oggetto Relazione progetto Programmazione a Oggetti

gruppo Tutino Giuseppe, mat. 2075515

titolo Building Manager

Introduzione

Building Manger è un'applicazione che gestisce i sensori collocati in un edificio. È possibile aggiungere nuovi sensori, modificarli, cercarli ed eliminarli. Esistono tre tipologie di sensori che è possibile gestire: luminosità, rumore e temperatura.

Descrizione del modello

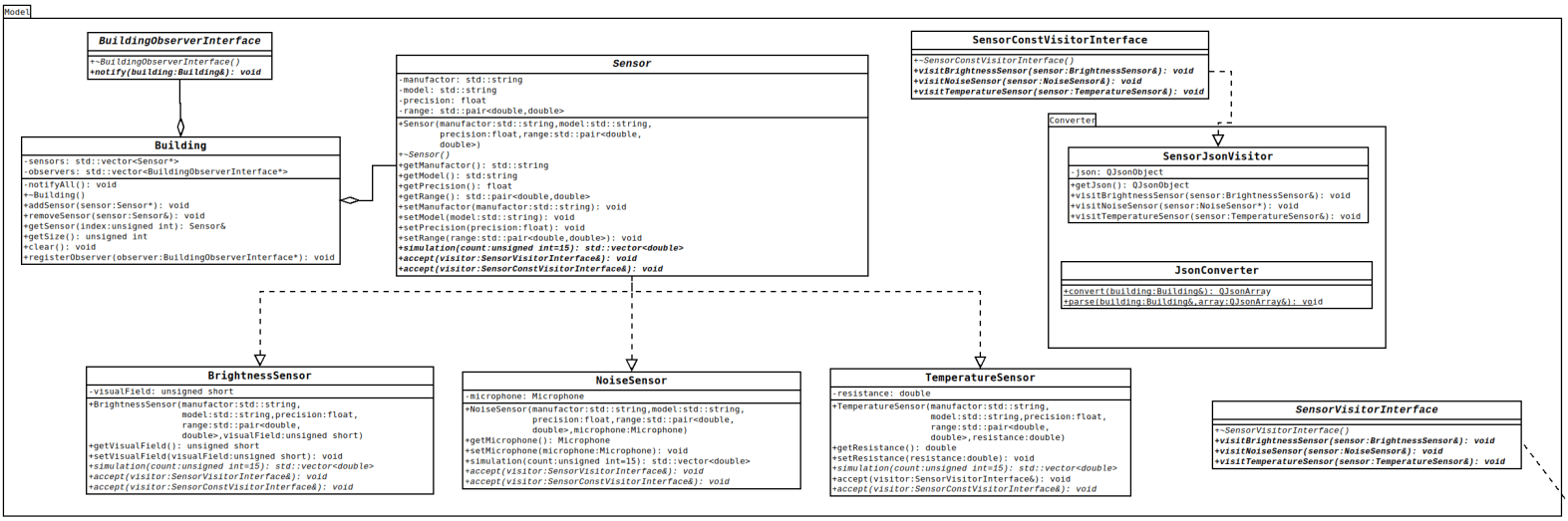
Il modello logico contiene la gerarchia dei sensori, una classe Building che fa da contenitore per i sensori, alcune classi di servizio per convertire i sensori in formato JSON, Visitor per Sensor e un Observer per Building.

Il modello parte da una classe astratta *Sensor* che rappresenta le informazioni comuni a tutti i sensori, ovvero il produttore, modello, precisione e range delle misurazioni, per ciascuno dei quali sono implementati metodi *getter* e *setter*. Poi abbimo le tre classi concrete che rappresentano tre tipi di sensori, ovvero luminosità, rumore e temperatura. Ogni sensore concreto ha delle proprietà intrinseche, quello di luminosità ha il campo visivo, rumore ha la tipologia di microfono e quello di temperatura ha la resistenza.

Si è scelto di utilizzare il *design pattern Visitor* per consentire la visualizzazione dinamica dei sensori e anche la loro scrittura su file JSON. A tal fine sono state realizzate le classi astratte Sensor*VisitorInterface* e Sensor*ConstVisitorInterface*, le quali differiscono unicamente per il fatto che la seconda lascia necessariamente inalterato l'oggetto visitato, ovvero consente di sfruttare la *const correctness* offerta dal compilatore. Di conseguenza, in *Sensor* sono stati inseriti i metodi virtuali puri *accept* per accettare i due tipi di *Visitor*.

Si è scelto anche di utilizzare un *Obsever* per Building, in modo da poter segnalare quando la lista dei sensori cambia.

Ecco il diagramma del modello:



Insieme al codice è fornito il diagramma completo.

Polimorfismo

L’utilizzo principale del polimorfismo riguarda il *design pattern Visitor* e il metodo *simulation* nella gerarchia di *Sensor*.

Il *design pattern Visitor* viene utilizzato per costruire il widget che mostra le informazioni dello specifico sensore (*SensorInfoVisitor*), per la conversione in JSON (*SensorJsonVisitor*), e per la visualizzazione dei diversi dialog di modifica dei sensori (SensorEditVisitor).

SensorInfoVisitor implementa Sensor*ConstVisitorInterface,* si occupa di mostrare dinamicamente gli attributi dello specifico sensore ed anche un’icona in base alla tipologia, costruendo un *QWidget* che verrà mostrato nel *SimulationPanel*.

SensorJsonVisitor implementa Sensor*ConstVisitorInterface,* si occupa di trasformare il sensore che riceve in input in un *QJsonObject, poi in un’altra classe (JsonConverter)* tutti questi *QJsonObject* verranno inseriti in un *QJsonArray* che verrà poi salvato su file.

SensorEditVisitor implementa *SensorVisitorInterface (non const perché è necessario l’utilizzo dei metodi setter),* si occupa di richiamare il *Dialog* di modifica corretto per ogni sensore (*EditBrightnessSensorDialog*, *EditNoiseSensorDialog, EditTemperatureSensorDialog*).

Il metodo *simulation* virtuale puro in *Sensor* viene implementato in ogni sottoclasse concreta, usa la libreria random e nello specifico la distribuzione uniforme per ricavare le misurazioni, cambia il suo comportamento nelle varie sottoclassi alterando la misura utilizzando gli attributi propri della sottoclasse (campo visivo, tipologia di microfono e resistenza.).

Persistenza dei dati

Per la persistenza dei dati viene utilizzato il formato JSON, un file contenente un array di oggetti. Ogni oggetto rappresenta un sensore e la serializzazione delle sottoclassi viene gestita aggiungendo un attributo “type”. Un esempio della struttura dei file è dato dal file “save.json” fornito assieme al codice, che contiene una lista di vari sensori di prova.

Funzionalità implementate

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

- funzionalità di ricerca case insensitive per produttore e modello
- scelta del numero di misurazioni per simulazione
- drag and drop dei file di salvataggio JSON
- scorciatoie da tastiera (mostrate anche nelle voci del menù)

Le funzionalità grafiche:

- barra dei menù in alto
- utilizzo di icone nelle voci del menù e nella visualizzazione delle info del sensore
- gestione del ridimensionamento
- utilizzo di icone nei pulsanti
- utilizzo di colori e stili grafici
- effetti grafici come cambio dello stile dei pulsanti al passaggio del mouse
- animazione del grafico della simulazione

Le funzionalità elencate sono intese in aggiunta a quanto richiesto dalle specifiche del progetto.

Rendicontazione ore

Attività	Ore Previste	Ore Effettive
Studio e progettazione	10	10
Sviluppo del codice del modello	10	11
Studio del framework Qt	10	8
Sviluppo del codice della GUI	10	12
Test e debug	5	6
Stesura della relazione	5	3
totale	50	**50**