Programmazione ad Oggetti 2023/2024

Relazione progetto

Relazione scritta da

Gusatto Derek - 2042330

in gruppo con

Tecchiati Veronica - 2034309

Indice

1 Introduzione

AsQlepio rappresenta un avanzato sistema di gestione dei dati provenienti da sensori medici, focalizzato sulla monitoraggio accurato di parametri vitali come la pressione del sangue, il battito cardiaco, la frequenza respiratoria e la temperatura corporea. Il nostro progetto si propone di offrire una soluzione completa e intuitiva per la raccolta, l'analisi e la visualizzazione di dati critici per la salute, mirando a migliorare l'efficienza e la precisione nella gestione delle informazioni mediche.

A differenza di approcci convenzionali, AsQlepio si distingue per la sua interfaccia user-friendly e per la capacità di integrare diverse tipologie di sensori in un'unica piattaforma. L'obiettivo principale è fornire agli operatori sanitari e ai pazienti un ambiente centralizzato per il monitoraggio in tempo reale e la registrazione storica di parametri vitali cruciali.

2 Descrizione del modello

Il modello prevede una gerachia, come riportato nella Figura1, incentrata sulla base astratta `Sensor` che rappresenta un sensore generico, con attributi standard (identificatore, nome, affidabilità), un range di misurazione, come definito dalla classe `Range` e una distribuzione della misurazione. Questa classe è astratta, poiché definisce un metodo virtuale puro *`isHealthy`* che deve essere implementato dalle classi derivate. Per quanto riguarda la distribuzione della misurazione è stato implementato uno strategy pattern, in modo che ogni classe concreta derivata da `Sensor` debba applicare una distribuzione concreta (normale, uniforme, sinusoidale). Dalla classe `Sensor` deriva la classe `HealthyRange` aggiunge un ulteriore range per definire un intervallo sano. Questa classe viene estesa dalle classi `OxygenSaturation`, `HeartRate`, `BreathFrequency`, e `Temperature` che aggiungono dettagli specifici per ciascun tipo di sensore. L'unica classe rappresentante un sensore che deriva direttamente da `Sensor` è `BloodPressure` poichè essa contiene due range ed oltre ad implementare il medoto virtuale aggiunge il metodo

isHealthyDiastolic per controllare il secondo parametro di questo tipo di sensore. `HeartRate` e `BreathFrequency` sono classi foglie della gerachia derivante da `Sensor`, ma estendono anche la classe `StressMesurable` che ha lo scopo di rappresentare sensori che possono essere misurare sia sotto sforzo che a riposo, offrendo un attributo booleano per indicare in quale contesto la misurazione è effettuta.

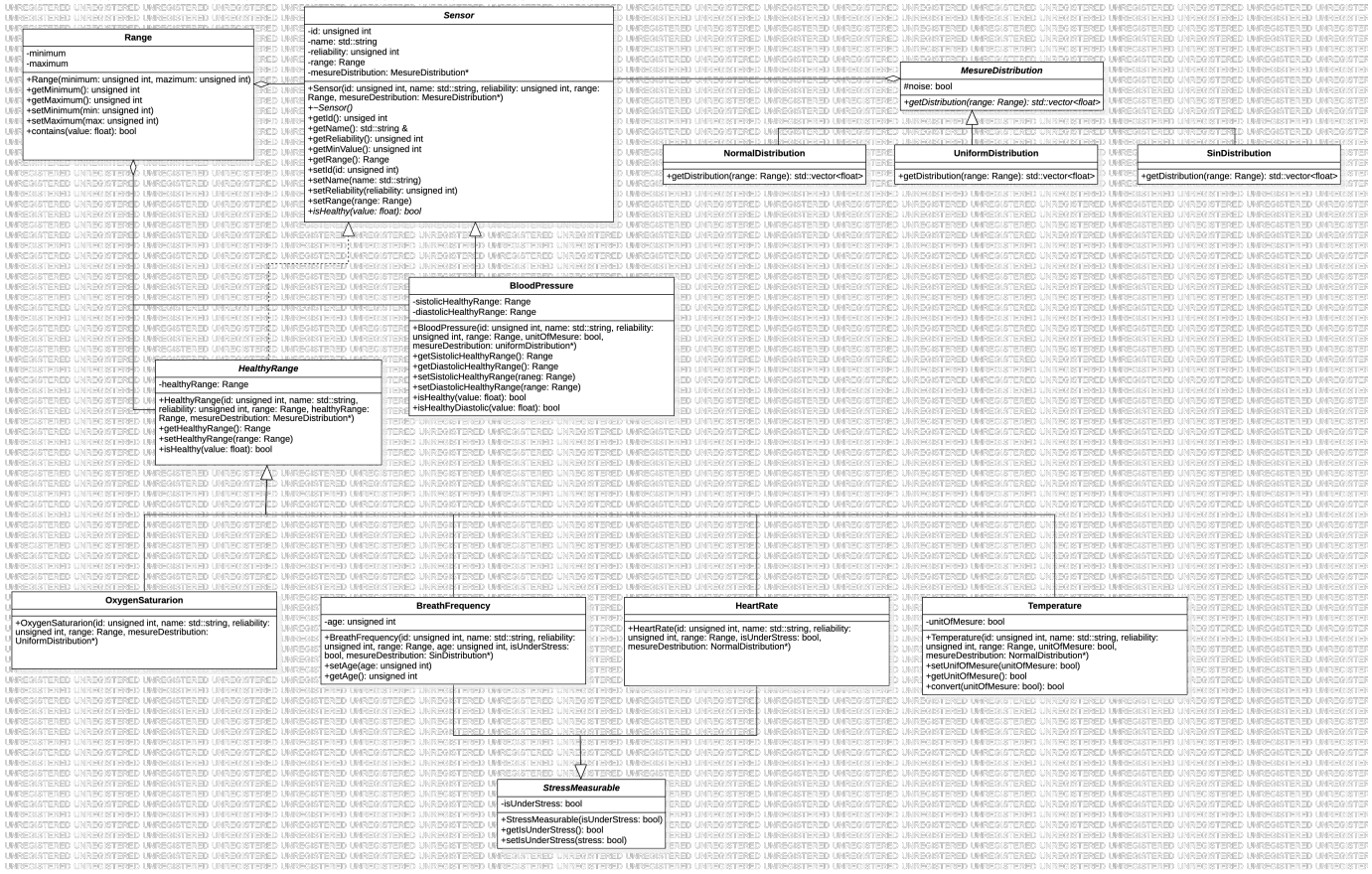

Figure 1: Figura 1: Diagramma UML - modello delle classi

Tutte le classi sono inoltre provviste di metodi *`QJsonObject toJson() const`* e *`static fromJson(QJsonObject json)`* per garantice il salvataggio e il cari-

camento dei dati salvati in locale in formato JSON, tramite le classi `SensorSaver` e `SensorLoader`.

3 Polimorfismo

L'utilizzo più banale dei polimorfismo è nella gerarchia principale dei sensori, dove `Sensor` è classe base astratta e le classi che la estendono devono implementare i metodi virtuali che essa definisce. Un altro esempio molto semplice di polimorfismo è l'ereditarietà e la ridefinizione di metodi nelle classi derivate, per esempio *`QJsonObject toJson() const`*.

Un esempio di polimorfismo non banale è invece lo *Strategy Pattern* implementato sulla distribuzione della misurazione che consente di incapsulare diversi algoritmi di generazione di distribuzioni in classi separate, rendendoli intercambiabili e facilitando l'estensione del sistema senza dover modificare le classi esistenti. In questo modo, ogni classe generatrice di una distribuzione implementa l'interfaccia `MesureDistribution` e fornisce la propria logica di generazione della distribuzione. Ogni sensore nel sistema può avere una strategia di distribuzione associata, per esempio per `HeartRate` si è scelto una `NormalDistribution`. L'utilizzo del design pattern *Strategy* in questo contesto facilita l'estensione del sistema, consentendo l'aggiunta di nuovi algoritmi di generazione delle distribuzioni senza dover modificare le classi esistenti.

Un altro utilizzo polimorfismo si ha con il design pattern *Visitor* nella gerarchia `Sensor`. Esso viene utilizzato per la costruzione dei widget per mostrare le diverse tipologie di sensori, inserendo i campi specifici di ciascun tipo di sensore, oltre che per l'impostazione dell'icona e per il salvataggio delle modifiche.

4 Persistenza dei dati

Per la persistenza dei dati viene utilizzato il formato JSON con un unico file per tutti i sensori. Tale file contiene un vettore di oggetti, perlopiù salvati semplicemente con coppie chiave-valore. La serializzazione delle sottoclassi prevede l'aggiunta di un attributo *type*. Un esempio di file di salvataggio JSON, fornito assieme al codice, è *sensors.txt* che contiene un sensore di ogni tipo e ne illustra le strutture.

5 Funzionalità implementate

Dal punto di vista funzionale sono state implementate:

- gestione di cinque tipologie di sensori;
- creazione e modifica dei sensori;
- visualizzazione della simulazione di un sensore;
- conversione e salvataggio in formato JSON;
- funzionalità di ricerca con filtro.

Mentre dal punto di vista della GUI sono state implementate:

- barra dei menù, in alto;
- salvataggio prima della chiusura;
- utilizzo di immagini e pulsanti nella visualizzazione degli oggetti;
- scorciatoie da tastiera (mostrate anche nelle voci del menù);
- diversa visualizzazione per ogni tipologia di sensore.

Nel corso dello sviluppo, sono state identificate numerose opportunità per migliorare ulteriormente l'interfaccia grafica. Tuttavia, a causa delle restrizioni di tempo imposte dal progetto, ci si è concentrati sulla realizzazione delle funzionalità essenziali.

6 Ambiente di sviluppo ed esecuzione

Il progetto è stato realizzato e testato in ambiente **Linux**:

- Sistema Operativo: *Ubuntu 22.04 LTS*
- Compilatore: *gcc 11.3.0*
- Framework Qt: *Qt 6.2.4*
- Strumento di build: *qmake 3.1*

6.1 Esecuzione

Per eseguire il progetto è sufficiente eseguire i seguenti comandi:

```
$ qmake AsQlepio.pro
$ make
$ ./AsQlepio
```

7 Rendicontazione ore

7.1 Divisione del lavoro

L'analisi dei requisiti e la progettazione sono state svolte assieme, nella fase di realizzazione poi il lavoro è stato suddiviso come segue:

- Gusatto Derek:
  - Modello: sviluppo della gerarchia di sensori
  - View: Grafico di simulazione
  - JSON
  - Controller: collegamento Model e View
  - Bug fixing GUI e funzioni
- Tecchiati Veronica:
  - Modell: Visitors
  - View: sviluppo della GUI
  - Controller: funzionalità della GUI
  - Bug fixing GUI e funzioni

7.2 Ore effettive

Attività	**Ore previste**	**Ore effettive**
Studio e progettazione	10	10
Sviluppo del codice del modello	10	15
Studio del framework Qt	10	9
Sviluppo del codice della GUI	10	8
Test e debug	5	7
Stesura della relazione	5	4
Totale:	50	**53**