Oggetto : Relazione progetto Programmazione a Oggetto

Studentessa : Martina Dall'Amico

Titolo : EnergySystem

Introduzione :

EnergySystem è una applicazione per generare previsioni per una azienda energetica "EniGreen "che produce e vende energia pulita. Energysystem permette di aggiungere, modificare, cercare dei sensori all'interno dei due impianti dell'azienda : quello Solare e quello Eolico .L'applicazione permette inoltre di scegliere uno o più sensori dell' impianto per simulare un grafico che l'azienda userà a posteriori per calcolare i diversi scenari di profitto. I sensori che è possibile gestire sono quelli per la Temperatura, per l'Umidità e per la Radiazione ciascuno con le proprie caratteristiche grafiche.

EnergySystem è caratterizzato dalla possibilità di testare più sensori di un impianto e vedere le influenze che gli agenti atmosferici esterni hanno su di essi. L'applicazione inoltre segnala se in una simulazione alcuni sensori andrebbero danneggiati per aiutare l'azienda a comprendere i limiti che potrebbero avere con i sensori acquistati.

Ho scelto questo tema perché trovo molto interessante il tema delle energie pulite inoltre i vari sensori e le funzionalità scelte mi hanno permesso di applicare in modo non banale il polimorfismo.

Descrizione del modello :

Il modello logico si sviluppa in una principale gerarchia con una classe astratta : Sensore_Astratto che contiene tutte caratteristiche comuni ai vari sensori ovvero : codice, nome, descrizione, l'immagine,se è o meno funzionante, il costo e l'impianto in cui è stato istallato. Escluso il codice che è univoco per ogni sensore ,il costo e l'impianto perché una volta istallato non si può spostare per ciascun attributo sono implementati i metodi get e set. Dalla classe astratta derivano tre classi concrete : S_Temperatura, S_Umidita e S_Radiazione che presentano le funzionalità specifiche per i sensori. S_Temperatura rappresenta un sensore di cui si può scegliere la scala di misurazione , la precisione, il livello massimo e minimo che si può misurare prima di compromettere il suo funzionamento. Il S_Radiazione serve per tenere controllata la radiazione che arriva dal sole e per questo sensore si può settare l'angolo a cui prende la radiazione, il livello di fusione e il livello minimo al quale produrrà energia.La terza classe concreta : S_Umidita permette di stabilire il livello al quale si creerà della muffa bloccando il sensore e il livello di massimo che può misurare .

Le classi naturalmente hanno come metodi polimorfi: la funzione clona per clonare un oggetto e le funzioni getValori e getValoreParametri che permettono di restituire una matrice di valori in base alla quantità di giorni e sensori su cui si vuole fare una simulazione. Nella vista i grafici hanno una rappresentazione diversa in base al sensore di pertinenza è quindi stata aggiunta la classe IVisitor per sfuttare la funzione visitor e distinguere i sensori senza utilizzare i cast. Alla classe astratta sono stati quindi aggiunti i metodi di accept per accettare i Visitor.I metodi del visitor sono stati implementati concretamente nella classe smistatore.

Secondo il design pattern MVC ,che divide e fa comunicare il modello con il controller e il controller alla vista ,in questo progetto al fine di mantenere divisa la pare grafica e realizzare in modo corretto

l’incapsulamento, nella parte logica si trova la classe modello che si comporta come un contenitore per tutti i sensori e sviluppa le funzioni di aggiunta, rimozione, modifica, salva , carica , ricerca e simula. La ricerca avviene tramite un oggetto parser che permette di filtrare i sensori mentre simula avviene tramite un oggetto simulazione che restituisce la matrice di valori.

La funzione aggiungi permette l’aggiunta solo se il codice inserito non è già prenotato da un altro vettore. La funzione rimuovi toglie il sensore dal vettore che lo contiene in base al codice univoco e la funzione modifica richiama la funzione specifica di ogni sensore. La funzione ricerca accetta una stringa su cui agisce diversamente in base al fatto che sia costituita da soli numeri o numeri e lettere. Infatti la funzione filtrasensori () se riceve solo numeri attua la ricerca su campi dati numerici generali e specifici di ogni sensore mentre se si scrive una stringa ricerca sugli attributi non numerici. La ricerca attribuisce un punteggio ad ogni carattere numero o lettera che trova nella ricerca e poi ritorna un vettore in ordine di similitudine alla ricerca.

L’oggetto simulazione si basa su un vettore di sensori , sottoinsieme di tutti quelli presenti nel modello, su cui prima è statio generato un numero casuale in base al range di misurazione di ogni sensore (livelli massimi o minimi se presenti) e dal quale per ogni giorno richiesto dalla Gui produce 24 risultati, uno per ogni ora, alzandosi o abbassandosi di un range statico definito dalla classe rispetto al valore dell’ora precedente. Questo permette alla simulazione per i sensori di temperatura, umidità e radiazione di seguire i ritmi naturali del sole con un una salita nelle prime ore della giornata con un picco dopo circa 12 seguito da una discesa. Questo permette di avere grafici realistici anche se generati da dati casuali .

La persistenza dei dati è stata realizzata con due funzioni una di carica e una di salva che hanno necessitato della definizione dell’operatore di output nelle classi sensori.

Polimorfismo :

Il polimorfismo “EnergySystem” è stato sviluppato attraverso queste funzioni :

virtual Sensore_Astratto* clona() const = 0;

virtual void accept(IVisitor& visitor) = 0;

virtual double getValore() =0;

virtual double getValoreParametri(double precedente,bool s) =0;

L’utilizzo meno banale è l’uso del design patter Visitor che ho utilizzato per smistare i sensori nel modello al fine di dividere la loro rappresentazione in 3 grafici diversi. Questo passaggio è stato fondamentale perché permette la divisione in n vettori distinti in base a n classi concrete di sensori disponibili senza dover sapere a priori che tipo dinamico sono gli oggetti e permette di aumentare l’estendibilità e la mantenibilità del codice.Per ogni classe concreta è stat implementato il metodo accept e grazie alla classe concreta Smistatore si attua la vera divisione dei sensori.

Il metodo clona utilizza la definizione standard mentre i metodi per restituire i valori sono stati implementati per restituire valori diversi a seconda dei limiti propri di ogni sensore. Questo è un polimorfismo non banale dati i pochi distinguo di questi sensori e dal fatto che non si abbiano a disposizione dei veri dai ma dei dati casuali ma concettualmente realizzano il fatto che ogni sensore sarebbe responsabile in modo opportuno della generazione dei suoi dati in base ai parametri su cui è stato settato.

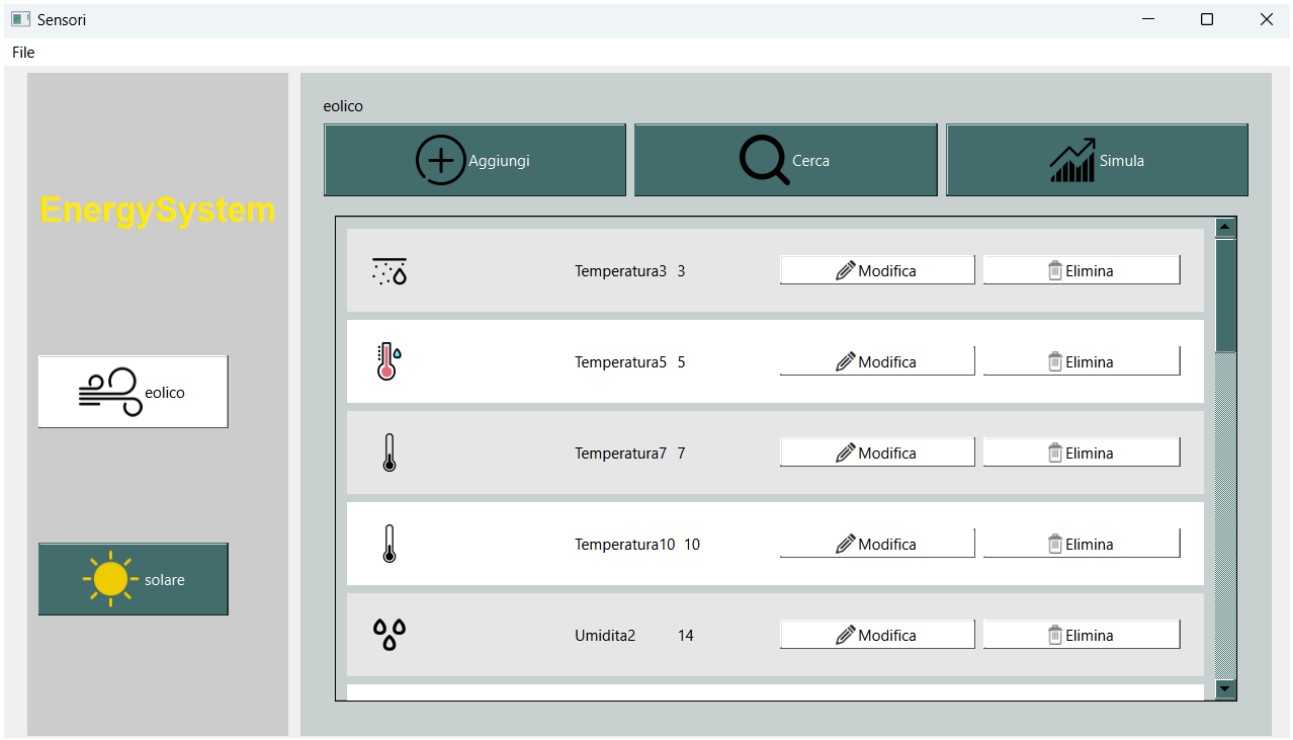
Persistenza dei dati

Per implementare la persistenza dei dati è stato deciso un formato con la funzione toostring() per convertire una stringa e questo accoppiato all'operatore di output è stato usato per il salvataggio su un file di testo mentre per il caricamento è stato usato un oggetto Parser che dividesse le stringhe per decostruirle in base al formato di creazione e restituire i valori per la creazione degli oggetti sensore. Un file di esempio delle struttura dei file è presente nel file "prova" che contiene diversi tipi di sensori per i due impianti.

Funzionalità implementate

La gui è stata pensata per essere facile e comprensibile ,per non disorientare l'utente e per essere in grado di restituire i feedback corretti che possano indirizzare gli utilizzatori.

Il fatto che l'interfaccia sia userfrendly è dato dalla semplicità in fatto in alto a sinistra sono state implementate le classiche funzionalità di salva e carica con due shortcut. Il resto dell'interfaccia è stata divisa in tre aree principali : a sinistra i 2 bottoni che fanno scegliere gli impianti , in alto a destra i 3 bottoni delle azioni principali : aggiungi, cerca e simula e al centro l'area principale delle liste di sensori.



Questa è come si presenta la schermata principale.

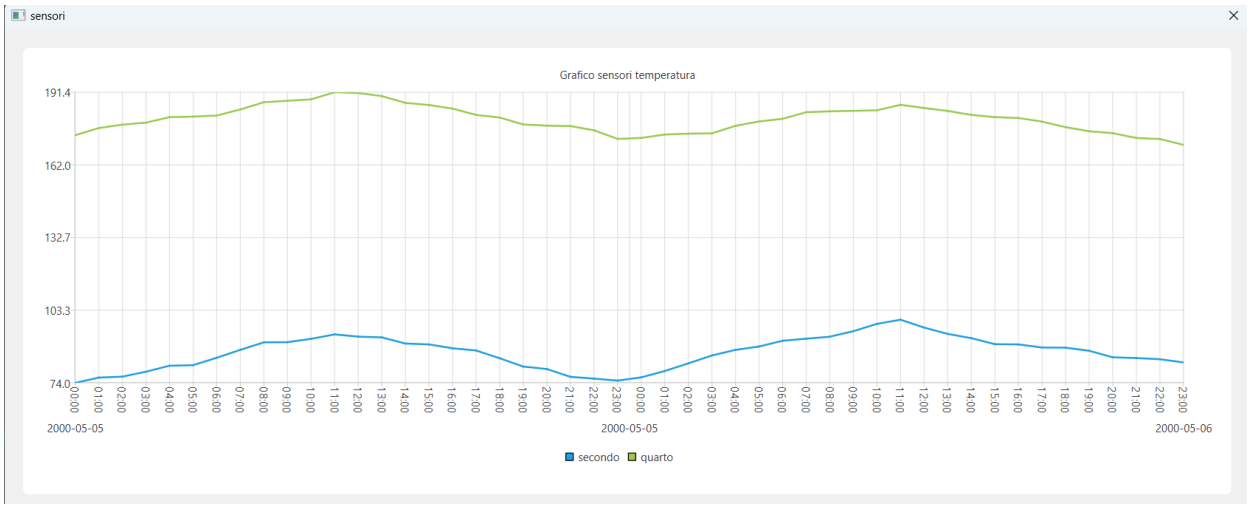
La funzione aggiungi ha implementati tutti i controlli sui codici univoci, sui campi se numerici o completabili con stringhe i campi rimangono precompilati in caso si presenti un messaggio di errore.

Modifica ha le stesse caratteristiche di aggiungi per i campi che si possono modificare e si presenta già precompilata in base al sensore scelto.

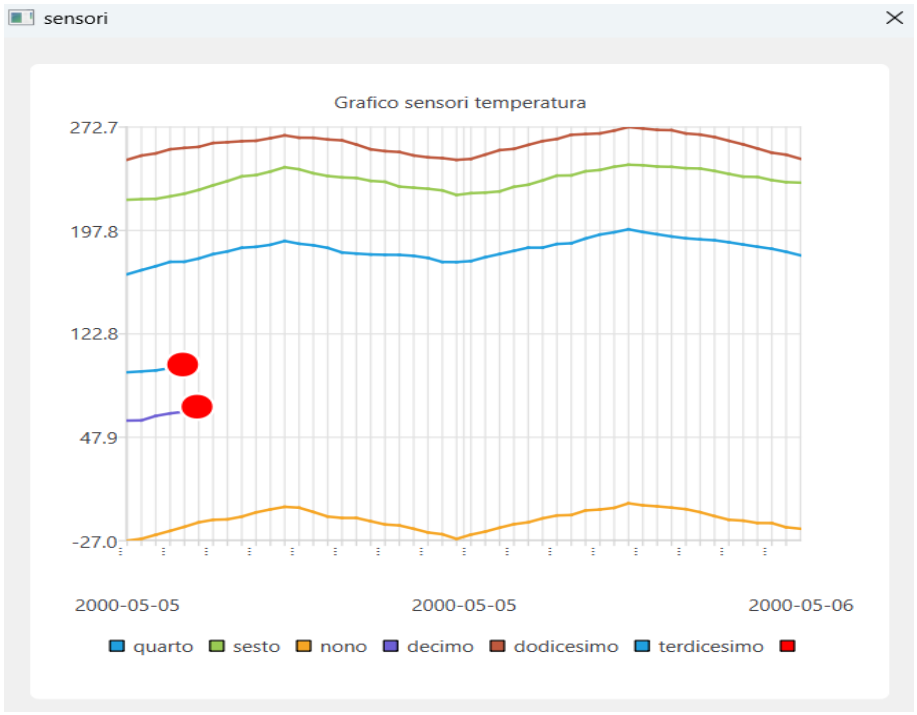
Per rendere più accessibile il widget costituito dalla lista di sensori gli sfondi sono resi diversamente e si aggiornano in caso di eliminazione o di aggiunta.

La simulazione permette di scegliere il giorno di partenza e di conclusione della simulazione, controllando che il secondo sia successivo al primo, e presenta da 0 a tre grafici distinti in base al fatto che non si sia scelto alcun sensore, o tutti di una tipologia o di più tipologie distinte.

Nell' asse X si troveranno le varie ore del giorno con i giorni scelti per la simulazione.



Dato che EnergySystem è stata implementata per aiutare in previsioni in base a range di dati possibili può succedere che dei sensori si rompano e debbano essere riparati (ovvero modificati nel nostro modello) e per evidenziare in che momento si sarebbero rotti i sensori nei grafici si trova un pallino rosso.



Ogni sensore è stato inoltre associato a una icona e ho scelto di usare dei dialog per i grafici , l'aggiunta e le modifiche perché potessero rimanere aperti anche mentre si consultava il resto dell'applicazione.

Per qualsiasi eliminazione prima avviene la richiesta se si è sicuri per evitare di dover ripetere l'aggiunta successivamente.

Con l’implementazione grafica e logica ho cercato di mantenere i principi SOLID e la possibilità di ampliamento del codice in futuro.

Rendicontazione ore :

Attività	Ore previste	Ore effettive
Progettazione	10	12
Sviluppo parte logica	10	9
Studio del frameworkQt	10	10
Sviluppo della gui	10	14
Test e debug	5	9
Stesura relazione	5	5
	50	60

Il monte ore è stato superato in quanto dopo lo studio della libreria di Qt il passaggio tra la parte grafica, il controller e il modello ha necessitato di diversi passaggi di verifica per vedere se ogni bottone e funzionalità era collegato correttamente . Anche la progettazione ha preso del tempo in più ma mi ha permesso di risparmiare nella progettazione logica poiché avevo sopito i miei dubbi e come dovevano essere strutturate le varie classi.