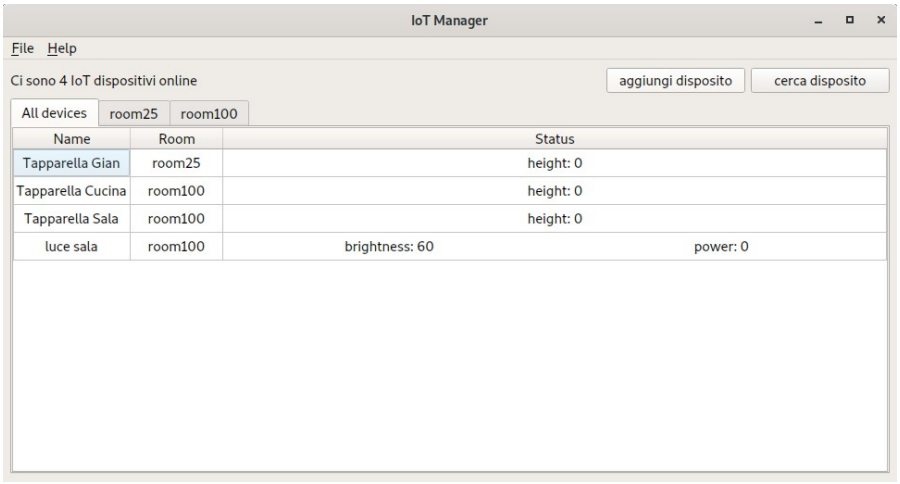
Univeristà degli studi di padova
Dipartimento di Matematica “Tullio Levi-Civita”

Corso di Laurea in Informatica



Qontainer

Relazione del progetto di Programmazione ad Oggetti di Gianmarco Pettinato matricola 1068299
Anno Accademico 2018-2019

Introduzione

Recentemente mi sono appassionato al mondo dei dispositivi IoT (*internet of things*), e visto che l'applicazione da sviluppare poteva contente qualsiasi cosa ho deciso di improntare lo sviluppo intorno a questi dispositivi e ad l'interazione che una persona può avere con essi.
Dopo diverse ricerche la codifica scelta per i file del progetto è JSON che è largamente utilizzata in questo campo data la semplicità della sintassi.
Lo sviluppo si è stato di tipo test driven utilizzando la libreria di Qt per testare tutti i componenti per diminuire la possibilità di errori nella logica del programma.
Ho utilizzato due strategie differenti per lo sviluppo, il componente dati (Model) è stato sviluppato con una strategia bottom-up ovvero partendo dall'elemento della struttura dati fino a la classe che comprende una struttura dati utilizzabile e navigabile. Il componente grafico (UI) è stato sviluppato invece con un approccio top-down ovvero prima è stata definita l'interfaccia globale poi sono state sviluppate tutte le funzionalità relative, di conseguenza è stato fatto un Modello particola che estende quello principale per adattarlo alle librerie di Qt per le UI.
Così facendo il Model originale rimane indipendente dall'interfaccia ed è possibile utilizzare lo stessa classe anche per un'applicazione non Qt.

Strumenti utilizzati

Per l'intera durata del progetto ho utilizzato un computer con installata una distribuzione di Manjaro derivante da Arch linux con kernel 5.0.21 il compilatore installato in questa macchina è il gcc in versione 8.3.0.
All'inizio dello sviluppo la versione di Qt presa in considerazione era la 5.12 dopo aver appurato problemi di compatibilità con le versioni precedenti sono passato alla 5.9.8 e fatto il porting del codice non funzionante, come per esempio QJsonDocument che perde il metodo operator[] e la possibilità di de-referenziare facilmente il contenuto delle varie parti.

Analisi dei requisiti

I requisiti principali del progetto erano:

- Una gerarchia di classi polimorfa
- Una struttura dati navigabile
- Un puntatore smart che evitasse la condivisione di memoria
- La separazione tra la logia del programma e l'interfaccia utente.
- Un'interfaccia grafica che permettesse la navigazione della struttura dati in modo agevole.

I requisiti sono stati rispettati implementando quanto richiesto e separando le classi per componente.

La struttura dati

La struttura dati scelta per il progetto è una lista doppiamente concatenata circolare.
Si è scelta questa struttura dati perché a seguito di un'implementazione piuttosto semplice si ottiene una struttura dati con un tempo di accesso, nel caso pessimo, ad un punto N qualsiasi della struttura dati in un tempo di N/2.
Questa struttura dati permette anche di riutilizzare gli iteratori perché una volta arrivati ad "past end" puntano a nullptr e incrementandoli ancora una volta tornano a puntare al primo valore dell'anello.

I Componenti

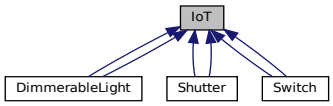
I componenti del progetto sono Container, IoT, QModel, UI.

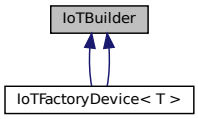
Il Componete Container.

Il componente Container contiene la classe template omonima e la classe template SmartPtr che implementa lo SmartPointer.
La classe template Container al suo interno ha due sotto classi, la classe Node, la quale rappresenta i nodi della struttura dati, e la classe Iterator che permette all'utilizzatore della struttura dati di muoversi all'interno di essa.

Il Componete IoT.

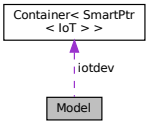
Il componente IoT contiene la classe IoT, che è la base della gerarchia, da essa derivano le classi Light Shutter e Switch che implementano un particolare dispositivo per la casa intelligente.
All'interno del componente IoT si trova anche la classe IoTBuilder il quale si occupa di servire istanze degli oggetti, da esso deriva la classe template IoTFactoryDevice che si occupa di fornire le "fabbriche" per ogni tipo di oggetto della gerarchia.
Il file iotdevices.h contiene i puntatori alle istanze di IoTFactoryDevice che servono per popolare la mappa statica di IoTBuilder.

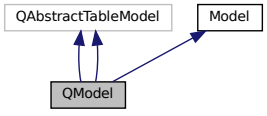




Il Componete QModel.

Il componente QModel contiene le classi Model e QModel.
La classe Model è la classe principale con cui si naviga e si riempie la struttura dati, essa permette il caricamento dei dati da file e la ricerca e il filtro dei dispositivi, attraverso questa classe è possibile anche impostare lo stato dei dispositivi e modificarli.
La classe QModel estende la classe Model e QabstractTableModel di Qt permettendo la comunicazione tra la struttura dati e l'interfaccia grafica.





Il Componete UI.

Il componente UI contiene il main dell’applicazione e tutte le classi necessarie alla creazione dell’interfaccia utente con cui interagire.

La classe MainWindow è il contenitore di tutti i moduli della UI, inizializza MainContent con un’istanza di QModel.

La classe MainContent estende QTabWidget ed è atta alla visualizzazione del contenuto del Model si occupa di inizializzare l’interfaccia di modifica e ricerca e di passare il risutato di queste interfacce alla struttura dati.

La classe InteractiveIot estende QDialog e si occupa di prendere in inputi i dati dell’utente per aggiungere o modificare un dispotivo IoT, la sua interfaccia cambia al cambiare della classe del dispositivo intelligente.

La classe ResearchView estende QDialog una volta istanziata si occupa di filtrare il contenuto del model per permettere all’utente di filtrare il contenuto della struttura dati e modificare ed eliminare specifici dati.

La classe CustomDelegate estende la classe QstyledItemDelegate che si preoccupa di modificare come vengono visualizzati i dati all’interno delle celle della vista tabellare, rendendo trasparente all'utente il tipo di codifica utilizzata per i dati.

Il Funzionamento

Inserimento

Per aggiungere un dispositivo alla struttura dati il model si aspetta in ingresso una stringa formattata in JSON come da esempio.

```
{
	"class":"classe del dispositivo",
	"name": "nome del dispositivo",
	"room": "stanza del dispositvo",
	"serial": "seriale del dispositivo",
	"status": "dipende dalla classe del dispositvo"
}
```

per ottenere le istruzione per impostare lo status del device smart bisogna istanziarlo utilizzando il metodo *IoTBuilder::getDevice* che una volta ricevuta la stringa ritorna un puntatore con l'oggetto richiesto.
Una volta ottenuto l'oggetto si invoca il metodo *getDeviceInstruction* dall'istanza ritornata dal metodo precedente che è poilimorfo e in base alla sua classe di implementazione viene restituito un oggetto QJsonDocument coerente con le impostazioni dell'istanza corrente.
Una volta utilizzato l'oggetto per le istruzioni si utilizza il metodo *setDevice* per impostare il dispositivo corrente e per aggiungerlo alla struttura si richiede la serializzazione dell'oggetto attraverso *JsonSerialize* che è un metodo che tutti gli oggetti che tipano IoT* ereditano.
Una volta ottenuta la serializzazione la si usa per richiamare sul model il metodo *addDevice* che si occupa di verificare se il seriale è già stato utilizzato all'interno della corrente strutta dati e di inserirlo se mancante.

Modifica

Per modificare un dispositivo i passaggi sono analoghi, ma il model mette a disposizione due metodi differenti per questo scopo.
setDeviceStatus(const std::string& status);
Questo metodo è meno efficiente in quanto scorre tutta la struttura dati per trovare il dispositivo con il seriale corrispondente e poi lo modifica.
setDeviceStatus(const std::string& status, int index)
Questo metodo prende l'oggetto che si trova in posizione index, ne verifica il seriale e se combacia procede alla modifica dell'oggetto.

Eliminazione

Per eliminare un determinato elemento c'è il metodo removeDevice all'interno del model che ha come parametro formale una stringa completa della serializzazione dell'oggetto.
Di questa stringa prende il seriale e lo ricerca dopo di che invoca il metodo di eliminazione sull'iteratore.

Caricare

Per caricare una struttura dati nel model si usa il metodo *load* che come parametro formale ha una stringa che rappresenta il path di un file formattato come segue.

```
{"0":{
    "class": "shutter",
    "name": "Tapparella Sala",
    "room": "room25",
    "serial": "CCCCCC10",
    "status": {
        "height": 0}
     }
,"1":{
    "class": "shutter",
    "name": "Tapparella Cucina",
    "room": "room100",
    "serial": "CCCCCC11",
    "status": {
        "height": 0}
     }
}
```

Se prima di caricare un nuovo file nella struttura dati c'erano degli oggetti, essi vengono eliminati.

Salvare

Per salvare il corrente stato della struttura dati il model mette a disposizione il metodo *save* che ha come parametro formale una stringa che rappresenta il path di salvataggio del file, se non viene fornito sovrascriverà l'ultimo file caricato.

Filtrare

Per filtrare i dispositivi contenuti nel model, si utilizza il metodo *filterDevicesForAttribute* questo metodo prende in ingresso una stringa che è una serializzazione, anche parziale, di un dispositivo restituisce una stringa che è la serializzazione di tutti i dispositivi che soddisfano i parametri richiesti.

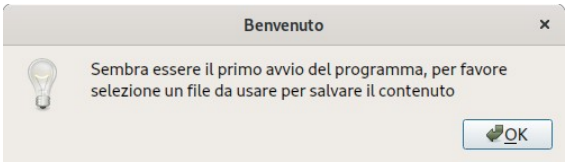
Come compilare

Per compilare correttamente il progetto bisogna compilare ogni componete separatamente e in ordine di dipendenza per semplificare l'operazione viene fornito uno script "MakeScript.sh" che si occupa di compilare e di creare un collegamento all'eseguibile.
Lo script esegue il qmake con il nome del componente seguito dall'estensione pro e successivamente esegue il comando make, in quanto ogni componente è stato sviluppato come progetto indipendente.

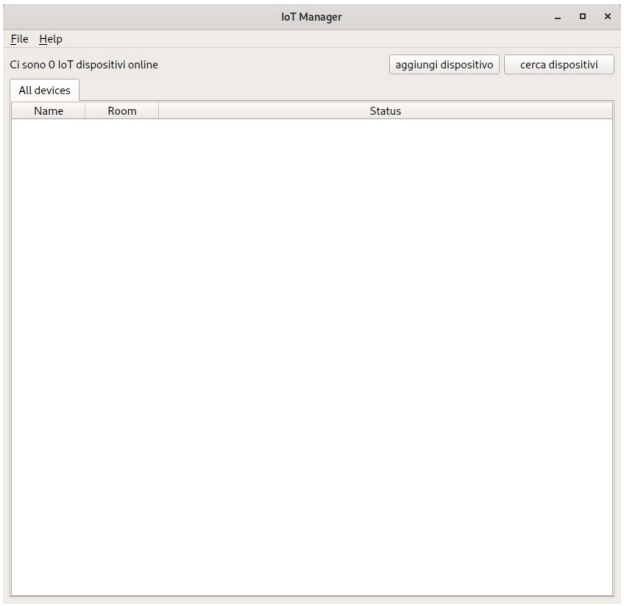
Istruzioni per l'uso della UI

Il Primo Avvio

Al primo avvio si verrà accolti da un messaggio che vi chiederà di selezionare dove salvare i dati dell'esecuzione corrente e successivamente si aprirà una finestra per selezionare il percoso.



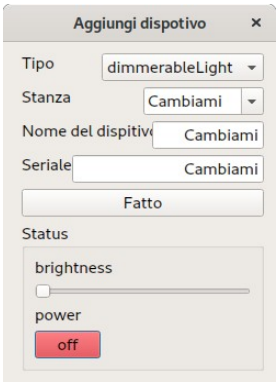
Dopodiché verrà presentata l'interfaccia grafica del programma vuota



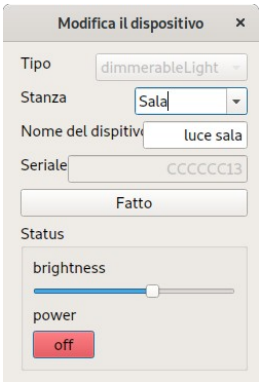
Da qui sarà possibile utilizzando i pulsati appositi aggiungere un dispositivo o aprire la funzione di ricerca.

L’aggiunta o modifica di un dispositivo

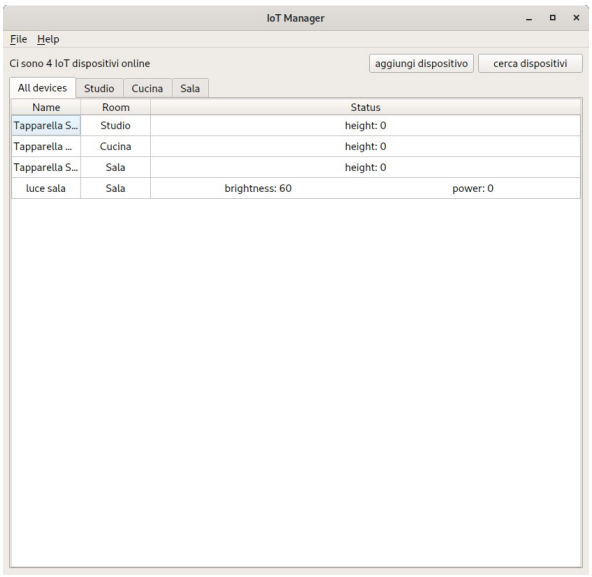
Quando si aggiunge un dispositivo viene caricata questa interfaccia



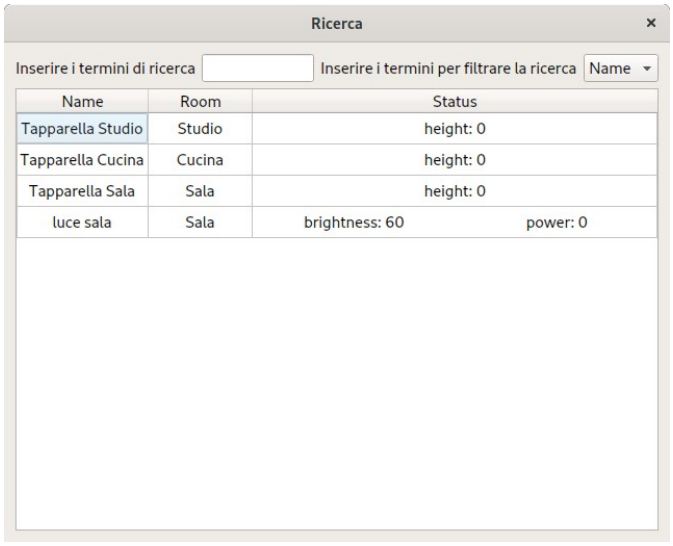
I campi con scritto “Cambiami” non permetteranno di completare l’operazione se non modificati.
Per aggiungere una stanza dopo aver scritto il nome bisogna confermare la creazione con il tasto invio sulla tastiera altrimenti non verrà recepito il cambiamento.
Una volta popolato il programma sarà possibile modificare i vari dispositivi facendovi doppio click o facendo click destro sull’elemento desiderato e poi selezionare la voce Modifica.



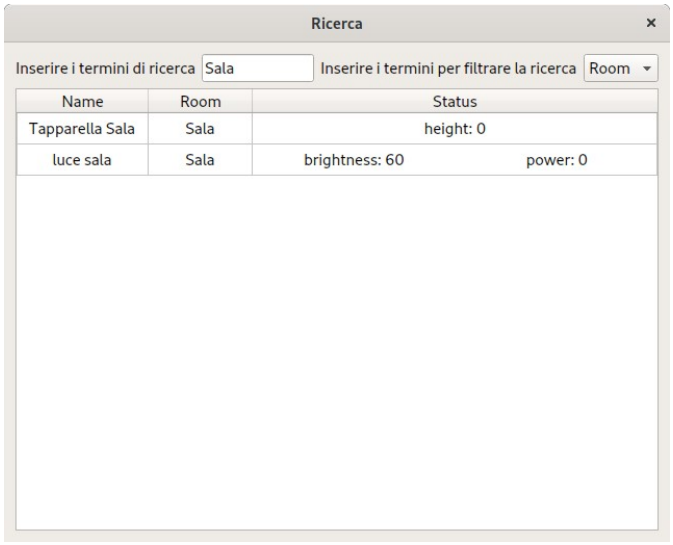
Con questa interfaccia sarà possibile modificare il nome dell’oggetto smart, la sua stanza e lo stato del dispositivo.



La ricerca di un dispositivo.
Quando si vuole cercare un dispositivo facendo click sul pulsante “cerca dispositivi” si aprirà la seguente finestra di ricerca in tempo reale

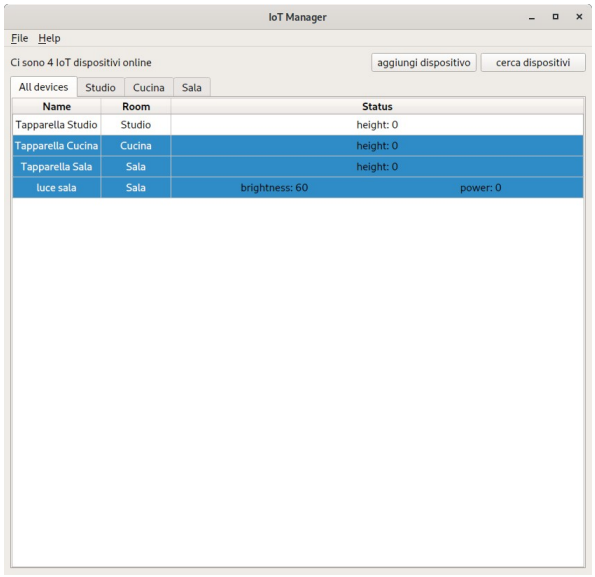


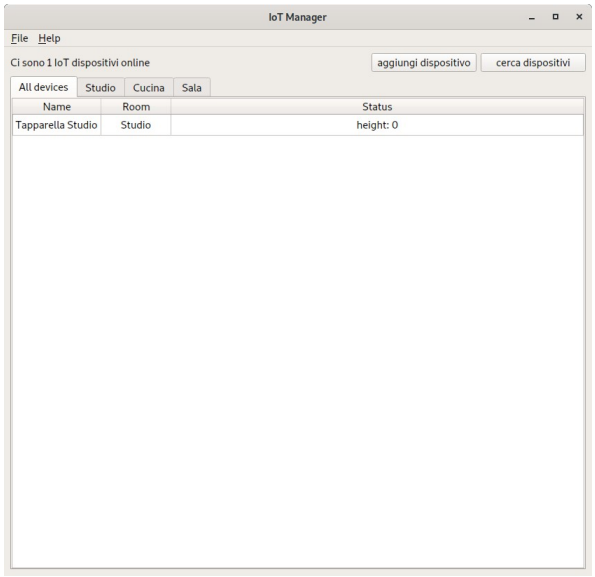
Una volta modificati i campi di ricerca verranno visualizzati i dispositivi corrispondenti



L’eliminazione di uno più dispositivi.

Quando si vuole eliminare uno o più dispositivi si selezionano le righe interessate e si fa tasto destro su gli elementi che si vogliono eliminare e si seleziona la voce Rimuovi.





Aprire Salvare ed Uscire

Le funzionalità di apertura di un file, il salvataggio di esso e la chiusura del programma sono situate nel menù di intestazione File.

Attenzione il programma non dispone di una funzionalità di salvataggio automatico, quindi una volta chiuso il programma ogni modifica non salvata andrà perduta.

La distribuzione del tempo

Il progetto ha richiesto all'incirca 60 ore compresa la stesura della relazione.
Circa 5 ore sono state spese nella fase di progettazione e organizzazione dei componenti di lavoro.
Il tempo di sviluppo della gerarchia principale e del suo Builder è stato abbastanza breve, circa 10 ore
La maggior parte del tempo, più o meno 30 ore, è stata spesa facendo testing e sviluppo sul Container e sul Model.
Per sviluppare l'interfaccia grafica è sono bastate 8 ore.
La relazione ha preso circa 6 ore di tempo per essere completata.
Il restante è stato speso facendo test generali sull'applicazione e cercando di ottimizzare l'uso della memoria.

Pensieri finali

Questo progetto mi ha permesso di prendere dimestichezza con il linguaggio C++ ed in particolare con la libreria Qt.
Lavorando con questa libreria mi ha insegnato ad utilizzare in maniera efficiente la documentazione e di costruire più velocemente un'applicazione completa.
Questa versione dell'applicazione ha però dei difetti congeniti, la struttura dati non è la più robusta e l'uso di un puntatore smart non standard non aiuta, sarebbe preferibile l'utilizzo di strutture dati standard in quanto più veloci ed efficienti