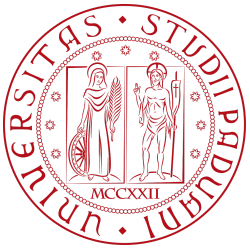
SENSOR MANAGEMENT

Progetto di Programmazione ad Oggetti
Anno accademico: 2023/2024

Nome: Giacomo Toso
Matricola: 1235002



1. Introduzione:

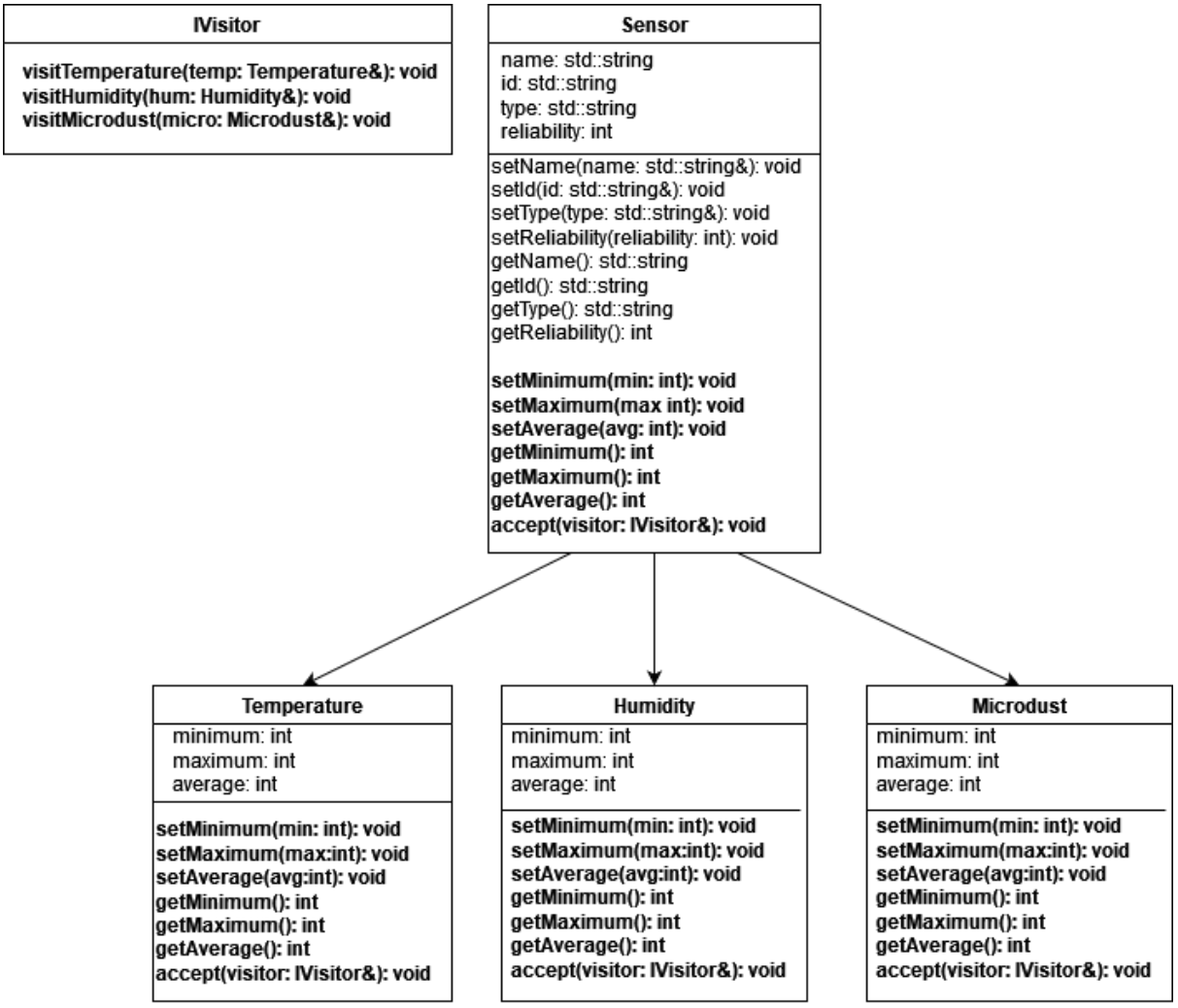
Sensor Management è un'applicazione scritta in C++ con l'utilizzo del framework Qt per la creazione e gestione di sensori atmosferici.
L'applicazione permette di creare sensori di vario tipo e di visualizzare i dati meteorologici dei vari mesi dell'anno. Questi dati riguardano temperatura, umidità e micropolveri e verranno visualizzati in un grafico. I sensori che si potranno creare saranno appunto di queste tre tipologie ed estrarranno i dati dal database. Oltre alla possibilità di creare, rimuovere e/o modificare i sensori si ha la possibilità di aggiungere manualmente dati atmosferici all'interno del database e di salvare in esso tutte le modifiche apportate. Una volta aperto il programma, si avranno tre principali schermate: quella dell'editor del database, dove si possono aggiungere e/o modificare i dati, quella del tutorial, dove si ha una breve descrizione dell'applicazione e quella del simulatore, dove si gestiscono i sensori e si visualizzano i dati, ovvero la schermata principale del programma.

2. Modello logico

2.1 Architettura del modello

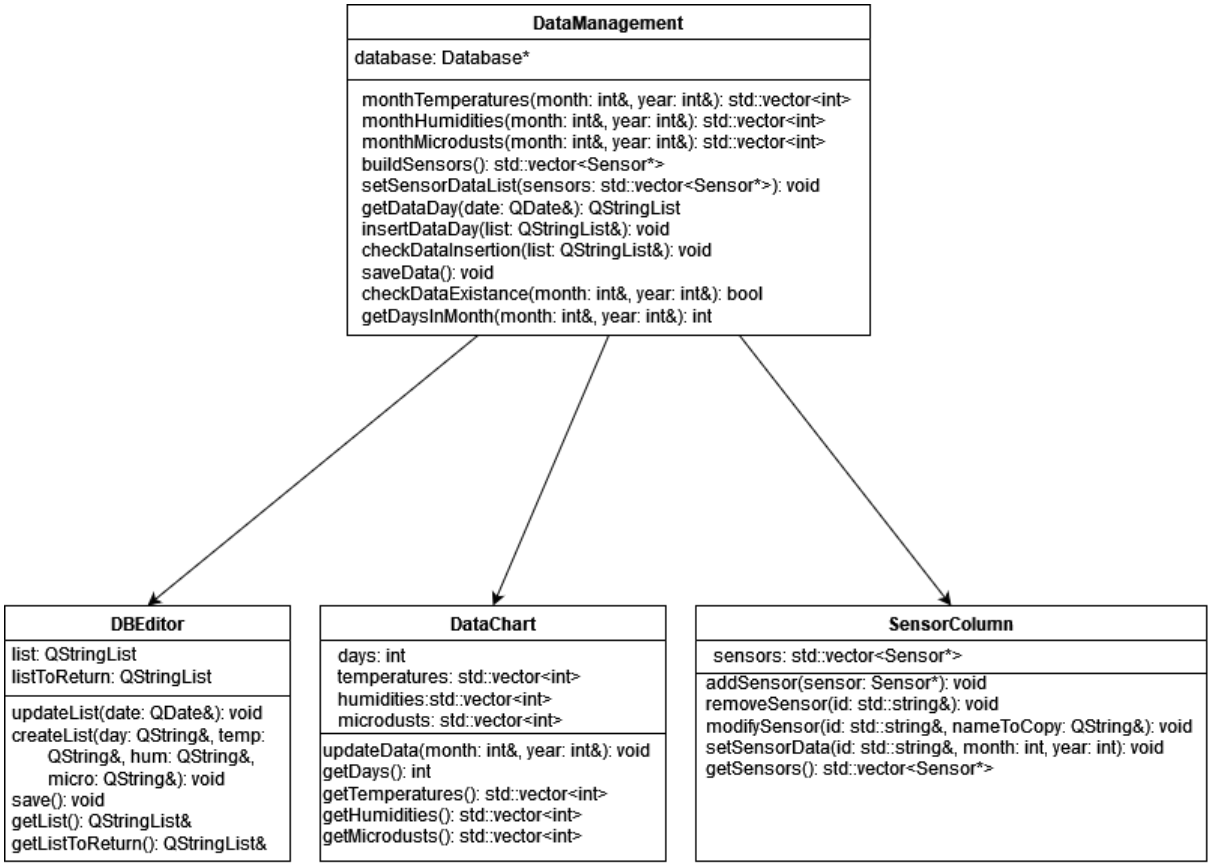
Il modello logico è diviso in due parti principali, quella riguardante il modello dei sensori e il modello delle varie schermate principali, e quella riguardante la gestione dei dati.
Esso è stato strutturato tramite due gerarchie principali, una riguardante il sensore, da cui derivano le tre tipologie esistenti, e l'altra riguardante la gestione dei dati, ovvero la classe che avrà accesso a tutti i dati presenti nel database. Da questa classe derivano tutte le classi che hanno necessità di accedere a tali dati e che fungono come base su cui costruire le schermate dell'interfaccia grafica.
Dove possibile sono stati usati alcuni tipi predefiniti di Qt come QDate o QString per evitare l'eccessivo numero di conversioni.

2.1.2 Gerarchia modello sensori



La classe astratta Sensor contiene i dati comuni a tutti i sensori, come il nome, l'id (univoco), il tipo e l'affidabilità.
I metodi virtuali puri della classe Sensor e successivamente ereditati ed implementati dalle tre sottoclassi, sono stati evidenziati dal testo in grassetto.
L'utilizzo del pattern visitor verrà spiegato più dettagliatamente nella sezione "Polimorfismo".

2.1.3 Gerarchia modello gestore dati



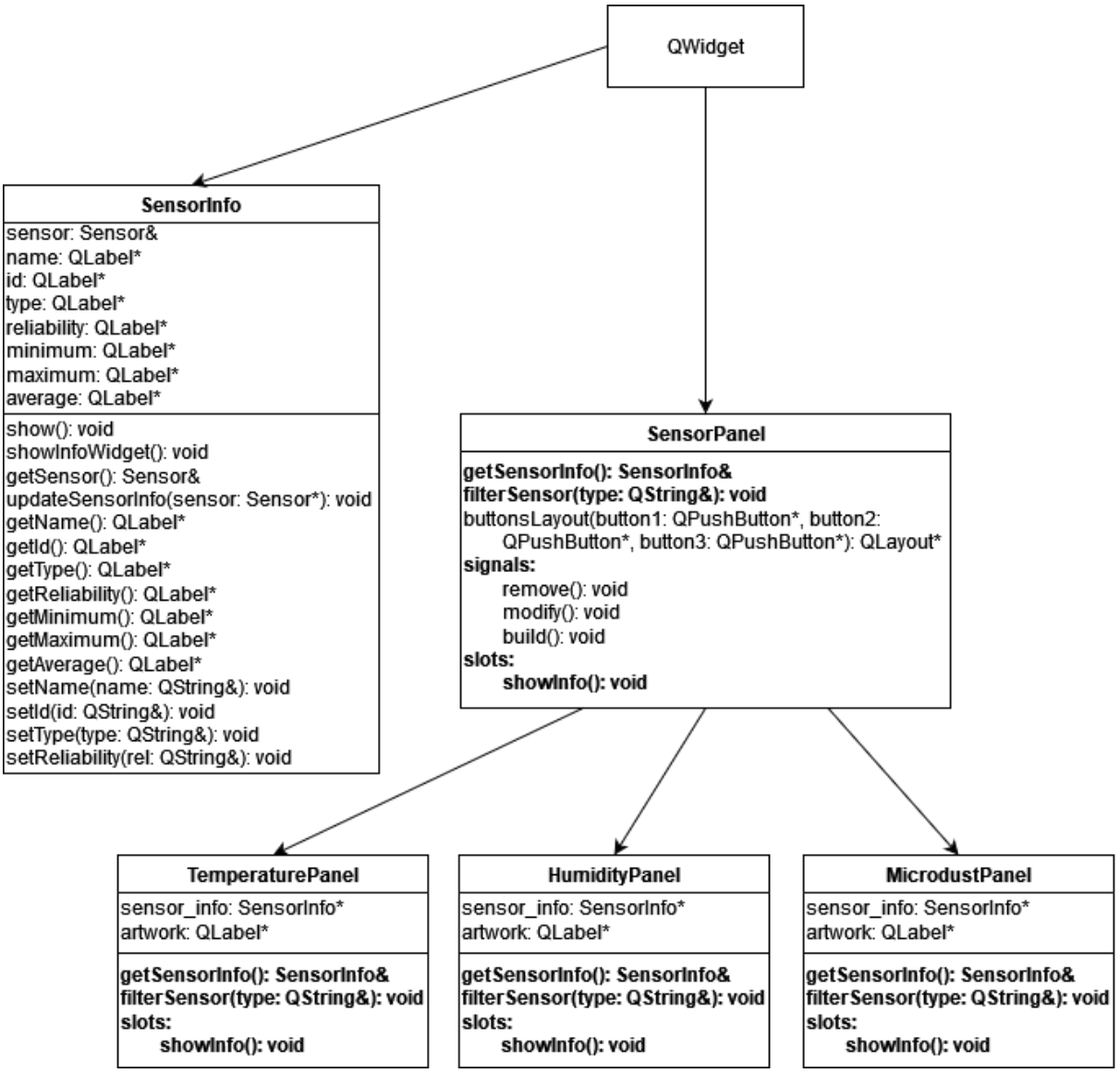
La classe DataManagement ha un accesso diretto alla classe Database, in modo da poter avere accesso ai dati e da permettere questa operazione anche alle relative sottoclassi. Per questo motivo il suo unico campo dati privato è un puntatore a Database.
Nella classe DBEditor sono stati usati alcuni dei tipi predefiniti di Qt (QDate, QString, QStringList), il tutto per evitare l'eccessivo numero di conversioni.

- La classe DBEditor fornisce tutte le funzioni necessarie alla classe DBEditorView, ovvero la classe responsabile della creazione della schermata dell'editor del Database.
- La classe DataChart fornisce tutte le funzioni necessarie alla classe DataChartView, ovvero la classe responsabile della creazione del grafico in cui si visualizzano i dati.
- La classe SensorColumn fornisce invece tutte le funzioni necessarie alla classe SensorColumnView, ovvero la classe responsabile della creazione della colonna di sensori nel simulatore.

Le tre relative sottoclassi sono usate come modelli per le relative schermate dell'applicazione.

L'implementazione e utilizzo della classe Database verrà spiegata più dettagliatamente nella sezione "Persistenza dei dati".

2.1.4 Gerarchia pannelli grafici sensori

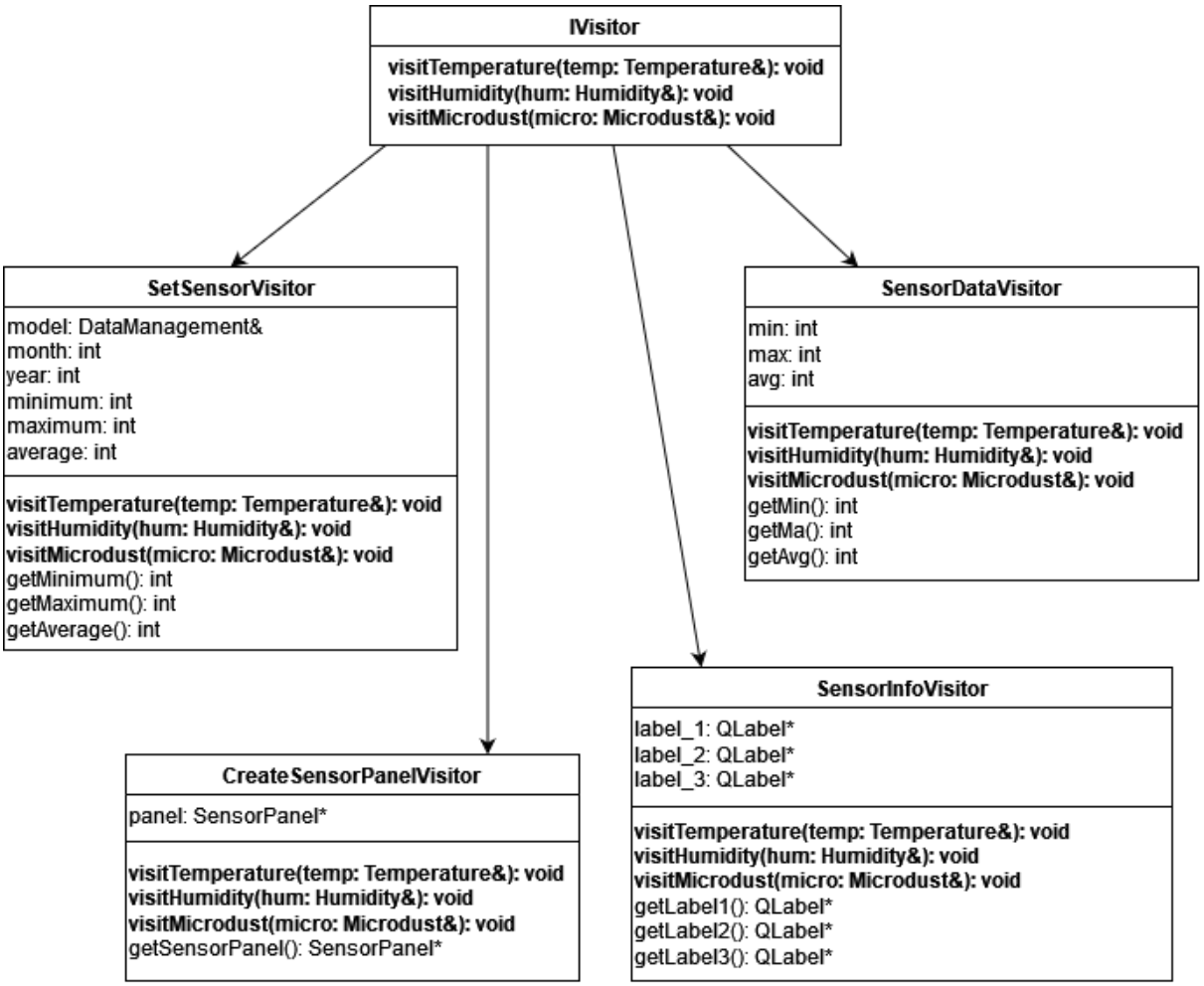


È stata utilizzata una gerarchia anche per rappresentare i pannelli dei vari sensori.
La classe astratta SensorPanel eredita dalla classe QWidget e da essa derivano le tre classi concrete per rappresentare i pannelli dei sensori.
La classe SensorInfo eredita anch'essa da QWidget ed è funzionale a rappresentare le informazioni comuni a tutti i sensori. Le classi concrete possiedono appunto un puntatore a SensorInfo e per ottenere i dati destinati alla propria tipologia di sensore è stato utilizzato il pattern visitor.
I metodi virtuali puri della classe SensorPanel ed ereditate dalle relative sottoclassi sono evidenziati dal testo in grassetto.

3. Polimorfismo

Oltre alle classi astratte Sensor e SensorPanel, alla classe DataManagement, ai rispettivi metodi virtuali e virtuali puri utilizzati e alle rispettive gerarchie implementate, il polimorfismo è stato implementato anche grazie all'utilizzo del design pattern visitor, un metodo fondamentale per lo sviluppo dell'applicazione.

3.1 Gerarchia Visitor



Sono state implementati quattro visitor concreti derivanti dalla classe astratta IVisitor, ognuno destinato ad una funzione:

- **SetSensorVisitor**: la sua funzione è quella di estrarre i dati necessari al sensore in base al tipo. Se il sensore è di tipo Temperatura dovrà estrarre dal database i valori temperatura, se è di tipo Umidità dovrà estrarre i valori umidità e se è di tipo Micropolveri i valori micropolveri.

- **SensorDataVisitor**: la sua funzione è quella di trasformare un sensore in un SensorData da aggiungere al database. Il visitor farà in modo di estrarre i dati del sensore necessari a SensorData in base al tipo.
- **SensorInfoVisitor**: la sua funzione è quella di creare le informazioni del sensore da aggiungere e visualizzare sul relativo pannello. A seconda del tipo del sensore le informazioni verranno aggiunte e visualizzate in un certo modo.
- **CreateSensorPanelVisitor**: la sua funzione è quella di creare il pannello per un dato sensore da aggiungere alla colonna di sensori. Il pannello verrà raffigurato in un certo modo in base al tipo di quest'ultimo.

4. Persistenza dei dati

Per la persistenza dei dati sono stati utilizzati due file .csv, uno contenente i dati atmosferici (temperatura, umidità, micropolveri) per ogni giorno dell'anno e uno contenente i sensori.

4.1 Classi Data, SensorData e Database

- **Data**: è la classe destinata a memorizzare i dati atmosferici di ogni giorno, i suoi campi dati sono la data, la temperatura, l'umidità e le micropolveri.
- **SensorData**: è la classe destinata a memorizzare i dati di ogni sensore presente nel sistema, i suoi campi dati sono quelli della classe Sensor e quelli dei sensori concreti che ereditano da esso.
- **Database**: è la classe destinata a memorizzare ogni Data e SensorData presente nel sistema, i suoi campi dati sono un std::list<Data*> e un std::list<SensorData*>.
  La classe Database presenta vari metodi per la lettura dei file .csv e per l'aggiunta e rimozione di dati Data e SensorData in questi ultimi.

5. Funzionalità implementate

Suddividiamo le funzionalità implementate in due categorie, funzionali e grafiche.

5.1 Funzionali

- conversione e salvataggio dei dati in formato .csv;
- gestione di tre tipologie di sensori;
- utilizzo di un filtro per la ricerca di ogni tipologia di sensore.

5.2 Grafiche

- utilizzo della topbar in alto con pulsante per tornare alla home;
- utilizzo di icone per i pulsanti;
- utilizzo di colori e stili grafici;
- utilizzo di QMessageBox per segnalare azioni effettuate;
- utilizzo del ToolTip per segnalare l'azione di un pulsante;
- possibilità di navigare attraverso le schermate tramite interfaccia;
- pannelli personalizzati per ogni tipologia di sensore;
- utilizzo di immagini e pulsanti nella visualizzazione dei sensori.

6. Rendicontazione ore

Attività	Ore previste	Ore Effettive
Studio e progettazione	10	10
Sviluppo del codice del modello	10	13
Studio del framework Qt	10	9
Sviluppo del codice della GUI	10	15
Test e debug	5	9
Stesura della relazione	5	5
Totale	50	61

7. Note

Nel simulatore sono già stati aggiunti dei sensori, tre con i dati già inseriti e tre con i dati da settare (impostati a 0).
I sensori da testare sono nominati con nome_to_test e per settarli bisogna premere il pulsante “setta” sul pannello.
I dati dei sensori potranno essere visualizzati premendo il pulsante “info”.