oggetto	Relazione progetto Programmazione a Oggetti
gruppo	Lapenna Francesco, mat. 2072134 francesco.lapenna.1@studenti.unipd.it https://github.com/Fraa23/SDC_proj_PAO
titolo	SDC

Introduzione

SDC, che sta per “Sistema Domotico Centralizzato”, è un’applicazione che permette la simulazione da remoto di un insieme di sensori. I sensori supportati sono sensori di temperatura, umidità e luminosità. Ognuno di essi ha un proprio modo di prendere misurazioni durante una simulazione e mostra informazioni e icone diverse.

L’utente ha la possibilità di aggiungere nuovi sensori, modificare e cancellare i sensori esistenti ed avviare una simulazione sui sensori esistenti. Nello specifico la funzione di simulazione su un sensore simula la misurazione di un dato ogni 10 minuti per 24 ore.

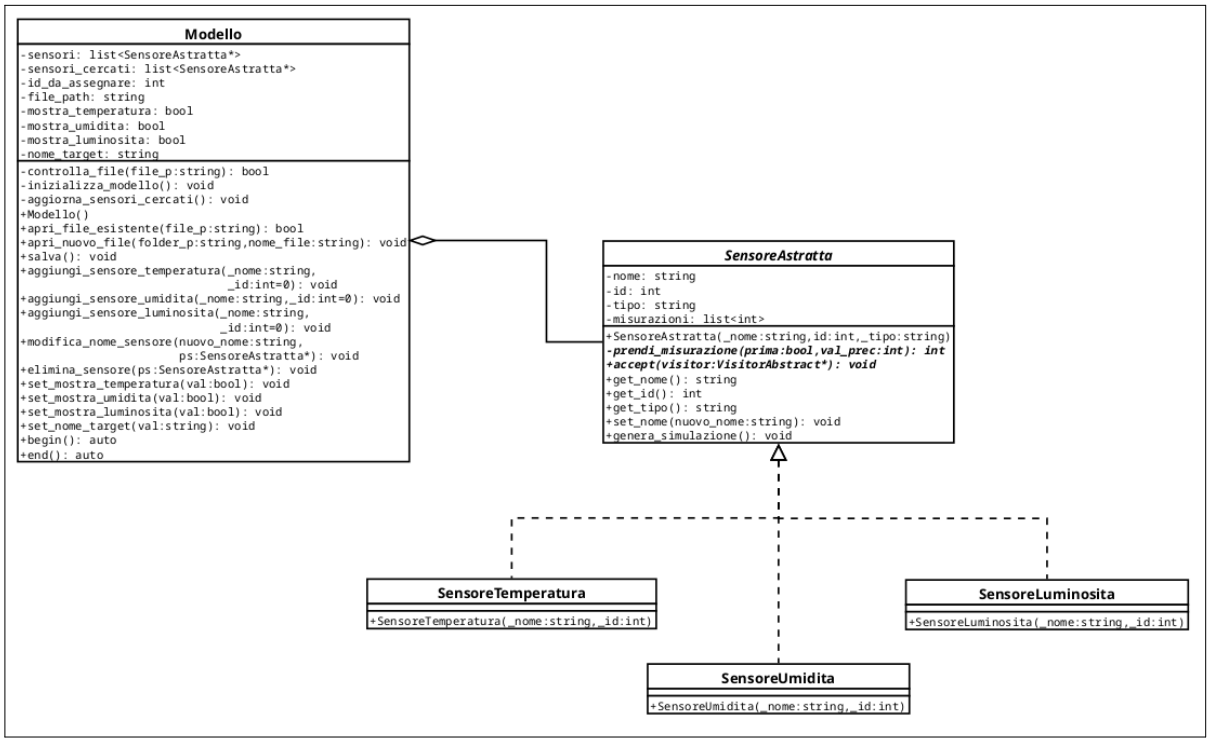
L’applicazione inoltre consente la ricerca di un sensore (in base al nome e al tipo) particolarmente efficiente dato che la lista dei sensori pertinenti viene aggiornata automaticamente senza la necessità di premere “pulsanti di ricerca”.

L’applicazione permette di creare un nuovo sistema di sensori e salvarlo per poi riusarlo in una successiva apertura dell’applicazione.

Descrizione del modello

L’applicazione adotta il design pattern Model-View, per cui la vista non comunica mai con i sensori se non attraverso il modello. Di conseguenza sarebbe possibile usare il modello anche senza usare la GUI includendo la classe “*modello.h*.”

Gerarchia di sensori e modello



La gerarchia di sensori comprende una classe astratta, in quanto contiene il metodo virtuale puro *prendi_misurazione,* da cui derivano le tre classi concrete di sensori. Queste tre classi implementano in modo diverso il metodo *prendi_misurazione* il che simula come diversi sensori nella realtà misurino i dati in modo diverso. La classe base astratta contiene le informazioni di base di un sensore (nome, id, tipo) con i relativi getter e setter e contiene una lista di misurazioni poi necessaria per la simulazione.

La classe base astratta inoltre ha un metodo *acept(visitor)* che permette di usare il design pattern del Visitor. Nello specifico è stato utilizzato per mostrare elementi grafici diversi a seconda del tipo di sensore (si veda sezione polimorfismo).

Come detto in precedenza è il modello a gestire un sistema di sensori e ad implementare tutte le funzioni dell'applicazione.

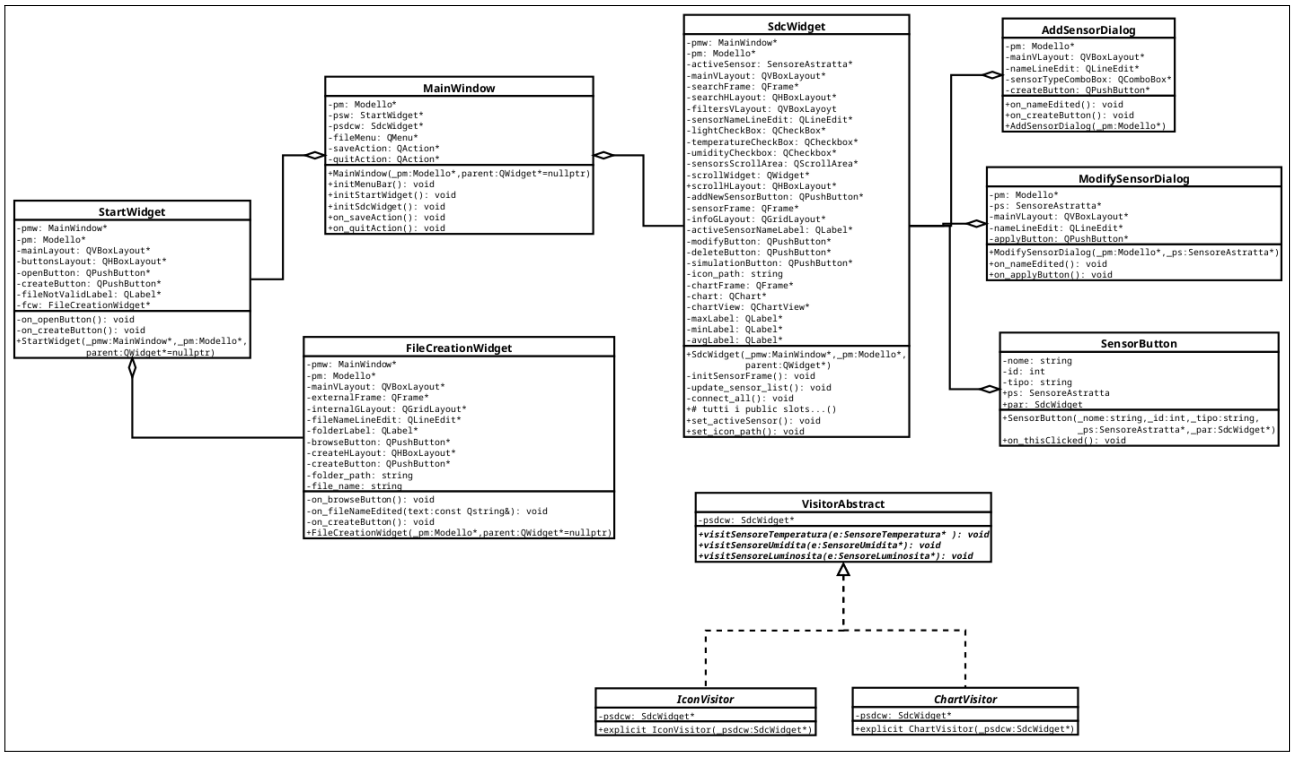
Il modello si comporta come una classe contenitore, infatti i suoi elementi principali sono due liste di puntatori a *SensoreAstratta*. La prima contiene tutti i sensori che l'utente inserisce nel sistema, invece la seconda contiene solo quelli che rispettano i vincoli di ricerca stabiliti dall'utente (se l'utente non stabilisce vincoli le due liste coincideranno).

Per prima cosa il modello deve essere inizializzato. Per farlo vengono chiamate innanzitutto le funzioni *apri_nuovo_file* oppure *apri_file_esistente* le quali prendono un percorso file, controllano la validità del file con *controlla_file* e nel caso il controllo vada a buon fine il modello viene inizializzato con *inizializza_modello*. A questo punto vengono caricati nel modello i sensori letti dal file di salvataggio.

Possono essere aggiunti nuovi sensori tramite le funzioni *aggiungi_sensore_…,* possono essere modificati i nomi dei sensori esistenti tramite *modifica_nome_sensore* e si può eliminare un sensore tramite elimina_sensore. Quando un sensore viene creato gli viene assegnato automaticamente dal sistema un identificativo. Al salvataggio ovviamente l'applicazione memorizza sia gli id assegnati che quelli ancora da assegnare in modo tale che alla riapertura dell'applicazione ogni sensore abbia sempre lo stesso id e che non si possano avere due sensori con lo stesso id.

Il sistema di ricerca permette di filtrare i sensori per tipo e per nome attraverso apposite variabili private e metodi setter. Ogni volta che viene invocata una funzione che modifica la lista di tutti i sensori, e che di conseguenza potrebbe potenzialmente modificare anche quella dei sensori cercati, viene aggiornata la lista dei sensori cercati.

Vista e Visitors



Inserisco anche il diagramma UML della vista e dei visitor nel caso in cui sia necessaria.

Spiegazione breve: la MainWindow è la schermata principale sulla quale vengono “montate” le classi “Widget” nel corso dell’utilizzo dell’applicazione. SdcWidget è il widget principale. Le due classi “Dialog” sono invece dei dialog esterni alla MainWindow.

Un oggetto della classe SensorButton rappresenta uno specifico sensore. SensorButton estende QPushButton in modo tale da contenere anche un puntatore al sensore a cui si riferisce. Di conseguenza quando viene premuto da il segnale di mostrare le informazioni relative al sensore a cui si riferisce.

Infine ho implementato due visitor che derivano dalla classe base astratta *VisitorAbstract*.

Polimorfismo

I principali utilizzi del polimorfismo nell’applicazione sono tre. In ordine crescente di rilevanza:

1. La funzione *prendi_misurazione* della gerarchia di sensori. Essa permette ad ogni sensore di prendere misurazioni in maniera diversa.
2. La classe IconVisitor che permette alla vista di mostrare un’icona diversa a seconda del tipo dinamico del sensore che visita (icona che rappresenta il tipo del sensore).
3. La classe ChartVisitor che costruisce (e mostra) un widget diverso a seconda del tipo dinamico del sensore che visita. Ad esempio per un sensore di temperatura mostra un LineChart e le informazioni su minimo e massimo. Per un sensore di umidità mostra anche l’informazione del’umidità media, mentre per un sensore di luminosità invece di un LineChart mostra un BarChart.

Persistenza dei dati

Per la persistenza dei dati vengono usati file csv. Le informazioni memorizzate sono il successivo id da assegnare e per ogni sensore inserito nel sistema vengono memorizzati nome, id e tipo. Si allega un file di esempio di persistenza dei dati (Salvataggi/file_esempio.csv)

Funzionalità implementate

- Creazione di un nuovo file di sensori
- Apertura di un file di sensori esistente
- Riconoscimento di file in formato non valido (si provi ad aprire nonvalido1.csv)
- Inserimento di un nuovo sensore
- ScrollBar contenente tutti i sensori aderenti ai filtri
- Modifica del sensore selezionato
- Eliminazione del sensore selezionato
- Simulazione sul sensore selezionato
- Icone diverse in base al tipo del sensore
- Statistiche diverse in base al tipo del sensore
- Sistema di ricerca “in tempo reale” (senza premere pulsanti cerca)
- Menu bar
- Scorciatoie per save e quit

Rendicontazione ore

Attività	Ore Previste	Ore Effettive
Studio e progettazione	10	12
Sviluppo del codice del modello	10	12
Studio del framework Qt	10	8
Sviluppo del codice della GUI	10	14
Test e debug	5	6
Stesura della relazione	5	5
totale	50	**57**