UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN
INGEGNERIA DELL'INFORMAZIONE

PROGETTAZIONE E SVILUPPO DI UN SOFTWARE AUSILIARE PER L'IMPLEMENTAZIONE DELLA REGOLAZIONE NELLE CENTRALI DI TRATTAMENTO DELL'ARIA

Relatore: Prof. Stefano Vitturi

Laureando: Ovidiu George Macovei

ANNO ACCADEMICO 2021 - 2022
Data di laurea 19/07/2022

AM. *Ai miei genitori, ad immensum.*

Ai pochi ma essenziali.

A me. *Per aspera, ad astra*.

Indice

Capitolo 1

Introduzione

La tesi è basata su un progetto dell'azienda Fast S.p.A. per la realizzazione di un software ausiliare per la regolazione delle centrali di trattamento aria.

1.1 Azienda

Fast S.p.A è un'azienda metalmeccanica che si occupa della produzione di macchine per il trattamento dell'aria fin dal 1990. Fu fondata sotto la volontà di Giordano Riello, noto impreditore della zona e fa parte della holding *"Giordano Riello International Group S.P.A" (GRIG)* con sede ufficiale a Bevilacqua in provincia di Verona. Attualmente l'azienda è gestita dal presidente *Raffaella Riello* e dal marito *Paolo Gasparini*, amministratore delegato.

Figura 1.1: Logo Fast S.p.A

Le applicazioni dei prodotti riguardano diverse realtà come piscine, Data Center, teatri, ospedali e offre diverse soluzioni per il settore alimentare, industriale, farmaceutico e ambienti potenzialmente esplosivi. Gli articoli si dividono in varie categorie, suddivise in base ai loro utilizzi anche se la maggior parte, al giorno d'oggi, sono personalizzate in base alla volontà dei clienti. Le centrali trattamento aria, nominate con la sigla FM, rappresentano il prodotto standard che offre l'azienda e sono disponibili 109 taglie in base alla dimensione ed alla portata d'aria che varia da 1.000 m^3/h a 100.000 m^3/h. Per quanto riguarda le unità di trattamento aria primaria ad alta efficienza energetica l'azienda offre due possibilità di scelta con le $Hygromax$ e le $Etamax$ dove è previsto un recuperatore di energia e sono disponibili con 5 taglie per ciascuna. Per le

piscine e le unità di deumidificazione sono presenti le $Alfamini$ e $Alfamax$ anche queste con 5 taglie disponibili ad ognuna. Per ultime ma non meno importanti rimangono i Roof-Top che sono destinati ad applicazioni industriali e commerciali a medio ed elevato affollamento.

1.2 Principale software aziendale

Ogni prodotto che viene venduto ha la possibilità di essere visto, personalizzato e interpretato direttamente dal cliente. Tutto questo è possibile con il software $airCalc$ sviluppato dall'azienda $ISC\ GmbH$. Questo programma distribuito nella versione Windows, è dedicato interamente alla selezione dei vari componenti per la realizzazione di un'unità trattamento aria. L'applicazione permette all'utente che la utilizza di comporre un'intera centrale suddividendola in sezioni specifiche a seconda dei tipi di elementi presenti. Quest'ultimi si dividono in $component$, che solitamente danno il nome delle sezioni, in $small\ component$ e $accessories$.

L'interfaccia utente come riportato in Figura 1.2, presenta due parti. Nella prima è possibile selezionare i componenti e le varie sezioni visualizzando attraverso un CAD interno, il disegno della centrale in due dimensioni con viste laterali e dall'alto. Nella seconda l'utilizzatore si può personalizzare ogni sezione ed ogni componente andando a modificare le diverse variabili che compongono la selezione.

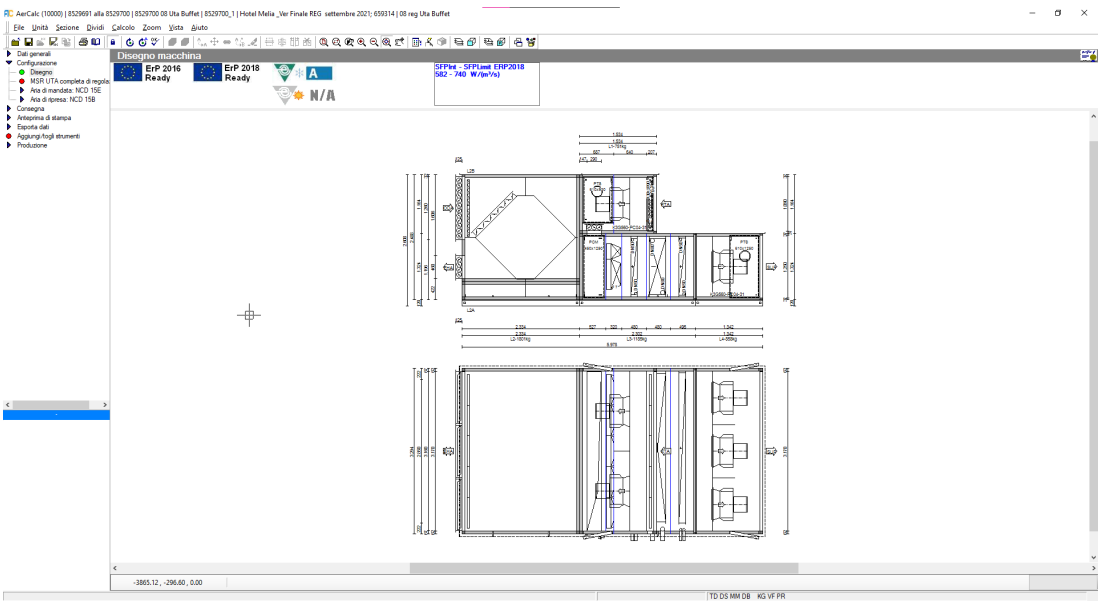


Figura 1.2: Interfaccia utente di Aircalc

Il risultato finale viene calcolato dal software attraverso un pulsante e produce tutti i dati necessari per la realizzazione del prodotto, una stampa della scheda tecnica, una del capitolato ed una con i prezzi singoli per ogni componente e materiali usati.

Il database ufficiale del programma è fornito tramite un file Microsoft Access protetto da password ma è possibile usare un database locale per tutti i dati messi presenti nel software. Questo è fondamentale in quanto c'è la possibilità di inserire, togliere o aggiornare l'applicazione da parte degli admin. E' possibile inoltre distribuire questo file a tutti gli utenti con licenza della propria azienda attraverso un costruttore di aggiornamento offerto dai sviluppatori per permettere una veloce sincronizzazione con le ultime modifiche effettuate.

1.3 Obiettivo

Una parte importante nella selezione delle unità trattamento aria è quella della regolazione che costituisce circa il 25% del costo totale. Il software $airCalc$ include la possibilità di gestirla attraverso un'interfaccia utente presente al suo interno. Il primo di questi è lavorare con un file $Excel$ esterno ed è quello che attualmente sta sfruttando l'azienda $FastS.p.A.$. Ciò implica la creazione di un file molto complesso e pesante in quanto si deve prevedere tutte le possibili combinazioni di tutti i vari componenti e l'interazione con diverse tabelle del database. Con l'andare degli anni, il documento è diventato molto pesante e la manutenzione e l'aggiunta di nuove funzionalità è diventata abbastanza complicata e quasi impossibile da parte di altre persone ad esclusione del proprio creatore. Inoltre i tempi di apertura della sezione della regolazione su $airCalc$ ed i tempi di calcolo sono diventati molto lunghi, con l'ordine di minuti. L'azienda si è quindi trovata nella posizione di trovare una soluzione alternativa, con diverse proposte. La prima era quella di continuare all'interno del software di $ISC\ GmbH$ con una gestione semplificata, per cui meno domande con minor tempo di calcolo e apertura. Questo però implica una minor personalizzazione del prodotto per cui è stata scartata. La seconda proposta era la realizzazione di un programma esterno ad $airCalc$ con tempi di calcolo ridotti e una massima personalizzazione. Questa soluzione era poco gradita però da parte degli utilizzatori commerciali in quanto dovevano utilizzare due applicazioni diverse. Dopo varie riunioni si è arrivato alla definitiva soluzione grazie alla collaborazione con i proprietari del software di $airCalc$, che hanno aggiunto la possibilità di apertura di un $form$ con lo stesso pulsante con cui prima si accedeva all'interfaccia della regolazione. Di fatto un programma esterno integrato nel

software, con tempi di lettura e di calcolo molto veloce e la possibilità di utilizzare una sola applicazione.

L'obbiettivo del lavoro di tesi è stato quindi lo sviluppo di questo nuovo software integrato che deve interagire con quello principale e configurare la parte della regolazione per le centrali di trattamento d'aria.

Capitolo 2

Regolazione

2.1 Regolazione Generale

Un sistema è l'insieme di parti meccaniche, elettriche, idrauliche o elettroniche assemblate e connesse tra di loro in modo da ottenere un'unica entità che funziona in un determinato modo. La regolazione ha il compito di svolgere azioni tese ad ottenere dei valori prefissati delle grandezze caratteristiche di un processo, in modo automatico.

Per poter attuare quindi l'automazione il sistema ha bisogno di diversi componenti che possono fungere da controllori o da attuatori. Lo strumento principale di cui si ha bisogno per poter effettuare una regolazione è sicuramente il PLC cioè il $programmable\ logic\ controller$. Esso ha il compito di ricevere dei segnali da parte del sistema e tramite le funzioni impostate nella sua memoria, attuare una serie comandi che portino ai valori impostati. I segnali ricevuti derivano dai trasduttori che sono i dispositivi elettrici o elettronici disposti nel campo (ossia la zona controllata) e hanno la funzione di trasformare le grandezze fisiche in segnali elettrici. La natura dei segnali può essere di diverso tipo come la posizione, velocità. forza, temperatura, umidità, pressione, portata ed accelerazione. Il PLC poi, può agire sul campo inviando segnali elettrici agli attuatori che sono invece dispositivi che producono un'azione fisica. Anche in questo caso possono essere di diverse tipologie come relè, elettrovalvole, motori, generatori di calore o lampade.

2.2 Regolazione nelle centrali di trattamento aria

La regolazione nelle centrali di trattamento aria consiste nell'applicare una serie di attuazioni per permette di controllare il flusso, la portata e la temperatura dell'aria. La logica secondo cui viene fatta l'automazione è dettata soprattutto dai controllori

programmabili che hanno più o meno diverse funzionalità. Nel caso dell'azienda $Fast\ S.p.A.$ viene utilizzato principalmente il $pCO5$ marca $Carel$ e con uso meno frequente $RDT940$ marca $Sauter$.

L'esempio riportato in Figura 2.1 è lo schema secondo il manuale $Carel$ di una tipica unità trattamento aria e aiuta a capire meglio che elementi e dispositivi servono e come funziona la regolazione.

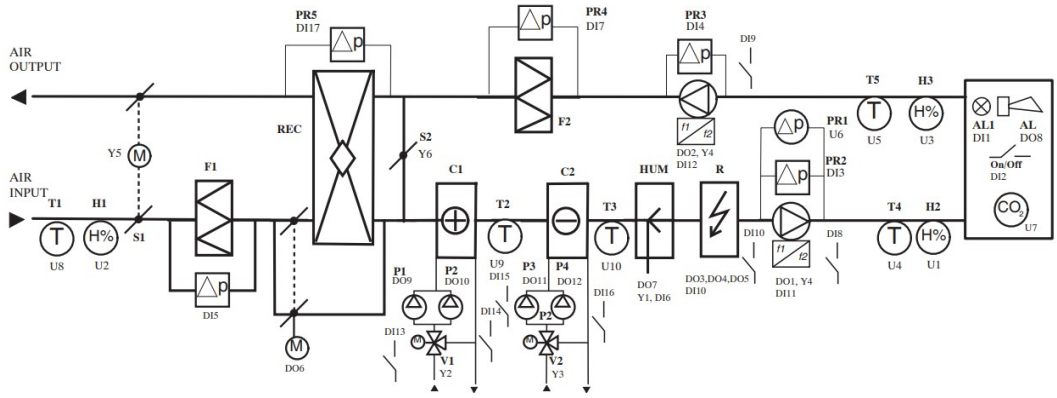


Figura 2.1: Schema regolazione con logica Carel

Per quanto riguarda il flusso di entrata aria dall'esterno, detto anche $mandata$, si incontrano le sonde di temperatura (T1) e dell'umidità (H1) che si trovano all'esterno della centrale. Principalmente non sono obbligatorie, se non nel caso del cosiddetto "$freecooling$" cioè quando si sfrutta la differenza di temperatura tra l'aria esterna e quella dell'ambiente risparmiando cosi l'uso delle batteria per riscaldare o raffreddare. Di conseguenza devono essere presenti ulteriori due sonde di temperatura e umidità all'inizio del condotto dell'aria di ripresa dall'ambiente, in questo esempio si tratta di $T5$ e $H3$.

Ogni componente fisico presente nella centrale UTA[1] può costituire motivo per avere uno o più elementi per la regolazione. Come si vede nell'esempio e seguendo il flusso dell'aria che entra, detta anche aria di "mandata", si può avere per:

- $Serrande$
  - Un servocomando modulante oppure on/off per controllare il flusso d'aria
- $Filtri$
  - Un pressostato collegato ad un allarme per capire quando il filtro è sporco e deve essere sostituito

[1] UTA: abbreviazione per Unità di Trattamento dell'Aria

- $Recuperatore$
  - Un servocomando modulante oppure on/off per la possibile serranda di Bypass
  - Un servocomando modulante oppure on/off per la possibile serranda di ricircolo dell'aria
- $Batterie$
  - Un'elettrovalvola per controllare il flusso dell'acqua della batteria
- $Ventilatori$
  - Un trasduttore per controllare la portata d'aria
  - Un potenziometro per la potenza del motore

L'umidificatore e la batteria elettrica sono gli unici elementi che non hanno bisogno di ulteriori dispositivi per essere controllati e quindi regolati. Rimane il fatto che tutti gli elementi devono essere collegati al quadro elettrico posto in una delle sezioni della centrale ed a sua volta può contenere dispositivi come la sonda CO2 ed allarmi generali o di fumo. Il terminale $pCO5$ può essere a bordo macchina o in remoto ma rimane l'obbligo di connettere tutti i dispositivi presenti nella centrale. Ogni elemento dispone di ingressi oppure uscite che possono essere analogiche o digitali. Nell'esempio analizzato gli ingressi analogici sono indicati con la lettera U mentre quelli digitali sono iniziano con DI ed in entrambi i casi sono seguiti da un numero progressivo per ogni elemento. Per quanto riguarda le uscite analogiche sono indicate con Y e quelle digitali con DO ed anche in questo caso sono accompagnate dal numero progressivo.

Capitolo 3

Software

Il nuovo software integrato ad $Aircalc$ sviluppato in $Visual\ Studio$, ha il compito di produrre un output sotto forma di stringa contenente campi specifici per identificare i componenti occorrenti alla regolazione, secondo quanto richiesto dal programma principale.

3.1 Interfaccia grafica

Il nuovo $form$ riportato in Figura 3.1 è aperto senza la barra del titolo ed in modale, cioè senza la possibilità di minimizzare la finestra ed utilizzare il software principale fino alla chiusura. Avrà una dimensione preimpostata di 1400x870 pixel con la possibilità da parte dell'utente di ridimensionarlo con la modalità $drag\ and\ drop$ grazie ad una funzione scritta a codice e applicata ad un bottone posto nell'estremità bassa destra. E' possibile inoltre spostare la finestra grazie ad un altro algoritmo applicato ad un bottone che prende il posto della barra del titolo e con il testo "$move\ window$" utilizzando sempre la tecnica citata per il bottone precedente.

Il layout della finestra è suddiviso in due parti: una parte contenente tutti i controlli ed una costituita da un oggetto $TreeView$ del pacchetto di contenitori base Windows. Quest'ultimo contiene tutte le domande attive rappresentate da nodi posti sullo stesso livello ed ognuno ha come figlio un altro nodo con la risposta selezionata o di default, per dare la possibilità all'utente di avere un quadro generale di tutte le scelte fatte. Nella prima parte invece è presente un $label$ con il compito di mostrare la domanda corrente e un $group\ box$ contenente 7 $Radio\ Button$ per la selezione delle risposte. Inoltre ci sono due pulsanti per andare avanti e indietro con le domande, e altri due per salvare la configurazione e produrre l'output e l'altro per uscire. Nella zona bassa della finestra è dedicata alla gestione degli errori con un label avente testo fisso e altri

4 dinamici abbinati a dei controlli chiamati "$glow$" che compaiono se sono presenti errori o incompatibilità nella scelte delle risposte.

Figura 3.1: Form software

La cura della parte grafica riguarda l'utilizzo dei colori usati. Infatti la scelta dello sfondo scuro, sia per il $form$ che per il $TreeView$, è dovuta al progresso grafico degli ultimi anni dove la maggior parte dei sistemi operativi e dei nuovi software utilizzano i temi scuri. Per quanto riguarda il testo delle $label$ il colore scelto è il bianco per avere un contrasto elevato e riuscire a leggere chiaramente e velocemente. I bottoni hanno invece uno sfondo trasparente con i bordi dello stesso colore del testo. Per i tasti $Forward$ e $Backward$ si è scelto il bianco, per il $Save$ verde lime e mentre il rosso domina sul $Exit$. I $glow$ sono piccole animazioni di cerchi che appaiono e scompaiono gradualmente, di colore rosso, per attirare l'attenzione. Nell'oggetto albero, il testo dei nodi è colorato a seconda dello stato delle domande e risposte. Se la domanda è senza risposta viene utilizzato il color "salmone", se ha la risposta il bianco mentre se contiene un errore il rosso. I nodi contenenti il testo delle risposte sono visibili solo se la risposta è valida quindi colorati di verde.

Per la realizzazione della parte grafica sono stati usati, oltre che il pacchetto standard di Windows, tre pacchetti esterni gratuiti trovati in rete e scaricati tramite il gestore di pacchetti NuGet implementato in Visual Studio. Il primo si chiama "DrakeUI" , il secondo "MagicUI" mentre l'ultimo è "SaaUI". Infatti grazie a questi è stato possibile avere una maggiore scelta di controlli e personalizzazione.

3.2 Interazione con l'utente e struttura

Per limitare le possibilità di errore, il software è pensato per aggiornarsi automaticamente e limitare le scelte possibili. Infatti l'utente deve rispondere solo alle domande che vengono mostrate a video e non a tutti quesiti presenti nel database. Inoltre anche le risposte che si presentano, possono essere parziale.

All'apertura del nuovo $form$ o schermata principale verranno caricate in memoria tutte le domande con relative risposte mentre in background verranno caricati tutti i codici possibili di output.

Le domande mostrate a video sono attivate in base alla configurazione della centrale di trattamento dell'aria. Infatti dopo il caricamento, viene eseguito un algoritmo che scansiona i vari componenti presenti ed in base a questi attiva le relative domande. I principali sono le batterie con la loro quantità e funzione, i filtri, le serrande ed il loro collocamento, i ventilatori e l'eventuale presenza degli inverter e motore, il recuperatore ed infine l'umidificatore. In questa fase sono settate anche le risposte per ogni quesito non mostrando a video le opzioni che risultano incompatibili con gli elementi presenti nella centrale. Un esempio può essere la batteria elettrica con input utente su Aircalc di 2 step, può avere come risposte attive un numero minore o uguale a 2 di entrate e uscite digitale a seconda della scelta dell'utente ma non può averne 3 nonostante quest'ultima risposta sia presente nella lista di tale quesito.

Inizialmente tutte le domande hanno una risposta di default quindi l'utente può salvare direttamente la preconfigurazione. Inoltre ha la possibilità di cambiare la risposta alle domande attive. Quest'ultime,che sono elencate nel TreeView collocato nella parte destra della finestra, vengono mostrate a video con le risposte disponibili scorrendo una ad una tramite i bottoni $Forward$ e $Backward$ oppure cliccando due volte sul nodo con il testo della domanda. Ad ogni cambiamento di risposta, il software ripercorre tutte le domande facendo una sorta di $Refresh$ cioè aggiornamento. Ogni risposta ha una variabile booleana associata che può scatenare una serie di attivazioni e disattivazioni di altre domande. Infatti un esempio può essere la regolazione scelta dall'utente se fatta con un processore Carel oppure Sauter. Per la prima opzione si attiva un ulteriore domanda per poter scegliere il tipo di terminale remoto mentre per la regolazione Sauter, la scelta non è disponibile e la domanda non si attiverà. Inoltre, per ogni quesito, se la risposta è stata selezionata dall'utente rimane invariata, mentre se non è stata vagliata manualmente si attiva la risposta di default che può essere univoca o che può cambiare in base ad altre variabili.

Ad ogni domanda e risposta quindi è associata una variabile booleana, contenuta

in una struttura $Dictionary$[2] come valore mentre la chiave è una stringa con il nome della variabile. Questo permette un controllo molto veloce e una facilitazione degli algoritmi.

Quando l'utente ha finito di effettuare le proprie scelte ha due opzioni. La prima è di salvare la configurazione attraverso il bottone $Salva$ e quindi memorizzare l'output nel programma principale. Questo è possibile solo se non sono presenti errori altrimenti per salvare è necessario cambiare le scelte delle domande con risposta non valida che vengono segnalate a video con l'id del quesito e una possibile spiegazioni in modo che si abbia un veloce riscontro. Se invece si esce senza salvare, viene mostrato un $pop-up$ [3] che informa l'utilizzatore l'uscita senza salvataggio. In questo caso si può salvare direttamente, nel caso non ci fossero errori, oppure uscire senza memorizzare le scelte cosi mantenendo la precedente configurazione.

3.3 Classi e algoritmi

La progettazione del software è basata su una programmazione orientata agli oggetti. Essa infatti permette di creare una classe o appunto meglio, un oggetto, che ha delle caratteristiche o degli attributi ben precisi ed un insieme di funzionalità. Questo tipo di dato è usato per modellare un insieme di oggetti dello stesso tipo.

Un'importante premessa è che esistono due variabili globali quindi accessibili da tutte le classi. Si tratta della variabile ***Language*** e ***LevelUser*** che vengono settate nella classe principale chiamata $FormMSR$.

Il programma quindi è un insieme di oggetti che interagiscono tra loro. In questo caso il software ne presenta otto come si può notare dalla Tabella 3.1. Di seguito vengono descritte ed approfondite nel dettaglio.

Classe	Descrizione
Answer	Singola risposta
SingleQuestion	Singola domanda con le relative risposte
Questions	Contenitore di tutte le domande
SinglePart	Singolo codice di output
Parts	Contenitore di tutti i codici di output divisi per gruppi
Section	Singola sezione di Aircalc
MSR	Configurazione e calcolo del risultato finale
FormMSR	Interfaccia grafica e interazione con l'utente

Tabella 3.1: Tabella riassuntiva delle classi

[2]Dictionary: struttura a coppia chiave-valore
[3]pop-up: Finestra con un messaggio informativo

3.3.1 Classe: Answer

La classe $Answer$ rappresenta un oggetto $risposta$ con lo scopo di identificare una singola possibile scelta per ogni domanda. Essa è caratterizzata da alcune variabili di tipo diverso che permettono di identificarla univocamente. Nel dettaglio le proprietà sono:

- **ID** di tipo $Integer$. Lo scopo di questa variabile è quello di identificare velocemente che posizione occupa nell'insieme di risposte associate ad ogni singola domanda. Il valore che può assumere è rappresentato da un numero intero dentro l'intervallo che va da 0 a 7 inclusi.

- **Text** di tipo $ArrayList$. Proprietà contente il testo delle risposte e per ogni indice una lingua diversa.

- **Checked** di tipo $Boolean$. Variabile molto importante per il controllo con lo scopo di identificare se in quel momento la risposta è selezionata o meno. Infatti può assumere il valore $True$ oppure $False$ e può cambiare a seconda della scelta fatta dall'utente. Per ogni domanda è possibile solo una risposta con questa variabile uguale a $True$ ed ad ogni cambio scelta la proprietà delle altre risposte assumono il valore $False$ attraverso un algoritmo mostrato successivamente.

- **BoolVariable** di tipo $String$. Il testo di questa stringa rappresenta il nome della variabile booleana associata alla risposta. Inizialmente, quando viene creato l'oggetto, la stringa è vuota mentre successivamente se il suo testo cambia con il caricamento dei dati, può essere molto utile per poter interagire con la proprietà generale $Dictionary$. Infatti in quest'ultimo può essere presente una coppia chiave-valore con la chiave uguale a questa variabile.

- **VariableActivate** di tipo $ArrayList$. Lista che contiene un insieme, anche vuoto, di stringhe con il nome della proprietà "BoolVariable" di altre risposte oppure anche domande contenute nella struttura $Dictionary$. Infatti se una di queste variabili ha valore uguale a True allora la risposta/domanda è visibile e selezionabile dall'utente.

- **MessageError** di tipo $String$. Semplice stringa contenente il testo di un eventuale possibile errore.

- **ErrorName** di tipo $String$. Il nome della variabile booleana che genera l'errore.

- **DefaultCheck** di tipo $String$. La stringa contiene il nome di una variabile nella struttura $Dictionary$. Questo permette di identificare se la risposta deve avere la proprietà "*Checked*" descritta sopra con il valore uguale a *True*. Ciò è utile anche per la preconfigurazione al primo avvio del programma.

- **LevelUserAccess** di tipo $ArrayList$. La lista contiene un insieme di $Integer$ che esprimono il livello che l'utente in $Aircalc$. Infatti a seconda del grado, la risposta può non essere visibile o selezionata.

Menzione particolare per la proprietà "**ActivateFields**" di tipo $Dictionary$ che è comune a tutte le classi. Infatti a differenza delle altre proprietà che vengono inizializzate nel costruttore con valori predefiniti o passati come parametro per valore, questa viene passata come parametro per riferimento in modo tale da interagire sempre con lo stesso oggetto indifferentemente dalla posizione in cui ci si trova. Tutto ciò permette di gestire la parte di attivazione e disattivazione delle domande e risposte nel programma.

Funzioni

Viste le proprietà, la classe presenta anche alcune funzioni utili per il funzionamento del programma e l'interazione con gli altri oggetti.

La prima dev'essere per forza la funzione di creazione dell'oggetto. Infatti è presente sia il costruttore vuoto che quello con i parametri. Nel primo caso tutte le proprietà associate alla classe vengono inizializzate con i valori predefiniti del tipo di dato. Infatti a parte la proprietà ID che assume il valore -1 e $Checked$ uguale a "False", i dati di tipo $String$ sono stringhe vuote. Per quanto riguarda gli $ArrayList$, essi vengono creati senza nessun elemento contenuto. L'unica eccezione è la presenza del parametro $Dictionary$ come descritto precedentemente. Nel secondo caso invece, oltre a quest'ultimo parametro passato per riferimento, tutti gli altri invece sono passati per valore.

Per vedere se una risposta è visibile o meno, la classe si avvala della funzione ***IsAnswerVisible***. Premesso che il numero di elementi presenti nella proprietà $VariableActivate$ non può essere maggiore di 2, il risultato si ottiene con $Algorithm$ 1 scritto di seguito con pseudocodice . La funzionalità di tale algoritmo è semplice e non necessita particolari spiegazioni.

Algorithm 1 IsAnswerVisible

```
Output: Boolean
 1: visible ← False
 2: numberElement ← VariableActivate.Count
 3: if LevelUserAccess.Contain(LevelUser) then
 4:     if numberElement = 1 AND ActivateFields(VariableActivate(0)) then
 5:         visible ← True
 6:     else if numberElement = 2 then
 7:         if ActivateFields(VariableActivate(0)) OR
    ActivateFields(VariableActivate(1)) then
 8:             visible ← True
 9:         end if
10:     end if
11: end if
12: return visible
```

Un'altra funzione interessante da vedere è *Algorithm 2*: **SetBoolVar**. Ha il compito di cambiare la il valore delle variabili booleane nella struttura $Dictionary$. Infatti agisce sulla variabile che ha come chiave il testo della proprietà *BoolVariable* presente in questa classe. Inoltre non restituisce nessun output mentre la stringa "bDefault" è la chiave con valore sempre uguale a True nel Dizionario.

Algorithm 2 SetBoolVar

```
    if BoolVar != "bDefault" then
2:      if ActivateFields.ContainsKey(BoolVariable) then
            if Checked then
4:              ActivateFields(BoolVariable) ← True
            else
6:              ActivateFields(BoolVariable) ← False
            end if
8:      end if
    end if
```

La classe presenta molte funzioni che impostano e ritornano il valore delle proprietà della classe , ed oltre alle due viste prima, l'ultima che merita di essere vista è *Algorithm 3*: **ShowAnswerError** che controlla se la risposta è visibile, se contiene un errore e ritorna la proprietà con il testo dell'errore, altrimenti una stringa vuota.

Algorithm 3 ShowAnswerError

Output: String
errorText ← String.empty
if ActivateFields(BoolVariable) AND ActivateFields(ErrorName) **then**
3: errorText ← MessageError
end if
return errorText

3.3.2 Classe: SingleQuestion

La classe *"SingleQuestion"* rappresenta l'oggetto domanda. Tra le sue proprietà troviamo alcune simili a quelle elencate sopra e qualcuna di nuova. In particolare:

- **ID** di tipo $Integer$. Numero progressivo che identifica univocamente la domanda.
- **Text** di tipo $String$. Testo del quesito.
- **VariableActivate** di tipo $ArraList$. Ha la stesso funzione descritta precedentemente però associata all'oggetto attuale.
- **ActivateFields** di tipo $Dictionary$. L'oggetto comune a tutte le classi e fondamentale per il funzionamento del software.
- **AnswerList** di tipo $ArrayList$.Vettore contenente l'insieme delle risposte associate alla domanda.
- **Checked** di tipo $Boolean$. Variabile che ritorna se la domanda ha una risposta attiva.
- **manualChecked** di tipo $Boolean$. Proprietà che indica se una delle sue risposte è stata selezionata dall'utente oppure no. Infatti se l'utente sceglie una delle risposte di questa domanda, la variabile assume il valore True e non può piu' cambiare. Serve per l'automazione del software in quanto, la scelta fatta rimane se la domanda è stata selezionata manualmente mentre può cambiare in base alla configurazione della centrale e alle impostazioni di default.
- **LevelUserAccess** di tipo $ArrayList$. Come nella classe precedente.

Funzioni

La classe in totale presenta 21 funzioni, alcune semplici ed altre più articolate, oltre al costruttore vuoto e con parametri. Tra le prime ci sono sicuramente quelle del *Get* e *Set* ritornano ed assegnano semplicemente il valore delle proprietà.

La maggior parte degli algoritmi di quest'oggetto interagisce con le risposte contenute nella variabile *AnswerList* , quindi di conseguenza non fa altro che usare le funzioni della classe precedente scorrendo dentro la lista tramite un ciclo for.

Il principale algoritmo di questa classe è sicuramente *Algorithm 4*: *CheckAnswer* in quanto permette di avere una singola risposta attiva per domanda e la variazione dei valori delle coppie chiave-valore presente nella *ActivateFields*.

Algorithm 4 CheckAnswer

```
Input: AnswerID
    boolAnswer ← AnswerList(AnswerID).GetBoolVariable()
    for all Answer in AnswerList do
        if Answer.GetId() = AnswerID then
          Answer.Check()
          Answer.SetBoolVariable()
 4:     else
            Answer.Uncheck()
            if boolAnswer != Answer.GetBoolVariabile then
                Answer.SetBoolVariable()
            end if
        end if
 8: end for
```

Per concludere, esiste una funzione che controlla se esistono risposte visibili oppure disponibili all'utente e in caso di esito negativo, cioè senza nessuna scelta, neanche la domanda verrà resa visibile.

3.3.3 Classe: Questions

Volendo riuscire a caricare in memoria tutte le domande, è necessario avere a disposizione un oggetto che faccia da contenitore. La classe $Questions$ contiene l'insieme dei quesiti con le relative risposte. In realtà è un oggetto tanto semplice quanto utile per distinguere la parte della regolazione con la parte degli output ed in generale con la parte meccanica della centrale che verrà descritta in seguito. Le sue proprietà sono soltanto due, di cui una è la variabile $Dictionary$ comune a tutte le classi. L'altra è un $Array$ definito con il nome $QuestionList$ che funge da raccolta e immagazzina

una domanda per ogni indice. Infatti quando vengono caricati i dati, la posizione della domanda nel vettore viene trasferito anche nella proprietà ID di quest'ultima.

Funzioni

Tutti gli algoritmi presenti in questa classe hanno il compito di impostare o ritornare dei valori delle proprietà descritte nelle classi precedenti. Le uniche eccezioni sono la funzione in cui vengono importati i dati, quindi popolare le domande con relative risposte, mentre l'altra è un algoritmo molto semplice come si può vedere da *Algorithm 5*, che funge da aggiornamento per quanto riguarda la visibilità, le scelte fatte e quelle disponibili, in base sempre ai cambiamenti avvenuti nelle variabili presenti nella proprietà globale del $Dictionary$. Di seguito è visibile il suo pseudocodice.

Algorithm 5 Refresh

```
for all Question in QuestionList do
    if Question.IsNotManualChecked() then
      Question.SetDefaultAnswer()
    end if
end for
```

3.3.4 Classe: SinglePart

Il software ha sempre un obiettivo e cioè fornire un output. Per fare ciò la classe $SinglePart$ è uno dei due oggetti di cui il programma usufruisce. Infatti, come detto in precedenza, $Aircalc$ richiede un certo formato per il valore finale che deve essere messo a disposizione. Si tratta di una stringa abbastanza lunga che deve comprendere, oltre che un insieme di codici aziendali anche altri valori. Una $SinglePart$ contiene tutti i campi necessari per l'output ed altri ausiliari per la selezione dello stesso elemento con caratteristiche diverse ed esprime un solo codice articolo. Infatti le sue proprietà sono ben 48 che andranno ad essere popolate nel caricamento dei dati a seconda del tipo di elemento, come vedremo nella prossima classe. Le variabili obbligatorie per l'output sono in tutto 33 e tra le più importanti ne fanno parte:

- **Codice** - codice articolo
- **Text** di tipo vettore stringa che contiene il testo in diverse lingue per il prodotto
- **Quantità** il numero di elementi
- **Meh** - l'unità di misura della quantità

- da **P01** a **P14** - le porte IN/OUT per il collegamento col PLC
- **Price** - prezzo dell'articolo

Le altre variabili sono obbligatorie solo per una struttura ben definita dell'output ma non sempre sono popolate e rimangono con il loro valore di default che può essere 0 se di tipo numero oppure una stringa vuota se descrittivo.

Funzioni

Le funzioni della classe sono solamente due e, mentre la prima è banale in quanto ritorna il testo in base alla lingua selezionata, la seconda è anch'essa semplice ma fondamentale per l'output. Infatti il suo compito è quello di ritornare la stringa totale dell'articolo concatenando tutte le variabili obbligatorie separate dal simbolo "|" .

3.3.5 Classe: Parts

Come per le domande, anche per la parte di output si usa un contenitore di oggetti per ogni singolo record[4]. Le variabili della classe sono tutti $ArrayList$ in quanto ha il solo compito di immagazzinare oggetti $SinglePart$ e divisi per categoria. Infatti troviamo i seguenti vettori:

- **GruppoSerrande**
- **GruppoValvole**
- **GruppoSonde**
- **GruppoQuadroElettrico**
- **GruppoRaccordiETubature**
- **GruppoMontaggioECablaggio**
- **GruppoPLC**

La suddivisione in diversi gruppi o contenitori è stata fatta per una ricerca più veloce in quanto sono categorie ben diverse e si sa in anticipo a quale gruppo attingere per l'output.

[4]record: un'insieme di caratteristiche per lo stesso oggetto

Funzioni

Tra le funzioni di questa classe troviamo una di $upload$ dei dati, cioè creazione di $SinglePart$ con i dati presenti nel database e popolazione dei vari gruppi. L'altra ha il compito di ritornare una $SinglePart$ in base al gruppo e all'indice nel proprio $array$. Sono algoritmi molto semplici però fondamentali per l'output finale.

3.3.6 Classe: Section

Finita la parte con domande e risposte si arriva alla parte meccanica della centrale. Infatti questa classe è un ausilio con le caratteristiche principali di ogni singola sezione presente in $Aircalc$. Presenta quindi tra le proprietà una serie di variabili che servono per definire l'output in base alle caratteristiche meccaniche. Di seguito quindi l'elenco di queste:

- **UN** di tipo $Integer$. Diminutivo di unità e serve per capire se la sezione è di mandata o ripresa dell'aria. Può assumere il valore 1 se è di mandata o 2 se di ripresa come definito anche da $Aircalc$.

- **Name** di tipo $String$. Il nome è dato dal componente principale della sezione.

- **NumberSection** di tipo $Integer$. Caratterizza la sezione in base alla posizione che ha seguendo il flusso di mandata o ripresa. Nel primo caso con UN=1 il conteggio parte dall'entrata dell'aria dall'esterno verso l'ambiente mentre nel secondo caso è esattamente il contrario.

- **Length** di tipo $Double$. La lunghezza della sezione. Serve principalmente per avere una misura preventiva per la lunghezza dei cavi elettrici da utilizzare sapendo la posizione del quadro elettrico ed i vari componenti da collegare.

- **Width** di tipo $Double$. La profondità della sezione. Il compito è uguale alla proprietà precedente.

- **Height** di tipo $Double$. L'altezza della sezione. Ha sempre la stessa funzione delle due proprietà precedenti determinando il caso in cui le sezioni sono sovrapposte.

- **Components** di tipo $ArrayList$. Vettore che contiene una raccolta di stringhe che esprimono i vari componenti presenti nella sezione.

- **FirstSE** di tipo $Boolean$. Variabile che indica se è la prima sezione lungo il flusso di entrate od uscita.
- **LastSE** di tipo $Boolean$. Come la precedente con la differenza che indica l'ultima sezione al posto della prima.
- **BatteryType** di tipo $String$. Con le diverse tipologie di batterie si ha bisogno di distinguerle attraverso questa variabile perché a seconda della tipologia le domande hanno risposta diversa e di conseguenza anche un output diverso.
- **PreBattery** di tipo $Boolean$. Variabile collegata direttamente ad una risposta dell'utente che esprime il fatto che la prima batteria nel flusso dell'aria ha una funzione di preriscaldamento o preraffreddamento. Proprietà valida solo per le sezioni con la presenza di batterie.
- **PostBattery** di tipo $Boolean$. Come la precedente con la differenza che indica se l'ultima batteria ha una funzione di post-riscaldamento o post-raffreddamento.
- **NumberBatteryInOrder** di tipo $Integer$. Assegna il numero della batteria in ordine di sequenza. Infatti nella maggior parte dei casi il numero di batterie è maggiore di 1 e vengono messe in sequenza. Servirà poi in futuro per la gestione delle $split$ che sono due batterie consecutive ma che hanno lo stesso funzionamento.

Funzioni

Le funzioni presenti in quest'oggetto, a parte quella del costruttore vuote o col parametro, hanno solo il compiti di Set e Get cioè quella di impostare, in fase di caricamento, e ritornare le variabili di classe.

3.3.7 Classe: MSR

Una volta definiti tutti gli oggetti che compongono il software, è necessario farli interagire tra di loro per avere un risultato finale ed in questa classe avviene tutto ciò, per cui si può definire la parte principale del programma. Innanzitutto vengono caricati i dati nei vari oggetti descritti precedentemente e, successivamente viene prodotto l'output per l'applicazione principale. Le variabili di classe sono poche ma essenziali. Nel dettaglio:

- **O** di tipo $airCalcMacroWrapper.CAclMkrHelper$. Variabile messa a disposizione da Aircalc e che verrà descritta in seguito.

- **ActivateFields** di tipo $Dictionary$. E' la variabile descritta negli oggetti precedenti che viene creata in questa classe e passata per riferimento nelle altre.
- **AllQuestions** di tipo $Questions$. Oggetto creato per contenere le domande con relative risposte descritto nelle sottosezioni precedenti.
- **AllParts** di tipo $Parts$. Oggetto creato per contenere tutti i codici di output.
- **UTA** di tipo $ArrayList$. Array che contiene tutte le $Section$ in ordine di flusso dell'aria a partire dalla mandata cioè dall'aria che entra dall'esterno verso l'ambiente interno.
- **TabIdraulic** di tipo $DataSet$. Una raccolta di 8 tabelle in cui sono presenti i codici dei raccordi e tubi per le valvole delle batterie, divisi in base al numero dei collettori e delle vie. Infatti la scelta è fra due o tre vie ed uno o due collettori.

Funzioni

Per quanto riguarda le funzioni si possono suddividere in varie regioni a seconda del loro impiego. Infatti si possono trovare algoritmi di **Upload**, **Get&Set Questions/Answer**, **SetMechanicParts** e **Solve&Solutions**. Mentre nelle prime due regioni la difficoltà è banale, nelle ultime due la complessità degli algoritmi aumenta, soprattutto per quanto riguarda le definizioni e le combinazioni delle parti meccaniche. Nelle pagine successive vengono descritte e fatti alcuni esempi di funzioni con vario grado di difficoltà.

Get and Set Questions and Answers L'algoritmo principale di questa regione usato per il controllo degli errori ogni volta che si cambia una risposta è *Algorithm 6*: *AnswerControlAndErrors*. Il suo compito è di ritornare un array con tutti gli errori dopo aver controllato tutte le scelte delle domande.

Quest'ultimo algoritmo, aggregato all'insieme delle varie funzioni che restituiscono o impostano certe variabili delle classi $Questions$ e $Answer$, permette il corretto funzionamento del software in quanto oltre a restituire il testo con gli errori, che blocca il salvataggio della configurazione, agisce anche sulle risposte di default. Infatti se alla domanda non è stata fatta una scelta manuale, vuol dire che la risposta sarà in base a determinate variabili presenti nel dizionario e può succedere che cambi. Un esempio può essere che se è stato selezionato il PLC della Carel, la risposta di default della domanda riguardante il protocollo di comunicazione è il *Protocollo BacNetIP-Ethernet*, mentre per la scelta di Sauter si opta di default per il *Protocollo ModBus RTU - RS485*.

Algorithm 6 AnswerControlAndErrors

```
Output: arrayError
    arrayError ← new ArrayList()
    for i=0 to AllQuestions.CountQuestions()-1 do
        if AllQuestions.IsActiveQuestion(i) then
            answerChecked ← WhichAnswerIsChecked(i)
            for j=0 to AllQuestions.CountAnswers(j)-1 do
 6:             if AllQuestions.AnswerError(i,j) != String.Empty then
                    if AllQuestions.IsManualChecked(i) then
                        arrayError.Add(AllQuestions.AnswerError(i,j))
                    else
                        AllQuestions.SetDefaultAnswer(i)
                    end if
                end if
            end for
12:     end if
    end for
        return arrayError
```

Upload Per funzionare, il software ha bisogno di informazioni in ingresso che vengono presi dal database. Principalmente tutte le variabili di classe presenti contengono dati e gli algoritmi di $upload$ hanno il compito di popolarle. Infatti in quest'oggetto si trovano le funzioni che caricano tutte le informazioni. La variabile $AllQuestion$ viene definita nel costruttore e passa questo compito alla classe $Question$ così come succede per $AllParts$.

La variabile $ActivateField$ viene popolata con tutte le coppie chiave-valore dove la chiave è il nome della domanda o risposta associata mentre il valore di default è $False$ tranne che per la chiave "$bDefault$" e per la chiave "$bInterni$" che ha come valore un'altra funzione che ritorna $True$ solo se l'utente che utilizza AirCalc ha determinati permessi.

L'upload di $TabIdraulic$ salva in diverse tabelle tutte le combinazioni tra vie e collettori sia per il gruppo tubi sia per il gruppo raccordi. Infatti si possono trovare in tutto otto tabelle in quanto le combinazioni totali sono quattro e cioè 2 vie + 1 collettore, 2 vie + 2 collettori, 3 vie + 1 collettore e 3 vie + 2 collettori. I codici tra i due gruppi sono diversi e dipendono dal diametro dei tubi e delle valvole.

La parte meccanica si trova tutta nell'array UTA. Infatti con l'aiuto della variabile O, che comunica con AirCalc e che verrà spiegata in seguito, l'algoritmo popola il vettore con oggetti di tipo $Section$ che rappresentano tutte le sezioni presenti nella centrale. Ogni sezione presenta al suo interno tutti i componenti necessari per la

regolazione.

Set Mechanic elements La comunicazione con il software principale permette di controllare gli elementi meccanici presenti nella centrale però, per lo sviluppo di questo programma si devono impostare le variabili booleane che attivano domande e risposte in base alla configurazione delle unità di trattamento aria. Infatti gli algoritmi presenti in questa regione servono a testare componenti come le serrande, batterie, recuperatori, umidificatori , filtri ed eventuali presenze di inverter nei ventilatori.

Le serrande, per esempio, possono avere risposte diverse a seconda del tipo, delle possibili combinazioni e del processore con cui sono controllate. A seconda della funzione, i tipi di serranda possono essere:

- **Immissione** : l'aria viene presa dall'esterno della struttura.
- **Espulsione** : l'aria viene espulsa verso l'esterno della struttura.
- **Ricircolo** : l'aria rimane all'interno della centrale, cambiando percorso dal flusso di mandata a quello di ripresa o viceversa.
- **Bypass** : l'aria passa direttamente nella sezione senza interagire con gli elementi presenti all'interno di quest'ultima.
- **Mandata** : l'aria pulita e trattata che verrà emessa nell'ambiente desiderato.
- **Ripresa** : l'aria esausta prese nell'ambiente e che deve essere espulsa.

Controllando quindi la presenza di queste serrande, si attivano le domande relative ad ognuna di esse. Per le risposte invece, bisogna seguire certe regole a seconda della logica funzionale del controllore. Infatti se nel caso di serrande bypass, mandata e ripresa la scelta può essere On/Off mentre per le altre è dettata dalla presenza della serranda di ricircolo. Se esiste quest'ultima, obbliga anche le serrande di immissione ed espulsione, ad avere lo stesso tipo di comando. Difatti per questa combinazione è previsto un commando analogico in modo tale da avere un controllo modulante. Se non è previsto il ricircolo, le serrande di immissione ed espulsione possono avere sia comandi On/Off sia modulanti, indifferentemente che siano presenti entrambe oppure singolarmente. L'unica eccezione riguarda la presenza di immissione+espulsione con il controllore Sauter che gestisce solo il caso On/Off e non anche quello modulante, come invece fa il pCO5 della Carel.

Un altro esempio può essere trovare la sequenza di batterie ed il loro tipo. Infatti le combinazioni posso essere multiple ed a seconda del loro utilizzo si possono avere

funzioni diverse. Inizialmente esiste una domanda in cui viene chiesto all'utente se la prima batteria trovata nel flusso di mandata dev'essere di preriscaldamento o preraffreddamento. Se la risposta è positiva le variabili booleane associate ad esse cambiano, scalando tutte di un posto, cioè la prima batteria diventa di pre-trattamento, la seconda sarà prima, la terza diventerà seconda e cosi via. Tutto questo perché nel caso si volesse una batteria con funzione di preriscaldamento o preraffreddamento si avrà bisogno di una sonda di temperatura in più.

Classe: Output Questa regione è il contenitore degli algoritmi adibiti al risultato cioè alla concatenazione delle stringhe di tutte le $Parts$ che compongono l'output finale. La funzione fondamentale, $Algorithm\ 7 : RequiredCode$ visibile in pseudocodice , per ogni elemento principale come filtri, batterie ,recuperatori, umidificatori e ventilatori esegue una serie di algoritmi "ElementCode", dove al posto di "Element" è sostituito con il nome vero e proprio del componente, per esempio "FilterCode" con parametro la sezione in questione e che avrà il compito di ritornare la stringa finale con il codice giusto. Per gli elementi classificati secondari come serrande, dove la funzione "DampersCode" con lo stesso parametro di prima, controlla l'eventuale presenza e si occupa di ritornare la stringa con il risultato per ogni sezione.

Algorithm 7 RequiredCode

```
Output: finalString
    finalString ← String.Empty
    for all section in UTA do
        Switch section.GetName()
          Case Element:
        partialCode ← ElementCode(oSection)
        if partialCode != String.Empty then
            finalString ← finalString + partialCode
        end if
         End Switch

7:      partialCodeSC ← DampersCode(oSection)
        if partialCodeSC != String.Empty then
            finalString ← finalString + partialCodeSC
        end if
    end for
    finalString ← finalString + ControllerCode()
        return finalString
```

Per trovare il codice giusto, bisogna seguire alcune regole che lo determina a seconda dei dati a disposizione nelle $SinglePart$ e di quelli forniti da Aircalc. Per ogni

componente si ha informazioni differenti e di conseguenza anche logiche diverse. Di seguito vengono spiegate brevemente le regole degli algoritmi per i vari componenti.

- **FilterCode(section)**: Funzione che ritorna il codice del pressostato per la sezione passata nel parametro. Avendo a disposizione tre differenti codici in base alla perdita di pressione, l'algoritmo interroga la variabile calcolata presente in Aircalc e controlla che questa cade nel range dei valori dei pressostati a disposizione, determinando così il codice giusto. Inoltre verranno scelte altre due $SinglePart$ che servono per il montaggio e cablaggio in quanto esprimono un costo di materiale e tempo.

- **BatteryCode(section)**: La stringa rilasciata da quest'algoritmo dipende principalmente dal tipo di batteria. Difatti per ogni tipologia, il risultato finale è diverso.

  Considerando la batterie elettriche l'utente ha la possibilità di scegliere il numero di ingressi-uscite per il collegamento e generando così un il primo codice, valido solo per il calcolo finale per la scelta del controllore. Insieme a questo viene aggiunto il codice del gruppo cablaggio in base alla potenza calcolata da Aircalc.

  La regolazione delle batterie ad espansione oppure a condensazione viene attuata tramite collegamenti Input/Output e l'utente ne ha la facoltà di scegliere la quantità. Le scelte disponibili sono da 0 fino 4 per le batterie ad espansione e da 0 a 2 per quelle a condensazione. Ovviamente il codice del cablaggio è incluso.

  Per le batterie a vapore, l'utente può solo scegliere se predisporre una serie di 3 ingressi/uscite nel controllore oppure no. Infatti l'unico codice da generare è quello che aumenta il numero di ingressi/uscite senza nessun cablaggio.

  La maggior parte delle batterie usate però sono quelle ad acqua. La regolazione prevede quindi l'utilizzo di una valvola a due o tre vie che l'utente ha la possibilità di scegliere. La scelta del codice giusto dipende in primo luogo dal diametro dei collettori e da quello della valvola. Principalmente questi devono essere uguali. In secondo luogo la scelta si basa sulla perdita di carico della batteria, che si calcola attraverso Aircalc, e della valvola stessa. Per avere un buona regolazione, la perdita di carico finale è espressa dalla seguente formula dove kv indica la perdita di carico della valvola e batteria:

$$0,3 < kvValve/(kvValve + kvBattery) < 0,5 \tag{3.1}$$

Oltre alla valvola, vengono inseriti anche i codici del raccordo filettato, servocomando e cablaggio. Vengono inoltre interrogate le variabili di temperatura di ingresso e uscita per aggiungere eventuali codici di alcune resistenze per riscaldare o raffreddare i raccordi. Infatti se la temperatura è minore di 0°C si rischia il congelamento per cui si devono riscaldare i tubi mentre se la temperatura supera i 100°C si fa il ragionamento inverso.

- **RecuperatorCode(section)**: I codici principali di output dei recuperatori sono principalmente associati agli ingressi-uscite. Questi dipendono dal tipo di recuperatore. Il primo è del tipo rotativo e prevede un'entrata analogica se l'utente sceglie una regolazione modulante mentre è digitale se on/off abbinati ovviamente ad un codice per il cablaggio. Sono obbligatorie inoltre due sonde di temperatura, una posizionata all'esterno della centrale mentre l'altra in ripresa o ambiente. Inoltre l'utente ha la possibilità di aggiungere una sonda antigelo con la conseguente aggiunta di codice di quest'ultima.

  La seconda tipologia è quella con la presenza di due batterie di recupero o reintegro. In questo caso il collegamento avviene tramite un'entrata digitale con il relativo gruppo cablaggio. C'è la possibilità di abbinare una pompa alle batterie che aggiunge ulteriormente i codici della pompa, montaggio e cablaggio. Le sonde di temperature sono obbligatorie come nel primo caso. La sonda antigelo non è prevista.

  La terza e ultima tipologia è il recuperatore a flussi incrociati dove non è previsto il collegamento ma ci sono sempre le sonde di temperatura. Anche in questo caso è possibile integrare una sonda antigelo come nel recuperatore rotativo.

  Gli elementi in comune a tutte le tipologie sono quindi le sonde di temperatura e la presenza o meno delle serrande ma quest'ultime vengono trattate nel codice $DamperCode$.

- **FanCode(section)**: La scelta per i codici di output riguardante i ventilatori dipende dalla gestione del motore. Infatti si può gestire tramite potenziometro oppure da processore. Nel primo caso si prevede un entrata ed un'uscita digitale per ogni ventilatore presente in mandata o ripresa. Nella gestione da processore è uguale con l'aggiunta di un'uscita digitale. Se il controllo avviene a portata o pressione costante,viene previsto anche un'entrata analogica per le sonde di qualità aria cioè CO2 (anidride carbonica) e VOC (gas organici), oltre naturalmente ad un trasduttore con relativo codice. Tutto ciò verrà montato per cui serve anche il codice che esprime il gruppo montaggio ed uno per il cablaggio. La presenza

inoltre del inverter porta ad avere un ulteriore output per il cablaggio con cavi schermati.

- **HumifierCode(section)**: La regola principale dettata dalla presenza di un umidificatore nella centrale è l'obbligo di avere una sonda di umidità. La sua posizione dipende in primis dalla scelta del PLC e poi dall'utente che ha la facoltà di collocarla meglio. Infatti se la marca del controllore è Sauter, si è obbligati ad avere la sonda di umidità nel flusso di ripresa. In questo caso le possibilità sono di avere una sonda a canale, a filo oppure in ambiente. Secondo Carel invece la sonda può essere situata anche nel flusso di mandata anche se è fortemente consigliato di metterla in ripresa. Di solito alla sonda di umidità viene abbinata anche una sonda di temperatura ed infatti se l'utente sceglie l'opzione della sonda di umidità in ambiente, il codice corrispondente le contiene entrambe.

  L'output con i codici da ritornare dipende soprattutto dal tipo di umidificatore utilizzato. Il primo è l'umidificatore a vapore ed ha come output tre codici che rappresentano i gruppi di connessione, cablaggio e I/O . Per la connessione il codice è unico così come per il cablaggio dove in questo caso varia la quantità perché dipendente dalla distanza dal controllore. Per quanto riguarda l'I/O quest'umidificatore necessità di due uscite, una analogica e l'altra digitale ed infine un'entrata digitale.

  La seconda tipologia di umidificatore è definita a pacco. In questo caso il collegamento avviene solo con un'entrata digitale con funzionamento on/off. Gli altri codici obbligatori sono quelli del gruppo di connessione e cablaggio dell'elettrovalvola mentre se la centrale non usa l'acqua di rete bisogna aggiungere i codici di connessione e cablaggio della pompa.

  Il terzo tipo è l'umidificatore a pressione. Attualmente però non servono codici di output per la sua regolazione.

- **DampersCode(section)**: Funzione che controlla la presenza delle serrande della sezione passata nel parametro. La scelta del codice dipende dalla forza esercitata sulla serranda espressa in Nm e dal numero di leve che determina anche la quantità. Entrambi i dati sono reperibili in Aircalc e la scelta del codice giusto è selezionato confrontando il dato associato nella $SinglePart$ con il rapporto tra la forza esercitata ed il numero di leve che deve essere maggiore o uguale rispetto al rapporto delle due come si può vedere dalla seguente formula:

$$SinglePart.GetForceNm() > iNm/iLeve \tag{3.2}$$

Un'altra discriminante per la scelta della serranda è il ritorno a molla in quanto la presenza di quest'ultimo componente significa codice differente. Inoltre collegandosi alle porto I/O, il servocomando ha un'entrata digitale nel caso on/off mentre è analogico se modulante. Inoltre viene aggiunto un codice aggiuntivo contenente una protezione se la centrale verrà posizionata all'esterno .

- **ControllerCode()**: Algoritmo che controlla tutti i codici presenti nella stringa finale per determinare il numero di ingressi uscite e ritornare la stringa con il codice del controllore a seconda della marca scelta dall'utente. Nel caso di Carel, il $Pco5+\ Large$ avente 13 uscite digitali, 6 analogiche e 18 ingressi digitali e 6 analogici, se il numero totale di ingressi/uscite presente nella configurazione si può aggiungere una piccola estensione, ovviamente con un codice in più avente altre 4 uscite digitali, una analogica, 4 ingressi digitali e 4 analogici. Se l'utente invece seleziona la marca Sauter, il controllore $RDT940$ ha un numero ristretto di ingressi/uscite e viene utilizzato se la regolazione prevede pochi elementi nella centrale. Infatti in questo caso si dispone di 5 uscite digitali e 2 analogiche e 6 ingressi digitali e 2 analogici.

3.3.8 Classe: FormMSR

Il programma ha bisogno di interagire con l'interfaccia video ed in questa classe vengono gestiti tutti i controlli software-utente. Infatti tutti gli automatismi sono mostrati all'utilizzatore grazie alle funzioni sviluppate nell'oggetto $FormMSR$ che ha il compito di gestire tutti gli eventi e mostrare a video il risultato. La prima funzione è quella che apre la finestra Windows e inizializza tutti gli oggetti descritti in precedenza per avere a disposizione dati con cui lavorare. La nuova finestra viene aperta in modalità modale cosicché l'utente non può agire sul programma principale finché questa non viene chiusa da uno specifico bottone che permette di salvare i cambiamenti oppure lasciare le scelte fatte precedentemente ma solo dopo aver risposto ad un pop-up che avvisa il mancato salvataggio. Inoltre con altri due algoritmi scritti a codice si può spostare e ridimensionarla visto che per scelta progettuale i noti tre bottoni di chiusura,ridimensionamento e riduzione ad icona sono stati tolti.

Ci sono poi diverse funzione che sono collegabili all'evento $Click$ a seconda da dove questo avviene. Se vengono schiacciati i bottoni $Forward$ o $Backward$ sono innescati due diversi algoritmi che aggiornano la variabile di classe dell'indice della domanda ed a seconda di quest'ultimo cambia il $label$ con la domande e tutti i $Radio\ Botton$ con le relative risposte. Se il click avviene sui bottoni $Save$ ed $Exit$ agisco-

no due algoritmi banali che salvano l'attuale configurazione e permettono il calcolo della regolazione in Aircalc oppure escono dal software. L'ultimo evento, ma il più complesso, associato al singolo click è quello che cambia le risposte alle domande. Infatti selezionando una risposta della domanda che viene mostrata a video, entrano in gioco diverse funzioni che possono cambiare il numero di domande, mostrare eventuali errori e l'output finale. Infatti, ogni volta che si cambia una risposta attraverso i $Radio\ Button$, la nuova scelta viene salvata e deselezionata la precedente, vengono aggiornate tutte le variabili presenti in $ActiveFields$ e quindi di conseguenza tutto l'automatismo fatto dal programma. Inoltre l'oggetto $TreeView$ mostra la nuova configurazione e nel caso ci fossero errori manuali saranno mostrati attraverso le $label$ della sezione "Gestione Errori" che prima erano nascoste.

L'ultimo evento verificabile in questo programma è quello del doppio click su una domanda presente nel $TreeView$. Quando ciò succede la funzione ad esso associato, aggiorna la variabile indice collegata alla domanda e la mostra a video, cambiando il testo della label e di conseguenza anche le risposte sui $Radio\ Button$.

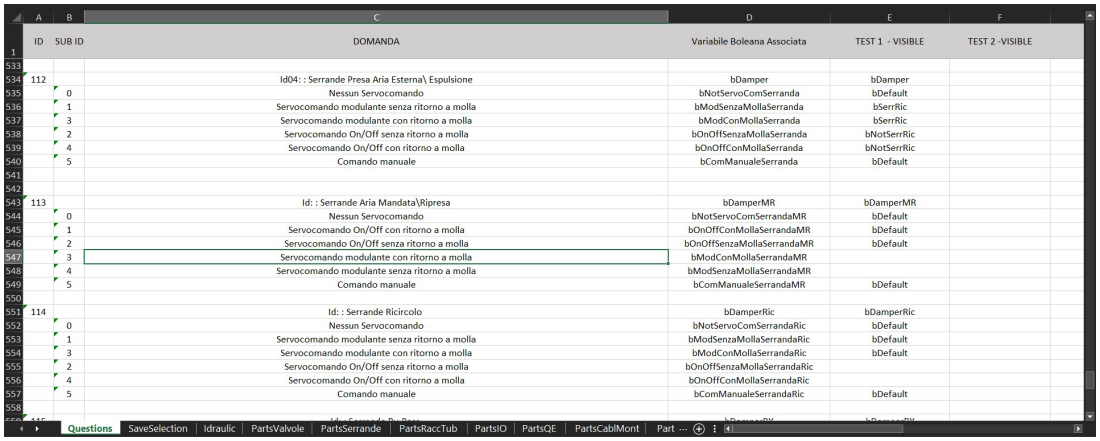
Un evento quindi è un messaggio che scaturisce un azione e nella programmazione viene inviato da un oggetto. Gli eventi messi a disposizione da Microsoft e utilizzati in questo software sono il $Click$, il $DoubleClick$, il $TextChanged$ che compie un'azione quando il testo cambia ed il $CheckedChange$ che agisce al cambiamento di stato di un $RadioButton$ oppure $CheckBox$.

3.4 Database

Un software ha bisogno di lavorare con dati ed il database raccoglie e immagazzina tutte le informazioni con cui il software deve interagire. Per il momento la scelta della basi di dati è ricaduta su un file Excel in quanto, almeno per la fase iniziale dopo il rilascio, è più semplice fare delle modifiche ed è accessibile anche al personale diverso dallo sviluppatore.

Come si può notare dalla Figura 3.2, il file è composto da diversi fogli ed ognuno di essi contiene informazioni essenziali per il funzionamento del software. Il primo in ordine di importanza è quello che contiene le domande e risposte. Nel dettaglio, è composto da 15 colonne, ciascuna con la propria mansione. Infatti le prime due contengono l'ID della domanda, che è un progressivo crescente, e della risposta, che invece inizia sempre da 0 per la prima scelta e cresce fino ad arrivare al numero totale di risposte per quel quesito. Segue con colonna il $Testo$ vero e proprio cioè la stringa che verrà mostrata a video. Proseguendo si trova $BoolAssociate$, cioè il nome della

variabile booleana associata, caricata nel dizionario $ActivateField$ e che risulta fondamentale per il funzionamento del software così come le successive due colonne che contengono il nome di quest'ultime variabili e verificano se la domanda o risposta risulta attiva o visibile. Esistono poi, due campi con il messaggio d'errore e la variabile booleana che lo scaturisce nel caso fosse attivo. Ciò è molto utile in quanto il testo può indicare l'ID della domanda in cui succede l'irregolarità, riuscendo a trovarla velocemente, e una possibile spiegazione del perché è sbagliata. Infine, le altre colonne contengono la traduzione, una per ogni lingua diversa, le note per delle spiegazioni extra e l'ultima con elenco di numeri separati dalla virgola che indicano la visibilità a seconda del livello utente.



ID	SUB ID	DOMANDA	Variabile Boleana Associata	TEST 1 - VISIBLE	TEST 2 -VISIBLE
112		Id04: : Serrande Presa Aria Esterna\ Espulsione	bDamper	bDamper	
	0	Nessun Servocomando	bNotServoComSerranda	bDefault	
	1	Servocomando modulante senza ritorno a molla	bModSenzaMollaSerranda	bSerrRic	
	3	Servocomando modulante con ritorno a molla	bModConMollaSerranda	bSerrRic	
	2	Servocomando On/Off senza ritorno a molla	bOnOffSenzaMollaSerranda	bNotSerrRic	
	4	Servocomando On/Off con ritorno a molla	bOnOffConMollaSerranda	bNotSerrRic	
	5	Comando manuale	bComManualeSerranda	bDefault	
113		Id: : Serrande Aria Mandata\Ripresa	bDamperMR	bDamperMR	
	0	Nessun Servocomando	bNotServoComSerrandaMR	bDefault	
	1	Servocomando On/Off con ritorno a molla	bOnOffConMollaSerrandaMR	bDefault	
	2	Servocomando On/Off senza ritorno a molla	bOnOffSenzaMollaSerrandaMR	bDefault	
	3	Servocomando modulante con ritorno a molla	bModConMollaSerrandaMR		
	4	Servocomando modulante senza ritorno a molla	bModSenzaMollaSerrandaMR		
	5	Comando manuale	bComManualeSerrandaMR	bDefault	
114		Id: : Serrande Ricircolo	bDamperRic	bDamperRic	
	0	Nessun Servocomando	bNotServoComSerrandaRic	bDefault	
	1	Servocomando modulante senza ritorno a molla	bModSenzaMollaSerrandaRic	bDefault	
	3	Servocomando modulante con ritorno a molla	bModConMollaSerrandaRic	bDefault	
	2	Servocomando On/Off senza ritorno a molla	bOnOffSenzaMollaSerrandaRic		
	4	Servocomando On/Off con ritorno a molla	bOnOffConMollaSerrandaRic		
	5	Comando manuale	bComManualeSerrandaRic	bDefault	

Questions | SaveSelection | Idraulic | PartsValvole | PartsSerrande | PartsRaccTub | PartsIO | PartsQE | PartsCablMont | Part ...

Figura 3.2: Fogli Excel usato come Database

Il secondo foglio contiene solo tre campi. Il primo custodisce l'identificativo della selezione su cui si sta lavorando su Aircalc che è sempre univoco. Il secondo ospita una codifica rapida delle scelte fatte con domande e risposte mentre l'ultima tratta di un stringa molto lunga, contenente tutti i componenti fisici della selezione meccanica, fatta sul software principale, per avere un controllo rapido nel caso di cambiamenti.

Un'altra pagina è quella della parte idraulica. Infatti sono presenti otto tabelle contenenti i codici dei gruppi di raccordi e dei tubi, differenziate dal numero di collettori e vie delle valvole. Per ognuna è presente un codice univoco a seconda della dimensione del diametro di collettore e valvola. La grandezza si misura in pollici e può variare da mezzo pollice fino ad arrivare a sei pollici che è la misura massima compatibile con le batterie usate nelle centrali trattate.

Gli ultimi sei fogli sono i contenitori dei vari gruppi presenti nella classe $Parts$. Trattano quindi di serrande, valvole, sonde, quadro elettrico, montaggio e cablaggio ed infine di PLC. Hanno tutti la stessa struttura e presentano campi comuni e altri che si differenziano per la propria mansione. Il dato principale è sicuramente il codice

aziendale, che verrà fornito una volta scelto il componente. Segue poi una descrizione associata, il codice fornitore, delle note private e l'unità di misura. Le successive quattro colonne contengono la quantità di ingressi digitali e analogici cosi come le uscite. Ciò risulta una parte molto utile per il conteggio finale di input/output che inciderà sulla scelta del controllore e della sua eventuale espansione. I restanti campi sono ristretti ai singoli gruppi in quanto una colonna risulta utile solo per un determinato gruppo. Un esempio può essere il valore del kvs che riferisce solo alle valvole, cosi come la presenza di ritorno a molla per i servocomandi delle serrande oppure il valore massimo e minimo per la scelta del pressostato differenziale nella sezione dei filtri.

3.5 Interazione con il software principale

Come si è visto in precedenza, il nuovo software ha bisogno di interagire con quello principale sia per riuscire a capire quali componenti meccaniche sono presenti nella selezione, sia per estrarre determinati valori calcolati. Gli sviluppatori di Aircalc quindi mettono a disposizione la classe $CAclMkrHelper$ che si trova nel codice sorgente sotto il namespace[5] $airCalcMacroWrapper$. Grazie a quest'oggetto, dichiarato e utilizzato nelle classi MSR e $FormMSR$, è stato possibile individuare i vari componenti delle diverse sezioni ed è risultato fondamentale per l'output finale in quanto sono state selezionate le $SinglePart$ in base ai valori calcolati su Aircalc.

Principalmente sono state usate solo due funzioni di quelle che mette a disposizione la classe: **GetValue** e **SetValue**. La prima viene usata nella maggior parte dei casi mentre la seconda viene usata solo in fase di salvataggio della configurazione per comunicare ad Aircalc che la regolazione è stata fatta in modo corretto. Entrambe utilizzano un parametro di tipo stringa, il quale indica la variabile da impostare o ottenere e ritornano valori sempre sotto forma di stringa. Per capire qual è la composizione della stringa, quindi il nome delle variabili, bisogna fare un passo indietro e capire com'è impostata la struttura di ogni selezione.

La struttura di ogni selezione è ad albero e rappresentata come nella Figura 3.3. Il nodo padre **HDR** contiene tutte le informazioni di progetto come lingua, data di inizio e identificativo univoco della selezione. Successivamente si trova **GD** sigla di $General\ Data$ e si possono trovare dati relativi alle certificazioni e proprio della regolazione. In ordine poi si trovano **UN** ovvero $Unit\ Data$ dove sono i collocati dati relativi all'insieme delle sezioni che compongono il flusso di mandata o ripresa e **DS** cioè $Delivery$

[5]namespace: in italiano spazio dei nomi che il programmatore definisce per indicare una collezioni di nomi di entità

$Section$ che indicano come la centrale verrà spedita . Scendendo di livello si trovano i nodi delle SE (section), CO (component), SC (small component) , AC (accessories),OP (openings) e RT (doors). Quest'ultimi, a parte gli AC e RT, interagiscono molto per la realizzazione del nuovo software. Tutti i nodi elencati fino ad ora possono contenere ulteriori strutture che raggruppano variabili serventi il programma.

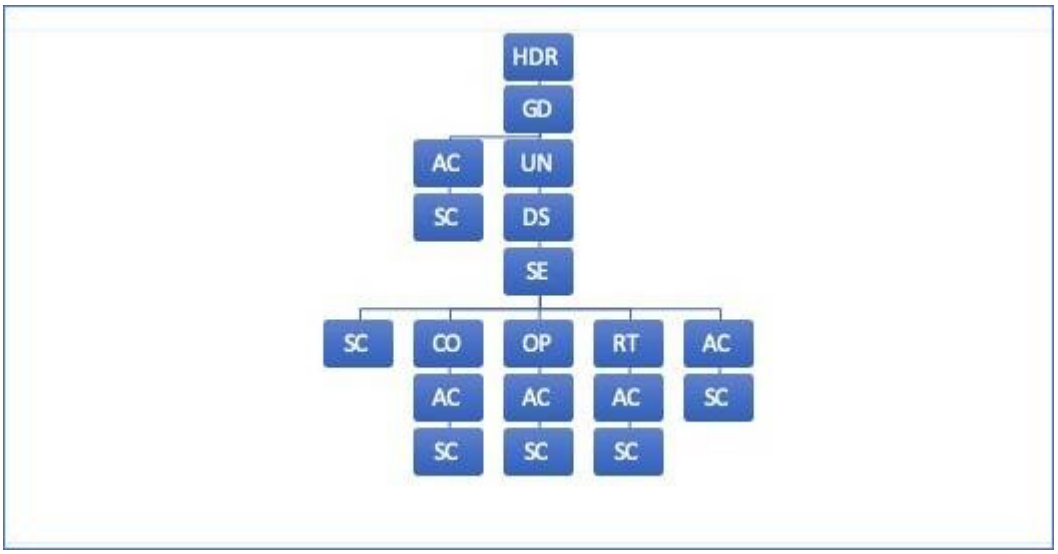


Figura 3.3: Struttura Aircalc della selezione

Detto ciò, si può spiegare la struttura della stringa parametro usata nelle funzioni elencate sopra ed è:

"nomeNodo.nomeContenitore.nomeVariabile"

Di seguito sono riportati due esempi, realmente usati.

Ex.1 Per ottenere il valore della variabile calcolata con il nome EndPa che si riferisce alla perdita di carico del componente batteria che ha come indice 3 nell'insieme di tutti i componenti viene usata la stringa:

"CO3.OUT.EndPa"

Se si volesse ottenere il valore impostato nella fase prima del calcolo della stessa variabile si dovrebbe usare:

"CO3.INP.EndPa"

nella funzione GetValue che diventerebbe:

GetValue("CO3.INP.EndPa")

Ex.2 Le variabili presenti nella parte bassa dell'albero sono tutte numerate, invece per la parte alta non esiste un indice perché le informazioni sono univoche e valgono per tutta la selezione. Infatti se si volesse impostare la variabile "valid" contenuta nella struttura "MSR" del nodo "GD", la stringa dovrebbe essere:

"GD.MSR.valid"

nella funzione SetValue che diventerebbe :

SetValue("GD.MSR.valid")

In Aircalc l'utente con il privilegio di amministratore ha a disposizione una finestra di debug in cui è possibile interrogare e impostare qualsiasi variabile cosi come da codice descritto sopra.

Capitolo 4

Conclusione

4.1 Sviluppi futuri

Il software verrà rilasciato con una versione iniziale, abbastanza basilare, dove l'output descritto negli obiettivi deve contenere una stringa contenente tutti i componenti necessari per la regolazione. Dopo un periodo iniziale di assestamento e utilizzo da parte degli utenti, il programma verrà ampliato con ulteriori funzionalità, sia progettuali che tecniche a seconda delle richieste da parte degli utilizzatori e dello sviluppo continuo delle centrali di trattamento aria.

Gli aggiornamenti già programmati e da fare nel prossimo futuro sono i seguenti:

- Calcolo della componentistica del quadro elettrico

  Il programma, per adesso, calcola solo i componenti necessari per attuare la regolazione però manca tutta la parte dei collegamenti elettrici. Infatti nella classe $Section$ sono state predisposte le variabili lunghezza ed altezza delle sezioni per effettuare questa funzionalità. Inoltre a seconda della configurazione meccanica saranno installate anche alcune cassette di derivazione per il collegamento tra componenti e quadro elettrico.

- Database

  Come detto in precedenza, l'attuale database si trova su un file Excel. Una volta superato il primo periodo di prova ed ampliamento, i dati verranno spostati direttamente sul database ufficiale di Aircalc. Questo provocherà anche un cambiamento nella parte di codice sorgente per l'interazione, però avrà il beneficio di non dover fare due aggiornamenti separati.

- Schema P&I

  Una volta calcolati tutti i componenti utilizzati per la regolazione, del quadro e delle cassette di derivazione, verrà implementata una stampa con lo schema dell'impianto elettrico da allegare alla scheda tecnica della centrale di trattamento aria.

- Template ingressi-uscite

  Si tratta di uno schema fisso e molto pratico per la realizzazione dei collegamenti tra PLC e componentistica. Infatti si creerà una tabella dove verrà associato ogni ingresso e uscita ad una determinata porta del processore, aiutando il personale addetto al montaggio ed a tenere traccia dei collegamenti.

Oltre ai punti appena descritti, verranno valutate le proposte e richieste da parte degli utilizzatori che potranno mantenere il software in continua evoluzione visto che non esiste un numero limite di aggiornamenti.

4.2 Epilogo

In questo progetto, lo scopo era di creare un software che potesse interagire con Aircalc ed estrarre i codici dei componenti necessari ad attuare la regolazione, mantenendo un'interfaccia utente simile a quella già esistente con domande e risposte a scelta multipla. L'obiettivo è stato raggiunto in pieno attraverso una programmazione ad oggetti, replicando la versione precedente eseguita attraversa un foglio Excel, con tempi di apertura e calcolo all'ordine del secondo, decisamente migliori rispetto a prima che raggiungevano i minuti.

La difficoltà incontrata nella realizzazione del programma è stata in primo luogo nella scelta progettuale di utilizzare una struttura tabellare per quanto riguarda i quesiti e le scelte che ha comportato l'utilizzo di un dizionario contenente coppie chiavi-valori associate ad ogni domanda o risposta. In secondo luogo la complessità maggiore è stata estrarre tutti i codici finali in base alla configurazione della selezione riguardante la centrale di trattamento dell'aria, dato che sono numerose le combinazioni possibili ed anche un singolo elemento può cambiare radicalmente le scelte di output. Un altro fattore che ha inciso è il differente modo con cui eseguono la regolazione i due produttori Carel e Sauter. Difatti, spesso i controllori agiscono con scelte differenti su alcuni componenti presenti nella selezione.

Concludendo, quindi, il software funziona correttamente ed il risultato finale è soddisfacente. La realizzazione attraverso una programmazione ad oggetti porta, oltre

che tempi d'attesa e di calcolo migliori, il vantaggio di poter integrare infinite nuove funzionalità che prima erano impossibili da attuare.

Bibliografia

[1] *Fast S.p.A.*, https://www.fastaer.com

[2] *Software airCalc*, https://www.aircalc.info/

[3] *Manuale tecnico Carel pCO5*, https://www.carel.it/documents/10191/0/+030220960/d8e283a5-71cf-43a1-8bae-70c7b4059303?version=1.1

[4] *Manuale tecnico Sauter RDT921*, https://www.sauter-flex-hvac-vision.com/wp-content/uploads/RDT900/RDT921_Applicazioni_rev1.pdf

[5] *Cay S. Horstmann, Concetti di informatica e fondamenti di Java, quinta edizione*, 2013.

[6] *Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser, Algoritmi e strutture dati in Java*, 2015.

[7] *Sito ufficiale NuGet*, https://www.nuget.org

[8] *Gestire e generare eventi* https://docs.microsoft.com/it-it/dotnet/standard/events/