Documento di specifiche

Klaudio Merja

Mat. 2075538

https://github.com/klamerja/SensorFlowUNIPD

Indice

SensorFlow

Progetto in itinere di Programmazione ad Oggetti
LT in Informatica
Università degli Studi di Padova

1 Introduzione

SensorFlow è un software di gestione per sensori in ambito domotico. Ogni sensore è identificato tramite un UUID ed è caratterizzato da un nome, dalla tipologia e dalla distribuzione dei dati generati.
Le tipologie di sensori per cui l'applicazione fornisce supporto sono:

- **Temperatura e umidità**: permette di analizzare la temperatura (in °C) e l'umidità (in percentuale)
- **Pressione atmosferica** (in hPa - ettopascal)
- **Elettricità**: permette di analizzare il consumo istantaneo (in W - watt) e la tensione elettrica (in V - volt)
- **Qualità dell'aria**: permette di analizzare i livelli di CO2 (in ppm - parti per milione), il PM2.5 ed il PM10 (in $\mu g/m^3$)

Le operazioni principali che l'applicazione permette di svolgere sono:

- aggiunta/rimozione dei sensori
- modifica delle informazioni relative ai singoli sensori
- visualizzazione dei dati generati

Una delle caratteristiche fondamentali del software è quella di poter visualizzare i dati generati dal sensore in tempo reale.
I dati, per fornire una simulazione del sensore, sono generati secondo una tipologia di distribuzione tra le seguenti:

- Casuale
- Uniforme
- Gaussiana

L'utente ha la possibilità di decidere quale distribuzione adottare per ogni singolo sensore e di modificarla in un secondo momento.

2 Descrizione del modello

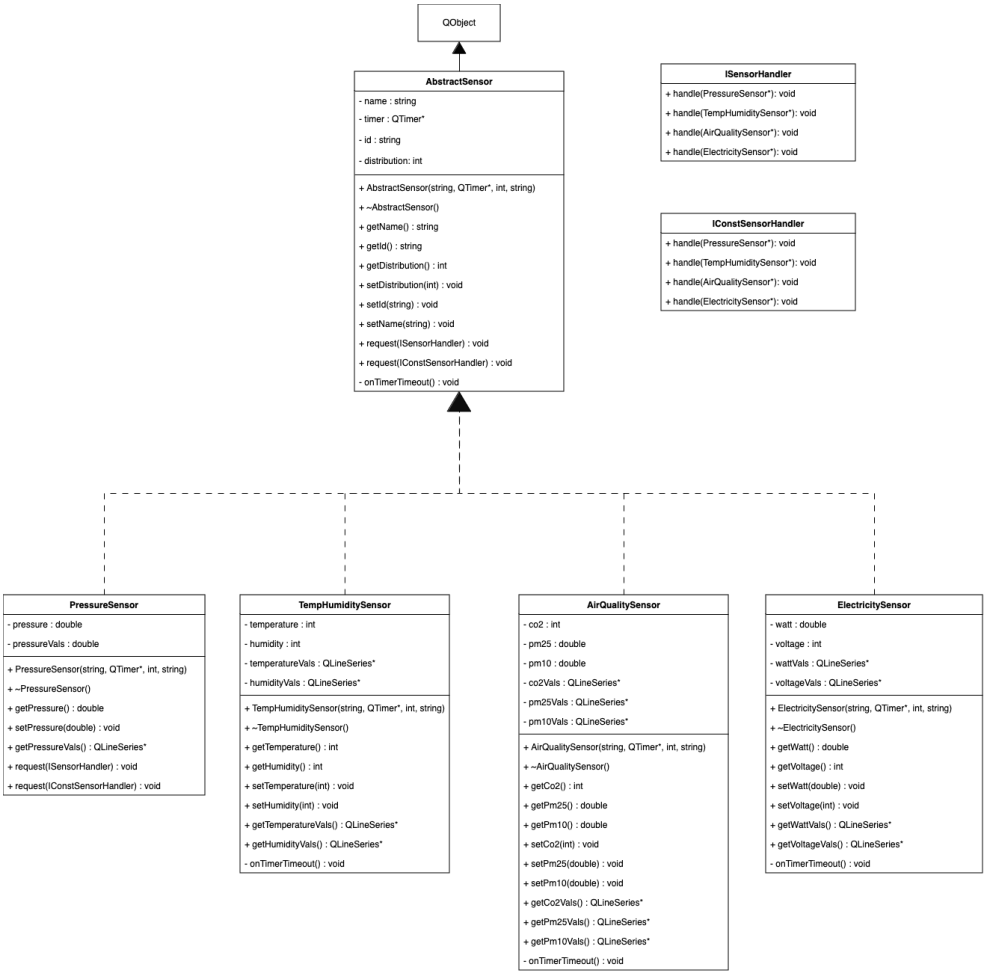

Figura 1: Diagramma delle classi dei sensori

Il modello logico è strutturato in due parti: la prima parte comprende le classi che descrivono i vari sensori utilizzabili all'interno dell'applicativo, mentre la seconda parte si occupa di creazione, lettura e aggiornamento del file JSON che si occupa del salvataggio dei sensori. `AbstractSensor` è la classe base astratta che rappresenta le informazioni comuni a tutti i sensori che possono essere creati, ovvero il nome, il timer, l'identificatore univoco UUID e la tipologia di distribuzione. Oltre ai relativi metodi *getter* e *setter* per le varie variabili d'istanza, è presente il metodo `onTimerTimeout`, che si occupa di effettuare delle azioni ad ogni *timeout* emesso dal timer: solitamente, le azioni che vengono performate sono quelle di aggiornamento dei valori dei dati che vengono generati dal sensore, oltre a cancellare valori dalle eventuali serie che risultano non necessarie ai fini del funzionamento del software e, in particolare, per la generazione del grafico. Le classi figlie di `AbstractSensor` sono:

- `PressureSensor`: descrive il sensore della pressione atmosferica. In particolare, ne indica il valore istantaneo della pressione stessa e gli ultimi venti

valori necessari alla generazione del grafico tramite una `QLineSeries`;

- `TempHumidity`: descrive il sensore della temperatura e dell'umidità tramite i valori degli stessi e gli ultimi venti valori necessari alla generazione dei relativi grafici (sempre tramite una `QLineSeries`);
- `AirQualitySensor`: descrive il sensore della qualità dell'aria. I parametri che vengono monitorati e di cui vengono riportati i relativi venti valori per la generazione del grafico sono:
  - CO_2
  - PM2.5
  - PM10
- `ElectricitySensor`: descrive il sensore dell'elettricità, che monitora il consumo istantaneo di energia in Watt e il voltaggio della rete elettrica. Anche per questi ultimi, sono presenti gli ultimi venti valori contenuti all'interno di una `QLineSeries`

3 Polimorfismo

L'utilizzo principale del polimorfismo riguarda il *design pattern* `Visitor` nella gerarchia `AbstractSensor`. Esso viene utilizzato per:

- per recuperare le `QLineSeries` dai singoli sensori, necessari per la generazione dei vari grafici relativi ai sensori
- per riconoscere la tipologia di sensore

Le classi che permettono l'esecuzione delle funzionalità sopra elencate sono:

- `JSONhandler`: per il salvataggio dei sensori viene utilizzato un vector di `AbstractSensor`, da cui bisognerà ricavare il tipo di sensore che andrà salvato all'interno del JSON;
- `DataPanel`: la seguente classe riceve in input un `AbstractSensor`; a seconda della tipologia di sensore, l'applicativo dovrà generare un certo numero di grafici di un determinato tipo. Risulta quindi necessario il polimorfismo per modellare la generazione dei grafici a seconda della tipologia di sensore;
- `ItemCard`: la seguente classe si occupa della generazione delle varie card che rappresentano i singoli sensori. Ogni card è caratterizzata dal nome del sensore, dai bottoni di eliminazione e modifica, ma soprattutto dalla tipologia di sensore che rappresenta la card. Per ottenere questa informazione, risulta necessario l'utilizzo del polimorfismo.

Le classi sopra elencate, in quanto effettuano soltanto operazioni di lettura e non di modifica, implementano tutte un `IConstSensorHandler`, che svolge le funzioni sopra indicate a seconda del tipo concreto del sensore (partendo quindi da un `AbstractSensor`).

4 Persistenza dei dati

È necessario avere dei dati persistenti per quanto riguarda tutti i sensori che un utente vuole utilizzare e gestire. Per la persistenza dei dati dei sensori viene quindi utilizzato il formato JSON, caratterizzato da un oggetto contente un array di oggetti `sensors`. Ogni oggetto presente all'interno dell'array `sensors` rappresenta i dati fondamentali di un singolo sensore:

- **id**: identificativo univoco del sensore
- **name**: nome del sensore
- **type**: tipologia del sensore
- **distribution**: tipologia di distribuzione dei dati del sensore

Si riporta un esempio di file JSON `test.json` nella cartella radice del progetto, contenente un sensore di ogni tipo e di ogni tipologia di distribuzione dei dati per illustrare velocemente il funzionamento del programma.

5 Funzionalità implementate

Le funzionalità implementate all'interno del programma sono:

- Funzionali
  - creazione, gestione e modifica di quattro tipologie di sensori
  - quattro tipologie di distribuzione dei dati
  - funzionalità di ricerca mediante RegEx
  - salvataggio dei sensori in formato JSON
  - shortcut da tastiera
    - CTRL+N per creare un nuovo file JSON
    - CTRL+O per aprire un file JSON
    - CTRL+S per salvare le modifiche nel file JSON
    - CTRL+T per creare un nuovo sensore
    - CTRL+SHIFT+Backspace per eliminare un sensore che ha il focus
- Estetiche
  - barra dei menù superiore per gestire le funzionalità relative al file JSON e per eliminare o aggiungere un sensore

Figura 2: MenuBar in MacOS

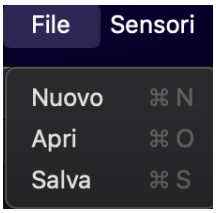

Figura 3: Menu di gestione del file JSON

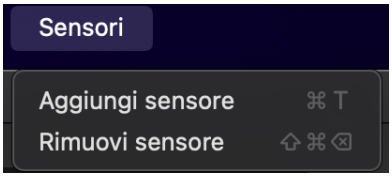

Figura 4: Menu di gestione del sensore

– Presenza di frecce direzionali per cambiare grafico nel caso in cui fossero più di uno

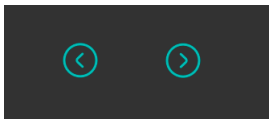

Figura 5: Freccie direzionali

– bottoni clickabili per ogni sensore per effettuare modifica o eliminazione sul singolo

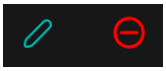

Figura 6: Item Card

– cambio del colore del bordo tramite hover del mouse

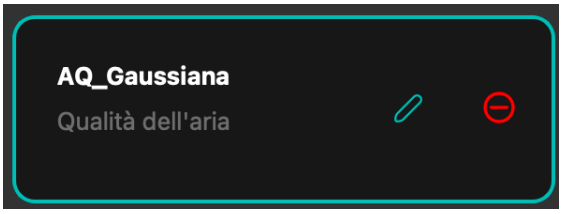

Figura 7: Item Card

6 Rendicontazione ore

Attività	**Ore Previste**	**Ore effettive**
Studio e progettazione grafica	5	6
Studio del framework Qt	15	18
Sviluppo del codice del modello	8	9
Sviluppo del codice della GUI	20	22
Test e debug	5	8
Stesura della relazione	5	5
Totale	58	**68**

Il monte ore è stato superato di 10 ore circa a causa di una serie di problemi; una delle attività che ha subito più ritardo nella tabella di marcia è stato sicuramente lo studio del framework Qt in quanto sottostimato come tempo e consuntivato prendendo spunto dalle stime orarie fornite dal professore. L'attività, ovviamente, comprende anche delle ore di sviluppo codice del software, utilizzato per prendere confidenza con il framework e ridurre il tempo sprecato. Un altro evento che ha richiesto più tempo del dovuto è stato sicuramente la fase di Test e debug; i principali problemi riscontrati durante lo sviluppo che hanno richiesto molto tempo per il debug sono:

- **conflitti di focus**: il programma crashava a seguito della perdita di focus del sensore, che causava l'eliminazione del data panel (come da progetto), che però aveva appena acquisito il focus;
- **perdita di focus dovuta ai popup menu**: data la poca esperienza con il framework, uno dei problemi che ha richiesto del tempo nella fase di test e debug è quella dei popup menu del QMenu all'interno della QMenuBar. Questi, nonostante la focus policy fosse impostata a `Qt::NoFocus`, il popup menu acquisiva comunque il focus, non permettendo così l'eliminazione tramite menù.