Università Di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Relazione per

Desktop Application in C++ con GUI in QT

Alunno
Orlando V. M. Ferazzani

Università Di Padova

Student ID
2058653

Chapter 1

Introduzione

Cognome	**Nome**	**Matricola**
Ferazzani	Orlando V. M. Ferazzani	2058653
Donanzan	Davide	2034337

Table 1.1: Componenti Del Gruppo

1.1 L'idea

Questa Desktop Application è la realizzazione di un progetto che io, Orlando, e il mio collega, Davide, avevamo in mente da tempo, in scala minore. Si tratta di un software che ti permette di monitorare i valori di sensori adibiti alla produzione vitivinicola. Questi sensori sono divisi per gruppi, dove ogni gruppo è un insieme di sensori collegati da un fattore logico comune, che può essere lo stesso posto fisco (quindi tutti i sensori di una cantina ad esempio) o fattore di produzione (monitorare tutto il processo di produzione di un certo vino). I nostri sensori misurano temperatura, pressione, umidità. In particolare, i tipi di sensori creati e creabili sono:

a. Sensore di Temperatura atmosferica: utile per la cantina e la vigna. Sensore Di tipo "Non a Contatto"

b. Sensore di Temperatura del Mosto: Sensore di tipo "a contatto"

c. Sensore di Umidità dell'Aria: Sensore di tipo "Capacitativo"

d. Sensore di Pressione Atmosferica: Sensore di tipo "Pressorio"

e. Sensore di Pressione della Botte: Sensore di tipo "Pressorio"

La scelta dell'argomento, quindi, non è stata per niente difficoltosa, ma, tutt'altro. Infatti, questo progetto è la rampa di lancio per portare a termine l'idea iniziale, di collegare tutti i sensori mediante dispositivi IoT, utilizzando l'aiuto di un'IA per elaborare i dati. Questi dati saranno poi visibili attraverso una dahsboard divisa per tipo di sensore.

1.2 Tipi Di Sensori

Come anticipato nel capitolo precedente, la nostra application sfrutta diversi tipi di sensori. I tre MacroTipi sono:

1. Sensori Di Temperatura: per lo scopo del progetto, si utilizzano sensori non a contatto per le temperature dell'ambiente e a contatto per il mosto.

   (a) I sensori a contatto sfruttano il rapporto tra temperatura e conduttività elettrica. Utilizzano quindi materiali conduttori.

   (b) I sensori non a contatto sfruttano invece gli infrarossi.

2. Sensori di Umidità: Per lo scopo del progetto i sensori di umidità sono di tipo capacitativo.

   (a) I sensori capacitativi sfruttano variazione della permettività elettrica di un materiale dielettrico al variare dell'umidità. Da un punto di vista costruttivo, un sensori di umidità capacitivo è costituito da un condensatore, tra le cui armature viene inserito un opportuno materiale dielettrico la cui costante dielettrica varia con l'umidità.

3. Sensori Di Pressione: Usano una membrana più o meno rigida, che è deformata dalla pressione dell'ambiente da misurare. Questa deformazione è poi tramutata in segnale elettrico da una piezoresistenza.

Chapter 2

Modello

Alla base della nostra gerarchia è presente la classe base astratta Sensor, che include come campo dati un vecotor di Data, altra classe della gerarchia. Tutte le altre classi relative ai sensori, derivano da sensor e ogni sensore specifico poi, ha una funzione di generazione dei dati. Queste funzioni sono state create in modo che mimino, attraverso diverse distribuzioni, l'effettivo comportamento del fenomeno fisico misurato. Ad esempio, il sensore di temperatura della vigna misurerà dei dati generati con distribuzione Normale.

2.1 Implementazione

L'idea generale dei sensori è la seguente: hanno un valore base, che rappresenta il valore ottimale del fenomeno misurato (20 °C per la temperatura ad esempio), valore che può essere superato o meno, sempre rispettando una certa soglia, che è data dalla variabile **threshold**.

```
1     class Sensor
2     {
3     private:
4         string name; // nome che lo identifica
5         vector<Data> infoArray; // valori effettivi - dati misurati
6         double expectedValue;   // valore atteso dallo specifico
    sensore
7         double threshold;       // soglia
8     }
9
```

Listing 2.1: Implementazione parte privata di Sensor

Il metodo virtuale "generate()" all'interno di Sensor rende la gerarchia di sensori uno Strategy Pattern, che è uno dei pattern fondamentali della OOP. Consiste nell'isolare un algoritmo all'interno di un oggetto, in maniera tale da risultare utile

in quelle situazioni dove sia necessario modificare dinamicamente gli algoritmi utilizzati da un'applicazione.

```
1    void VinesTemperatureSensor::generate() {
2        const double pi = acos(-1);
3        random_device rd;
4        default_random_engine generator(rd());
5        normal_distribution<double> distribution(getExpValue(),
   stdDeviation);
6        //una misurazione all'ora
7        for (unsigned int hour = 0; hour < 24; ++hour) {
8            Data d;
9            d.setTime(hour);
10           double temperature = getExpValue() + amplitude * sin(2 *
   pi * hour / 24.0);
11           temperature += distribution(generator);
12           d.setValue(temperature);
13           push(d);
14       }
15   }
16
```

Listing 2.2: Implementazione funzione distribuzione Normale

Altra classe "worth mentioning" è la classe Group, che è quella con cui l'utente finale effettivamente si interfaccia, in quanto ogni gruppo è associato ad una Tab all'interno della mainwindow. Un Group come specificato sopra, è un insieme di sensori collegati da un comune fattore, logico o fisico.

```
1        class Group
2        {
3        private:
4            string groupName;
5            vector<Sensor *> sensors;
6
7        }
8
```

Listing 2.3: Implementazione class Group

Come si può notare, Group ha come campi dati solamente il suo nome e un vector di sensori. Tutti i metodi sono poi definiti nella parte pubblica. Un metodo interessante è sicuramente "removeSensor", che è implementato in tre modi differenti, così da dare al programma completezza di implementazione. Effettivamente poi, viene usato solamente uno.

2.2 Scelte Progettuali del Modello

Tutti i campi dati dei sensori sono di tipo double in quanto rendono la misurazione dei fenomeni più realistica. Ogni tipo di sensore inoltre, ha una frequenza di campionamento differente, anche se tutte le misurazioni sono effettuate nell'arco di 24 ore, e l'ora della misurazione è visibile grazie al campo dati Time, che è anch'esso una classe della gerarchia. I campi bool sono invece marcati come const in quanto abbiamo deciso di rendere non modificabile il tipo dei sensori una volta creati, mentre è modificabile la expected value e la threshold. Nei sensori di tipo Must, è modificabile anche il timer.

2.3 Polimorfismo

La classe base Sensore contiene i metodi virtuali che poi sono ridefiniti nella maniera più consona in ogni specifica classe. Nel secondo livello della gerarchia (temperatura, umidità e pressione) alcuni dei metodi virtuali astratti vengono definiti in quanto applicabili a tutte le sottoclassi. Poiché alcuni invece, non sono ridefiniti, queste classi riamngono anche loro astratte. L'ultimo livello della gerarchia, invece, ridefinisce le funzioni rimaste astratte con lo scopo principale di differenziare la generazione e il salvataggio dei dati dei sensori concreti. L'ultimo esempio di polimorfismo risiede nei visitor. Per completezza implementativa, dalla classe astratta SensorVisitor, deriva la classe concreta SensorInfo che ridefinisce le funzioni della classe padre, con lo scopo di mostrare e modificare i dati di specifiche classi di sensore. Questa struttura lascia spazio a implementazioni di funzionalità future attraverso i visitor.

Chapter 3

GUI

3.1 Introduzione alla Gui

Come interfaccia grafica ci siamo ispirati ad una classica desktop application, minimale ma funzionale. La nostra mainWindow è composta da vari widget, tutti definiti in altri file così da non creare disordine, e disposti in questo modo:

- mainWindow che contiene tutti i widget: è composta anche da un QTabWidget che server per navigare tra i gruppi di sensori.
- Tab costituisce una tab di visualizzazione all'interno della mainwindow.
  - Ogni Tab è composta da un Head e da un Body. Queste sono classi create da noi, che derivano da QWidget.
  - Le tab sono create dinamicamente quando si crea un gruppo.
  - Nella parte sinistra della schermata, si trova una lista di QPushButtons. Questa è la lista di tutti i sensori del gruppo attualmente aperto.
  - Al centro invece, si hanno le specifiche dei sensori, i tasti per intervenire su di essi, il tasto per generare i grafici e il grafico stesso.
- All'interno della visualizzazione dei dettagli di un sensore si vede:
  - Il nome e l'icona del sensore;
  - La expected Value;
  - La threshold;

Con l'utilizzo dei visitors, i sensori con attributi particolari, ad esempio il timer dei sensori di temperatura del mosto, vengono comunque visualizzati nella lista.

3.2 Funzionalità Presenti

- Creazione, Apertura, Modifica e Cancellazione di un gruppo.
  - La modifica consiste nella rinominazione del gruppo stesso
- Creazione, Apertura, Modifica e Cancellazione di un sensore.
  - La modifica consiste nella rinominazione o nel cambio di alcune variabili del sensore (Expected Value o Threshold e Timer).
- Visualizzazione dei dati generati dai vari sensori sotto forma di grafico
- possibilità di generare nuovamente i dati, sovrascrivendo quelli precedenti (solamente al salvataggio questi saranno sovrascritti sul JSON).
- Controllo della rilevazione dei dati attraverso la funzione isInThreshold(). A livello di interfaccia, il colore del sensore cambia, in base a come è stata superata la threshold.

```
1        if (!(sensor->getArray().empty()))
2        {
3            if (sensor->isInThreshold() == 1)
4                btn->setStyleSheet("background-color: red; font-
   weight: bold;");
5            if (sensor->isInThreshold() == -1)
6                btn->setStyleSheet("background-color: dodgerblue;
    font-weight: bold;");
7        }
8
```

Listing 3.1: Implementazione funzione connection per il cambio del colore del sensore.

Chapter 4

Conclusioni

4.1 Rendicontazione delle ore di lavoro

Attività	**Ore Previste**	**Ore Effettive**
Progettazione	10	8
Sviluppo del codice del modello	20	20
Studio del framework Qt	5	7
Sviluppo del codice della GUI	5	15
Test e debug	10	15
Stesura della relazione	5	5
Totale	**55**	**70**

4.2 Considerazioni Finali

Il progetto, se pur impegnativo e challenging, è stato un buon parco giochi per ciò che il futuro ci potrebbe riservare. Oltre a stimolare il nostro pensiero da "programmatori" ci ha aiutato a sviluppare la nostra abilità di problem solving e team-work, tutte caratteristiche necessarie per lavorare in ambienti di sviluppo.

Speriamo che Tell Me Wine possa essere inoltre il primo tassello per procedere con l'idea originale. Vi ringraziamo per l'opportunità.