Relazione progetto Programmazione ad Oggetti

Progetto svolto singolarmente da Squarzoni Matteo, mat. 2068240
Titolo: OOP-Sensors

Introduzione

Il progetto sviluppato mette a disposizione un'applicazione software in grado di creare, modificare, cancellare e visualizzare dati di sensori IoT. I sensori fruibili all'interno dell'applicazione hanno tre tipologie: sensori di movimento, sensori di luminosità e sensori di temperatura. Uno dei punti principali dell'applicativo è quello di mettere a disposizione un pannello dove è possibile generare delle simulazioni di misurazioni dei vari sensori visibili tramite appositi grafici. La simulazione avviene staticamente ed è in grado di generare un grafico, a seconda dei valori inseriti dall'utente per descrivere i vari sensori. Nello specifico i sensori di movimento offrono un numero di movimenti rilevati, quelli di luminosità offrono misurazioni in lux ed infine quelli di temperatura in ℃. Le simulazioni per le tre tipologie di sensori offrono 24 valori randomici (concettualmente uno ad ogni ora del giorno), in particolare per i sensori di movimento vengono misurati il numero di movimenti rilevati nell'arco di ogni singola ora. Tutti i dati come detto precedentemente sono generati randomicamente, a solo scopo simulativo, per la generazione di un grafico per la visualizzazione di tali dati.

Descrizione del modello

Il modello logico è relativamente semplice, è presente una classe base astratta *AbstractSensor* da cui derivano le classi *TemperatureSensor, MotionSensor e LuminositySensor* come è possibile vedere nella figura sottostante che descrive parte fondamentale del modello logico (è consigliabile visionare la figura prima di continuare a leggere).
Nella figura sono assenti i vari metodi getter per ricavare i dati degli oggetti. Nella classe *AbstractSensor* sono presenti le generalità comuni ad ogni sensore come il nome, l'identificatore (può essere una combinazione sia di numeri che di lettere), il brand produttore, se il sensore supporta o meno una possibile smart app, se il sensore è indoor o outdoor. Inoltre è presente il metodo virtuale puro *generateRandomHistory* che genera una serie di numeri random per popolare il vettore *history* (storico misurazioni), il metodo è virtuale in quanto utilizza membri specifici delle tre classi concrete. Viene poi implementato l'overloading dell'operatore di uguaglianza dove nel caso di questo progetto si impone che due sensori siano uguali se i loro nomi sono uguali. C'è inoltre un metodo *modifyData* che serve per modificare i dati nel sensore ed il metodo viene ridefinito per ogni classe concreta.
Nelle tre classi concrete *TemperatureSensor, MotionSensor e LuminositySensor* viene dichiarato come membro private il vettore *history*, quest'ultimo non viene ereditato dato che non è detto che ogni vettore *history* debba contenere per forza dei tipi int per tutti i sensori come nel caso dei tre tipi di sensori svilup-

pati, il tipo int utilizzato in questo caso è da intendere come approssimazione ad un numero intero per la precisione delle varie misurazioni anche allo scopo della generazione dei grafici. In maniera più specifica negli oggetti di tipo *TemperatureSensor* sono presenti campi dati per identificare temperatura minima e massima misurabile, negli oggetti di tipo *LuminositySensor* sono presenti campi dati per identificare luminosità minima e massima misurabile, negli oggetti di tipo *MotionSensor* sono presenti campi dati per identificare la sensibilità (rappresentata da un numero) e il raggio di rilevamento del sensore, questi ultimi due campi dati non incidono sulla simulazione di misurazioni di un *MotionSensor*.
Nella struttura del modello è presente anche un *SensorContainer*, ossia un contenitore di sensori, che permette di gestire gli inserimenti, modifiche, cancellazioni e ricerche per nome, il tutto sotto un punto di vista logico. All'interno di un oggetto *SensorContainer* non ci potranno essere due sensori con lo stesso nome, l'utente dunque sarà costretto ad utilizzare nomi diversi per ogni sensore inserito.
Si è scelto poi di implementare il *design pattern Visitor* che porta ad avere il metodo virtuale puro *accept* in *AbstractSensor*, e il *design pattern Observer* per il quale nella classe *AbstractSensor* è presente un vettore di *SensorObserverInterface**.
Sono presenti poi altre classi per la gestione della persistenza dei dati dove un file Json viene rappresentato dalla classe *JsonFile* che si avvale di oggetti di tipo *Reader* e *SensorJsonVisitor* per la lettura e la scrittura all'interno di file Json.

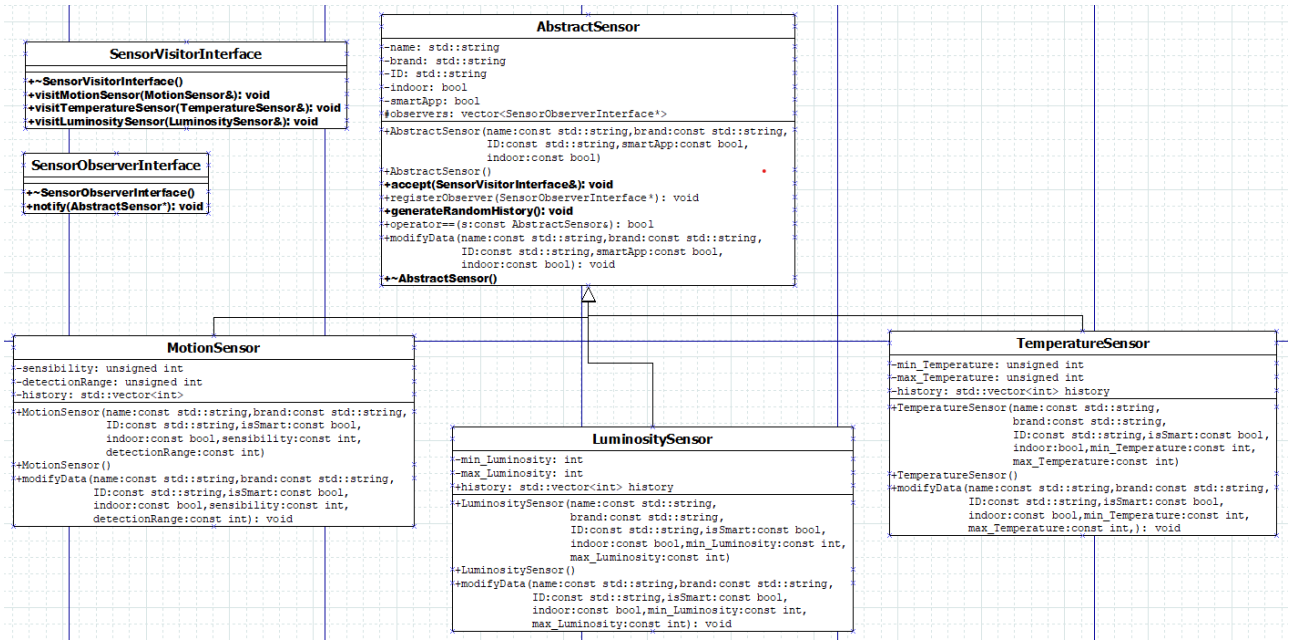

Figure 1: Modello logico

Viene proposto anche una breve idea riguardo la struttura del modello della GUI, interamente realizzata in Qt, nella figura successiva. I grafici sono stati realizzati tramite QtCharts e sono inoltre state implementate una finestra di dialogo per l'inserimento e una per la modifica.

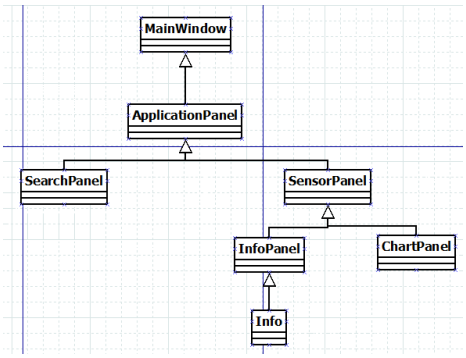

Figure 2: Modello GUI

Polimorfismo

L'utilizzo principale del polimorfismo riguarda il *design pattern Visitor* nella gerarchia *AbstractSensor*, viene utilizzato in particolare per la parte della GUI per la costruzione dei widget (ad esempio il pannello info, il grafico della simulazione e la finestra di modifica) grazie alle classi *SensorInfoVisitor*, *SensorChartVisitor* e *ModifyWindowVisitor*; è stato utilizzato anche per il caso della persistenza dei dati grazie alla classe *SensorJsonVisitor*. In generale è necessario svolgere operazioni diverse in base alla tipologia di sensore, dunque l'uso del visitor facilita la struttura del codice rendendolo più leggibile e sostenibile per eventuali modifiche. Inoltre è stato utilizzato il *design pattern Observer*, grazie al metodo virtuale puro *notify* infatti è possibile notificare il cambiamento di alcuni dati ed è possibile visualizzare questi cambiamenti anche nella parte grafica, in particolare per il pannello info e quello del grafico.

Persistenza dei dati

Per la persistenza dei dati come anticipato è stato utilizzato il formato JSON. In particolare sono state sfruttate le classi messe a disposizione dal framework Qt, all'interno del file viene salvato un vettore di sensori, inoltre viene aggiunto un attributo "type" ad ogni sensore salvato nel file JSON per mantenere traccia del tipo dell'oggetto, l'attributo "type" è necessario e di fondamentale importanza quando si decide di aprire un file per la costruzione degli oggetti contenuti in quest'ultimo. Viene proposto un piccolo dataset "esempio.json" dove sono presenti alcuni sensori sulla quale poter iniziare a fare delle operazioni.

Funzionalità implementante

Le funzionalità implementate vengono suddivise in due categorie: funzionali ed estetiche.
Partendo da quelle funzionali:

- gestione di tre tipologie di sensori
- persistenza dei dati (apri e salva) in formato JSON
- ricerca per nome per substring e case-insensitive
- pulsante di refresh per azzerare la ricerca
- gestione contenitore di sensori

Mentre quelle estetiche sono:

- gestione dei widget rendendoli disponibili all'interazione solo quando possibile
- inserimento di icone
- utilizzo di una toolbar come barra dei menu in alto
- parziale gestione del ridimensionamento per la *MainWindow*
- alcuni sensori hanno grafici diversi per la simulazione rispetto ad altri
- messaggi di errore tramite apposite finestre in caso di inserimento di dati errati
- scorciatoie da tastiera per apri (ctrl+o), salva (ctrl+s), chiudi (ctrl+q)
- opportune finestre di dialogo per inserimento e modifica

Ambiente di sviluppo

Il progetto è stato completamento sviluppato sulla macchina virtuale ufficiale del corso, dalle prove svolte compila senza errori o warning. Come editor è stato utilizzato vscode oltre al classico terminale di Linux per la compilazione tramite gli appositi comandi di qmake, tutto il codice relativo alla parte della GUI dunque è stato scritto manualmente. Per la compilazione una volta scaricata la cartella del progetto è necessario prima immettere sul terminale il comando *qmake*, successivamente il comando *make* ed infine *./OOP-Sensors*.

Rendicontazione ore

Attività	**Ore Previste**	**Ore Effettive**
Studio e progettazione	10	10
Sviluppo del codice del modello	10	11
Studio del framework Qt	8	10
Sviluppo del codice della GUI	12	15
Test e Debug	5	9
Stesura della relazione	5	6
Totale	**50**	**61**

Il monte ore previsto era inizialmente di 50 ore, tuttavia il progetto ha richiesto più ore di quelle previste, dunque il monte ore è stato di poco superato (indicativamente intorno alle 60 ore), a causa dell'inesperieza per quanto riguarda il framework Qt, inoltre le fasi di testing e di debugging per eliminare la presenza di errori legati alla parte grafica ed altri errori legati all'utilizzo della struttura *SensorContainer* si sono protratte per svariate ore aggiuntive rispetto a quelle inizialmente pattuite.