UNIVERSITÀ DEGLI STUDI DI PADOVA

Progetto di Programmazione ad Oggetti
Anno accademico 2023/2024

SenseNet

Alessandro Di Pasquale - 2075544
Malik Giafar Mohamed - 2075543

Indice

1. Introduzione
2. Modello Logico
3. Polimorfismo
4. Persistenza dei Dati
5. Funzionalità
6. Rendicontazione delle Ore
7. Suddivisione Attività Progettuali

1 - Introduzione

SenseNet è un software con interfaccia grafica progettato per la gestione e il monitoraggio di sensori di rete. Il programma è destinato a monitorare le connessioni tra gli host all'interno di una specifica rete, la quale risulta essere statica e immutabile, tramite appositi sensori.
Su ciascuna connessione è possibile aggiungere, modificare o rimuovere un sensore a discrezione dell'utente (vedi in dettaglio: 5 - Funzionalità). Ogni tipo di sensore può essere aggiunto una sola volta per connessione, poiché la presenza di più sensori identici sulla stessa connessione fornirebbe valori duplicati, risultando quindi ridondante.
Ogni sensore è identificato da un codice univoco e un nome (si veda la Fig.1 e, nello specifico, 4 - Persistenza dei Dati) e si suddividono in:

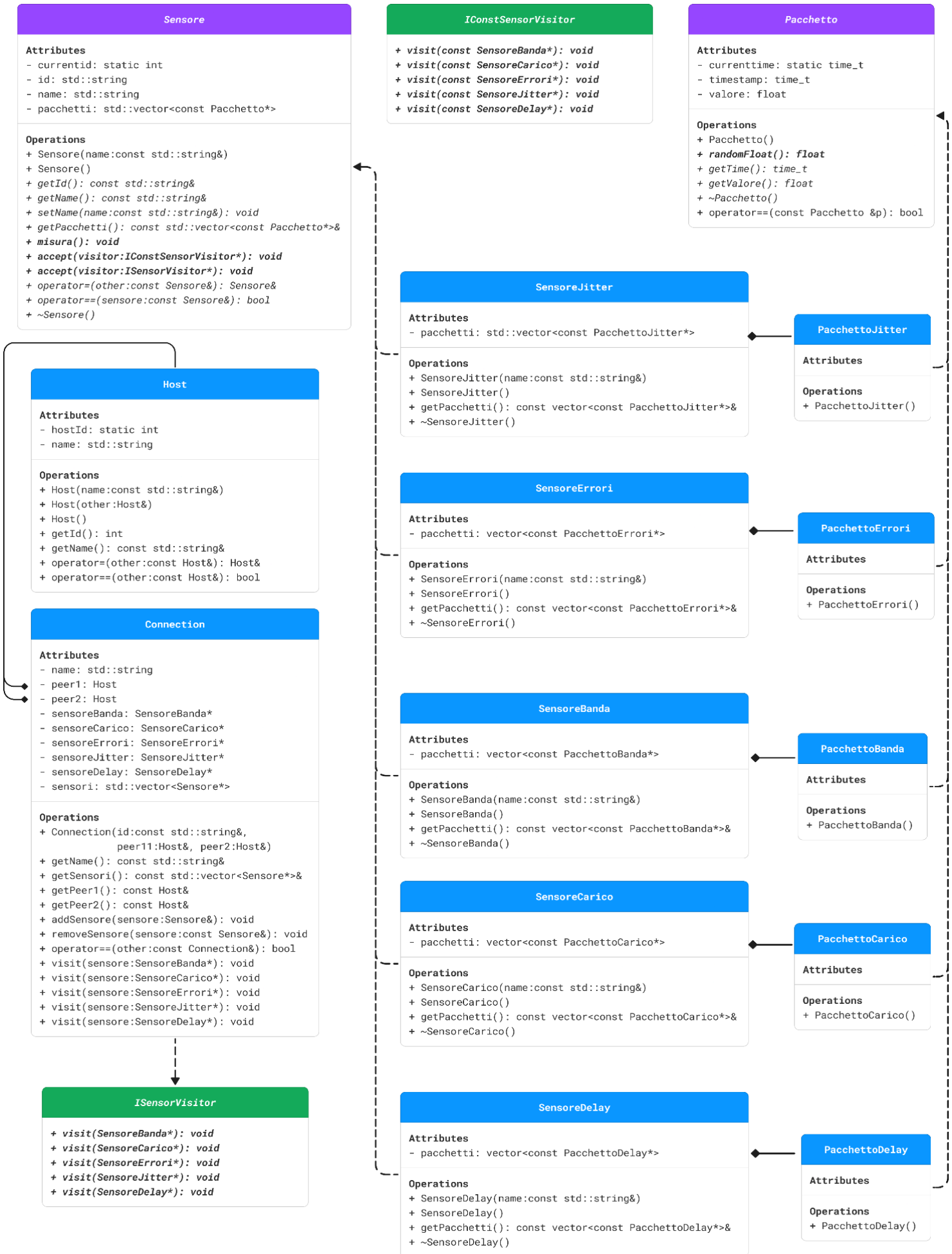
- *Sensore di Banda:* fornisce quanta banda (in Mbps) è in utilizzo, con una sensibilità massima di 1 Gbps per ogni misurazione.
- *Sensore di Errori:* fornisce la quantità complessiva di errori (come packet loss, collisioni, etc.) con una sensibilità massima di 500 errori a misurazione.
- *Sensore di Carico:* permette di visualizzare (in %) quanto è occupata la connessione rispetto alle sue massime possibilità di carico.
- *Sensore di Delay:* offre la visualizzazione del tempo di risposta (in ms) a cui la connessione opera, potendo considerare un delay fino a 1000 ms per misurazione.
- *Sensore di Jitter:* ogni sua misurazione illustra il variare della stabilità di connessione (in ms) ed è capace di percepire variazioni di massimo 50 ms ad ogni misurazione.

Ogni sensore conterrà una serie di pacchetti, che includeranno i dati reali di una misurazione insieme a un timestamp per facilitare l'inserimento dei dati in un grafico. I pacchetti saranno suddivisi nelle stesse categorie dei sensori e dovranno necessariamente essere associati al tipo di sensore corrispondente.

Inoltre, l'interfaccia grafica di SenseNet offre la possibilità di visualizzare più misurazioni di qualsiasi sensore in un determinato periodo di tempo, utilizzando appositi grafici con le relative unità di misura e livelli di sensibilità.

2 - Modello Logico

Nel diagramma da noi creato, la composizione è rappresentata dal collegamento con un quadrato pieno, mentre l'implementazione è raffigurata da una freccia tratteggiata. In italic gli elementi virtuali ed in bold-italic gli elementi virtuali puri. (si è cercato di rimanere il più fedeli possibile alla "sintassi" del software DIA)


Sensore
Attributes
- currentid: static int
- id: std::string
- name: std::string
- pacchetti: std::vector<const Pacchetto*>
Operations
+ Sensore(name:const std::string&)
+ Sensore()
+ getId(): const std::string&
+ getName(): const std::string&
+ setName(name:const std::string&): void
+ getPacchetti(): const std::vector<const Pacchetto*>&
+ misura(): void
+ accept(visitor:IConstSensorVisitor*): void
+ accept(visitor:ISensorVisitor*): void
+ operator=(other:const Sensore&): Sensore&
+ operator==(sensore:const Sensore&): bool
+ ~Sensore()
IConstSensorVisitor
+ visit(const SensoreBanda*): void
+ visit(const SensoreCarico*): void
+ visit(const SensoreErrori*): void
+ visit(const SensoreJitter*): void
+ visit(const SensoreDelay*): void
Pacchetto
Attributes
- currenttime: static time_t
- timestamp: time_t
- valore: float
Operations
+ Pacchetto()
+ randomFloat(): float
+ getTime(): time_t
+ getValore(): float
+ ~Pacchetto()
+ operator==(const Pacchetto &p): bool
Host
Attributes
- hostId: static int
- name: std::string
Operations
+ Host(name:const std::string&)
+ Host(other:Host&)
+ Host()
+ getId(): int
+ getName(): const std::string&
+ operator=(other:const Host&): Host&
+ operator==(other:const Host&): bool
Connection
Attributes
- name: std::string
- peer1: Host
- peer2: Host
- sensoreBanda: SensoreBanda*
- sensoreCarico: SensoreCarico*
- sensoreErrori: SensoreErrori*
- sensoreJitter: SensoreJitter*
- sensoreDelay: SensoreDelay*
- sensori: std::vector<Sensore*>
Operations
+ Connection(id:const std::string&,
peer11:Host&, peer2:Host&)
+ getName(): const std::string&
+ getSensori(): const std::vector<Sensore*>&
+ getPeer1(): const Host&
+ getPeer2(): const Host&
+ addSensore(sensore:Sensore&): void
+ removeSensore(sensore:const Sensore&): void
+ operator==(other:const Connection&): bool
+ visit(sensore:SensoreBanda*): void
+ visit(sensore:SensoreCarico*): void
+ visit(sensore:SensoreErrori*): void
+ visit(sensore:SensoreJitter*): void
+ visit(sensore:SensoreDelay*): void
ISensorVisitor
+ visit(SensoreBanda*): void
+ visit(SensoreCarico*): void
+ visit(SensoreErrori*): void
+ visit(SensoreJitter*): void
+ visit(SensoreDelay*): void
SensoreJitter
Attributes
- pacchetti: std::vector<const PacchettoJitter*>
Operations
+ SensoreJitter(name:const std::string&)
+ SensoreJitter()
+ getPacchetti(): const vector<const PacchettoJitter*>&
+ ~SensoreJitter()
PacchettoJitter
Attributes
Operations
+ PacchettoJitter()
SensoreErrori
Attributes
- pacchetti: vector<const PacchettoErrori*>
Operations
+ SensoreErrori(name:const std::string&)
+ SensoreErrori()
+ getPacchetti(): const vector<const PacchettoErrori*>&
+ ~SensoreErrori()
PacchettoErrori
Attributes
Operations
+ PacchettoErrori()
SensoreBanda
Attributes
- pacchetti: vector<const PacchettoBanda*>
Operations
+ SensoreBanda(name:const std::string&)
+ SensoreBanda()
+ getPacchetti(): const vector<const PacchettoBanda*>&
+ ~SensoreBanda()
PacchettoBanda
Attributes
Operations
+ PacchettoBanda()
SensoreCarico
Attributes
- pacchetti: vector<const PacchettoCarico*>
Operations
+ SensoreCarico(name:const std::string&)
+ SensoreCarico()
+ getPacchetti(): const vector<const PacchettoCarico*>&
+ ~SensoreCarico()
PacchettoCarico
Attributes
Operations
+ PacchettoCarico()
SensoreDelay
Attributes
- pacchetti: vector<const PacchettoDelay*>
Operations
+ SensoreDelay(name:const std::string&)
+ SensoreDelay()
+ getPacchetti(): const vector<const PacchettoDelay*>&
+ ~SensoreDelay()
PacchettoDelay
Attributes
Operations
+ PacchettoDelay()

Il codice del modello logico è organizzato in due sezioni: la prima contiene le classi del core, che gestiscono la logica del programma, mentre la seconda si occupa della persistenza dei dati in formato JSON. Le classi astratte `Sensore` e `Pacchetto` sono fondamentali, rappresentando rispettivamente tutti i tipi di sensori e pacchetti menzionati sopra. Ogni sensore è contraddistinto da un ID univoco e immutabile, sebbene sia possibile modificare il nome del sensore tramite l'unico setter disponibile.Le classi concrete derivate da `Sensore` includono il metodo `getPacchetti()`, che restituisce diversi tipi di pacchetti in base al tipo di sensore. È presente anche il metodo `misura()`, che consente di aggiungere una nuova misurazione al sensore che lo invoca. Le classi `Host` e `Connection` sono identificate tramite il loro nome e, poiché vengono definite all'interno del programma, non possono essere modificate; di conseguenza, non dispongono di setter. Nel caso delle connessioni, il metodo `addSensore()` verifica, prima di aggiungere un sensore al vettore, che un sensore dello stesso tipo non sia già presente tra gli attributi della classe; in caso affermativo, il vettore rimane invariato.

3 - Polimorfismo

L'implementazione del polimorfismo non banale è ottenuta tramite l'utilizzo del Visitor Design Pattern, gestito dalle classi:

- `ISensorVisitor`: adoperato nel contesto dell'aggiunta dei sensori in una connessione, la quale impone che non vengano inseriti molteplici sensori dello stesso tipo (per approfondire: 1 - Introduzione). Più nello specifico:
  - `Connection`: utilizza il Visitor nel metodo `addSensore()` per confrontare gli attributi con il sensore da aggiungere (vedi 2 - Modello Logico).

- `IConstSensorVisitor`: ideato unicamente per operazioni *const*, utilizzato per il riconoscimento della tipologia di sensore, per determinare quale simulazione del grafico avviare per un determinato sensore. Più nello specifico:
  - `GraficoSensore`: è una classe che, nel costruttore, prende in input un oggetto di tipo `Sensore` (dunque generico) e che in relazione alla tipologia di sensore fornitagli aprirà una finestra raffigurante il grafico (simulato) con le rispettive proprietà ed unità di misura (Fig.7).

4 - Persistenza dei Dati

Per consentire il salvataggio e il trasferimento della configurazione dei sensori creata, si utilizza la persistenza dei dati tramite un file di salvataggio in formato JSON. Questo file è composto da un vettore di elementi, ognuno dei quali contiene il nome della connessione di riferimento e i sensori associati ad essa.

Per ogni sensore vengono riportati l'ID e il nome assegnato (sancito dall'utente al momento della creazione del sensore o modificato in seguito tramite GUI).

Segue un esempio (Fig.1):

```
[
    {
        "name": "A-B",
        "sensors": [
            {
                "id": "BS#1",
                "name": "Bandwidth Sensor"
            }
        ]
    },
    {
        "name": "B-C",
        "sensors": [
        ]
    }
]
```

[Fig.1]

Un file più completo, `sensors.json`, è fornito tra gli asset del progetto.
(N.B.: il numero di id nel file è comunque mantenuto unicamente per leggibilità e correttezza)

5 - Funzionalità

5.1 - Funzionalità Logiche

Vengono fornite molteplici funzionalità logiche, sia essenziali che non, tra cui:

- Creazione, modifica e rimozione di qualsiasi sensore (5 tipologie differenti).
  (n.b. L'aggiunta di un sensore ad una connessione avente già un sensore dello stesso tipo non verrà ritenuto errore, ma l'aggiunta sarà ignorata/annullata)
- Ricerca di un sensore all'interno della connessione corrente, tramite RegEx
  (n.b. Ognuna delle suddette operazioni, quando compiuta, comporta la visualizzazione globale di tutti i sensori.)
- Gestione della persistenza dei dati attraverso file di salvataggio in JSON
- *Save*: qualora si abbia già esportato o salvato il file JSON, questa funzionalità permette di sovrascriverlo comodamente senza dover chiedere di nuovo la destinazione di salvataggio. In caso fosse la prima volta in cui l'operazione di salvataggio viene effettuata, verrà richiesto di specificare un percorso di destinazione ed un nome del file
  (default: `./sensor.json`)
- *Export JSON*: questa funzione salva il file JSON chiedendo esplicitamente percorso e nome del documento di destinazione ogni volta che viene invocata. (default: `./sensor.json`)
- *Import JSON*: questa funzione consente di caricare nel programma in esecuzione una configurazione di sensori precedentemente salvata, permettendo esclusivamente la selezione di un file in formato JSON. L'importazione sovrascrive immediatamente la configurazione attuale. Se alcune connessioni nel file importato non corrispondono a quelle presenti nel programma, queste saranno ignorate insieme ai sensori a esse associati. Inoltre, se i tipi di sensori nel file JSON non corrispondono esattamente a quelli esistenti, il file di salvataggio sarà considerato non valido e ignorato, senza modificare la configurazione corrente dei sensori (vedi 1 - Introduzione).

- Combinazioni di tasti shortcut (n.b. in MacOS il tasto CTRL è sostituito dal tasto CMD)
- *CTRL+S* effettua l'operazione di *Save*.
- CTRL+E effettua l'operazione *Export JSON*.
- CTRL+I effettua l'operazione di *Import JSON*.
- CTRL+N apre la pagina di aggiunta di un nuovo sensore.

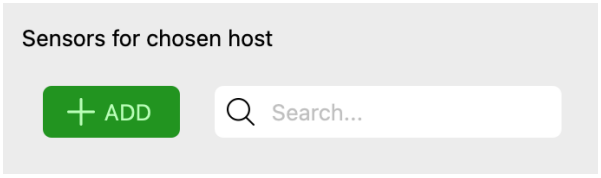
5.2 - Funzionalità Grafiche

Durante l'implementazione dell'interfaccia utente, si è pensato a renderla facilmente utilizzabile e piacevole alla vista.
le principali funzionalità grafiche sono:

- Toolbar di accesso rapido alle funzionalità di salvataggio e caricamento (Fig.2), riposizionabile in qualsiasi lato della finestra.

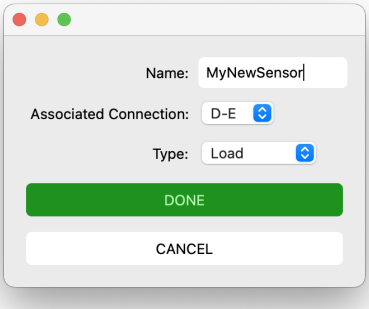
[Fig.2]

- Sezione di aggiunta e di ricerca dei sensori (Fig.3a)



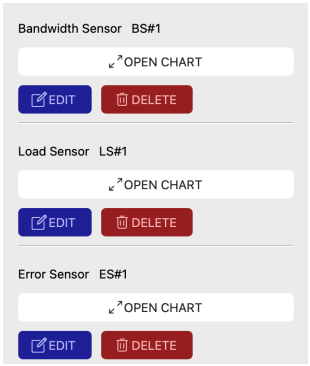
[Fig.3a]

- Menù di aggiunta (Fig.3b) che permette la selezione di nome, connessione e tipo di sensore desiderati; finché non è specificato un nome non è possibile premere DONE



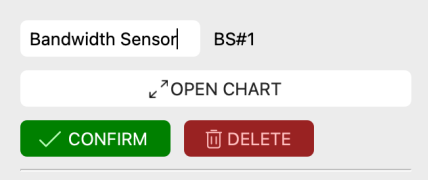
[Fig.3b]

- Sezione scorrevole di visualizzazione e gestione dei sensori (Fig.4)



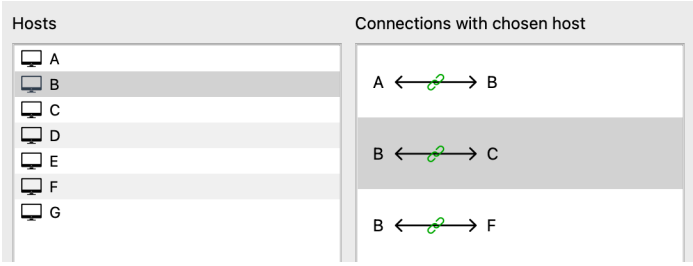
[Fig.4]

Nella suddetta, premere il tasto DELETE comporta la rimozione del sensore, OPEN CHART permette la visualizzazione del relativo grafico e il pulsante EDIT fornisce la possibilità di modificare (rinominare) il sensore (Fig.5), nel caso in cui si dovesse lasciare vuoto il nome, premendo CONFIRM due volte si reimposta il nome del sensore a quello originale.
Non viene quindi permessa la rimozione del nome.



[Fig.5]

- Per permettere in semplicità una corretta visualizzazione dei sensori nella connessione desiderata, ci si può avvalere delle colonne scorrevoli di selezione degli host e delle relative connessioni presenti sulla destra, come mostrato in Fig.6



[Fig.6]

- Ogni grafico delle misure si distingue per le informazioni e, talvolta, anche per la forma (Fig.7)

[Fig.7]

- Tutti i pulsanti “OPEN CHART”, ”EDIT”, “DELETE”, “ADD” hanno effetti visivi di *hover* ottenuti tramite l’utilizzo del QSS (e.g. Fig.8)

[Fig.8]

6 - Rendicontazione delle Ore

	Ore Previste	Ore Effettive
Progettazione del modello logico	5	6
Implementazione modello logico	15	17
Studio del framework Qt	5	5
Implementazione del modello MVC	10	11
Test e debug	10	10
Stesura della relazione	5	5
Totale ore	**50**	**54**

7 - Suddivisione Attività Progettuali

Il mio ruolo all'interno del progetto è consistito principalmente nello sviluppo del back-end dell'applicativo, nello specifico:

- la progettazione del modello logico (struttura, design patterns, ecc)
- l'implementazione delle classi del core (sensore, pacchetto, ecc)
- l'implementazione del controller
- il collegamento del controller con front-end e backend
- l'implementazione di slot e segnali
- l'implementazione dell'ISensorVisitor
- la correzione di alcuni bug implementativi del front-end

Questa suddivisione delle attività progettuali ha permesso un'equa distribuzione del carico di lavoro per questo progetto.

8 - Compilazione

Infine, vengono riportate le istruzioni di compilazione su ubuntu per il progetto:

- nella directory dove si trova il file `SenseNetCPPro.pro` sarà necessario eseguire il comando `qmake SenseNetCPPro.pro && make && ./SenseNetCPPro`