Manuale sviluppatore

Dream Corp.

06/05/2019

G&B

Informazioni sul documento	
Versione	Versione 1.0.0
Responsabile	Davide Ghiotto
Redattori	Matteo Bordin
	Davide Liu
Verificatori	Gianluca Pegoraro
Uso	Esterno
Destinatari	Dream Corp.
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Zucchetti SpA

Versione	Data	Descrizione	Autore	Ruolo
1.0.0	06/05/2019	Approvazione per il rilascio RA	Davide Ghiotto	Responsabile
0.1.2	30/04/2019	Verifica documento	Marco Davanzo	Verificatore
0.1.1	29/04/2019	Correzione errori in seguito all'esito della RQ	Davide Liu	Verificatore
0.1.0	8/04/2019	Approvazione per il rilascio RQ	Marco Davanzo	Responsabile
0.0.9	7/04/2019	Verifica documento	Gianluca Pegoraro	Verificatore
0.0.8	6/04/2019	Modifica contenuti §7	Pietro Casotto	Progettista
0.0.7	1/04/2019	Aggiunta immagini di §6 e §7	Michele Clerici	Progettista
0.0.6	28/03/2019	Modifica di §8.2	Michele Clerici	Progettista
0.0.5	24/03/2019	Scrittura §8	Davide Liu	Progettista
0.0.4	23/03/2019	Scrittura §6.3 e §7.1	Davide Liu	Progettista
0.0.3	21/03/2019	Scrittura §4, §5, §6.1 e §6.2	Pietro Casotto	Progettista
0.0.2	21/03/2019	Scrittura §1, §2 e §3	Michele Clerici	Progettista
0.0.1	20/03/2019	Creazione struttura del documento	Davide Liu	Progettista

Indice

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Questo documento ha come scopo quello di presentare le informazioni necessarie per estendere, correggere e migliorare il nostro plugin per Grafana. Il documento conterrà anche le informazioni necessarie per impostare l'ambiente di lavoro in modo che sia il più simile possibile all'ambiente dove è stato sviluppato. Questa guida è stata scritta avendo in mente sistemi Windows e Linux. Altri sistemi potrebbero creare problemi di compatibilità, anche se è una cosa poco probabile.

1.2 Scopo del progetto

Lo scopo del progetto è quello di realizzare un plugin$_{\mathbf{G}}$ per Grafana$_{\mathbf{G}}$ per integrare metodi di intelligenza artificiale$_{\mathbf{G}}$ al flusso dei dati raccolti con lo scopo di monitorare lo stato del sistema e migliorare il software utilizzato.

1.3 Note esplicative

Allo scopo di evitare ambiguità a lettori esterni al gruppo, si specifica che all'interno del documento verranno inseriti dei termini con un carattere G come pedice, questo significa che il significato inteso in quella situazione è stato inserito nel *Glossario v4.0.0*.

1.4 Riferimenti

1.4.1 Riferimenti per l'installazione

- **Git**
  - Windows: https://gitforwindows.org/
  - Linux: https://git-scm.com/
- **NodeJs**
  - https://nodejs.org/en/
- **Grafana**
  - https://grafana.com/

- **Webstorm**
  - https://www.jetbrains.com/webstorm/
- **InfluxDB**
  - https://www.influxdata.com/
- **Telegraf**
  - https://www.influxdata.com/time-series-platform/telegraf/

1.4.2 Riferimenti legali

- **Licenza MIT**
  - https://opensource.org/licenses/MIT

2 Installazione

2.1 Requisiti minimi

2.1.1 Ambiente di lavoro

- Grafana 5.4.x o superiore
- Node.js v10.15.3
- NPM v6.9

2.1.2 Browser

- Google Chrome v40.0.2214
- Safari v8.0
- Firefox v32

2.2 Set up

Scaricare ed estrarre la cartella bayesian-grafana-plugin, copiarla dal supporto fisico allegato alla documentazione oppure scaricare il codice tramite Git con il comando:

```
git clone https://github.com/dreamcorpsw/bayesian-grafana-plugin.git
```

Affinché Grafana possa vedere correttamente il plugin è necessario copiarlo nella cartella `/grafana/plugins` presente dentro la cartella di Grafana. Successivamente, aprire il terminale e raggiungere la root del plugin per lanciare il comando:

```
npm install
```

`NPM` provvederà a scaricare tutte le dipendenze necessarie, tra le quali la libreria JsBayes.
In caso di problemi controllare che tutti i requisiti siano stati soddisfatti, che funzionino correttamente e che siano inclusi nella variabile d'ambiente. Per maggiori informazioni riguardo le variabili d'ambiente e la loro configurazione controllare l'help del proprio sistema operativo.

2.3 Esecuzione

Per rendere operativo l'applicativo, eseguire sempre nel terminale della cartella dove è inserito il plugin il seguente comando:

```
npm run build
```

Successivamente, far partire il server di Grafana il cui launcher si trova nella cartella `bin` dentro la root principale di Grafana oppure tramite il comando `sudo service grafana-server start`.
A questo punto, aprire il browser e collegarsi all'indirizzo del server dove è installato Grafana seguito dall'eventuale numero di porta utilizzata.
I.E.: se il server è installato sulla stessa macchina da cui si vuole accedere e la porta utilizzata è quella di default, si può accedere tramite l'indirizzo `localhost:3000` .
Il plugin sarà disponibile nella barra laterale non appena verrà aggiunto dalla schermata dei plugin.

3 Impostare l'ambiente di lavoro

Questo capitolo ha lo scopo di configurare l'ambiente di lavoro in modo tale che sia lo stesso utilizzato dai membri del gruppo DreamCorp. Questa procedura è necessaria solo nel caso si volesse contribuire allo sviluppo di questo progetto, altrimenti può essere saltata.

3.1 Grafana

Grafana è la piattaforma sulla quale il plugin si appoggia. Questo è stato sviluppato utilizzando la versione 5.4.3, ma la sua compatibilità è stata anche testata sulla versione 6.0.0. È possibile scaricare Grafana da questo indirizzo[1].

3.2 InfluxDB

InfluxDB è un database specializzato nella memorizzazione di dati provenienti da serie temporali. Per farlo comunicare con l'applicativo innanzitutto è necessario scaricare da questo indirizzo[2] una versione minore o uguale ad 1.7.4. Successivamente è necessario riempire il database con i dati temporali provenienti dai vari sensori che monitorano l'hardware. Se

[1] `https://grafana.com/`
[2] `https://portal.influxdata.com/downloads/`

non si è in possesso di nessun dato è possibile utilizzare un Time-Series Data Collector come Telegraf[3].

3.3 JsBayes

JsBayes è la libreria utilizzata dal plugin per gestire le reti bayesiane. Per installarla bisogna aprire la console di Webpack e digitare il comando:

```
npm install jsbayes --save
```

È fondamentale aggiungere anche il comando `-save` per permettere a `npm` di memorizzare le dipendenze. Successivamente si può importare la libreria utilizzando l'istruzione:

```
var jsbayes = require('jsbayes');
```

3.4 Angular

Angular è un requisito fondamentale poiché utilizzato dall'applicazione che il nostro plugin andrà ad estendere. Non è necessario eseguire nessuna chiamata esplicita o utilizzare le librerie grafiche di Angular ma è necessario solamente utilizzare `$scope` per collegare il front-end in html al controller in javascript. Per realizzare la view sono utilizzate le direttive `Ng{nomeDirettiva}` come ad esempio `NgModel`.

Debug Per un migliore debug dei componenti Angular è raccomandato l'uso del seguente plugin *Angular Augury*, scaricabile da questo indirizzo[4] disponibile sia per Google Chrome che per Mozilla Firefox.

3.5 NodeJs

NodeJs e di conseguenza NPM, che è un tool utilizzato per l'installazione e la gestione di dei moduli di Node, possono essere scaricati da questo indirizzo[5]. Una volta completato il download basta lanciare l'installer e seguire le istruzioni mostrate dal prompt dell'installer.

[3] https://portal.influxdata.com/downloads/
[4] https://augury.rangle.io
[5] https://nodejs.org/en/download/

3.6 Travis

Il programma include la configurazione del servizio di continuous integration Travis CI. La configurazione `.travis.yml` presente nella root del progetto consiste in tre comandi principali:

```
- npm run build
- npm test
- npm run coveralls
```

In particolare l'ultimo comando permette di caricare i dati di code coverage appena creati all'interno del servizio Coveralls[6] proiettando così dei grafici sull'andamento dei test.

4 Test

Questo capitolo ha lo scopo di indicare agli sviluppatori come controllare in che modo opera il codice e la sua sintassi.

4.1 Test sul codice Javascript

Per eseguire i test sul codice bisogna eseguire il seguente comando:

```
npm run test
```

Eseguirà tutti i test nella cartella `test`. La configurazione del framework Jest è presente nella root del progetto con nome `jest.config.js`
Per eseguire invece i test di analisi statica sul codice è necessario il seguente comando:

```
npm run lint
```

4.2 Code coverage

I test di code coverage sono eseguiti automaticamente con il seguente codice

```
npm run coveralls
```

[6] https://coveralls.io

Verranno generati i risultati nel terminale al termine dei test di unità. Successivamente verranno caricati i risultati del code coverage sul servizio Coveralls tramite uno storico dei risultati ottenuti per poter meglio comprendere i miglioramenti ottenuti. Da notare che il comando appena citato è fruibile solamente tramite il già descritto servizio Travis e non nella build locale, nella quale i risultati verranno visualizzati solamente sul terminale e non verranno caricati su coveralls.

4.3 Lint

Con Javascript Lint è possibile controllare in maniera statica tutto il codice sorgente Javascript, senza dover effettuare la build. In particolare Lint controlla:

- Mancanza del punto e virgola alla fine della linea;
- Parentesi graffe senza i costrutti sintattici come l'*if* o il *while*;
- Codice che non verrà mai eseguito;
- Switch case senza un break;
- Indentazioni dei commenti senza commenti;
- Risoluzione delle ambiguità all'interno del codice;
- Statement che non fanno nulla;
- Errori sintattici generici come tre "+" per indicare l'incremento di una variabile.

5 Architettura generale

5.1 Pattern architetturale: MVC

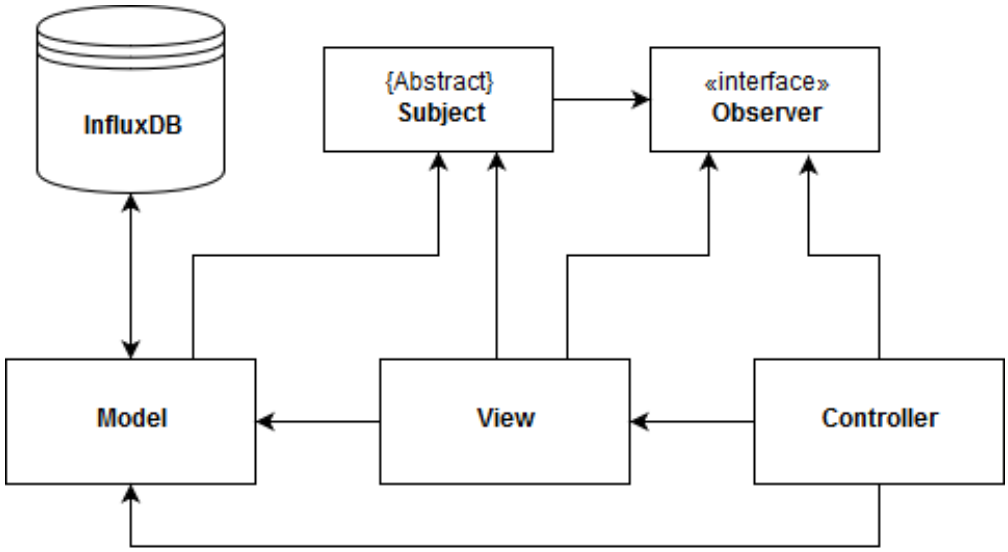

Figura 1: Architettura generale del pattern MVC

Il plugin aderisce al pattern architetturale MVC (Model-View-Controller). Il model è formato dalla classe `Looper` la quale contiene la struttura delle reti Bayesiane utilizzate e dalla classe `InfluxDB` che si occupa dell'interazione con il database InfluxDB[7]. Il controller è costituito da varie classi che svolgono funzione di controllo sui dati gestiti dal `Looper` validandoli. La view è composta dalle classi relative ai diversi tipi di panels presenti con le loro viste in html.

[7] https://www.influxdata.com/

5.2 Jsbayes

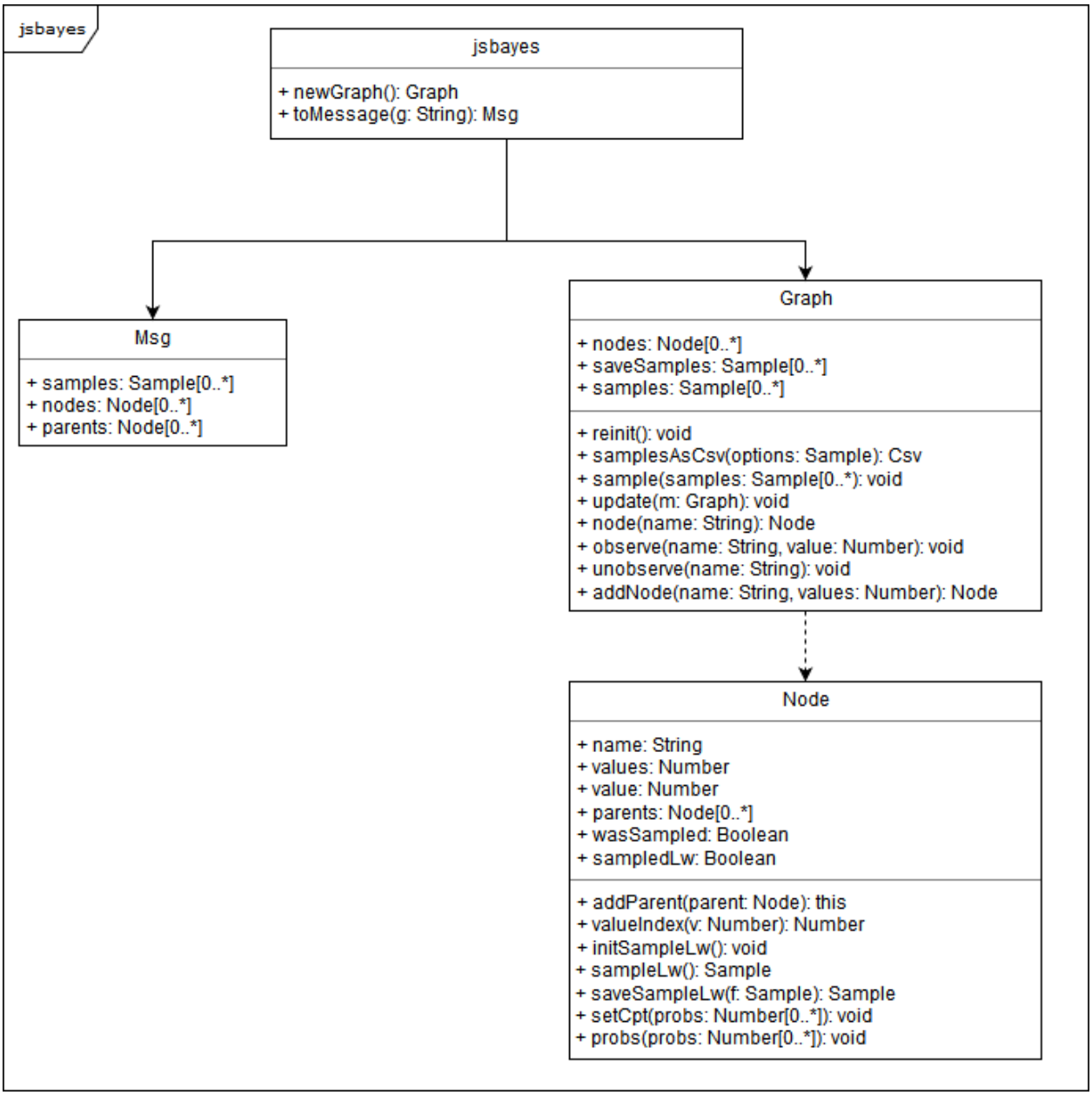

Figura 2: Diagramma delle classi della libreria JsBayes

La logica del plugin è gestita a carico della libreria JsBayes. Inizialmente viene utilizzato il metodo *newGraph()* che permette di creare una nuova rete logica. Il metodo più importante della classe `Graph` è il metodo *sample()* che effettua l'inferenza Bayesiana tenendo conto dei nodi osservati nella rete. I metodi *observe()* e *unobserve()* servono per monitorare o terminare il monitoraggio di un nodo. Per costruire la rete logica vengono utilizzati il metodo *addNode()* per aggiungere un nodo alla rete ed il metodo *addParent()* della classe `Node` per collegare un nodo figlio con i propri padri e viceversa.

6 Front-end

6.1 Le pagine del plugin

Il front-end è stato realizzato usando Html e AngularJs. AngularJs serve per collegare la view con il controller in modo che qualsiasi cambiamento che avvenga nella view sia immediatamente riflesso sul controller e viceversa.

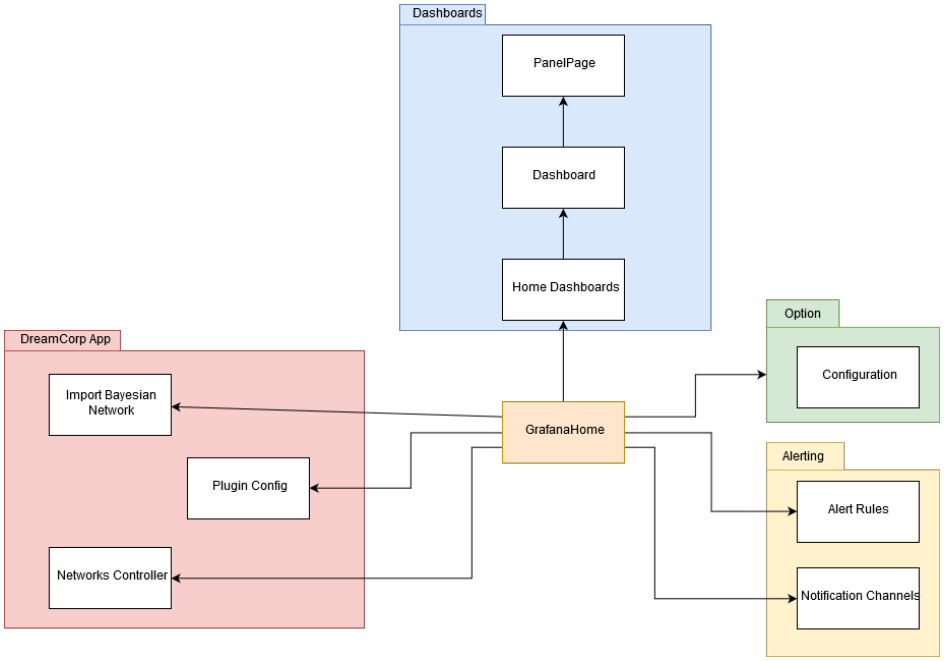

Figura 3: Diagramma delle pagine

6.2 I panel del plugin

Un panel è la visualizzazione di base di un singolo blocco di Grafana, ognuno di essi dispone di un Query Editor per estrarre i dati da visualizzare sullo schermo. Ci sono diversi tipi di panel che il sistema offre con lo scopo di visualizzare i dati nel migliore dei modi o disponendo di funzionalità aggiuntive. Il tipo di panel messo a disposizione dal plugin è il Bayesian Graph. Esso è stato costruito estendendo il già esistente panel Graph ed ampliando le sue funzionalità usando la libreria JsBayes per offrire il supporto alle reti Bayesiane.

6.2.1 Come creare il proprio panel

Per creare un nuovo panel basta estendere uno dei panel di Grafana:

```
class MyPanel extends GraphCtrl{
    constructor($scope, $injector, annotationsSrv, backendSrv) {
        super($scope, $injector, annotationsSrv);
        $scope.ctrl.panel.title = "My title";
        this.backend = backendSrv;
        this.mydata = new Data();
    }
```

Salvare il file con il nome *mypanel.js* e salvarlo nel percorso *src/panel/mypanel*.

6.3 Le tab del plugin

Le tab sono delle sotto-pagine all'interno di un panel che permettono di cambiare le varie impostazioni del panel stesso. Il panel Bayesian Graph rispetto ad un normale panel Graph dispone di un tab aggiuntivo chiamato *Bayesian Network*. All'interno di esso viene data la possibilità all'utente di associare o dissociare un nodo di una qualsiasi rete Bayesiana al flusso dati proveniente dal datasource del panel a patto che questo non sia già monitorato da un nodo della stessa rete.

6.3.1 Come creare la propria tab

Il modo corretto per creare una nuova tab è collegarla ad un panel già esistente tramite la seguente istruzione da inserire nel corpo del panel:

```
this.addEditorTab('MyPanel', MyTab);
```

Per costruire un nuovo tab è necessario creare un nuovo file ed è consigliato salvarlo nello stesso percorso del rispettivo panel per mantenere una struttura ordinata delle cartelle.

```
class MyTab{
constructor($scope, backendSrv){
    /*metodi e campi dati*/
}
```

7 Back-end

7.1 API di Grafana

Il plugin interagisce con il backend di Grafana sfruttando alcune API messe a disposizione dal sistema stesso.

7.1.1 Salvataggio dati

Salvataggio di una rete Durante l'importazione di una nuova rete Bayesiana, la struttura della rete sotto forma di file Json viene inserita all'interno della struttura di una Dashboard tramite un campo *network*. Viene poi utilizzata la seguente API di Grafana per salvare la nuova struttura della Dashboard:

```
return this.backendSrv
    .post('api/dashboards/import', {
        dashboard: this.dash,
        overwrite: true,
        inputs: inputs,
        folderId: this.folderId,
    })
```

7.1.2 Recupero dati

Recupero struttura rete tramite download della dashboard: Analogamente all'operazione di salvataggio di una rete, per recuperarne una basta recuperare la Dashboard che la contiene sfruttando, come al solito, un'API di Grafana:

```
return this.backend.get('/api/search?tag=bayesian-network')
```

Recupero dati di un alert Per ottenere il valore corrente di un nodo bisogna prima recuperare i dati contenuti nell'alert associato per poi estrarne le informazioni necessarie. Il tutto viene eseguito tramite la seguente API:

```
return await this.backend.get('/api/alerts/?panelId='+panelId)
```

7.2 API di InfluxDB

7.2.1 Scrittura

Ad intervalli regolari ogni nodo monitorato invia il suo valore verso un database InfluxDB in modo che esso possa essere memorizzato nel tempo. Questi valori verranno poi utilizzati per costruire i grafici relativi ai nodi non monitorati. Per salvare dei dati sul database viene richiamata la seguente API di InfluxDB:

```
return $.ajax({
    url:this.host+this.port+'/write?db='+this.database,
    type:'POST',
    contentType:'application/octet-stream',
    data: query,
    processData: false,
    error: function(test, status, exception) {
        console.log("Error: " + exception);
}
```

7.2.2 Lettura

La lettura dei dati dal database viene eseguita dai grafici associati collegati ad esso e attraverso apposite query impostate dall'utente vengono recuperati i dati desiderati. Questa funzione non è a carico dello sviluppatore poiché se ne occupa il sistema stesso.

8 Architettura di dettaglio

8.1 Desing Pattern

Durante la fase di progettazione, per risolvere alcuni problemi implementativi, abbiamo trovato utile utilizzare alcuni design pattern. Le diverse tipologie di pattern (creazionali, strutturali e comportamentali) sono state studiate per le nostre esigenze.

8.1.1 Façade: Looper

Classe per gestire il monitoraggio dei valori degli alert dei panel associati, il ricalcolo delle reti bayesiane e il salvataggio dei dati su InfluxDB.

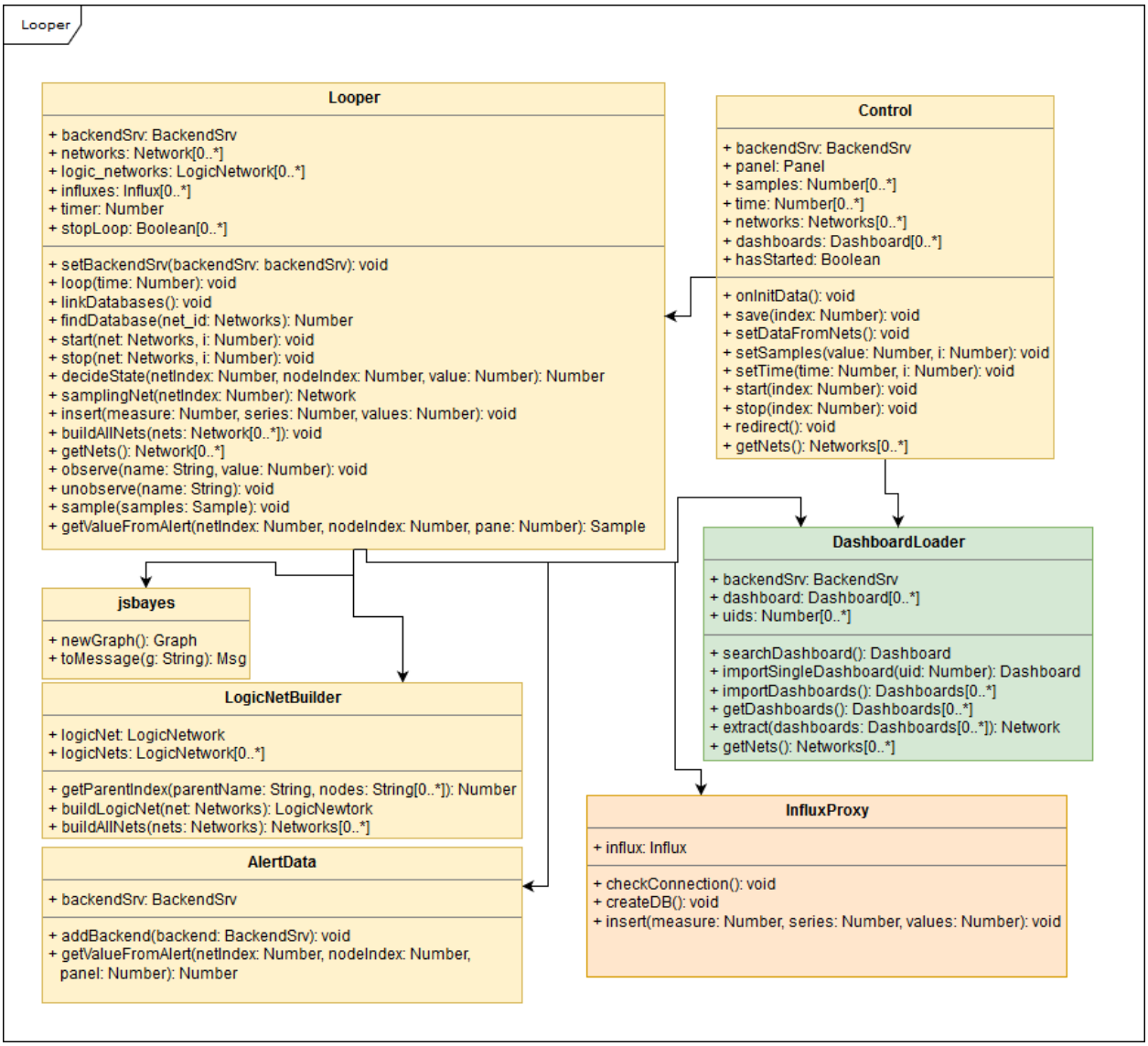

Figura 4: Diagramma delle classi del Looper nel pattern Façade

Descrizione Pattern strutturale utilizzato per nascondere la complessità di una funzionalità complessa, attraverso un'interfaccia più semplice che accede a funzionalità di sottosistemi complessi e diversi fra loro.

Motivazioni della scelta La necessità di nascondere la complessità della classe `Looper` ha richiesto l'implementazione di questo pattern: diversi metodi con funzionalità diverse tra loro e appartenenti a classi diverse sono utilizzate in modo controllato all'interno del metodo `loop()` della classe qui descritta.
Il metodo `loop()`, in particolare, è il centro della logica del nostro plugin e richiede la coordinazione di diverse funzionalità:

- Download struttura reti bayesiane: classe `DashboardLoader`;
- Creazione reti logiche bayesiane: classe `jsbayes`;
- Download valori alert dei panel associati ai nodi della rete: classe `AlertData`;
- Observe dei nodi e sampling delle reti bayesiane: classe `jsbayes`;
- Salvataggio dei valori delle probabilità dei nodi su InfluxDB: classe `Influx`.

Il controllo rimane centralizzato sulla classe `Looper`, però le singole funzionalità appartengono e vengono portate a termine dalle singole classi che effettivamente implementano quella funzionalità.

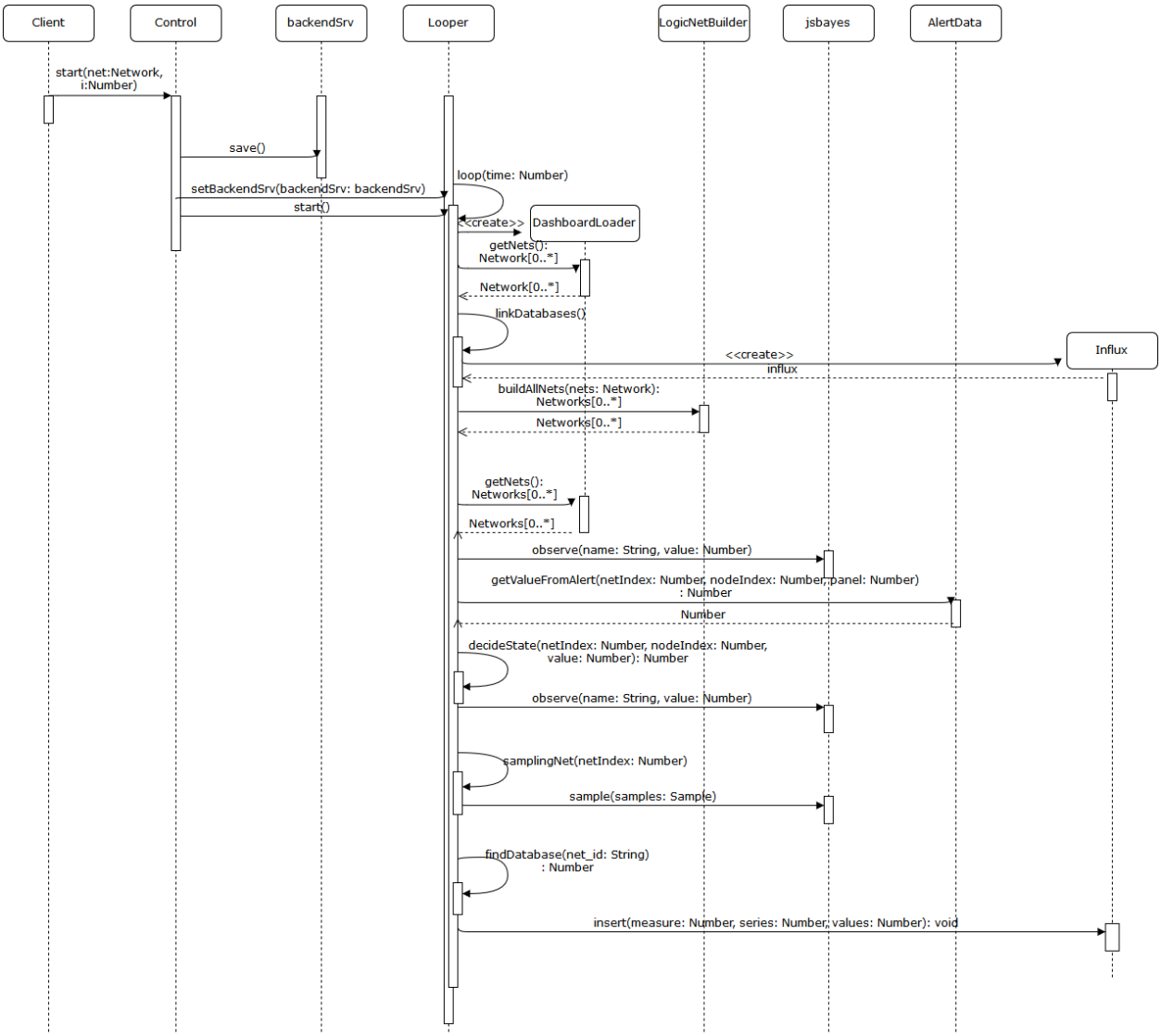

Figura 5: Diagramma di sequenza per l'operazione di inizio del monitoraggio

8.1.2 Façade: ImportNetCtrl

Classe per gestire l'import della struttura della rete bayesiana in formato json.

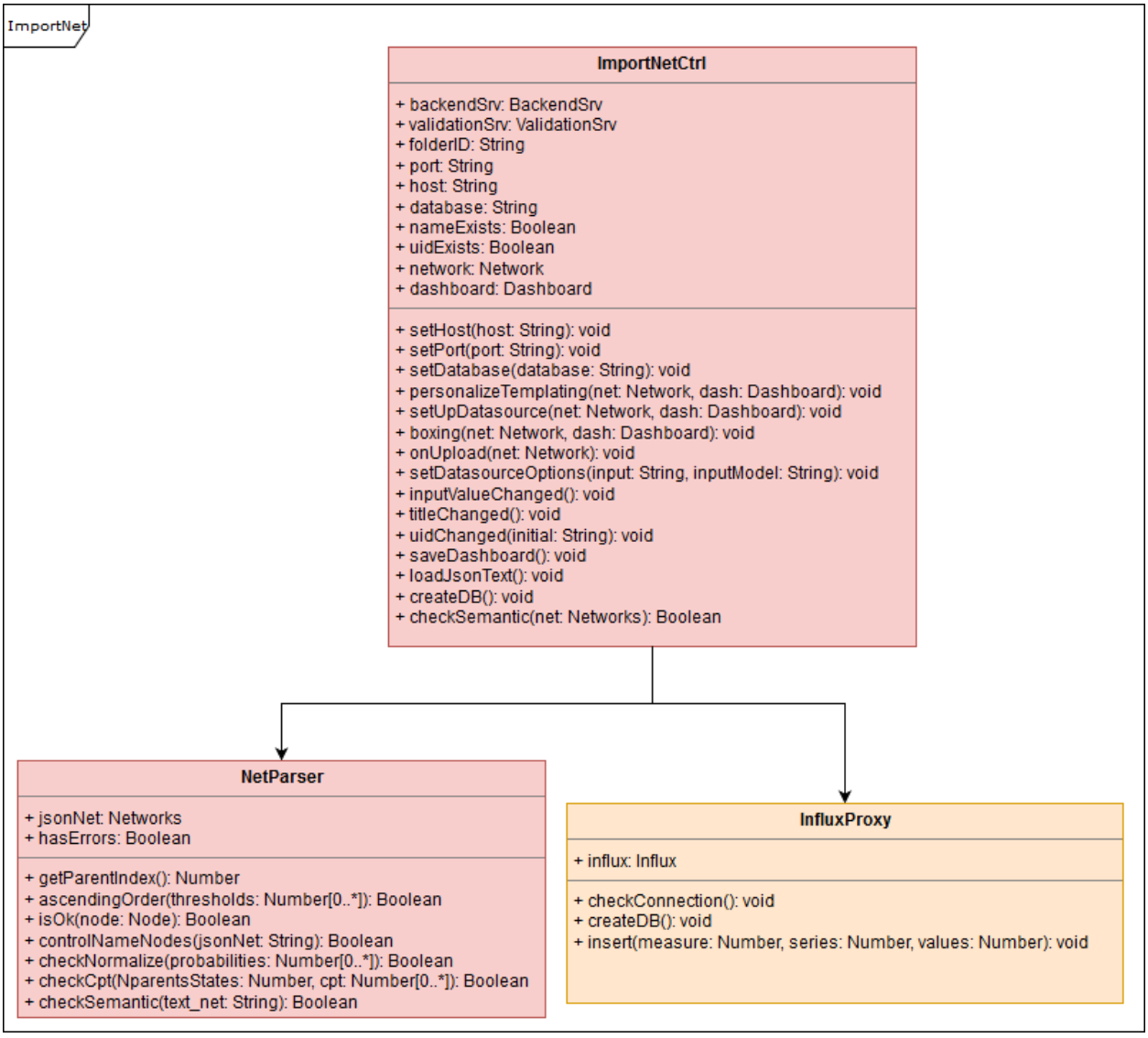

Figura 6: Diagramma delle classi di ImportNetCtrl nel pattern Façade

Descrizione Pattern strutturale utilizzato per nascondere la complessità di una funzionalità complessa, attraverso un'interfaccia più semplice che accede a funzionalità di sottosistemi

complessi e diversi fra loro.

Motivazioni della scelta La necessità di nascondere la complessità della classe `ImportNet` ha richiesto l'implementazione di questo pattern. Diverse funzionalità sono richieste per portare a termine il compito che si propone questa classe. Per ogni funzionalità non strettamente collegata all'import di una rete bayesiana, è stata creata una classe che gestisse, tramite metodi propri, le specifiche funzionalità, le quali sono:

- Controllo semantico della struttura della rete: classe `NetParser`;
- Creazione del database per i dati della probabilità dei nodi della rete: classe `Influx`.

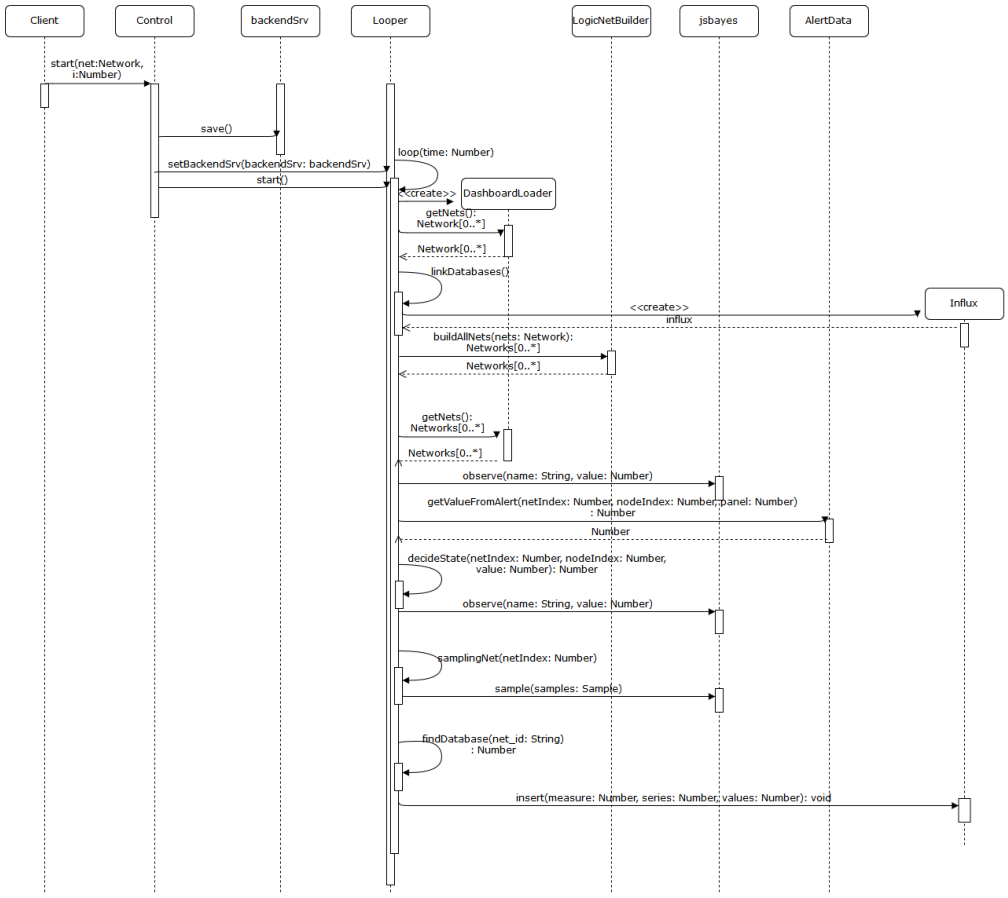

Figura 7: Diagramma di sequenza per l'operazione di inizio del monitoraggio

8.1.3 Singleton: NetHandler

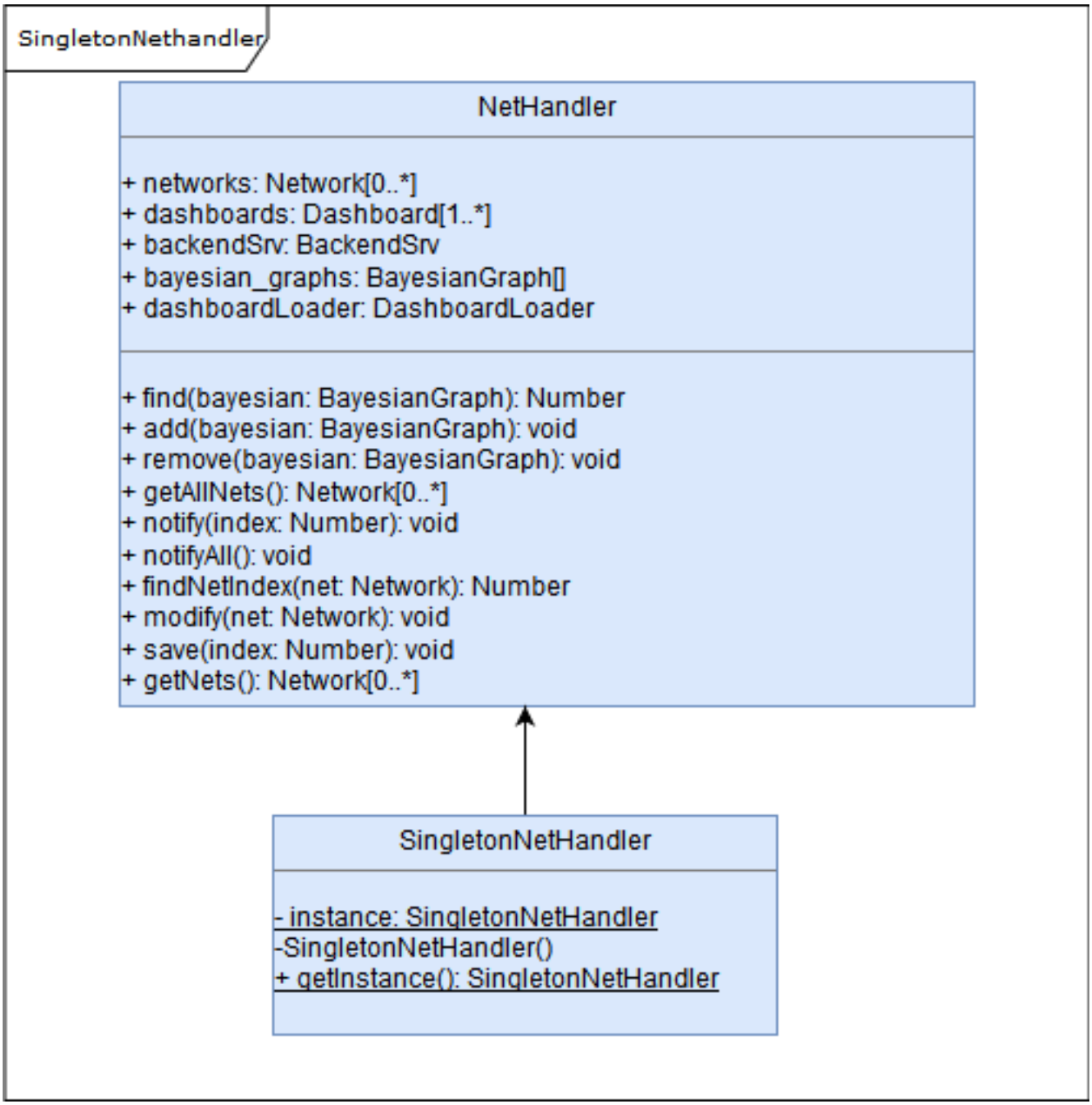

Figura 8: Diagramma delle classi del SingletonNetHandler

Descrizione Il singleton è un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza.

Motivazioni della scelta L'implementazione di questo pattern è stata necessaria per garantire la comunicazione e la consistenza delle strutture delle reti. Più istanze della classe `BayesianGraphCtrl` potrebbero voler modificare la struttura di una o più reti e utilizzando il singleton, implementato con la classe `SingletonNetHandler` e `NetHandler`, come punto di accesso globale per le modifiche possiamo garantire la consistenza di queste strutture. Inoltre, per poter modificare la struttura delle reti è necessario riuscire a scaricarne la versione aggiornata. La computazione richiesta per questa operazione può essere ridotta utilizzando sempre la classe `NetHandler` come controllo centrale, in grado di fornire i dati delle reti in tempi minori.

8.1.4 Observer: NetHandler-BayesianGraphCtrl

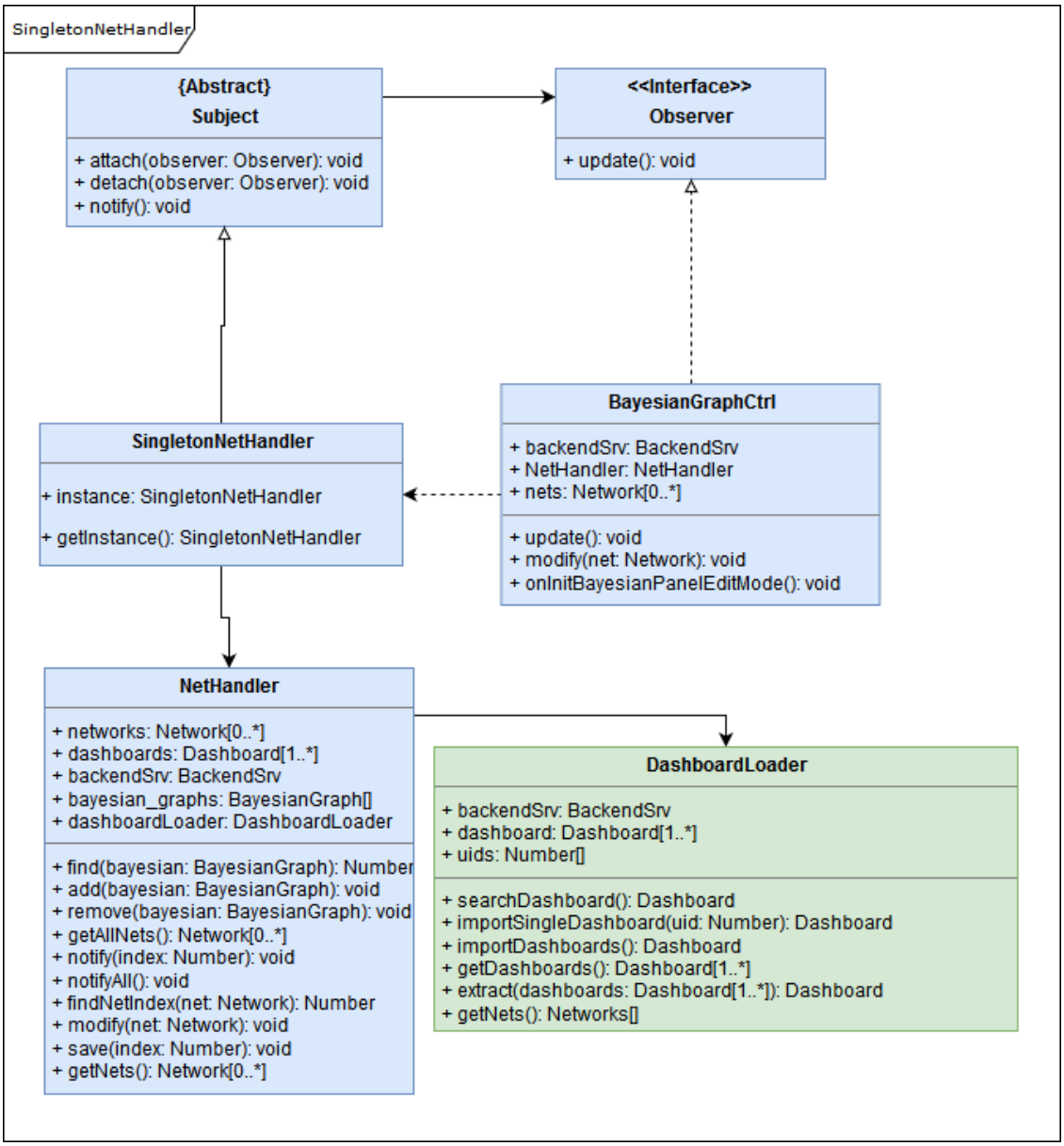

Figura 9: Diagramma delle classi del SingletonNetHandler come subject del pattern Observer

Descrizione Questo pattern permette di definire una dipendenza uno a molti fra oggetti, in modo tale che se un oggetto cambia il suo stato interno, ciascuno degli oggetti dipendenti da esso viene notificato e aggiornato automaticamente.

Motivazioni della scelta L'*Observer* nasce dall'esigenza di mantenere un alto livello di consistenza fra classi correlate, senza produrre situazioni di forte dipendenza e di accoppiamento elevato. Per questo motivo abbiamo modellato la relazione tra la classe `NetHandler` e la classe `BayesianGraphCtrl`. `NetHandler` funge da *subject* e contiene una lista di istanze della classe `BayesianGraphCtrl`. In questo modo le modifiche della rete di un panel si riflettono immediatamente su tutti gli altri panel attraverso la `notifyAll()` di `NetHandler`, mantenendo la consistenza dei dati delle reti. Specifichiamo le interazioni tra `SingletonNetHandler` e i `BayesianGraphCtrl`

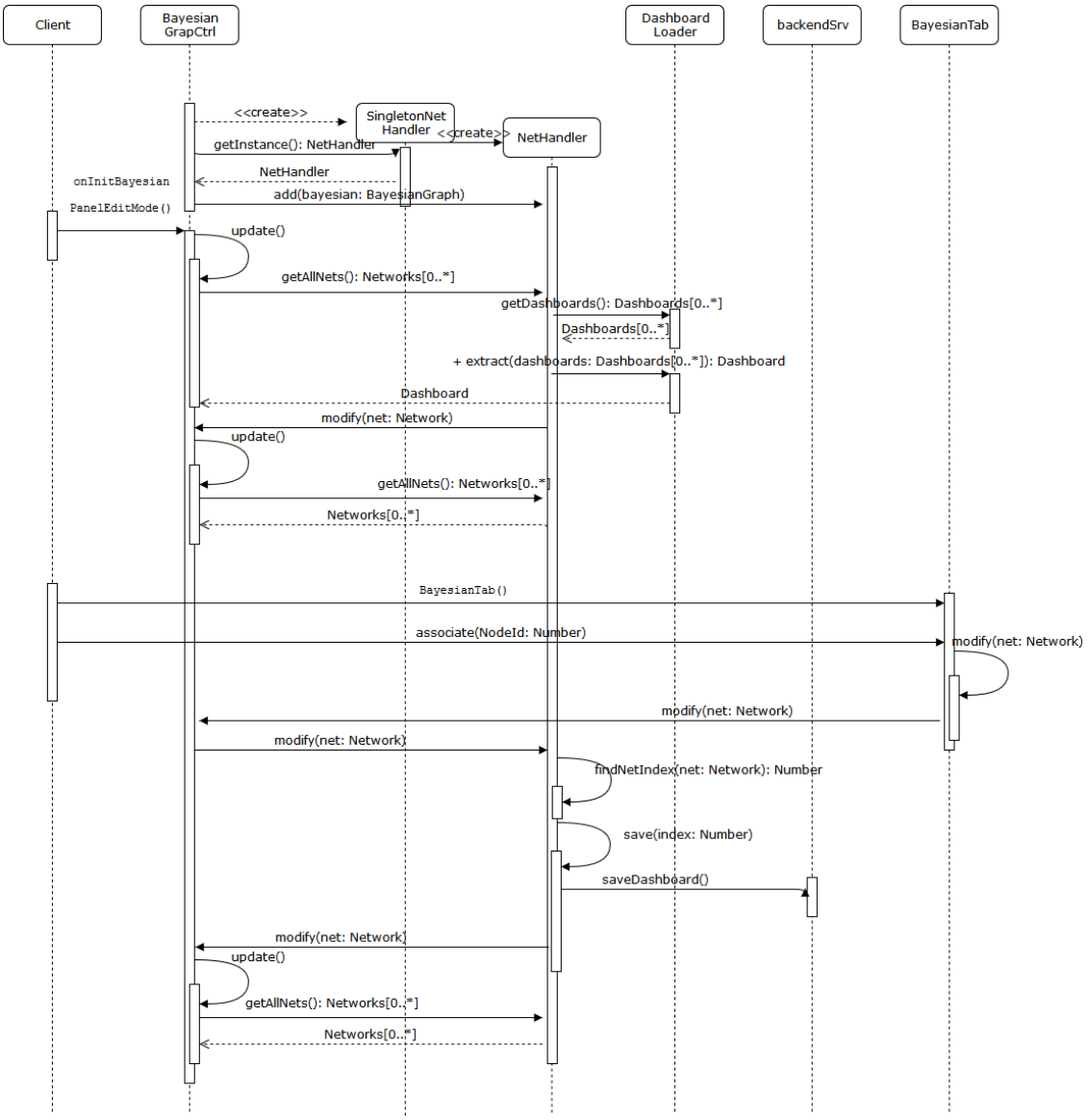

Figura 10: Diagramma di sequenza per l'associazione di un nodo ad una rete.

8.1.5 Proxy: Influx-InfluxProxy

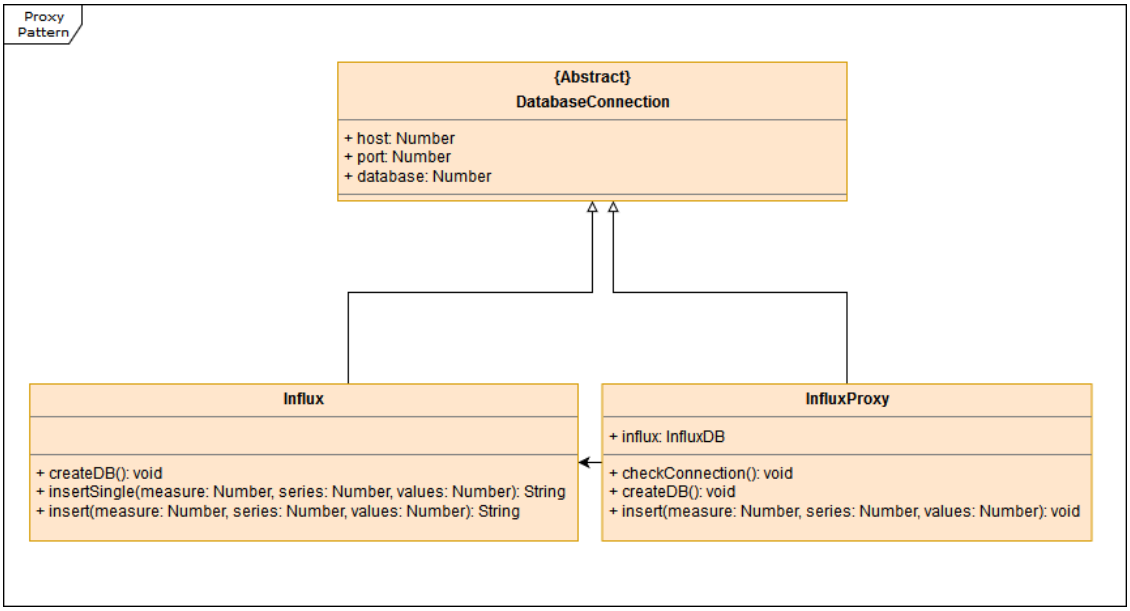

Figura 11: Diagramma delle classi del Proxy Pattern

Descrizione Questo pattern permette la rappresentazione locale di un oggetto che si trova in uno spazio di indirizzi differente In questa versione del proxy pattern, detta *remote proxy*, si fornisce un'astrazione sulla comunicazione che avviene tramite la rete, nascondendo quest'ultima al client che vi si interfaccia.

Motivazioni della scelta La presenza di una richiesta di connessione ad un database, possibilmente remoto, ha richiesto l'implementazione del proxy pattern al fine di nascondere, attraverso un'astrazione, la connessione stessa attraverso rete e host. Per questo motivo abbiamo creato una classe `DatabaseConnection`, in grado semplicemente di memorizzare le informazioni relative alla connessione. Successivamente abbiamo derivato due classi: `Influx` e `InfluxProxy`. Quest'ultima è l'unica classe che si interfaccia con il client direttamente e controlla, tramite un riferimento ad un'istanza della classe `Influx`, le effettive operazioni sul database le quali possono avvenire dopo aver garantito la connessione. La classe `InfluxProxy` ha infatti un metodo `checkConnection()` in grado di effettuare un controllo sulla risposta che viene fornita dal database specificato tramite informazioni come host, port e nome del database. In caso di risposta positiva, la connessione è pronta e le operazioni sul

database richieste dal client possono essere portate a termine. In caso contrario non verranno effettuate le operazioni.

8.2 Considerazioni

Un'analisi più approfondita richiede l'implementazione dei pattern *Singleton* e *Observer*. La classe che gestisce la comunicazione all'interno del pattern *Observer*, ossia il *subject*, è la stessa che funge da *singleton* per l'omonimo pattern. Questa classe è `NetHandler`.
Come gruppo abbiamo osservato che la compresenza di entrambi i pattern non invalida i pattern stessi, specifica solo meglio che la comunicazione tra classi e le loro dipendenze modellate come *subject/observer* è anche supportata dal fatto che il *subject* abbia una sola istanza, un accesso globale per tutte le istanze degli *observer*.
Il motivo di questa progettazione è intrinsecamente legato alla logica implementativa che abbiamo deciso di dare al nostro plugin e ai vincoli dettati dall'ambiente di sviluppo che è Grafana.