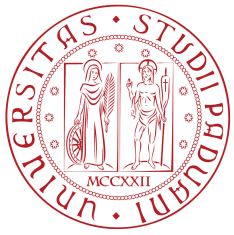


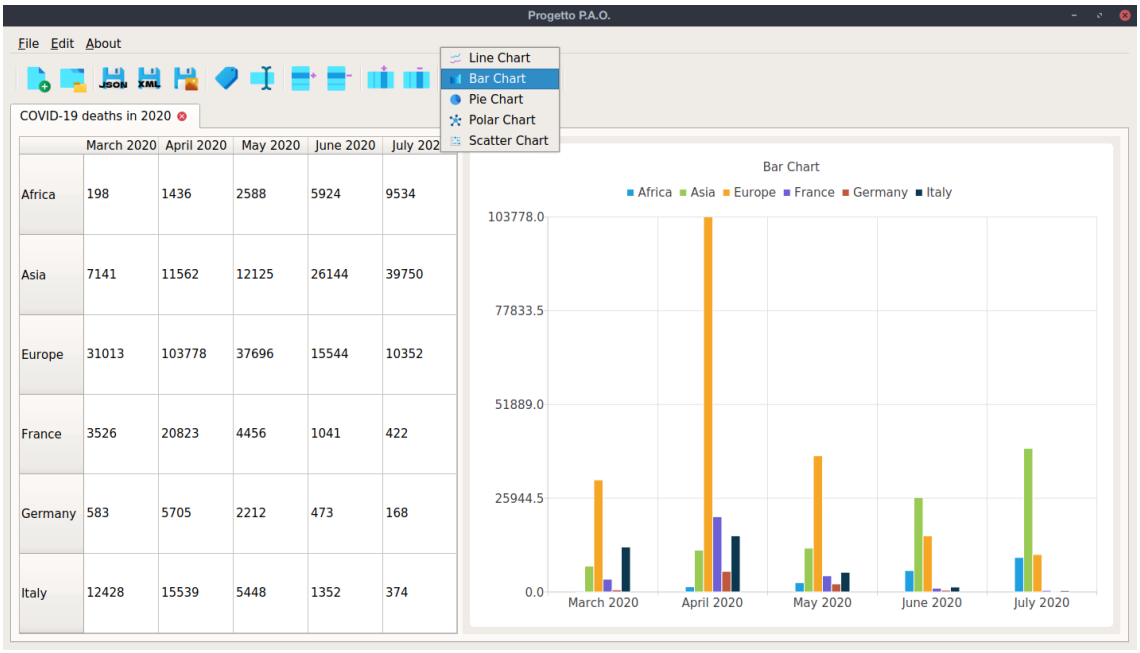
Relazione Progetto di Programmazione ad Oggetti a.a. 2021/2022
QtChartifier

Ivan Antonino Arena

Matricola n. 2000546

Indice

1 Introduzione

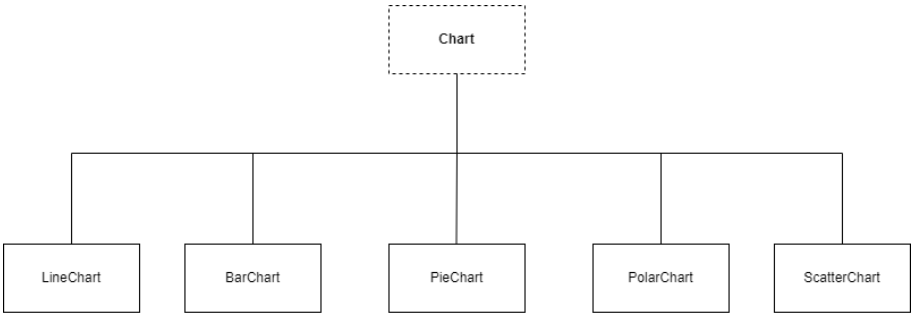


QtChartifier è un software che consente di creare, salvare e modificare cinque tipi diversi di grafici, mediante l'uso di una tabella per gestire i dati. Tra le funzioni disponibili vi sono le seguenti:

- Inserimento di dati numerici in una tabella di grandezza definita dall'utente;
- Modifica delle etichette di ogni riga e colonna ;
- Aggiunta e rimozione di righe e colonne e aggiornamento dei grafici in tempo reale in relazione ai cambiamenti della tabella;
- Visualizzazione di cinque grafici diversi per uno stesso set di dati;
- Gestione di più tabelle tramite un apposito sistema di tab management;
- Importazione ed esportazione di dati in formato JSON o XML ed esportazione di grafici in PNG.

2 Descrizione delle gerarchie dei tipi

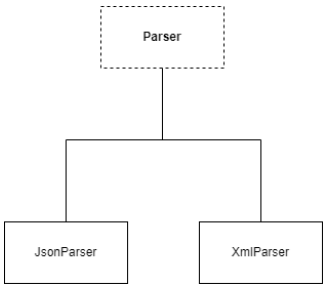
2.1 Chart



`Chart` è la classe base astratta atta alla creazione dei grafici; eredita pubblicamente `QChart` ed è ereditata dalle classi concrete `LineChart`, `BarChart`, `PieChart`, `PolarChart` e `ScatterChart`.

- `LineChart`: implementa il grafico a linee tramite `QLineSeries`, su due assi, `QCategoryAxis` per le etichette delle colonne della tabella sulle ascisse, e `QValueAxis` per i valori significativi della tabella sulle ordinate;
- `BarChart`: implementa il grafico a barre tramite una `QBarSeries` di `QBarSets` e due assi, `QBarCategoryAxis` e `QValueAxis`, analogamente a `LineChart`;
- `PieChart`: implementa il grafico a torta tramite `QPieSeries` e `QPieSlices`;
- `PolarChart`: implementa il grafico polare tramite `QSplineSeries` e due assi, `QCategoryAxis` e `QValueAxis`, analogamente a `LineChart`;
- `ScatterChart`: implementa il grafico a dispersione tramite `QScatterSeries` e due assi, `QCategoryAxis` e `QValueAxis`, analogamente a `LineChart`;

2.2 Parser

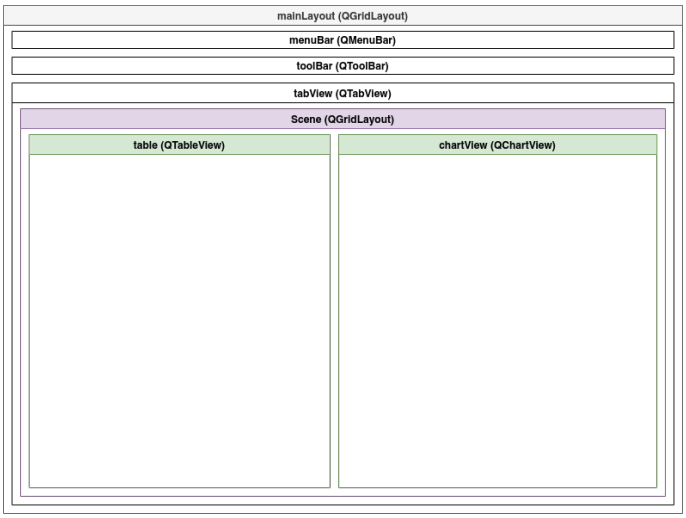


`Parser` è la classe base astratta del sistema di parsing dei file in formato JSON, gestito dalla classe derivata `JsonParser`, e XML, gestito dalla classe derivata `XmlParser`. È facilmente estensibile creando altre classi derivate per descrivere ulteriori formati.

2.3 DataTableModel

`DataTableModel` è la classe che definisce il modello dati utilizzato; eredita `QAbstractTableModel`.

2.4 View



`View` è la classe principale della GUI del progetto, quella che contiene tutti gli elementi grafici e che sarà renderizzata nella finestra dal file `main.cpp`. Eredita pubblicamente `QWidget` e ha come campi dati privati un `Controller`, i `QAction` (più un `QComboBox`) per `QMenu` e `QToolBar`, il `QLabel` della finestra iniziale, un `QTabWidget` per gestire più finestre (`Scene`) contemporaneamente.

2.4.1 Scene

`Scene` è una classe derivata da `QWidget` che rappresenta la schermata tramite un `QGridLayout`, il quale contiene, a sua volta, altri due widget grafici, `QTableView` e `QChartView`, utilizzati rispettivamente per renderizzare un'istanza di `DataTableModel` ed una di `QChart`.

3 Descrizione delle chiamate polimorfe

Oltre alle chiamate polimorfe già fornite dalla libreria Qt, ampiamente impiegate nel progetto (specificatamente nelle classi della GUI), il progetto presenta una serie di metodi virtuali implementati a partire dalle classi base astratte `Chart` e `Parser`.

3.1 Chart

`Chart` contiene un campo dati protetto di tipo `DataTableModel`, che memorizza il modello da rappresentare e tutti i metodi virtuali atti alla creazione e alla modifica di grafici, e i seguenti metodi:

- `virtual void mapData()`:

  Effettua il mapping dei dati del modello sul grafico;

- `virtual void insertSeries()`:

Aggiorna la struttura del grafico ogni volta che viene aggiunta una riga nella tabella;

- `virtual void removeSeries()`:

  Aggiorna la struttura del grafico ogni volta che viene rimossa una riga nella tabella;

- `virtual void insertSeriesValue()`:

  Aggiorna la struttura del grafico ogni volta che viene aggiunta una colonna nella tabella;

- `virtual void removeSeriesValue()`:

  Aggiorna la struttura del grafico ogni volta che viene rimossa una colonna nella tabella;

- `virtual void updateAxes()`:

  Ridimensiona le assi del grafico (ove presenti), basandosi sui valori minimi e massimi presenti nella tabella, ogni volta che avvengono modifiche;

- `virtual void replaceValue(QModelIndex, QModelIndex)`:

  Ogni volta che nella tabella viene modificato un valore, lo aggiorna anche nel grafico;

- `virtual void updateSeriesName(Qt::Orientation, int, int)`:

  Ogni volta che una delle etichette della tabella viene modificata, la aggiorna anche nel grafico;

- `virtual void clearChart()`:

  Viene invocato dal distruttore, effettua la distruzione profonda;

3.2 Parser

`Parser` contiene due metodi virtuali puri per la lettura e la scrittura da e su file:

- `virtual DataTableModel* load(QFile&) const = 0`:

  Legge un file di testo `QFile` adeguatamente formattato e lo converte in un'istanza di `DataTableModel`;

- `virtual void save(DataTableModel*, QFile&) const = 0`:

  Scrive un'istanza di `DataTableModel` su file di testo, formattandola adeguatamente per le future letture;

4 Descrizione dei formati di input e output

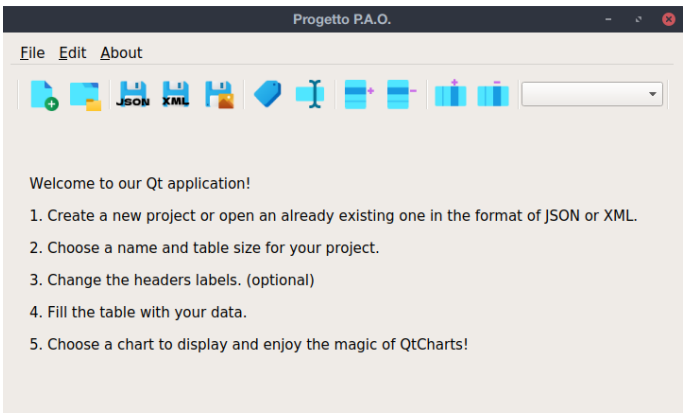
Il progetto comprende un apposito sistema di parsing che consente di leggere e salvare file in formato JSON (JavaScript Object Notation) e XML (eXtensible Markup Language). Per JSON la struttura è del tipo:

```
{
    "Columns": [
        PrimaColonna,
        ...,
        UltimaColonna
    ],
    "Rows": {
        PrimaRiga: [
            valore1,
            ...,
            valoreN
        ],
        ...,
        UltimaRiga: [...]
    }
}
```

Per XML, invece, la struttura è del tipo:

```
<Doc>
    <Columns>PrimaColonna, SecondaColonna, ..., UltimaColonna</Columns>
    <Rows>
        <PrimaRiga>valore1, valore2, ..., valoreN</PrimaRiga>
        ...
        <UltimaRiga>valore1, valore2, ..., valoreN</UltimaRiga>
    </Rows>
</Doc>
```

5 Manuale utente della GUI



La GUI è pensata per essere di immediata comprensione grazie alle icone esplicative e la schermata iniziale dell'applicazione presenta, in sintesi, tutte le operazioni possibili. Un breve manuale d'uso è, inoltre, presente nel menù *About*. Le numerose scorciatoie, elencate nei menù, accanto ai corrispondenti comandi, rendono l'esperienza utente più veloce e comoda.

6 Istruzioni di compilazione

Per compilare ed eseguire il file del progetto occorre spostarsi nella cartella `src` e lanciare i seguenti comandi da terminale:

```
$ qmake prog.pro
$ make
$ ./prog
```

7 Descrizione ore richieste

Attività	**Ore impiegate**
Analisi preliminare del problema	2
Progettazione di modello e GUI	4
Apprendimento della libreria Qt	9
Codifica di modello e GUI	18
Debugging	18
Testing	2
Stesura relazione LaTeX	3
Totale	**56**

Ho superato di poco le 50 ore previste per curare più nel dettaglio l'aspetto grafico, realizzando personalmente alcune delle icone, oltre che nella codifica di alcune funzionalità minori ma che contribuiscono ad aumentare la qualità e lo spessore del progetto.

8 Suddivisione del lavoro

Matricola	**Attività**
Ivan Antonino Arena 2000546	Progettazione modello e GUI Codifica modello e View Codifica del controller Codifica Chart, LineChart e BarChart Scrittura dataset XML
Lorenzo Pasqualotto 2008651	Progettazione modello e GUI Codifica del sistema di parsing Codifica di PieChart, PolarChart e ScatterChart Debug View Scrittura dataset JSON

9 Informazioni sullo sviluppo

Strumento	**Versione**
Ubuntu	21.10
G++	11.2.0
Qt	5.9.9
Qt Creator	6.0.2

Sono stati inoltre utilizzati i seguenti strumenti:

- **Git/GitHub:** per il controllo versione e la Kanban Board;
- **Diagrams.net:** per la progettazione delle gerarchie;
- **Overleaf:** per la stesura della relazione in LaTeX;