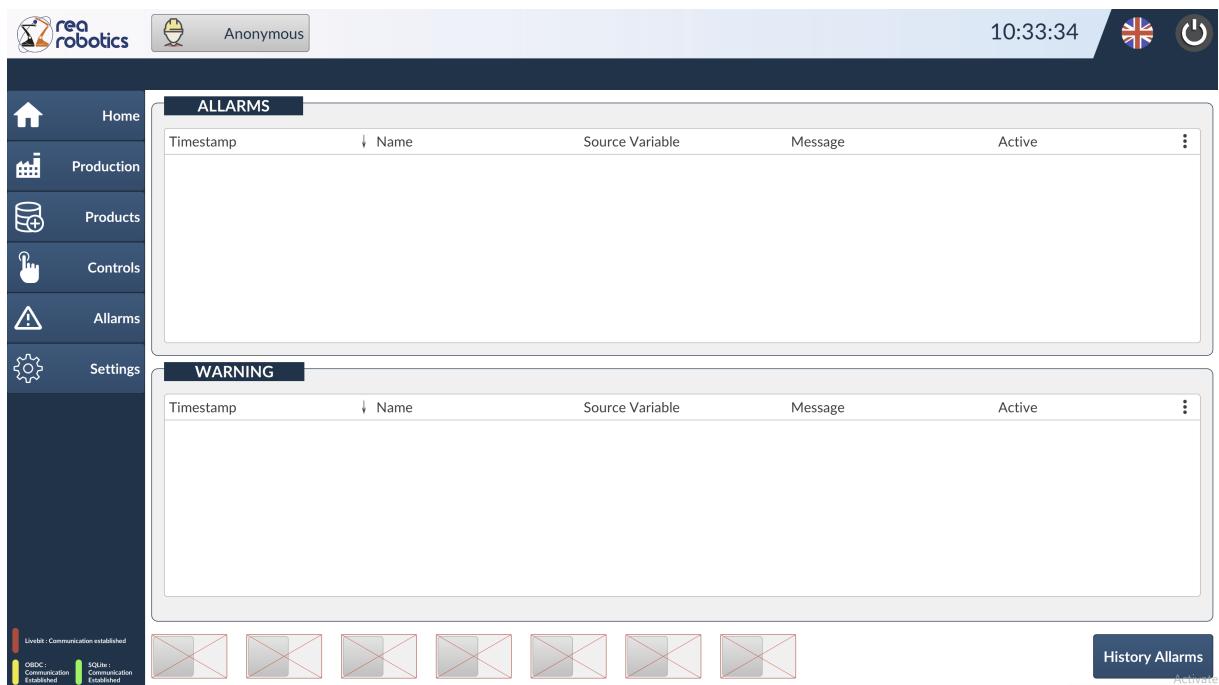


Figura 3.8: Illustrazione sezione *Allarmi*

```

1 public class DesignTimeConfiguraDB : BaseNetLogic
2 {
3     const string DB_SERVER = ".\\SQLEXPRESS";
4     const string DB_NAME = "PRG_OPTIX";
5     const string DB_USER = "sa";
6     const string DB_PASSWORD = "reaSQL";
7
8     private string _connectionString = $"Server={DB_SERVER};Database={DB_NAME};User Id={DB_USER};Password={DB_PASSWORD};";
9
10    ...

```

Per poi configurare tramite il metodo `ConfigureDB` l'accesso al database ed aggiungere eventuali tabelle necessarie al datastore OBDC di FT Optix™. Quindi, è stata copiata la libreria `sni.dll` per fornire comunicazione con SQL Server, per poi aggiungere le tabelle necessarie:

```

1 ...
2     sourcePath = Path.Combine(RuntimePath, "sni.dll");
3     destinationPath = Path.Combine(RuntimePath, "NetSolution", "bin", "sni.dll");
4     System.IO.File.Copy(sourcePath, destinationPath, true);
5 ...
6     //aggiungo le tabelle necessarie con le relative colonne
7     List<string> tables = new List<string>();
8     tables = ListTables();
9
10    if (tables.Count>0 )
11    {
12        foreach (string tbl in tables)
13        {
14            AddNewTable(db, tbl);
15        }
16    }
17 ...

```

Infine vengono gestiti i tipi di dati SQL, mappandoli correttamente:

```

1 ... static Type MapSqlType(string sqlType)
2 {
3     switch (sqlType.ToLower())
4     {
5         case "int":
6             return typeof(int);
7         case "varchar":
8         case "nvarchar":
9         case "char":
10        case "nchar":
11            return typeof(string);
12        // Aggiungi altri casi secondo necessità
13        default:
14            return typeof(object); // Tipo generico per altri tipi non gestiti
15        }
16    }
17 }
```

Questo script automatizza la configurazione del collegamento ad un database SQL Server garantendo che le tabelle e le colonne siano mappate correttamente nell'ODBC dell'HMI. Non di meno, lo script gestisce la copia delle librerie necessarie, le connessioni e soprattutto il recupero dinamico delle strutture delle tabelle, rendendo le tabelle facilmente scalabili.

## 3.4 Sviluppo back-end e integrazione con il PLC

In questa sezione analizziamo gli aspetti necessari per poter unire tutto ciò che è stato progettato finora. Lato back-end, dobbiamo garantire che lo SCADA comunichi correttamente con uno o più PLC, a seconda dell'impianto, ottenendo pieno controllo del sistema. Questo approccio permette di gestire qualsiasi caso d'uso o variabile che potrebbe comportarsi in modo imprevisto durante l'operatività della macchina. Ho deciso di focalizzare l'attenzione su due elementi chiave: la possibilità che lo SCADA determini esattamente il passo operativo del PLC in ogni momento, e la certezza che ci sia una comunicazione bidirezionale tra lo SCADA e il PLC. Per arrivare a ciò, è stata creata una macchina a stati, con un ciclo che inizia nel momento in cui la macchina viene avviata e che avanza esclusivamente in risposta ai segnali del PLC. Questa macchina rappresenta un'astrazione del comportamento del PLC, offrendo pieno controllo in qualsiasi momento. Il lavoro è stato suddiviso in più fasi:

1. Inizializzazione di tutti i tag del PLC e dello SCADA.
2. Sviluppo di un programma in grado di gestire tutti i casi d'uso e gli errori dell'impianto.
3. Verifica che tutto sia coerente tra FT Optix™ e il Database.

### 3.4.1 Indicizzazione tag variabili

L'idea a livello progettuale è stata di creare un contenitore, implementato sotto forma di classe, che raccogliesse tutti i percorsi delle variabili presenti su FT Optix™, inclusi quelli del PLC.

Il primo passo consiste nell'identificare, su FT Optix™, tutte le variabili di nostro interesse. È fondamentale che sia attivata la modalità avanzata in FT Optix™ per eseguire i passi seguenti. Attraverso la *Project View* vanno estrapolati i percorsi delle variabili. Ad esempio, consideriamo una variabile di modello scelta casualmente (3.9). Una volta estratto il percorso, si ottiene una stringa di questo tipo: SCADA/Model/Produzione/ResetProduction. Da quest'ultima conviene sempre escludere la prima parte del percorso, che rappresenta il nome del progetto, e utilizzare solo la parte rimanente. Questa scelta è motivata dal fatto che la prima parte, nel nostro caso: SCADA/ rappresenta il nome del Progetto, e se dovessimo riutilizzarne tutto il modello in contesto multi-progetto, dovremmo aggiornarne globalmente tutte. Riducendo il percorso alla sola parte: Model/Produzione/ResetProduction, si garantisce maggior flessibilità e portabilità.

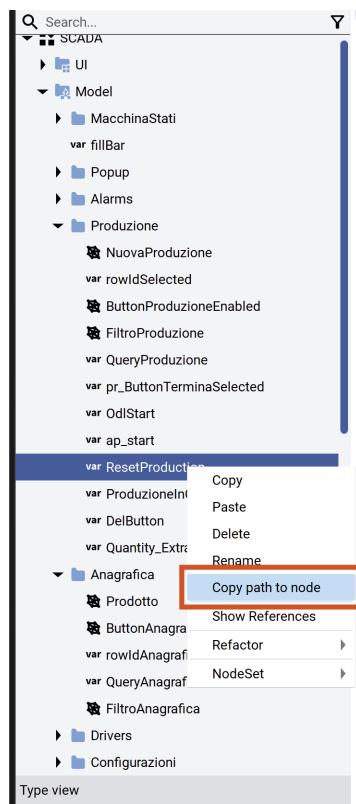


Figura 3.9: esempio di salvataggio percorso variabile di modello

Procediamo a creare la classe e ad inserire volta per volta tutti i percorsi delle variabili necessarie. Alla fine dovremo ottenere una classe del genere:

```

1 ...
2
3 public static class VariablePaths
4 {
5     ... // lista variabili per HMI
6
7     //cliente_to_rea -> produzione

```

```

8   public const string PathQueryProduzione          = "Model/Produzione/QueryProduzione";
9   public const string PathProduzioneFilterActive  = "Model/Produzione/FiltroProduzione/FilterActive";
10  public const string PathProduzioneTextFilter    = "Model/Produzione/FiltroProduzione/TextFilter";
11  public const string PathProduzioneAvviaEnabled  = "Model/Produzione/ButtonProduzioneEnabled/AvviaEnabled";
12 ...
13
14 //rea_to_cliente -> storico
15 public const string PathQueryStorico           = "Model/Storico/QueryStorico";
16 public const string PathStoricoDateFilter      = "Model/Storico/bdateFilter";
17 public const string PathStoricoDateFrom        = "Model/Storico/DateFrom";
18 public const string PathStoricoDateTo          = "Model/Storico/DateTo";
19
20 //anagrafica
21 public const string PathQueryAnagrafica         = "Model/Anagrafica/QueryAnagrafica";
22 public const string PathAnagraficaFilterActive  = "Model/Anagrafica/FiltroAnagrafica/FilterActive";
23 public const string PathAnagraficaTextFilter    = "Model/Anagrafica/FiltroAnagrafica/TextFilter";
24 ...
25
26 ... // lista variabili per HMI

```

Nella classe sono state inserite variabili secondo una precisa struttura: il percorso salvato in precedenza associato ad una stringa. Questo approccio consente maggiore controllo delle variabili all'interno della macchina a stati e, qualora sia necessario, permette di aggiungere nuove variabili semplicemente aggiornando la suddetta classe. È inoltre consigliato, ma non estremamente essenziale, suddividere le variabili secondo una logica ben definita in modo che siano facilmente rintracciabili, specialmente in impianti estremamente più strutturati.

Per la gestione delle variabili nella macchina a stati è stata definita una regola di suddivisione in due tipologie principali per fornire maggiore sicurezza:

1. **DB92-PLCtoHMI**: sono tutte le variabili gestite dal controllore logico programmabile e inviate all'HMI, che le può leggere ma non modificare.
2. **DB91-HMItоПLC**: variabili gestite sempre dal PLC, ma scrivibili solo dallo scada e leggibili dal PLC.

Questa classificazione rimane costante indipendentemente dall'impianto; ciò che cambierà sono la quantità e il tipo di segnale dei tag. Nel caso specifico, verranno considerate le variabili utilizzate in un impianto di piegatura, che sarà analizzato più dettagliatamente nel capitolo 4.

### 3.4.2 Sviluppo Macchina a Stati

Lato progettazione, è stato inserito un oggetto *RunTimeNetLogic* associato alla *MainWindow*. Per quanto concerne la programmazione, lo script associato è stato suddiviso in diverse parti. Inizialmente, vengono dichiarati alcuni script strettamente legati al database, che vengono richiamati all'interno della *StateMachine*. Successivamente, all'interno della funzione *Start* (riga 19), è stata definita una *PeriodicTask* con i seguenti attributi:

- *Maincycle*: contiene la logica periodica del sistema.

- 250ms: indica la frequenza con cui la logica viene ripetuta periodicamente.
- LogicObject: contiene le risorse necessarie per l'esecuzione.

Grazie a questa configurazione, dopo aver avviato l'impianto, il maincycle viene eseguito ogni 250ms. Questa frequenza garantisce una sincronizzazione quasi istantanea tra HMI e PLC, riducendo al minimo gli errori.

```

1 ...
2 public class script_maincycle : BaseNetLogic
3 {
4     private PeriodicTask myPeriodicTask;
5     private RuntimeNetLogicClienteToRea _prod;
6     private RuntimeNetLogicAnagrafica _art;
7
8     private RuntimeNetLogicClienteToReaLocale _prodLocale;
9     private RuntimeNetLogicAnagraficaLocale _artLocale;
10
11    public override void Start()
12    {
13        _prod = new RuntimeNetLogicClienteToRea();
14        _art = new RuntimeNetLogicAnagrafica();
15
16        _prodLocale = new RuntimeNetLogicClienteToReaLocale();
17        _artLocale = new RuntimeNetLogicAnagraficaLocale();
18
19        myPeriodicTask = new PeriodicTask(Maincycle, 250, LogicObject);
20        myPeriodicTask.Start();
21    }
22
23
24    public override void Stop()
25    {
26        myPeriodicTask.Dispose();
27    }
28
29    public void Maincycle()
30    {
31        StateMachine();
32    }
33
34    private void StateMachine()
35    {
36        ...

```

La struttura della StateMachine è stata organizzata in diverse sezioni. La prima si occupa di inizializzare le variabili, richiamando i percorsi definiti nella classe VariablePaths illustrata in precedenza, che contiene tutti i tag di interesse. Di seguito, una parte del codice illustrato:

```

1 ...
2     private void StateMachine()
3     {
4         //inizializzo variabili plc
5         PopUpNetLogic popup = new PopUpNetLogic();
6         var popupOK = Project.Current.GetVariable(VariablePaths.PathPopupOK);
7         var popupYes = Project.Current.GetVariable(VariablePaths.PathPopupYes);
8         var popupNo = Project.Current.GetVariable(VariablePaths.PathPopupNo);
9
10        var MachineStatusText = Project.Current.GetVariable(VariablePaths.PathMachineStatusText);
11        var MachineStatus = Project.Current.GetVariable(VariablePaths.PathMachineStatus);
12        var OdlStart = Project.Current.GetVariable(VariablePaths.PathOdlStart);
13        long OdlStartLong = OdlStart.Value;
14        var ap_start = Project.Current.GetVariable(VariablePaths.Pathap_start);
15        var pr_ButtonTerminaSelected = Project.Current.GetVariable(VariablePaths.Pathpr_ButtonTerminaSelected);
16        var ResetProduction = Project.Current.GetVariable(VariablePaths.PathResetProduction);

```

```

17 var ProduzioneInCorso           = Project.Current.GetVariable(VariablePaths.PathProduzioneInCorso);
18
19 var DB91_CambioProduzione      = Project.Current.GetVariable(VariablePaths.PathDB91_CambioProduzione);
20 var DB91_TerminaProduzione     = Project.Current.GetVariable(VariablePaths.PathDB91_TerminaProduzione);
21 ... // altre variabili plc

```

Dopo l'indicizzazione, la macchina statì è composta da vari passi definiti all'interno di uno switch-case e numerati da 0 a 199, ciascuno per uno specifico stato operativo. Questo approccio logico è ampiamente utilizzato negli impianti industriali in REA, in quanto consente di gestire in modo modulare e scalabile le varie operazioni dallo scada oltre a facilitare la manutenzione futura. Di seguito, una panoramica dei passi standard:

- 0 = "Inizializzazione": passo che identifica l'avvio della macchina, solitamente corrisponde alla schermata di caricamento dell'HMI.
- 1 = "Stato iniziale (IDLE)": passo in cui avviene un reset completo delle variabili HMI e DB91.
- 10 = "Controllo allineamento con PLC-SCADA": sincronizzazione tra SCADA e PLC.
- 20 = "Attesa avvio produzione": stato di attesa, con macchina operativa per la produzione.
- 70 = "Attesa esito da PLC - ok/ko": attesa del segnale di risposta dal PLC dopo un comando inviato dall'HMI.
- 80 = "Attesa RESET segnali da PLC - ok/ko": verifica di sicurezza, attesa ulteriore segnale dal PLC.
- 100 = "IN LAVORO": identifica che l'impianto è in piena attività produttiva.
- 150 = "Fine produzione + attesa handshake PLC": stato di attesa del segnale di handshake da parte del PLC.
- 165 = "Attesa reset richiesta termina produzione dal PLC": pulizia delle variabili per garantire la sicurezza operativa.
- 190 = "ANNULLAMENTO CARICAMENTO DA OPERATORE" annullamento di qualsiasi produzione attiva o in chiusura.
- 199 = "Termina produzione completata correttamente" controllo finale e ritorno al passo 20.

Come si può notare, non tutti i passi seguono un ordine sequenziale; questo consente di aggiungere eventuali estensioni future agli impianti. Partiamo con l'analizzare il codice di tutti i passi principali, escludendo l'implementazione di alcuni per l'impianto di piegatura, che analizzeremo successivamente nel capitolo 4.

Il codice di `case 0` rappresenta la prima iterazione del ciclo e quindi l'avvio della macchina. Qui si gestisce il caricamento iniziale, inclusa l'interfaccia HMI. Viene monitorata la variabile `fillBar`, cioè la barra di caricamento, per verificare che l'intero programma sia caricato (righe 7-10). Una volta completato il caricamento, lo stato viene impostato a 1, e nell'HMI viene caricata la pagina iniziale completa di tutta l'interfaccia (righe 13-20). Infine esce dal passo.

```

1 ...
2     case 0:
3         //-----
4         MachineStatusText.Value = "Inizializzazione";
5         //-----
6
7         if (Project.Current.GetVariable("Model/fillBar").Value <15)
8         {
9             Project.Current.GetVariable("Model/fillBar").Value += 1;
10        }
11        else
12        {
13            MachineStatus.Value = 1;
14
15            //cambia pagina
16            var myPanel = LogicObject.GetPanelLoader("PanelLoader1");
17            myPanel.ChangePanel("Main");
18
19            var mainPanel = LogicObject.GetPanelLoader("PanelLoaderScreens");
20            mainPanel.ChangePanel("Home");
21        }
22
23        break;
24 ...

```

Il `case 1` rappresenta uno dei passi più critici, poiché si occupa di controllare variabili, allarmi e sensori sia dell'HMI che del PLC. Questo passo è importante per garantire continuità in caso di blackout o spegnimenti anomali del pannello HMI, che potrebbero de sincronizzare PLC-SCADA. Al suo interno vengono eseguiti i seguenti reset:

1. `DB91_CambioProduzione.Value`: impostato a false (riga 7).
2. `ResetHMIProductVar()`: richiama la funzione di reset delle variabili HMI di produzione (riga 10).
3. `ResetPLCProductVar()`: richiama la funzione di reset delle variabili PLC (riga 13).

Concluso il reset, la macchina imposta il passo successivo, cioè 10, ed esce dal passo (riga 15).

```

1 ...
2     case 1:
3         //-----
4         MachineStatusText.Value = "Stato iniziale (IDLE)";
5         //-----

```

```

6      DB91_CambioProduzione.Value = false;
7
8      //Reset di tutte le variabili legate al caricamento programma su HMI
9      ResetHMIProductVar();
10
11
12      //Reset variabili PLC DB91
13      ResetPLCProductVar();
14
15      MachineStatus.Value = 10;
16
17      break;
18  ...

```

Dal `case 10` tutto il codice è stato strutturato con l'implementazione sia per il database locale, sia per il database aziendale. Tuttavia, in questa spiegazione ci concentriamo su una singola implementazione, poiché le logiche principali rimangono identiche. Per prima cosa avviene un check sulla variabile `DBExpress.Value` configurata nelle impostazioni dell'HMI che determina quale database è in utilizzo (riga 7). Fatto questo avviene un confronto diretto con la variabile di status nel PLC (riga 17): se maggiore di 0, il PLC sta già lavorando ad un ordine. In questo caso, tutte le variabili dello SCADA vengono aggiornate per riallinearsi con il PLC e continuare la produzione senza problemi (righe 20-29); se invece `DB92_ODP.Value` è uguale a zero, il sistema procede al passo successivo senza ulteriori azioni (riga 33).

```

1 ...
2     case 10:
3     //-----
4     MachineStatusText.Value = "Controllo allineamento con PLC-SCADA";
5     //-----
6
7     if (DBExpress.Value)
8     {
9         ... //implementazione per SSMS
10    }
11    else
12    {
13        //sincronizzo il campo /Status con lo stato corretto dell'ordine che sta girando sul PLC
14        _prodLocale.pr_StatusSyncro_Locale(DB92_ODP.Value);
15
16        //se in plc sto lavorando con una ricetta allineo tabella in running
17        if (DB92_ODP.Value > 0)
18        {
19            //Sincronizzo OdlStart
20            OdlStartLong = DB92_ODP.Value;
21
22            //mando prodotti al plc
23            SendProductDataToPLCLocale(OdlStartLong);
24
25            //mi metto in produzione in corso
26            ProduzioneInCorso.Value = true;
27
28            //vado in produzione
29            MachineStatus.Value = 100;
30        }
31        else
32        {
33            MachineStatus.Value = 20;
34        }
35    }
36
37    break;
38 ...

```

Il case 20 rappresenta lo stato in cui l'impianto è in attesa avvio produzione, monitorando continuamente le seguenti variabili (riga 12):

- `ap_start.Value`: è stato avviato un ordine dall'operatore.
- `OdlStartLong`: è maggiore di zero se esiste un ordine di produzione in attesa.

Se avvengono queste condizioni, i dati relativi all'ordine vengono inviati al PLC tramite funzione `SendProductDataToPLCLocale` (riga 15). Se avviene l'invio, viene impostata `ProduzioneInCorso.Value` a TRUE (riga 18), con cambio al passo successivo (riga 21). In caso contrario viene aperto un popup di avvertimento e conseguentemente cambiato il passo (righe 26, 27).

```

1 ...
2     case 20:
3     //-----
4     MachineStatusText.Value = "Attesa avvio produzione";
5     //-----
6     if (DBExpress.Value)
7     {
8         ... //implementazione per SSMS
9     }
10    else
11    {
12        if (ap_start.Value && OdlStartLong > 0)
13        {
14            //mando dati al plc
15            if (SendProductDataToPLCLocale(OdlStartLong))
16            {
17                //vado in produzione in corso
18                ProduzioneInCorso.Value = true;
19
20                //cambio stato
21                MachineStatus.Value = 25;
22            }
23            else
24            {
25                //Dati anagrafica non esistenti, popup e KO diretto
26                popup.OpenPopUp("Dati anagrafica non esistenti: crea prima l'articolo in Anagrafica", 0);
27                MachineStatus.Value = 21;
28            }
29        }
30    }
31
32    break;
33 ...

```

Il case 21 gestisce l'attesa di lettura del popup da parte dell'operatore, procedendo al ripristino delle variabili di produzione e al cambio del passo. Nel case 25 viene impostata la variabile di cambio produzione a TRUE (riga 13), per avvisare il PLC del cambio di ordine. Viene aggiornato l'attributo dello stato dell'ordine nel database (riga 16), e viene richiamata la funzione per abilitare/disabilitare i pulsanti della schermata di produzione. Infine cambiamo passo (riga 19).

```

1 ...
2     case 25:
3 //-----
4     MachineStatusText.Value = "Invio cambio produzione al PLC";
5 //-----
6     if (DBExpress.Value)
7     {
8         ... //implementazione per SSMS
9     }
10    else
11    {
12        //Handshake PLC - attendo risposta
13        DB91_CambioProduzione.Value = true;
14
15        //Aggiorno stato in db
16        _prodLocale.pr_UpdateStatusLocale(0dlStartLong, MachineStatus.Value);
17
18        //Aggiorno pulsanti
19        _prodLocale.pr_ManageProductionButtonsLocale(MachineStatus.Value);
20
21        //Cambio stato
22        MachineStatus.Value = 50;
23    }
24
25    break;
26 ...

```

I passi `case 50, 55, 60` verranno trattati nel Capitolo 4, essendo progettati secondo l'impianto su cui gira lo SCADA. Se tutto risulta corretto arriviamo al `case 70` in cui siamo in attesa dell'esito da parte del PLC relativo al cambio produzione:

1. Il PLC da l'OK (da riga 8): allora vengono resettate le variabili precedenti di cambio produzione e caricamento ordine, per poi cambiare passo.
2. Il PLC manda un KO (da riga 21): allora l'HMI mostra a schermo l'errore e rimane in attesa di risposta da parte dell'operatore.

```

1 ...
2     case 70:
3 //-----
4     MachineStatusText.Value = "Attesa esito da PLC - ok/ko";
5 //-----
6
7     //Se ricevo OK
8     if (DB92_CambioProduzioneOK.Value)
9     {
10         //reset bit di cambio produzione
11         DB91_CambioProduzione.Value = false;
12
13         //reset richiesta caricamento nuovo prodotto
14         ap_start.Value = false;
15
16         //cambio stato
17         MachineStatus.Value = 80;
18     }
19
20     //Se ricevo KO
21     if (DB92_CambioProduzioneKO.Value)
22     {
23         //mostra popup
24         popup.OpenPopUp("KO da PLC: dati errati", 0);
25
26         //Cambio stato
27         MachineStatus.Value = 71;

```

```

28     }
29
30     break;
31 ...

```

Il case 71 gestisce il reset della variabile del popup mostrato in errore e passa direttamente al 180 (ne discuteremo l'implementazione in seguito). Il case 80 attende che il PLC resetti entrambi i segnali precedenti (riga 12), successivamente cambia il passo per avviare la produzione (riga 15), e aggiorna nuovamente lo stato dell'ordine nel database (riga 18) e i pulsanti dell'HMI (riga 21).

```

1 ...
2 ...
3     case 80:
4     //-----
5     MachineStatusText.Value = "Attesa RESET segnali da PLC - ok/ko";
6     //-----
7     if (DBExpress.Value)
8     {
9         ... //implementazione per SSMS
10    }
11    else
12    {
13        if (!DB92_CambioProduzioneOK.Value && !DB92_CambioProduzioneKO.Value)
14        {
15            //Cambio stato
16            MachineStatus.Value = 100;
17
18            //Aggiorno status in db
19            _prodLocale.pr_UpdateStatusLocale(0dlStartLong, MachineStatus.Value);
20
21            //Aggiorno pulsanti
22            _prodLocale.pr_ManageProductionButtonsLocale(MachineStatus.Value);
23        }
24    }
25
26     break;
...

```

Il case 100 rappresenta il cuore del processo produttivo, in cui la macchina lavora sull'ordine e aggiorna continuamente i dati relativi alla produzione. Per prima cosa viene selezionato il database con annessi campi per l'aggiornamento dei dati durante la produzione. Successivamente vengono mandate continuamente richieste al PLC in cui viene controllato che i pezziDepositati, quindi i pezzi prodotti, siano uguali a quelli memorizzati dall'HMI. In caso di discrepanza, si aggiorna il database e si sincronizzano i valori in memoria (righe 10-14). Questo avviene finché non si raggiunge la quantità richiesta, sommata all'extra produzione (riga 18), che inizialmente sarà a 0. Di seguito se l'operatore preme il pulsante di fine produzione, resetta il flag e cambia stato a fine produzione (righe 21-24). Nel codice sottostante vengono illustrate anche alcune parti fondamentali delle funzioni ausiliarie di aggiornamento parametri richiamate precedentemente.

```

1 ...
2 ...
3     case 100:
4     //-----
5     MachineStatusText.Value = "IN LAVORO";
//-----

```

```

6 // Seleziona il data store e i nomi dei campi in base a DBExpress.Value
7 ...
8
9 // Aggiorna le quantità se necessario
10 if (DB92_PezziDepositati.Value != Mem_PezziDepositati.Value
11     DB92_PezziScarti.Value != Mem_PezziScarti.Value
12     DB92_QtaRiordino.Value != Mem_QtaRiordino.Value)
13 {
14     AggiornaQuantita(myStore, tableName, idField, quantityProducedField, totalRejectField);
15 }
16
17 // Verifica se è stata raggiunta la quantità richiesta più l'extra produzione
18 ControllaExtraProduzione(myStore, tableName, idField, quantityRequestedField, extraProductionField);
19
20 // Gestisce la richiesta di fine produzione
21 if (pr_ButtonTerminaSelected.Value)
22 {
23     pr_ButtonTerminaSelected.Value = false;
24     MachineStatus.Value = 148; // Codice di stato per "fine produzione"
25 }
26
27 // Funzione per aggiornare le quantità
28 void AggiornaQuantita(... //variabili richieste)
29 {
30     ...
31     // Aggiorna quantità pezzi prodotti
32     if (DB92_PezziDepositati.Value != Mem_PezziDepositati.Value)
33     {
34         string query = $"UPDATE {table} SET {quantityProducedFieldName} = '{(uint)DB92_PezziDepositati.Value}' WHERE {idFieldName} =
35             '{OdlStartLong}'";
36         store.Query(query, out Header, out ResultSet);
37         Mem_PezziDepositati.Value = DB92_PezziDepositati.Value;
38     }
39
40     // Aggiorna quantità scarti
41     if (DB92_PezziScarti.Value != Mem_PezziScarti.Value)
42     {
43         string query = $"UPDATE {table} SET {totalRejectFieldName} = '{(uint)DB92_PezziScarti.Value}' WHERE {idFieldName} =
44             '{OdlStartLong}'";
45         store.Query(query, out Header, out ResultSet);
46         Mem_PezziScarti.Value = DB92_PezziScarti.Value;
47     }
48
49     // Aggiorna quantità di riordino
50     if (DB92_QtaRiordino.Value != Mem_QtaRiordino.Value)
51     {
52         string query = $"UPDATE {table} SET {quantityProducedFieldName} = '{(uint)DB92_PezziDepositati.Value}' WHERE {idFieldName} =
53             '{OdlStartLong}'";
54         store.Query(query, out Header, out ResultSet);
55         Mem_QtaRiordino.Value = DB92_QtaRiordino.Value;
56     }
57
58 // Funzione per controllare l'extra produzione
59 void ControllaExtraProduzione(... //variabili richieste)
60 {
61     ...
62     string query = $"SELECT {quantityRequestedFieldName}, {extraProductionFieldName} FROM {table} WHERE {idFieldName} =
63         '{OdlStartLong}'";
64     store.Query(query, out HeaderExtra, out ResultSetExtra);
65
66     int quantityRequested = Convert.ToInt32(ResultSetExtra[0, 0]);
67     int extraProduction = Convert.ToInt32(ResultSetExtra[0, 1]);
68
69     // Richiesta per extra-produzione
70     if (DB92_PezziDepositati.Value >= quantityRequested + extraProduction)
71     {
72         MachineStatus.Value = 110; // Codice di stato per "extra produzione raggiunta"
73     }
74     break;
75 }

```

Illustriamo anche la gestione di extra produzione tramite popup, eseguita tramite il codice se-

guente. Il tutto è strutturato su due case, il 110 che gestisce l'apertura del popup e il 111 che gestisce le risposte attese dal popup. Nello specifico deve gestire sia la terminazione della produzione senza aggiunta di extra (righe 10-14), sia, in caso affermativo, il ritorno allo stato di produzione (da riga 18).

```

1 ...
2 ...
3 ...
4 ...
5 ...
6 ...
7 ...
8 ...
9 ...
10 ...
11 ...
12 ...
13 ...
14 ...
15 ...
16 ...
17 ...
18 ...
19 ...
20 ...
21 ...
22 ...
23 ...
24 ...
25 ...
26 ...
27 ...
28 ...
29 ...
30 ...
31 ...
32 ...
33 ...
34 ...
35 ...
36 ...
37 ...
38 ...
39 ...
40 ...
41 ...
42 ...
43 ...
...

```

Se il processo produttivo è avvenuto correttamente, ci troveremo al case 148 in cui viene richiesto all'operatore di confermare la chiusura ordine. Le varie risposte sono gestite al case 149, in cui vengono gestiti i comandi di reset popup e il DB91\_TerminaProduzione è il prossimo passo a cui passare. Nel caso in cui venga confermata la chiusura ordine, lo scopo del case 150 è di rimanere in attesa dell'handshake da parte del PLC, cioè la variabile DB92\_AckTerminaProduzione. Se tutto avviene in modo corretto, vengono resettate tutte le variabili di produzione (righe 10-14) e si passa al case 160 in cui viene fatto l'aggiornamento delle tabelle spostando la produzione appena conclusa in storico produzione. Il case 165 monitora il reset della variabile DB92\_AckTerminaProduzione di handshake e imposta il passo direttamente a quello finale.

```

1 ...
2     case 150:
3     //-----
4     MachineStatusText.Value = "Fine produzione + attesa handshake PLC";
5     //-
6
7     //rimango in attesa dell'ack PLC
8     if (DB92_AckTerminaProduzione.Value)
9     {
10         DB91_TerminaProduzione.Value = false;
11
12         ProduzioneInCorso.Value = false;
13
14         MachineStatus.Value = 160;
15     }
16
17     break;
18
19     case 160:
20     //-----
21     MachineStatusText.Value = "Aggiornamento tabelle e spostamento nello storico";
22     //-
23     if (DBExpress.Value) {
24         ... //implementazione per SSMS
25     }
26     else
27     {
28         //Aggiorno stato in db
29         _prodLocale.pr_UpdateStatusLocale(0d1StartLong, MachineStatus.Value);
30
31         //Aggiorno pulsanti
32         _prodLocale.pr_ManageProductionButtonsLocale(MachineStatus.Value);
33
34         //sposto su storico e elimino record
35         _prodLocale.pr_TerminateAllRunningLocale();
36         _prodLocale.pr_EliminaLocale(0d1Start.Value);
37
38         //reset variabili
39         ResetHMIProductVar();
40         ResetPLCProductVar();
41
42         MachineStatus.Value = 165;
43     }
44
45     break;
46 ...

```

I passi case 180, 181, 190 e 191 servono per gestire i vari KO in cui ci siamo imbattuti durante tutto il processo e che sono stati rispettivamente richiamati.

- case 180 = errore caricamento ricetta KO da PLC: segnala un qualsiasi errore nel caricamento della ricetta da parte del PLC, aggiornando lo stato nel database e resettando le variabili sia del PLC, sia dell'HMI, per poi passare al passo successivo.
- case 181 = Attesa rimozione segnali KO da PLC: il sistema è in attesa che i segnali di KO vengano rimossi dal PLC, passando poi al passo 199.
- case 190 = annullamento caricamento da operatore: gestisce l'annullamento manuale del caricamento della ricetta da parte dell'operatore, aggiornando stato, pulsanti e resettando variabili. Di seguito viene riportata una parte del codice.
- case 191 = Attesa fine produzione da PLC: il sistema attende una conferma di terminazione di produzione da parte del PLC, passando poi al passo finale.

```

1 ...
2     case 190:
3 //-----
4     MachineStatusText.Value = "ANNULLAMENTO CARICAMENTO DA OPERATORE";
5 //-----
6     if (DBExpress.Value)
7     {
8         ... //implementazione per SSMS
9     }
10    else
11    {
12        //aggiorno status
13        _prodLocale.pr_UpdateStatusLocale(OdlStartLong, MachineStatus.Value);
14
15        //aggiorno pulsanti
16        _prodLocale.pr_ManageProductionButtonsLocale(MachineStatus.Value);
17
18        ResetHMIProductVar();
19        ResetPLCProductVar();
20
21        DB91_TerminaProduzione.Value = true;
22
23        MachineStatus.Value = 191;
24    }
25
26    break;
27 ...

```

Infine, il `case 199` finalizza il processo e riporta la macchina in attesa di un nuovo ciclo produttivo, quindi in attesa di avvio produzione `case 20`. Questo passo vale sia in caso di avvenuta produzione precedente, sia nel caso in cui non sia stato prodotto nulla per errori di percorso.

```

1 ...
2     case 199:
3 //-----
4     MachineStatusText.Value = "Termina produzione completata correttamente";
5 //-----
6
7     if (!DB92_AckTerminaProduzione.Value)
8     {
9         MachineStatus.Value = 20;
10    }
11
12    break;
13
14    default:
15        break;
16    }
17 ...

```

In questo capitolo abbiamo analizzato i punti salienti del main cycle. Per visionare il codice completo far riferimento alla repository GitHub, più precisamente al file: `script_maincycle.cs`.



# Capitolo 4

## Caso Studio: Isole robotizzate di piegatura

In questo capitolo analizziamo la messa a punto dello SCADA su un impianto demo, nello specifico una cella di presso-piegatura robotizzata fornita da Salvagnini, uno dei maggiori clienti di REA. Dopo aver progettato l’HMI, la fase successiva consiste nel far comunicare PLC allo SCADA attraverso l’importazione dei tag del PLC e la configurazione di questi ultimi su variabili di modello.

### 4.1 Importazione Tag

Per quanto riguarda l’importazione dei tag, esistono due modalità principali da seguire in FT Optix™:

1. **Collegamento diretto:** Viene sfruttato il protocollo interno di FT Optix™ per stabilire una connessione diretta tra software e PLC.
2. **Importazione da file di progetto:** Viene utilizzato un file con estensione .ap17, generato da tecnici PLC attraverso il software *Siemens TIA Portal*. Questo metodo è spesso preferito per la sua praticità, in quanto permette di lavorare con un file già configurato sulle scelte progettuali dei tecnici PLC.

Per il nostro caso studio utilizzeremo la seconda modalità. L’intera procedura si svolge su ambiente FT Optix™, seguendo questi passaggi. In primo luogo, la configurazione avviene nella Project view, come mostrato in figura 4.1, nella sezione CommDrivers. Al suo interno viene creato un nuovo driver dalla lista di quelli supportati da FT Optix™. Nel nostro caso andremo a crearne uno di tipo S7 TIA PROFINET driver, compatibile con il PLC Siemens utilizzato. Fatto questo, viene aggiunta una nuova Station con annesso il percorso del file .ap17, nella sezione delle proprietà. Una volta completati i passaggi sopra descritti, da FT Optix™, dovremo

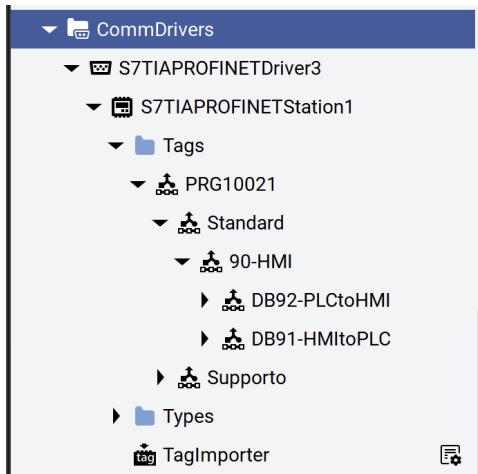


Figura 4.1: Illustrazione Tag importati

selezionare la cartella contenente i tag di nostro interesse e trovarci davanti ad una struttura contenente la cartella importata, con le relative variabili tag organizzate per tipologia (fare riferimento alla cartella generata in figura 4.1). Questo facilita il mapping e la configurazione successiva.

## 4.2 Configurazione variabili

Per quanto riguarda la gestione dei tag importati in precedenza, ci sono due strade da poter percorrere: lavorare direttamente sui tag importati con il rischio di avere accessi indesiderati e difficoltà nella manutenzione del sistema; oppure definire un livello di astrazione che separi i tag PLC dalle funzionalità dell’HMI. Per ovviare al problema della prima strada, adottiamo l’approccio che prevede sin da subito la creazione di variabili di modello in FT Optix™, per poi renderle intermediarie tra i tag del PLC e le funzionalità di FT Optix™. Questo approccio permette di rendere molto più sicuro l’utilizzo dell’HMI non avendo accessi diretti ai tag, e favorisce un maggior controllo dello SCADA. In secondo luogo si utilizza uno script per creare associazioni bidirezionali (*dynamic links*)<sup>[16]</sup> tra le variabili appena create e i tag importati dal PLC. Partiamo con il creare le variabili di modello, per cui è sufficiente creare all’interno della sezione Project view sotto la voce Model, una cartella Drivers in cui si definiscono tutte le variabili. Ovviamente le variabili devono essere configurate con nomi significativi per facilitare l’identificazione e l’uso. Una volta definite le variabili di modello, si passa alla parte di associazione, grazie ad un nuovo script di tipo Design-time NetLogic in cui viene creata una funzione che comprenda la seguente struttura:

- **Lista variabili di modello:** le variabili di modello sono ottenute tramite il metodo `GetVariable`, specificando il path all’interno della struttura di progetto. Per ottenere il path eseguiamo gli stessi passaggi illustrati nei capitoli precedenti.

- **Lista dei tag PLC:** analogamente, i tag del PLC sono identificati con il loro path sotto la sezione `CommDrivers`.
- **Creazione dynamic link:** l'associazione tra le variabili di modello e i tag del PLC avviene grazie al metodo `SetDynamicLink` fornito da FT Optix™, configurato con la modalità `ReadWrite` nel caso di lettura e scrittura su variabile, `Read` per la lettura e `Write` per la sola scrittura.

```

1 ...
2
3 public class AssociazioneVarModelloAPLC : BaseNetLogic
4 {
5     [ExportMethod]
6     public void Method1()
7     {
8         /*-----
9          * variabili di modello
10         */
11
12         //DB91
13         var modello000 = Project.Current.GetVariable("Model/Drivers/DB91/DB91_CambioProduzione");
14         ... // la lista prosegue con tutte le variabili di modello
15
16         /*-----
17          * variabili di PLC
18         */
19
20         // DB91
21         var varPLC000 = Project.Current.GetVariable("CommDrivers/S7TIAPROFINETDriver3/S7TIAPROFINETStation1/Tags/PRG10021/Standard/90-HMI/D"]
22         ... // la lista prosegue con tutte le variabili di PLC
23
24         /*-----
25          * dynamicLink per Optix
26         */
27         modello000.SetDynamicLink(varPLC000, DynamicLinkMode.ReadWrite);
28         ... // la lista prosegue con tutti i dynamicLink tra la prima e seconda lista
29
30     }
31 }
```

Questa modalità ha molteplici vantaggi. Prima di tutto, la sicurezza è migliorata: l'astrazione evita che l'HMI interagisca direttamente con i tag del PLC e permette di impostare la modalità in lettura e/o scrittura, che riduce significativamente il rischio di accesso non autorizzato. Inoltre, la manutenzione viene semplificata: la separazione tra modello e tag facilita eventuali modifiche future al sistema ed un eventuale reimpegno dell'HMI. Infine, la scalabilità: l'utilizzo di uno script per la configurazione consente di gestire facilmente un numero elevato di variabili e tag direttamente in un'unica funzione. Nonostante i numerosi vantaggi offerti da questo approccio, è importante evidenziare alcune criticità operative legate a sistemi complessi con migliaia di variabili. Più nello specifico, nei sistemi con elevato numero di variabili, il processo di messa a punto diventa un'operazione altamente dispendiosa in termini di tempo. Ciò accade perché ogni variabile di modello viene associata al corrispondente tag PLC tramite script. Una soluzione plausibile per sistemi avanzati potrebbe essere di automatizzare lo script di configurazione tramite il mapping delle variabili.

#### 4.2.1 Gestione FTP Robot, FTP Pressa e passi 50, 55 e 60

Nei case 50, 55 e 60 citati in precedenza, vengono gestiti tutti i caricamenti dei programmi, sia della pressa piegatrice, sia del robot dell’impianto. Per richiesta, è stato scelto sin dall’inizio di gestire entrambi i programmi lato interfaccia, con la seguente lettura dei dati tramite macchina a stati. Partendo dalla progettazione HMI, entrambi i pannelli sono stati inseriti nella sezione impostazioni, con accesso riservato solo agli utenti con autorizzazione appropriata, come in figura 4.2.

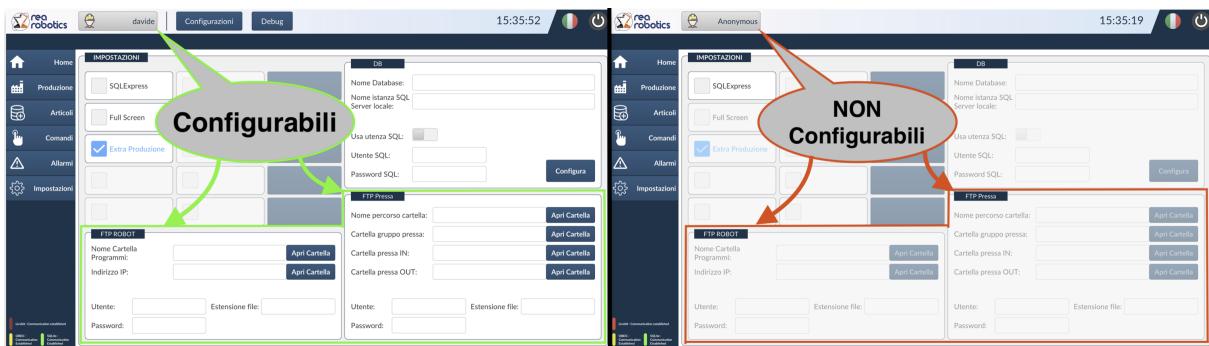


Figura 4.2: Illustrazioni Gestione FTP Robot ed FTP Pressa

Entrando nello specifico, in questa sezione sono stati inseriti 2 pannelli che rispettassero i seguenti requisiti:

- FTP Robot:** in cui è permesso selezionare il nome della cartella contenente i programmi robot, con estensione ed altri parametri specifici dell’impianto.
- FTP Pressa:** richiede informazioni più dettagliate, tra cui le cartelle di input ed output del programma, oltre al nome specifico per la pressa e le credenziali di rete per un’eventuale gestione tramite FTP.

Queste due sezioni sono fondamentali per poter riuscire a far comprendere all’impianto che tipo di istruzioni debba eseguire la macchina (robot e/o pressa) durante il caricamento di un ordine di produzione. I file contenenti le istruzioni del robot e quelli per la pressa vengono prodotti rispettivamente da chi si occupa della gestione robot e della gestione pressa. Per quanto riguarda, invece, la lettura tramite macchina stati, i passi da analizzare sono il 50, 55 e 60:

- **case 50:** in questo passo, il sistema si trova in attesa della richiesta da parte del PLC del caricamento del programma. Una volta ricevuta, in caso affermativo il sistema passa al passo 55; in caso di KO invece il passo viene impostato a 180.
- **case 55:** il sistema esegue una serie di operazioni in cui vengono lanciate le funzioni di gestione programma, con annesso controllo dell’esistenza del file nel percorso indicato,

copia del file dalla sorgente alla destinazione e verifica della consistenza del file copiato. Inoltre, ogni funzione ha una gestione degli errori con popup adeguati, restituendo valori in base all'esito (questa parte è presente in `script_maincycle.cs`). Se tutto avviene in modo corretto, si passa al passo 60.

- case 60: seconda ed ultima fase di attesa da parte dell'HMI nei confronti del PLC. Più nello specifico, l'HMI verifica che la variabile di richiesta caricamento programma desiderato sia resettata dal PLC, ed in base al risultato di questa verifica, il sistema decide quali saranno i prossimi passi da eseguire.

Quello che abbiamo analizzato in questo capitolo è la minima parte di tutto il lavoro che compone la gestione dei robot e di altri vari componenti presenti in un impianto; nel nostro caso la gestione era di due sole macchine con istruzioni ben definite. In contesti industriali più complessi, dove sono coinvolti numerosi apparati e macchine, la gestione può risultare molto più articolata e richiede soluzioni più sofisticate per garantire scalabilità, sincronizzazione e gestione di malfunzionamenti.



Figura 4.3: Prova messa in funzione dello SCADA su pannello prova ASEM



# Capitolo 5

## Conclusioni

Il lavoro svolto per questa tesi ha avuto come obiettivo principale lo sviluppo di una soluzione SCADA integrata per la gestione di sistemi industriali, con focus sulle tecnologie utilizzate in REA Robotics. Nello specifico, la finalità è stata di implementare una piattaforma in grado di gestire in modo efficiente e sicuro il flusso di dati tra HMI e PLC, non solo permettendo di rispondere con successo alla task iniziale fornita dall'azienda, ma anche garantendo funzionalità operativa, sicurezza ed affidabilità del sistema. Il progetto ha visto come cardini l'utilizzo di strumenti e software come FT Optix™, C#, SSMS e SQLite, per permettere la realizzazione di un'interfaccia che semplifica il monitoraggio e controllo dei macchinari. Nonostante sia stata raggiunta la task richiesta, sono emerse alcune criticità che meritano attenzione. Una delle principali sfide è stata la gestione separata dei due database, che ha comportato maggiore complessità nella progettazione da un lato e nella manutenzione futura dall'altro. Sarebbe auspicabile in futuro unificare la parte di database in un'unica implementazione generica che permetta una gestione più scorrevole, al fine di semplificare la mole di dati e codice presente e ridurre il rischio di incoerenze. Un'altra criticità riscontrata riguarda l'utilizzo dei recipe schema. La rigida dipendenza dalle regole imposte dal sistema di gestione di questi ultimi ha limitato la flessibilità del progetto, rendendo difficile adattare il sistema a future modifiche o implementazioni per altri impianti. In tal senso, una possibile evoluzione futura potrebbe prevedere il refactoring di alcuni script di codice per permettere la gestione dei dati più modulare e indipendente dalle regole imposte dai recipe schema, garantendo maggiore personalizzazione per i clienti. Guardando al futuro, sono diverse le opportunità di miglioramento e sviluppo. Per esempio, l'uso di C# per ottimizzare la lettura delle variabili dei PLC consentirebbe un aggiornamento in tempo reale dei dati, una maggiore reattività del sistema alle modifiche e soprattutto meno tempo da dedicare lato programmazione, grazie a una gestione intelligente dei dynamic link attraverso una lettura diretta delle variabili. Inoltre, l'integrazione di Intelligenza Artificiale per l'analisi dei dati e per la progettazione dei pannelli potrebbe aprire nuovi scenari, consentendo una gestione ancora più

efficiente lato software.

In conclusione, il progetto ha raggiunto gli obiettivi prestabiliti, complice un nuovo approccio nel contesto industriale di REA Robotics. Tuttavia, le criticità emerse durante lo sviluppo offrono sia spunti di riflessione, sia indicazioni per futuri miglioramenti. L'implementazione di soluzioni più moderne e flessibili, una gestione più modulare dei dati e l'introduzione di nuove tecnologie rappresentano sfide importanti per il futuro, con l'obiettivo di rendere questa tipologia di sistemi SCADA ancora più performante, sicura e adattabile alle esigenze in continua evoluzione dell'industria 4.0.

# Bibliografia e Sitografia

- [1] R. Robotics. «Homepage ufficiale di Rea Robotics.» (2025), indirizzo: <https://www.rearobotics.it/>.
- [2] A. sconosciuto. «Documento PDF da SpazioWeb.» (2025), indirizzo: <https://files.spazioweb.it/b3/2a/b32ab4b0-bad4-4cfb-ad2f-b36dd422eb86.pdf>.
- [3] R. Robotics. «Chi Siamo.» (2024), indirizzo: <https://www.rearobotics.it/chi-siamo/>.
- [4] S. Sistemi. «Cosa è SCADA?» (2025), indirizzo: <https://www.sielcosistemi.com/it/cosa-%C3%A8-scada.html>.
- [5] ExonSys. «SCADA Software Solutions.» (2022), indirizzo: <https://exonsys.com/en/software/scada/>.
- [6] A. S.r.l. «FactoryTalk® Optix™ Application Software.» (2025), indirizzo: <https://www.asemautomation.com/it/prodotti/251/factorytalk-optix.html>.
- [7] R. Automation. «FactoryTalk Optix: Configure the Main Window.» (2025), indirizzo: <https://www.rockwellautomation.com/en-us/docs/factorytalk-optix/1-00/contents-ditamap/getting-started/quick-start/configure-and-brand-the-main-window/configure-the-main-window.html>.
- [8] R. Automation. «FactoryTalk Optix: Data Grid Example.» (2025), indirizzo: <https://www.rockwellautomation.com/en-us/docs/factorytalk-optix/1-00/contents-ditamap/developing-solutions/object-examples/data-grid-example.html>.
- [9] R. Automation. «FactoryTalk Optix: Datastore Database.» (2025), indirizzo: <https://www.rockwellautomation.com/en-us/docs/factorytalk-optix/1-00/contents-ditamap/using-the-software/datastore-database.html>.
- [10] R. Automation. «FactoryTalk Optix: SQL Query.» (2025), indirizzo: <https://www.rockwellautomation.com/en-us/docs/factorytalk-optix/1-00/contents-ditamap/using-the-software/datastore-database/sql-query.html>.

- [11] R. Automation. «FactoryTalk Optix: Recipes.» (2025), indirizzo: <https://www.rockwellautomation.com/en-us/docs/factorytalk-optix/1-00/contents-ditamap/using-the-software/recipes.html>.
- [12] R. Automation. «FactoryTalk Optix: Developing Projects with C# - NetLogic.» (2025), indirizzo: <https://www.rockwellautomation.com/en-us/docs/factorytalk-optix/1-00/contents-ditamap/developing-solutions/developing-projects-with-csharp/netlogic.html>.
- [13] R. Automation. «FactoryTalk Optix: Users and Groups.» (2025), indirizzo: <https://www.rockwellautomation.com/en-us/docs/factorytalk-optix/1-00/contents-ditamap/using-the-software/users-and-groups.html>.
- [14] R. Automation. «FactoryTalk Optix: Alarms.» (2025), indirizzo: <https://www.rockwellautomation.com/en-us/docs/factorytalk-optix/1-00/contents-ditamap/using-the-software/alarms.html>.
- [15] R. Automation. «FactoryTalk Optix: ODBC Store Object Types.» (2025), indirizzo: <https://www.rockwellautomation.com/en-us/docs/factorytalk-optix/1-00/contents-ditamap/developing-solutions/object-and-variable-references/ftoptix-odbcstore/objecttypes/odbcstore.html>.
- [16] R. Automation. «FactoryTalk Optix: IUA Variable - SetDynamicLink3.» (2025), indirizzo: <https://www.rockwellautomation.com/en-us/docs/factorytalk-optix/1-00/contents-ditamap/developing-solutions/developing-projects-with-csharp/csharp-apis-reference/manage-dynamic-links/iuavariable-setdynamiclink3.html>.

# **Ringraziamenti**

Desidero esprimere la mia sincera gratitudine a tutte le persone che hanno contribuito alla realizzazione di questa tesi.

In primo luogo, vorrei ringraziare il mio relatore, Prof. Emanuele Menegatti, per la sua preziosa guida durante tutto il percorso.

Un ringraziamento speciale va a Laura e Denis, che mi hanno accompagnato durante l'esperienza di tirocinio in REA e nella scrittura della Tesi.

Un doveroso grazie ai miei familiari, ai miei amici e a Erika, che mi hanno sempre sostenuto e incoraggiato lungo tutto il percorso.

Infine, desidero ringraziare i miei colleghi più cari - Ludo, Simo, M1 e M2 - che hanno condiviso con me questa esperienza, offrendomi supporto e collaborazione.

Grazie di cuore a tutti.