oggetto Relazione progetto Programmazione a Oggetti

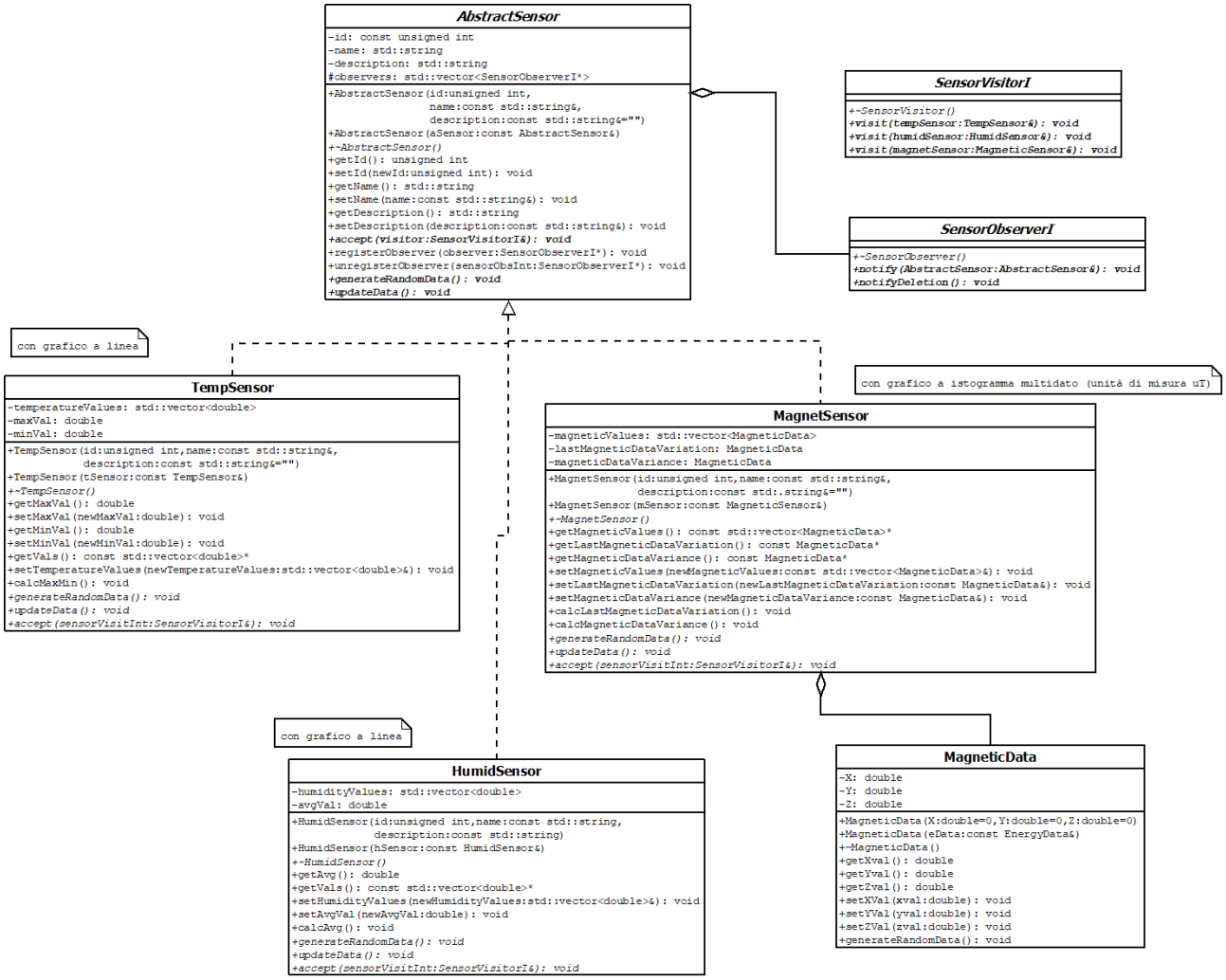
gruppo Bellon Filippo, mat [redacted]

titolo Sensors Monitor

Introduzione

Sensors Monitor è un gestore che permette di creare, modificare, eliminare e analizzare i dati raccolti da un insieme di sensori. Attualmente il progetto include tre tipi di sensori tra cui scegliere: sensori di temperatura, sensori di umidità e sensori di campi magnetici, tutti salvati all'interno di una memoria temporanea, svuotata ad ogni terminazione del programma. Ogni sensore viene rappresentato, oltre che graficamente da un'icona che ne identifica il tipo, anche da un insieme di dati specifici che variano da sensore a sensore: valori maggiori, minori, medi, varianze, etc..... È possibile quindi operare su ogni sensore utilizzando le operazioni di creazione, modifica ed eliminazione di quest'ultimo. Inoltre, è attiva una funzionalità di filtro, la quale permette di ricercare solo un certo numero di sensori che rispettano dei requisiti, in questo caso, utilizzando solo l'attributo *Nome*. Successivamente, all'interno di una barra posizionata in alto, troviamo le operazioni di salvataggio e apertura/caricamento di un file di sensori, il quale, in caso di assenza di errori, permettono rispettivamente di: salvare tutti i sensori della memoria all'interno di un file Json appositamente formattato; eliminare totalmente i sensori presenti all'interno della memoria e di inserire quelli ricavati dal file. Infine, in basso, è attiva una barra di stato che permette di seguire lo stato di salvataggio e di utilizzo di file.

Descrizione del modello



Come sia apprende dall'immagine, la classe *AbstractSensor* si presenta come la classe base per ogni tipo di sensore contenuto all'interno del programma. Si compone dei tre attributi comuni *Nome*, *Id* e *Descrizione*, oltre che ai relativi

metodi per la loro manipolazione. Sono inoltre presenti metodi per la gestione dei vari *Observers* e *Visitors* della classe, a cui si aggiungono due metodi qui definiti astrattamente ma implementati in maniera diversa all'interno di ogni classe derivata, che permettono la creazione e l'aggiornamento dei dati in maniera casuale.

Da notare inoltre come la classe *Observer* presenti due metodi distinti. Il primo, *Notify*, è il metodo normalmente utilizzato per notificare gli Observers di cambiamenti avvenuti nella classe; il secondo, *Notifydeletion*, è stato creato per prevenire problemi di incongruenza poiché in Qt, è il sistema che si occupa dell'eliminazione di un certo oggetto tramite un'apposita funzione, che però molte volte non segue l'ordine di eliminazione che ci interessa e quindi rischiamo di andare a lavorare con valori nulli.

Le classi derivate *TempSensor* e *HumidSensor*, derivano metodi e attributi dalla classe *AbstractSensor* a cui vanno ad aggiungere altri attributi specifici quali *MaxVal*, *MinVal* e *AvgVal*, con i relativi metodi per la loro gestione. Implementano quindi i metodi astratti, definiti nella superclasse, per la creazione e l'aggiornamento dei dati in maniera casuale.

La classe derivata *MagnetSensor*, oltre che derivare dalla classe *AbstractSensor*, si compone di valori i quali sono definiti attraverso la creazione di un ulteriore classe: *MagneticData*. Anche qui, come nelle altre due classi di sensori derivate, si vanno ad aggiungere altri attributi e metodi specifici, oltre ad implementare i vari metodi astratti.

Polimorfismo

All'interno di questo programma, persistono due configurazioni principali che fanno uso di polimorfismo non banale, ed entrambe sono utilizzate nell'ambito della visualizzazione e rappresentazione grafica di un determinato sensore.

La prima, rappresentata dalla classe *SensorInfoVisitor*, è quella che si occupa di riportare, all'interno di un widget appositamente creato, tutti i dati specifici del tipo di sensore in questione. Nel caso, per esempio, di un sensore di tipo *TempSensor*, il widget creato attraverso il Visitor conterrà i due valori specifici *MaxVal* e *MinVal*, appositamente formattati per essere distinti e mostrati correttamente. Di conseguenza, attraverso l'uso di questa classe, possiamo costruire widget che contengono diversi tipi di dati in base al sensore che stanno rappresentando.

La seconda, rappresentata dalla classe *SensorChartVisitor*, è invece quella che si occupa di costruire un widget in modo da rappresentare i valori contenuti in ogni sensore attraverso l'uso di un grafico. Nel caso, per esempio, di un sensore di tipo *MagnetSensor*, dove i valori sono indicati come triple di altri valori, il widget creato attraverso il Visitor, andrà a costruire un grafico a barre verticali (Istogramma), dove ogni tripla di barre indicherà rispettivamente *XVal*, *YVal*, e *ZVal*, attributi contenuti in *MagneticData*, il quale è il tipo dei valori presenti in *MagnetSensor*.

Persistenza dei dati

Per la persistenza dei dati viene utilizzato un file di tipo Json. In un file, i sensori vengono salvati all'interno di un array Json, ognuno con i rispettivi attributi, ai quali si aggiunge un attributo *Type*, utilizzato principalmente in lettura per la distinzione tra i vari tipi di sensori.

Funzionalità implementate

Le funzionalità implementate sono, per semplicità, suddivise in due categorie: funzionali ed estetiche.

Le prime comprendono:

- gestione di tre tipologie di sensori e relativi dati
- salvataggio e caricamento su/da file in formato JSON
- funzionalità di ricerca all'interno dell'attributo *Nome*

Le funzionalità grafiche:

- utilizzo di una toolbar per le operazioni di salvataggio e caricamento su/da file
- utilizzo di icone nella toolbar
- status bar in basso

- controllo della presenza di modifiche non salvate prima di uscire
- gestione del ridimensionamento
- visualizzazione distinta per ogni tipo di sensore
- utilizzo di immagini per la distinzione tra i vari tipi di sensori
- utilizzo di pulsante per l'esecuzione delle varie operazioni sui sensori
- utilizzo di un pulsante e di una barra di ricerca per il filtraggio dei sensori
- utilizzo di grafici differenti

Rendicontazione ore

Attività	Ore Previste	Ore Effettive
Studio e progettazione	10	15
Sviluppo del codice del modello	10	17
Studio del framework Qt	10	19
Sviluppo del codice della GUI	10	20
Test e debug	5	5
Stesura della relazione	5	2
totale	50	78

Il monte ore è stato superato in quanto lo studio della documentazione e l'implementazione della GUI hanno richiesto più tempo del previsto, in particolare, sono stati svolti parecchi test per prendere confidenza con il framework di Qt.

Note

Di seguito sono riportate alcune annotazioni riguardo il progetto svolto:

- Durante la compilazione tramite riga di comando su Ubuntu, si potrebbe incontrare un warning come questo:

```
View/SensorsPanel.cpp:72:74: warning: enum constant in boolean context [-Wint-in
-bool-context]
   72 |     scrollAreaSensorWidget->setFrameStyle(QFrame::Box || QFrame::Plain);
      |                                                                          ^~~~~
```

  Tuttavia, ciò si verifica proprio seguendo la documentazione fornita da qt per l'utilizzo di questo determinato metodo: https://doc.qt.io/qt-6.2/qframe.html#setFrameStyle . Di conseguenza ho ritenuto opportuno mantenerlo e notificare il warning tramite questa nota.