Ingegneria del software
Elaborato

Martina Buccioni, Cosimo Tanganelli

1 Introduzione

L'elaborato ha lo scopo di simulare un contesto fisico georeferenziato e strutturato in aree: dove al suo interno ci sono collocati device. Inoltre, sono presenti dei servizi che usano una composizione degli stessi device. In particolare, noi abbiamo scelto come riferimento al contesto fisico, una casa intelligente, composta da 7 stanze principali. I dispositivi fanno parte di 4 macrocategorie, come luci, termostati, allarmi e rilevazione presenza. I servizi sono molteplici e coprono tutte le aree della casa e tutti i device.

Inoltre il nostro elaborato si pone l'obbiettivo di associare ad ogni servizio un digitalTwin a cui vengono notificati le osservazioni raccolte dai vari service. Le notifiche ai digitalTwin vengano fatte sia quando viene acceso il servizio corrispondente, ma anche quando vengono accesi dispositivi che fanno parte del servizio.

Nel nostro caso, il digitalTwin oltre a raccogliere le informazioni, scriverà su un file.txt le informazioni stesse e infine verrà creato un grafico a barre dei vari utilizzi dei servizi.

2 Implementazione

Per l'implementazione del progetto si sono sviluppate 2 logiche ben distinte: la prima che gestisce il composite di servizi e dispositivi, e la seconda che si relaziona con il database.

2.1 Codice

Il codice usa principalemente due Design Pattern: Composite, Observer. Lo scopo del pattern **Composite** è quello di consentire la creazione di composizioni di dispositivi e servizi utilizzando un'interfaccia comune. Questo significa che è possibile trattare un singolo dispositivo o un gruppo di dispositivi come entità composite, consentendo un'interazione uniforme con essi.

Grazie all'utilizzo del pattern **Composite,** è possibile gestire in modo efficiente le operazioni sui dispositivi e sui servizi, indipendentemente dalla loro struttura o gerarchia. Ad esempio, è possibile accendere o spegnere un singolo dispositivo, ma anche eseguire azioni sulle composizioni di dispositivi come un gruppo.

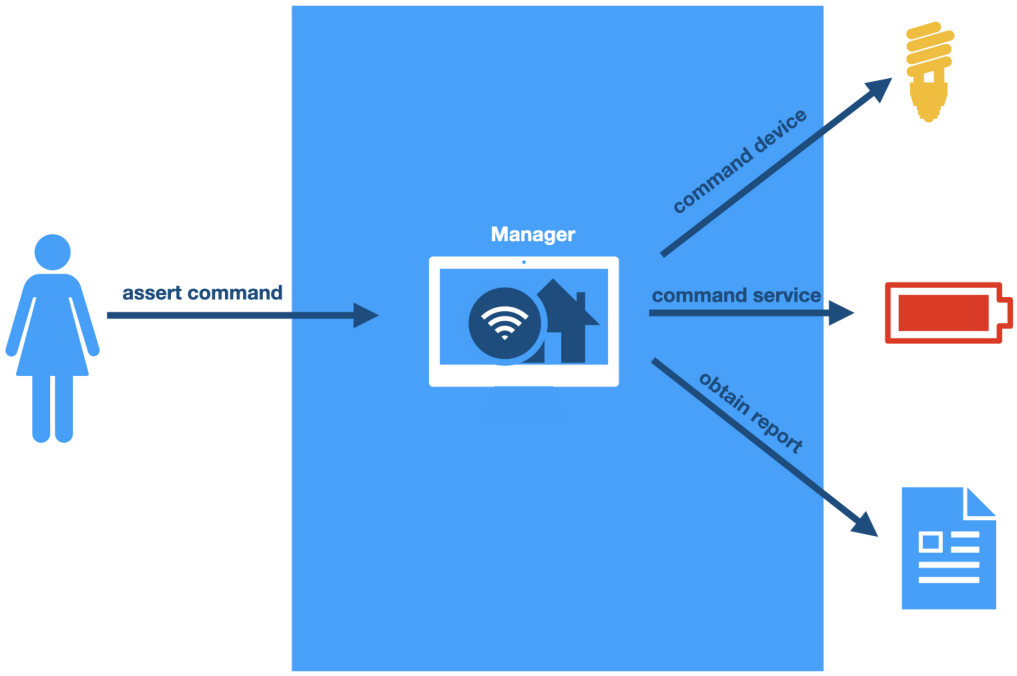
Oltre al pattern Composite, è stato utilizzato anche il pattern **Observer** che consente la gestione delle notifiche ai digitalTwin quando vengono raccolte nuove osservazioni dai vari servizi. I digitalTwin si registrano come osservatori presso i servizi rilevanti e vengono notificati automaticamente quando si verificano eventi di interesse. Questo permette al digitalTwin di aggiornarsi in tempo reale con i dati raccolti e prendere le azioni necessarie in base alle osservazioni ricevute.

Inoltre, nell'implementazione del lato back-end, si è adottato l'approccio dei **DTO (Data Transfer Object)** e dei **DAO (Data Access Object)** per gestire l'interazione tra il database e il lato di business.

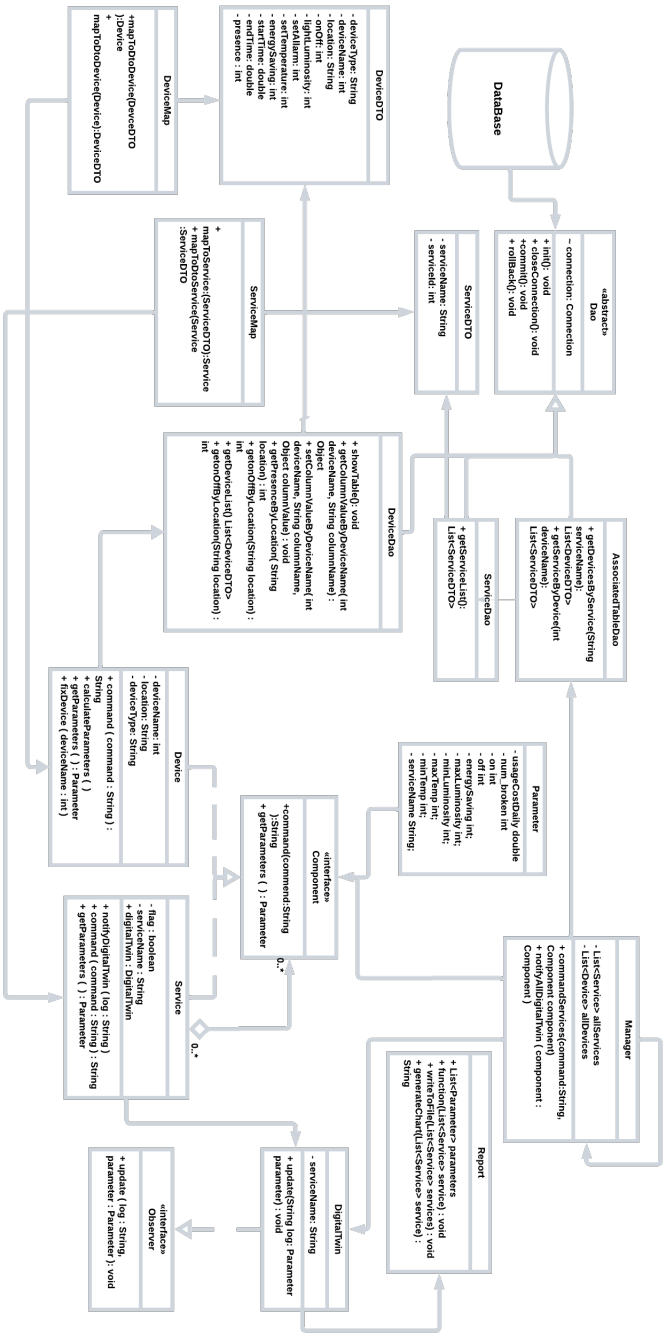
I DTO sono stati utilizzati per rappresentare in modo strutturato e coerente i dati delle tabelle coinvolte nel progetto, ovvero le tabelle "device", "servizi" e "associata". Ogni DTO corrisponde a una tabella specifica e contiene campi che rappresentano le colonne della tabella stessa. L'utilizzo dei DTO facilita la trasmissione dei dati tra il database e codice, permettendo una rappresentazione chiara e standardizzata delle informazioni.

D'altra parte, i DAO sono stati implementati per gestire l'accesso e la manipolazione dei dati nel database. Ogni tabella coinvolta ha un DAO corrispondente, che fornisce metodi per eseguire operazioni di creazione, lettura, aggiornamento ed eliminazione dei dati. I DAO astraggono il livello di persistenza dei dati, consentendo al resto dell'applicazione di interagire con il database in modo indipendente dalla sua specifica implementazione.

2.1.1 Use Case Diagram



2.1.2 Class Diagram



2.1.3 Classi principali

- **Component**

  La classe rappresentata dall'interfaccia Component definisce un'astrazione per un componente generico all'interno del sistema. Questo componente è responsabile di eseguire comandi specifici e fornire i parametri relativi.

  La classe contiene i seguenti metodi:

  - **command(String command):** Questo metodo prende in input un comando come stringa e restituisce una stringa come risultato. Il comando passato come argomento rappresenta un'azione o un'operazione specifica che deve essere eseguita dal componente. Il metodo può generare un'eccezione di tipo SQLException se si verificano errori durante l'esecuzione del comando.
  - **getParameters():** Questo metodo restituisce un oggetto di tipo Parameter che rappresenta i parametri associati al componente. I parametri possono essere utilizzati per configurare o personalizzare il comportamento del componente.

    L'interfaccia Component definisce un contratto che deve essere implementato dalle classi concrete che rappresentano i diversi tipi di componenti nel sistema. Le classi che implementano questa interfaccia avranno la responsabilità di fornire un'implementazione concreta dei metodi definiti nell'interfaccia.

- **Device**

  La classe Device rappresenta un dispositivo all'interno del sistema e implementa l'interfaccia Component. Questa classe contiene vari attributi che descrivono le caratteristiche del dispositivo, come il nome, la posizione, il tipo di dispositivo, la temperatura, il numero di volte che il dispositivo è rotto, acceso, spento, in modalità di risparmio energetico e i valori massimi e minimi di luminosità e temperatura.

  La classe Device fornisce un costruttore che accetta il nome del dispositivo, la posizione, il tipo di dispositivo e un oggetto DeviceDao che viene utilizzato per l'interazione con il database.

  - **command(String command):**La classe implementa il metodo command definito nell'interfaccia Component. Questo metodo consente di eseguire comandi specifici sul dispositivo, come accensione, spegnimento, regolazione della luminosità o della temperatura. La logica dei comandi è gestita all'interno di uno switch-case, in cui viene eseguita un'azione specifica in base al comando fornito come parametro.
  - **Altri metodi** La classe Device include anche altri metodi come CalculateParameters, che calcola i parametri del dispositivo, come i costi di utilizzo giornalieri e il numero di volte che il dispositivo è stato rotto, acceso, spento, in modalità di risparmio energetico e i valori massimi e minimi di luminosità e temperatura. Il metodo getParameters restituisce un oggetto Parameter che rappresenta questi parametri.

- **Service**

  La classe 'Service' rappresenta un servizio nel sistema e implementa l'interfaccia 'Component'. Questa classe contiene attributi che descrivono il nome del servizio, una lista di dispositivi associati al servizio, un'istanza di 'DigitalTwin', un flag e vari mappe per tenere traccia del numero di dispositivi guasti, accesi, spenti, in modalità di risparmio energetico e i valori massimi e minimi di luminosità e temperatura.

  Il costruttore della classe accetta il nome del servizio e inizializza gli attributi associati. Viene anche istanziato un oggetto 'DigitalTwin' associato al servizio.

- **Command(String string):**La classe implementa il metodo 'command' definito nell'interfaccia 'Component'. Questo metodo esegue un comando specifico su tutti i dispositivi associati al servizio utilizzando un ciclo for. Il risultato del comando viene memorizzato nella variabile 'com', che viene quindi utilizzata per notificare il 'DigitalTwin' con il messaggio corrispondente. Infine, il metodo restituisce il messaggio di notifica.
- **getParameters():**La classe implementa anche il metodo 'getParameters' dell'interfaccia 'Component'. Questo metodo restituisce un oggetto 'Parameter' che rappresenta i parametri del servizio, come il costo di utilizzo giornaliero, il numero totale di dispositivi guasti e le mappe che tengono traccia del numero di dispositivi guasti, accesi, spenti, in modalità di risparmio energetico e i valori massimi e minimi di luminosità e temperatura.
- **notifyDigitalTwin():**La classe ha anche il metodo 'notifyDigitalTwin', che aggiorna il 'DigitalTwin' con un messaggio e i parametri del servizio.

- **DigitalTwin** La classe DigitalTwin implementa l'interfaccia Observer ed è responsabile di rappresentare il "gemello digitale" del servizio nel sistema. La classe contiene attributi per il nome del servizio, un logger per l'applicazione e un oggetto Report.

  Il costruttore della classe accetta il nome del servizio e inizializza l'attributo serviceName.

  - **Update**La classe implementa il metodo update, definito nell'interfaccia Observer. Questo metodo viene chiamato quando il servizio notifica il DigitalTwin con un messaggio e i parametri aggiornati. Nel metodo, il messaggio viene registrato nel logger dell'applicazione per scopi di tracciamento. Viene quindi creato un oggetto Report e i parametri vengono aggiunti alla lista del rapporto.

- **Manager**La classe Manager rappresenta il gestore principale del sistema. Contiene una lista di tutti i servizi (allServices) e una lista di tutti i dispositivi (allDevices), insieme a un oggetto AssociatedTableDao per la gestione delle tabelle associate.

  La classe dispone di un costruttore che inizializza le liste dei servizi e dei dispositivi e inizializza anche l'oggetto AssociatedTableDao. La gestione dell'eccezione viene inoltrata nel costruttore.

  - **notifyAllDigitalTwin():** Il metodo notifyAllDigitalTwin è responsabile di notificare tutti i "gemelli digitali" associati a un componente (servizio o dispositivo). Se il componente è un dispositivo, vengono scansionati tutti i servizi per verificare se il dispositivo è presente nella lista dei dispositivi di un servizio. In tal caso, viene notificato il rispettivo "gemello digitale". Se il componente è un servizio, vengono notificati tutti i dispositivi associati al servizio. Il metodo notifyAllDigitalTwin viene richiamato dopo l'esecuzione di un comando su un componente.
  - **commandService():**Il metodo commandService esegue un comando sul componente specificato e quindi richiama il metodo notifyAllDigitalTwin per notificare i "gemelli digitali" associati.

- **Device Dao, ServiceDao, AssociatedTableDao:** Queste classi sono responsabili dell'interazione con il database e dell'esecuzione di query per recuperare o aggiornare i dati.

  - **DeviceDao:**La classe 'DeviceDao' estende la classe 'Dao' e contiene i metodi per inizializzare e chiudere la connessione con il database. Inoltre, fornisce metodi per eseguire query specifiche per la tabella dei dispositivi, come la visualizzazione di tutti i dispositivi ('showTable'), l'ottenimento del valore di una colonna per un dispositivo specifico ('getColumnValueByDeviceName'), l'impostazione del valore di una colonna per un dispositivo specifico ('setColumnValueByDeviceName'), l'ottenimento della presenza per una determinata posizione ('getPresenceByLocation'), l'ottenimento dell'accensione/spegnimento per una determinata posizione ('getonOffByLocation') e l'ottenimento di una lista di oggetti 'DeviceDto' che rappresentano i dispositivi presenti nella tabella ('getDeviceList').

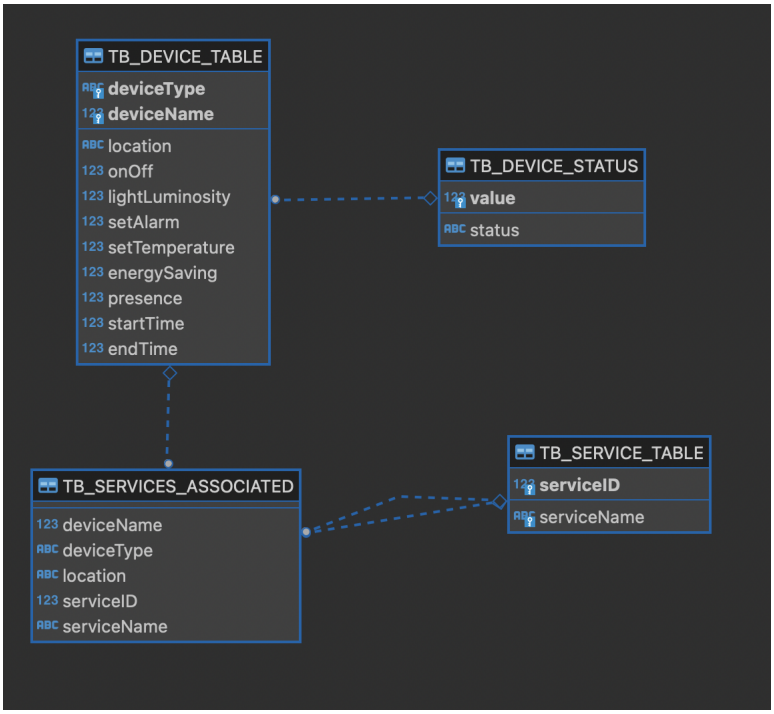
- **ServiceDao:** La classe ‘ServiceDao‘ estende la classe ‘Dao‘ e contiene il metodo ‘getServiceList‘ per ottenere una lista di oggetti ‘ServiceDto‘ che rappresentano i servizi presenti nella tabella dei servizi.
- **AssociatedTableDao:** La classe ‘AssociatedTableDao‘ estende la classe ‘Dao‘ e contiene i metodi per recuperare le informazioni associate tra dispositivi e servizi dalla tabella associata. I metodi ‘getDevicesByService‘ e ‘getServicesByDevice‘ consentono di ottenere rispettivamente una lista di oggetti ‘DeviceDto‘ associati a un determinato servizio e una lista di oggetti ‘ServiceDto‘ associati a un determinato dispositivo.

  Tutte queste classi utilizzano la connessione al database gestita dalla classe ‘ConnectionManager‘ e forniscono metodi per eseguire query e manipolare i dati del database.

- **DeviceDto, ServiceDto e AssociatedTableDTO:** Le classi DeviceDto, ServiceDto e AssociatedTableDTO sono classi di oggetti di dati che vengono utilizzate per rappresentare i dati recuperati dal database.

2.2 Database

2.2.1 DB Diagram



Il database è costituito da diverse tabelle che contengono informazioni sui dispositivi, lo stato dei dispositivi e i servizi associati. Ecco una panoramica delle tabelle nel database:

- **TB DEVICE STATUS:**

  La tabella ”TB DEVICE STATUS” memorizza lo stato dei dispositivi. Ha le seguenti colonne:

  - **status (VARCHAR(20)):** Rappresenta lo stato del dispositivo.
  - **value (INT, PRIMARY KEY):** Rappresenta il valore corrispondente allo stato.

- **TB DEVICE TABLE** La tabella "TB DEVICE TABLE" contiene le informazioni sui dispositivi. Ha le seguenti colonne:
  - **deviceType (VARCHAR(20)):** Rappresenta il tipo di dispositivo.
  - **deviceName (INT):** Rappresenta il nome del dispositivo.
  - **location (VARCHAR(20)):** Rappresenta la posizione del dispositivo.
  - **onOff (INT):** Rappresenta lo stato di accensione/spegnimento del dispositivo.
  - **lightLuminosity (INT):** Rappresenta la luminosità della luce del dispositivo.
  - **setAlarm (INT):** Rappresenta l'impostazione dell'allarme del dispositivo.
  - **setTemperature (INT):** Rappresenta l'impostazione della temperatura del dispositivo.
  - **energySaving (INT):** Rappresenta lo stato del risparmio energetico del dispositivo.
  - **presence (INT):** Rappresenta la presenza del dispositivo.

    La chiave primaria della tabella è costituita dalla combinazione delle colonne deviceName e deviceType. La colonna onOff è una chiave esterna che fa riferimento alla tabella "TB DEVICE STATUS".

- **TB SERVICE TABLE**

  La tabella "TB SERVICE TABLE" contiene le informazioni sui servizi. Ha le seguenti colonne:
  - serviceName (VARCHAR(20)): Rappresenta il nome del servizio.
  - serviceID (INT, AUTO INCREMENT, PRIMARY KEY): Rappresenta l'ID univoco del servizio.

- **TB SERVICES ASSOCIATED** La tabella "TB SERVICES ASSOCIATED" memorizza le associazioni tra dispositivi e servizi. Ha le seguenti colonne:
  - **deviceName (INT):** Rappresenta il nome del dispositivo associato.
  - **deviceType (VARCHAR(20)):** Rappresenta il tipo di dispositivo associato.
  - **location (VARCHAR(20)):** Rappresenta la posizione del dispositivo associato.
  - **serviceID (INT):** Rappresenta l'ID del servizio associato.
  - **serviceName (VARCHAR(20)):** Rappresenta il nome del servizio associato.

  Le colonne deviceName, deviceType e location fanno riferimento alle colonne corrispondenti nella tabella "TB DEVICE TABLE", mentre le colonne serviceID e serviceName fanno riferimento alle colonne corrispondenti nella tabella "TB SERVICE TABLE".

3 Altre tecnologie

3.1 Log4J

Log4j è una popolare libreria di logging per Java che fornisce un framework flessibile e configurabile per la gestione dei log.

Nel contesto del nostro progetto, abbiamo utilizzato Log4j per registrare informazioni, avvisi, errori e altri messaggi di log. Questa libreria permette di avere un controllo completo sulla registrazione dei log e di definire diversi livelli di log per gestire la verbosità delle informazioni registrate.

esempio codice:

```
loggerApplication.error(e.getMessage(), e);
```

3.2 Junit

JUnit è un framework di test unitari per il linguaggio di programmazione Java. È ampiamente utilizzato per automatizzare i test di unità e facilitare la scrittura di test robusti, ripetibili e facilmente eseguibili.

Nel nostro progetto, abbiamo utilizzato JUnit per scrivere e eseguire i test delle funzionalità del nostro codice. I test unitari sono stati creati come metodi annotati con l'annotazione '@Test', che indica a JUnit che il metodo è un test da eseguire.

esempio codice:

```
    @Test
public void testSelectDevice() throws Exception {
  Manager manager = Manager.getManager();
    // Creazione di un dispositivo
  DeviceDao dao = new DeviceDao();
  dao.init();
    Device device = new Device(1, "bagno", "luce",dao );

    // Aggiunta del dispositivo al manager
    manager.addDevices(Collections.singletonList(device));

    // Selezione del dispositivo dal manager
    Device selectedDevice = manager.selectDevice(1);

    // Verifica che il dispositivo selezionato corrisponda al dispositivo aggiunto
    assertEquals(device, selectedDevice);
};
```

3.3 Metodi per la velocizzazione del codice

3.3.1 Funzioni Lambda

Le funzioni lambda sono una caratteristica potente di Java che permette di scrivere codice più conciso, leggibile ed efficiente. Nel contesto del nostro progetto, abbiamo fatto uso delle funzioni lambda per semplificare l'iterazione e l'elaborazione di collezioni di oggetti.

Nel codice, le funzioni lambda sono state utilizzate per eseguire operazioni su liste di servizi e dispositivi. Ad esempio, abbiamo utilizzato stream() per creare un flusso di elementi da una lista e forEach() per eseguire una determinata azione su ciascun elemento del flusso.

Le funzioni lambda ci hanno permesso di specificare in modo conciso le azioni da eseguire per ciascun elemento. Abbiamo utilizzato le lambda per filtrare, mappare o eseguire altre operazioni specifiche su ciascun elemento delle liste.

Questo approccio ci ha permesso di scrivere codice più leggibile ed espressivo, riducendo la quantità di codice boilerplate. Le funzioni lambda ci hanno aiutato a migliorare la modularità del nostro progetto, consentendoci di scrivere logicamente le operazioni per ogni elemento in modo più chiaro e focalizzato.

Esempio codice:

```
    allServices.stream().forEach(s->{
     try {
        addDeviceToService(s);
     } catch (SQLException e) {
          e.printStackTrace();
     }
});
```

3.3.2 Lombok

La libreria Lombok offre l'annotazione @Data, che è estremamente utile per semplificare la creazione di classi Java. Applicando questa annotazione a una classe, Lombok genera automaticamente una serie di metodi comuni, riducendo la quantità di codice ripetitivo che altrimenti dovremmo scrivere manualmente.

Con @Data, Lombok genera i metodi toString(), equals(), hashCode() e getters/setters per tutti i campi della classe. Questo permette di ottenere in poche righe di codice una classe che implementa tutti questi metodi fondamentali per la gestione dei dati.

3.3.3 Classe Static Value

La StaticValue classe contiene una serie di variabili statiche, dichiarate come costanti (final), che memorizzano valori di stringa associati a diverse funzionalità, dispositivi e servizi.

Alcuni esempi di costanti presenti nella classe includono:

On: rappresenta il valore "accendi". Off: rappresenta il valore "spengi". IncreasedLuminosity: rappresenta il valore "aumenta luminosità". DecreasedLuminosity: rappresenta il valore "diminuisci luminosità". EnableEnergySaving: rappresenta il valore "inserisci risparmio energetico". DisableEnergySaving: rappresenta il valore "disinserisci risparmio energetico". luce: rappresenta il valore "luce". termostato: rappresenta il valore "termostato". La classe StaticValue contiene anche altre costanti associate a funzionalità specifiche come temperatura, presenza, allarmi e varie aree della casa.

L'uso di queste costanti può rendere il codice più leggibile e manutenibile, poiché evita l'uso di stringhe "hard-coded" (stringhe direttamente inserite nel codice) e consente di riferirsi ai valori desiderati utilizzando i nomi delle costanti definite nella classe.

Ad esempio, se desideri accendere tutte le luci in casa, puoi utilizzare la costante OnAllLights invece di scrivere direttamente la stringa "OnAllLights". Questo rende il codice più chiaro e meno soggetto a errori di battitura.

In generale, l'utilizzo di costanti statiche come quelle definite nella classe StaticValue può contribuire a una programmazione più robusta e comprensibile.

3.4 Classe Report

La classe Report svolge un ruolo importante nella generazione di report dettagliati. È responsabile per creare un grafico a barre e scrivere i dati su un file di testo.

Il metodo function(List¡Service¿ service) è il punto di ingresso per generare il report completo. Questo metodo prende in input una lista di oggetti Service che rappresentano i dati da analizzare. All'interno del metodo, vengono chiamati due metodi ausiliari: generateChart(service) e writeToFile(service).

Il metodo addToList(Parameter parameter) è utilizzato per aggiungere un oggetto Parameter alla lista parameters. Questo metodo svolge un ruolo importante nel popolare la lista con i dati necessari per generare il report.

Il metodo generateChart(List(Service) service) è responsabile per la creazione di un grafico a barre utilizzando la libreria JFreeChart. Questo metodo itera attraverso la lista di oggetti Service passati come argomento e estrae il valore UsageCostDaily da ciascun oggetto Service. I valori vengono poi utilizzati per popolare un dataset di tipo DefaultCategoryDataset. Una volta che il dataset è pronto, viene creato un oggetto JFreeChart che rappresenta il grafico a barre. Infine, la rappresentazione testuale del grafico viene restituita come risultato del metodo.

Il metodo writeToFile(List¡Service¿ services) si occupa di scrivere i dati dei servizi su un file di testo denominato "file.txt". All'interno del metodo, viene creato un oggetto FileWriter per gestire l'operazione di scrittura. Successivamente, vengono iterati gli oggetti Service nella lista services. Per ogni oggetto Service, i dati vengono estratti e scritti nel file di testo utilizzando il metodo write() dell'oggetto FileWriter. Alla fine, il writer viene chiuso per garantire che i dati siano correttamente salvati sul file.

3.4.1 Risultati

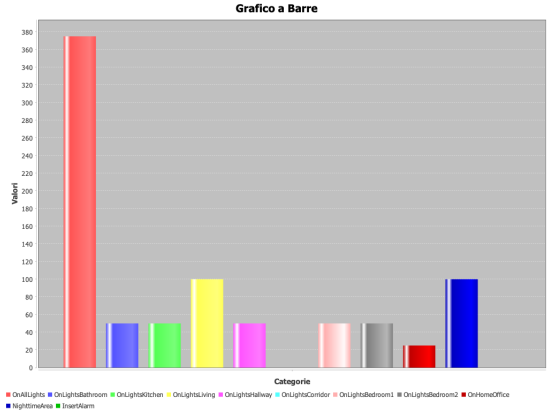


Figure 1: Grafico a barre del costo dell'elettricità rispetto ai vari service

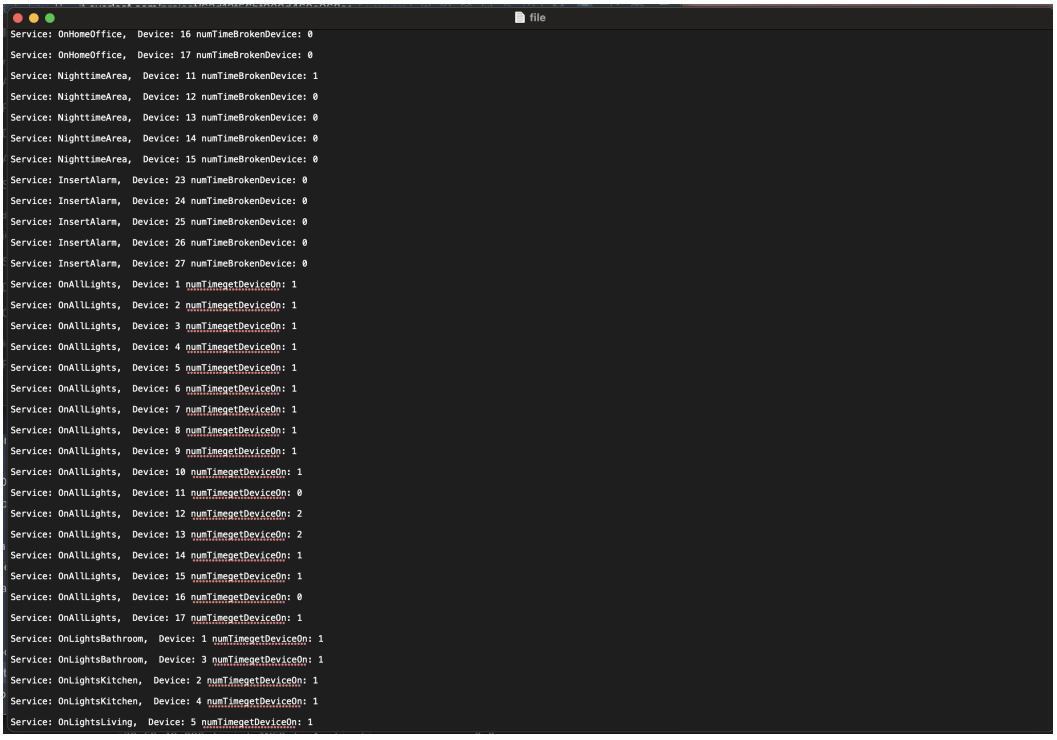


Figure 2: File txt dei vari parametri legati ai servizi