



Manuale Sviluppatore

7DOS - 8 Aprile 2019

Informazioni sul documento

Versione	1.0.0
Responsabile	Giovanni Sorice
Verifica	Marco Costantino Nicolò Tartaggia
Redazione	Giacomo Barzon Lorenzo Busin Michele Roverato
Stato	Approvato
Uso	Esterno
Destinato a	Prof. Tullio Vardanega Prof. Riccardo Cardin Zucchetti 7DOS
Email	7dos.swe@gmail.com

Descrizione

Questo documento contiene il manuale sviluppatore relativo al prodotto G&B del gruppo 7DOS.



Diario delle modifiche

Versione	Data	Descrizione	Autore	Ruolo
1.0.0	2019-04-08	<i>Approvazione del documento</i>	Giovanni Sorice	Responsabile
0.7.0	2019-04-08	<i>Verifica del documento</i>	Marco Costantino	Verificatore
0.6.0	2019-04-08	<i>Verifica del documento</i>	Nicolò Tartaggia	Verificatore
0.5.1	2019-04-04	<i>Stesura §7</i>	Giacomo Barzon	Amministratore
0.5.0	2019-04-04	<i>Stesura §6</i>	Giacomo Barzon	Amministratore
0.4.0	2019-04-04	<i>Stesura §4 e §5</i>	Giacomo Barzon	Amministratore
0.3.0	2019-04-04	<i>Stesura §3</i>	Giacomo Barzon	Amministratore
0.2.0	2019-04-04	<i>Stesura §2</i>	Michele Roverato	Amministratore
0.1.1	2019-04-04	<i>Stesura §1</i>	Giacomo Barzon	Amministratore
0.0.1	2019-04-01	<i>Prima stesura dello scheletro del documento</i>	Lorenzo Busin	Amministratore

Indice

1 Introduzione	3
1.1 Scopo del documento	3
1.2 Scopo del prodotto	3
1.3 Glossario	3
1.4 Riferimenti	3
1.4.1 Installazione	3
1.4.2 Legali	4
1.4.3 Informativi	4
1.5 Maturità del documento	4
2 Requisiti di sistema	5
2.1 Requisiti hardware	5
2.2 Browser compatibili	5
2.3 Sistemi operativi	5
3 Installazione	6
3.1 Installazione di Grafana	6
3.2 Installazione Node.js	6
3.3 Installazione plug-in	6
3.4 Installazione e configurazione InfluxDB	7
4 Configurazione dell'ambiente di lavoro	8
4.1 WebStorm	8
4.2 VSCode	8
4.3 TSLint e ESLint	8
5 Test	9
5.1 Test sul codice	9
5.2 Code Coverage	9
6 Architettura	10
6.1 Persistence Layer	11
6.1.1 NetworkAdapter	12
6.1.2 NodeAdapter	12
6.1.3 AbstractValue	13
6.1.4 NetBuilder	14
6.1.5 NetParser	14
6.2 Buisness Layer	15
6.2.1 NetReader	16
6.2.1.1 Flow	16
6.2.1.2 ClientPool	17
6.2.2 NetUpdater	17
6.2.3 NetWriter	18
6.2.4 NetManager	18
6.2.5 Results	19

6.2.6	TimeBasedNetUpdater	20
6.3	Database Layer	21
6.4	Presentation Layer	22
7	Struttura del JSON	24
Appendice		26
A	Glossario	26

1 Introduzione

1.1 Scopo del documento

Questo documento rappresenta il Manuale dello Sviluppatore relativo al prodotto software *G&B* sviluppato dal gruppo 7DOS. Il suo scopo principale è fornire tutte le informazioni necessarie per usufruire delle funzionalità fino ad ora implementate ed, eventualmente, per estendere e migliorare il prodotto.

1.2 Scopo del prodotto

G&B è un plug-in per il software di monitoraggio Grafana. Il suo scopo è estenderne le funzionalità permettendo di monitorare il sistema desiderato con una o più reti Bayesiane definita ad-hoc dall'utente, consentendo poi la visualizzazione dei dati calcolati mediante i panel di Grafana.

Il plug-in può essere utilizzato interamente in locale, e non dipende da servizi esterni per il suo funzionamento: una volta installato, è sufficiente impostare le *datasource_g* necessarie, caricare e configurare la rete Bayesiana scelta e avviare il monitoraggio.

1.3 Glossario

In appendice al documento è presente un glossario con tutti i termini necessari per la piena comprensione del documento. Il prodotto è diretto a operatori IT e addetti al monitoraggio di sistemi informatici, pertanto vocaboli valutati come di conoscenza comune o appartenenti a competenze informatiche di base sono stati ignorati.

La presenza di un termine all'interno del glossario è segnalata con una "g" posta come pedice (esempio: *Glossario_g*).

1.4 Riferimenti

1.4.1 Installazione

- **Git**
<https://git-scm.com/>;
- **Grafana**
<https://grafana.com/docs/installation/>;
- **Grafana Plug-in**
<https://grafana.com/docs/plugins/installation/>;
- **Node.js**
<https://nodejs.org/>;
- **NPM**
<http://www.npmjs.com/>;
- **InfluxDB**
<https://www.influxdata.com/>;

- JetBrains - WebStorm
<https://jetbrains.com/webstorm>;
- JetBrains - WebStorm for students
<https://jetbrains.com/student>;
- VSCode
<https://code.visualstudio.com/>;
- Coveralls
<https://coveralls.io/>.

1.4.2 Legali

- Licenza MIT
<https://opensource.org/licenses/MIT>.

1.4.3 Informativi

- Reti Bayesiane
https://en.wikipedia.org/wiki/Bayesian_network.

1.5 Maturità del documento

Il presente documento potrebbe essere soggetto ad incrementi futuri. Per questo motivo, non si pone l'obiettivo di risultare completo. Tutto ciò che riguarda la pianificazione degli incrementi, può essere trovato nel *Piano di Progetto v3.0.0* in §4.

2 Requisiti di sistema

2.1 Requisiti hardware

Per poter utilizzare il plug-in e tutto ciò che permette la sua esecuzione sono presenti dei requisiti hardware da soddisfare:

- RAM: 2 GB;
- Memoria interna libera: 10 GB;
- Processore: minimo *dual core_g*

2.2 Browser compatibili

Di seguito vengono riportate le versioni dei browser utilizzate durante lo sviluppo sui quali è garantito il funzionamento del nostro plug-in:

- Google Chrome v.73;
- Mozilla Firefox v.66;
- Safari v.11;
- Microsoft Internet Explorer v.11;
- Microsoft Edge v.41.

Affinché tutte le funzionalità offerte dal plug-in vengano eseguite correttamente, è necessario che *JavaScript_g* sia abilitato.

2.3 Sistemi operativi

Il plug-in è stato sviluppato con i seguenti sistemi operativi ed è pertanto compatibile con essi:

- Ubuntu 18.04;
- Windows 7 e 10.

3 Installazione

3.1 Installazione di Grafana

Per poter installare e utilizzare il plug-in è necessario scaricare e configurare precedentemente Grafana; le istruzioni di installazione sono reperibili al link

<http://docs.grafana.org/installation/>

che le riassume per ogni sistema operativo supportato.

Il plug-in è stato sviluppato utilizzando la versione 5.4.3. Per evitare errori nell'esecuzione, si raccomanda di avvalersi della stessa versione. Una volta terminata la configurazione descritta al riferimento precedente, in Windows avviare l'eseguibile

`grafana-5.4.3/bin/grafana-server.exe`

mentre in Linux eseguire il comando

`sudo service grafana-server start.`

A questo punto, aprire una finestra del browser e navigare all'indirizzo URL localhost:3000. La porta 3000 è quella di default per Grafana, ma su sistema operativo Windows potrebbe richiedere permessi aggiuntivi.

3.2 Installazione Node.js

Per usufruire delle funzionalità del plug-in è necessario installare il *framework_g* Node.js e il relativo gestore dei pacchetti chiamato Node Package Manager (*NPM_g*), attraverso l'apposita sezione al link

<https://nodejs.org/it/download/>.

In questo modo saranno disponibili i comandi `npm install` e `npm run build`, necessari rispettivamente per l'installazione di tutte le dipendenze specificate nel *package.json_g* e per l'esecuzione del processo di *build_g*. Verificare, alla fine, la disponibilità del comando `npm` tramite il comando `npm -v`. In caso negativo, scaricarlo dal link

<https://www.npmjs.com/get-npm>.

3.3 Installazione plug-in

Il plug-in completo è disponibile al link

<https://github.com/NicoloTartaggia/7DOS-plugin>

, scaricabile attraverso il bottone "Clone or download" e, in seguito, "Download ZIP", oppure attraverso comando

`git clone https://github.com/NicoloTartaggia/7DOS-plugin.`

Una volta scaricato ed estratto dall'archivio compresso, spostare il contenuto all'interno del percorso specifico:

- `grafana-5.4.3/data/plugins` in Windows;

- `var/lib/grafana/plugins` in Linux.

Dopodiché, da terminale spostarsi all'interno della cartella del plug-in e eseguire il comando

```
npm install
```

per installare tutti i moduli node necessari al funzionamento del plug-in. Terminata l'installazione, procedere con il comando

```
npm run build
```

per eseguire il processo di build. Infine, riavviare il server di grafana. In Windows riaprire l'eseguibile

```
grafana-server.exe;
```

in Linux eseguire il comando

```
sudo service grafana-server restart.
```

3.4 Installazione e configurazione InfluxDB

Grafana utilizza diverse *datasource_g* per prelevare i dati da elaborare e monitorare. In particolare, il prodotto sviluppato si appoggia ad *InfluxDB_g*. Di conseguenza, sono necessari due passaggi:

- Installare InfluxDB su Grafana: dirigersi alla sezione "Configuration" dal side-menu a sinistra e selezionare la voce "Data sources". Successivamente, aggiungere una datasource tramite il bottone "Add data source" e selezionare InfluxDB. Una volta installato, impostare InfluxDB come datasource di default spuntando il relativo checkbox nelle impostazioni;
- Scaricare InfluxDB al link

<https://portal.influxdata.com/downloads/>.

La versione di riferimento utilizzata durante lo sviluppo è la 1.7. Una volta scaricato, avviare il server. In Windows, eseguire

```
influxd.exe,
```

mentre in Linux eseguire il comando `sudo service influxdb start`.

L'istanza sarà interrogabile all'indirizzo localhost:8086.

4 Configurazione dell'ambiente di lavoro

Il seguente capitolo descrive come configurare l'ambiente di lavoro in modo corretto.

4.1 WebStorm

Per lo sviluppo del plug-in il team ha scelto di utilizzare l'*IDE* WebStorm, sviluppato da JetBrains. L'*IDE* è a pagamento, tuttavia esso può essere scaricato gratuitamente dal loro sito ufficiale in prova gratuita per trenta giorni oppure se si è in possesso di un'e-mail personale universitaria ottenere gratuitamente una licenza valida un anno.

Il software è disponibile per Microsoft Windows, Linux e MacOS.

4.2 VSCode

VSCode è una valida alternativa a WebStorm, che il team consiglia di utilizzare nel caso in cui quest'ultimo non fosse reperibile.

Il software può essere scaricato molto facilmente dal sito ufficiale ed è disponibile per Microsoft Windows, Linux e MacOs.

4.3 TSLint e ESLint

TSLint e ESLint verranno automaticamente installati con l'esecuzione del comando:

```
npm install .
```

Una volta installati correttamente, WebStorm li rileverà automaticamente tra le dipendenze presenti all'interno del package.json e procederà a segnalare tutti gli errori relativi all'analisi statica rilevati, senza la necessità di dover eseguire il comando:

```
npm run build .
```

5 Test

Questo capitolo descrive le procedure per eseguire i test sul funzionamento e sulla sintassi del codice.

5.1 Test sul codice

Per avviare i test sul funzionamento del codice TypeScript, è sufficiente eseguire da terminale il comando:

```
npm run test .
```

Il comando avvierà l'esecuzione di tutti i test presenti nella cartella `/src/test`. Per avviare i test sulla sintassi del codice è necessario lanciare i comandi:

- `npm run eslint` per verificare la sintassi JavaScript;
- `npm run tslint` per verificare la sintassi TypeScript.

Alternativamente questi due comandi possono essere eseguiti assieme lanciando:

```
npm run eslint && npm run tslint .
```

È importante notare che il comando congiunto non è compatibile con *Windows Powershell*. Al termine verranno segnalate le parti di codice che non rispettano le linee guida.

5.2 Code Coverage

Per avviare il calcolo del code coverage è sufficiente eseguire da terminale il comando:

```
npm run coverage .
```

Il comando esegue prima i test generando un report di coverage (in formato text-lcov) e poi manda direttamente il risultato a [coveralls.io](#). Se la repository non è pubblica è necessario generare e usare un token per l'account usato. La precisa procedura per fare ciò è presente nella documentazione di coveralls.io.

6 Architettura

All'interno della seguente sezione verrà descritta l'architettura dell'intero sistema.

Ogni componente logico del plug-in viene eseguito all'interno dello stesso, e quindi all'interno del browser, senza appoggiarsi a server esterni per l'esecuzione. Per questo motivo ogni panel è un componente a sé stante e indipendente.

Per la realizzazione dell'architettura del plug-in il team ha deciso di adottare uno stile architettonale a layer.

I principali layer che sono stati individuati nel corso dello sviluppo dell'applicazione sono i seguenti:

- **Presentation layer:** si occupa di gestire la componente grafica del panel;
- **Persistence layer:** si occupa di mantenere la struttura della rete;
- **Business layer:** incorpora tutta la logica relativa all'intero processo di ricalcolo delle probabilità;
- **Database layer:** composto dall'insieme di tutti i database esterni forniti dall'utente per il reperimento delle informazioni.

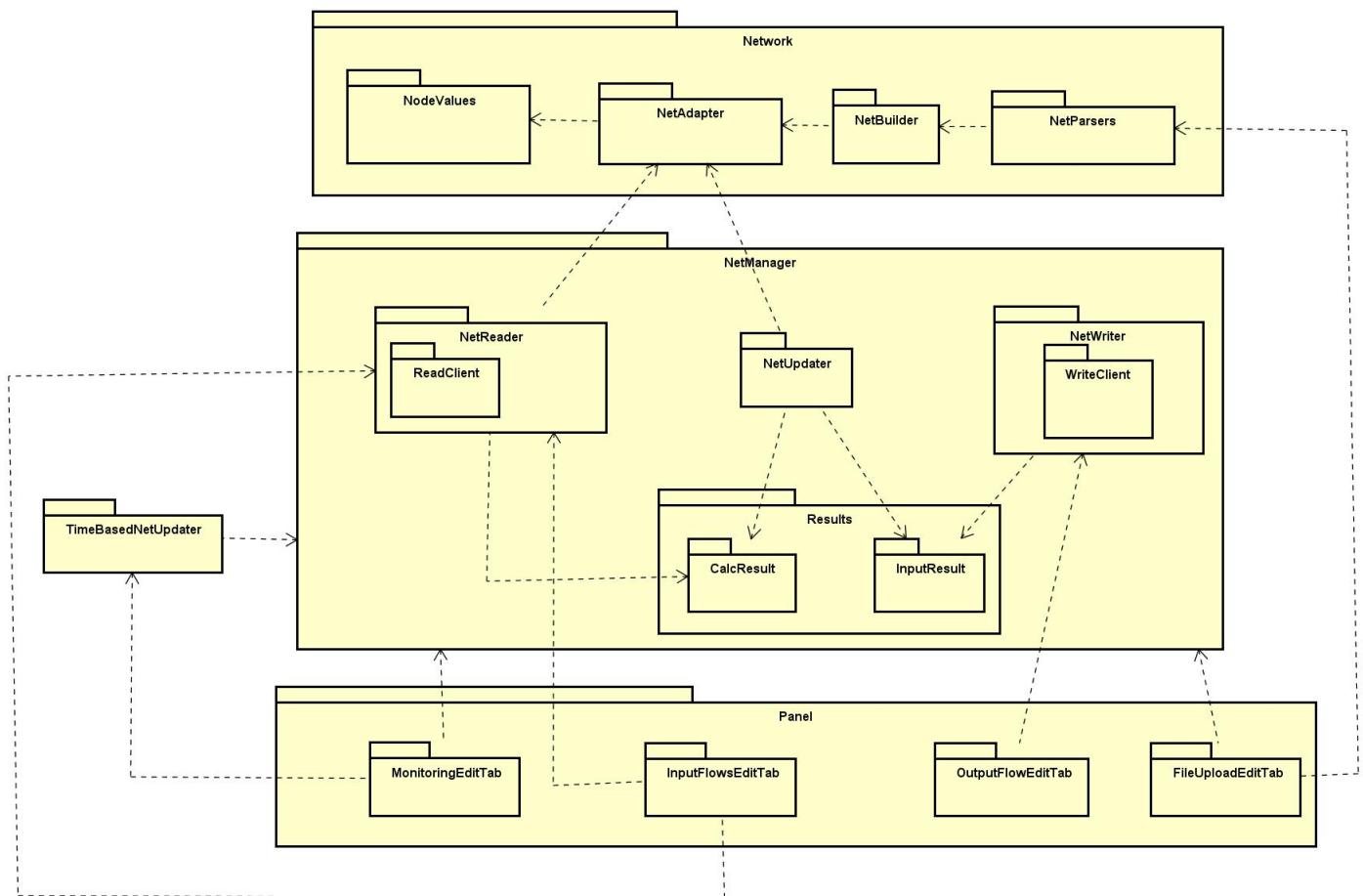


Figura 1: Diagramma dei package

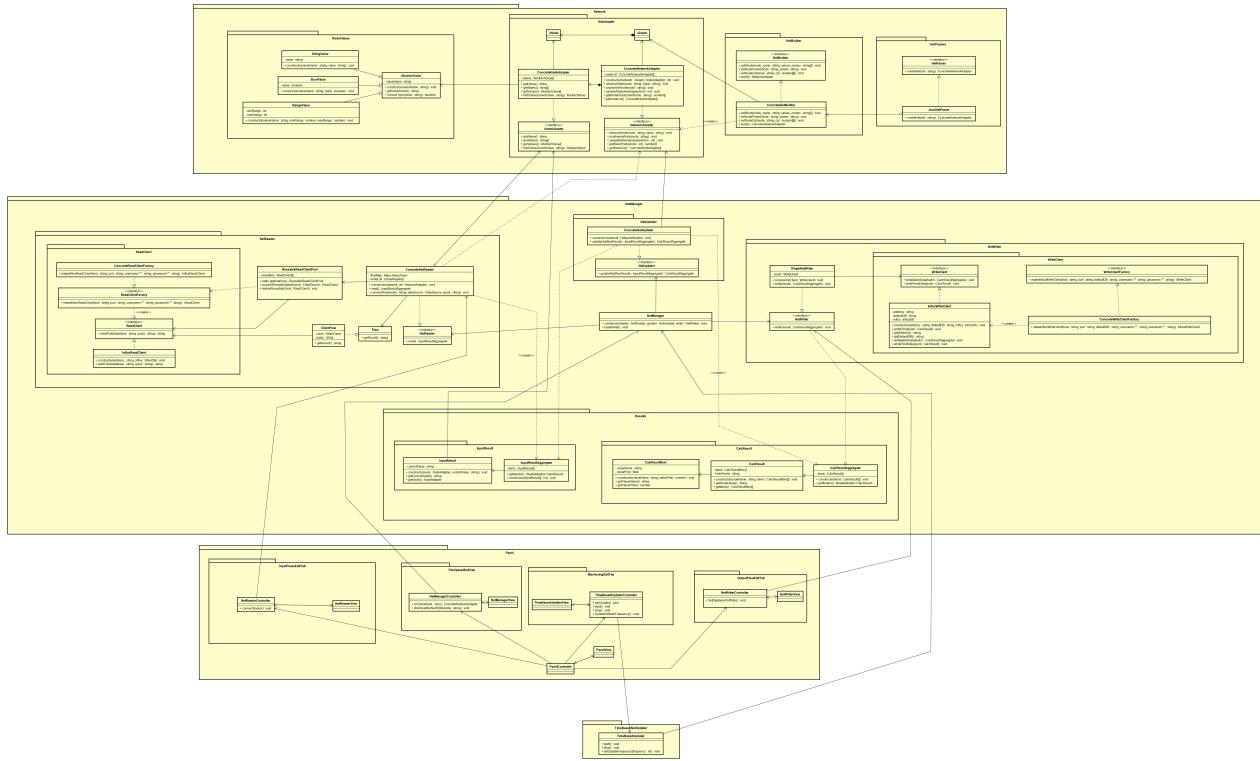


Figura 2: Diagramma delle classi complessivo

Analizziamo ora più in dettaglio ciascuno dei layer precedentemente descritti.

6.1 Persistence Layer

L'intera nostra applicazione è basata sulla libreria Javascript JSBayes consigliata dal proponente, la quale permette di mantenere l'intera struttura della rete ed esegue su di essa il calcolo delle probabilità di ogni stato associato ad ogni nodo della rete. Questa libreria tuttavia presenta non pochi problemi, tra i quali una pressoché totale assenza della gestione degli errori, una difficile interazione con altri oggetti, ed una scarsa estendibilità. Il persistence layer è composto principalmente da un insieme di classi che permettono di ovviare ai problemi precedentemente descritti.

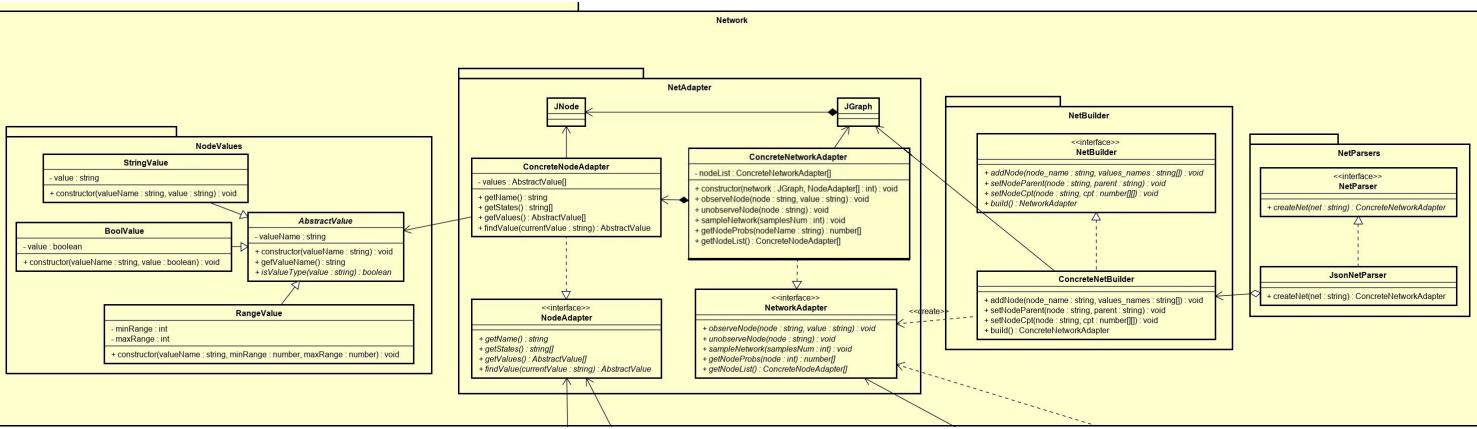


Figura 3: Diagramma UML persistence layer

6.1.1 NetworkAdapter

La classe `NetworkAdapter` ha il compito di schermare tutte le operazioni non sicure di JSBayes, cioè quelle che vanno a modificare la struttura della rete rischiando quindi di minare la potenziale integrità dei dati. Essa espone solamente i metodi necessari per effettuare il ricalcolo delle probabilità e che quindi non modificano in alcun modo lo stato.

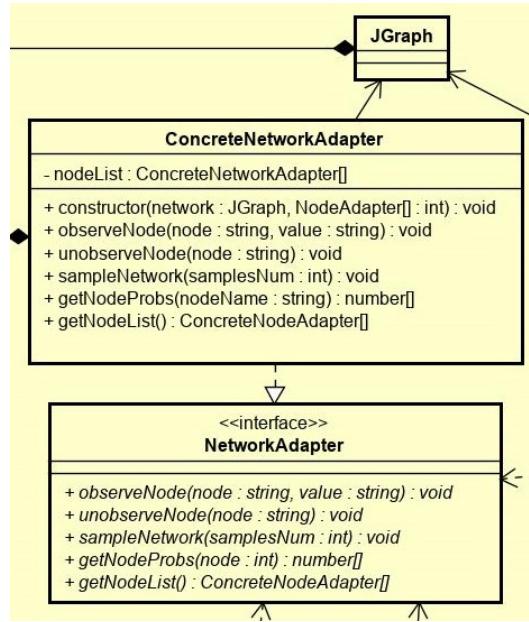


Figura 4: Diagramma UML NetworkAdapter

6.1.2 NodeAdapter

Per lo sviluppo delle funzionalità del plug-in il team ha avuto la necessità di aggiungere informazioni ai singoli nodi della rete bayesiana memorizzata da JSBayes. Estendere dalle classi offerte dalla libreria era impensabile a causa dei molteplici problemi descritti all'inizio di questa sezione. Per questo motivo abbiamo deciso di realizzare un adapter anche per

i singoli nodi di JSBayes ed includere all'interno di questi tutte le informazioni aggiuntive necessarie, come gli `AbstractValue` che verranno descritti in dettaglio all'interno della sezione seguente.

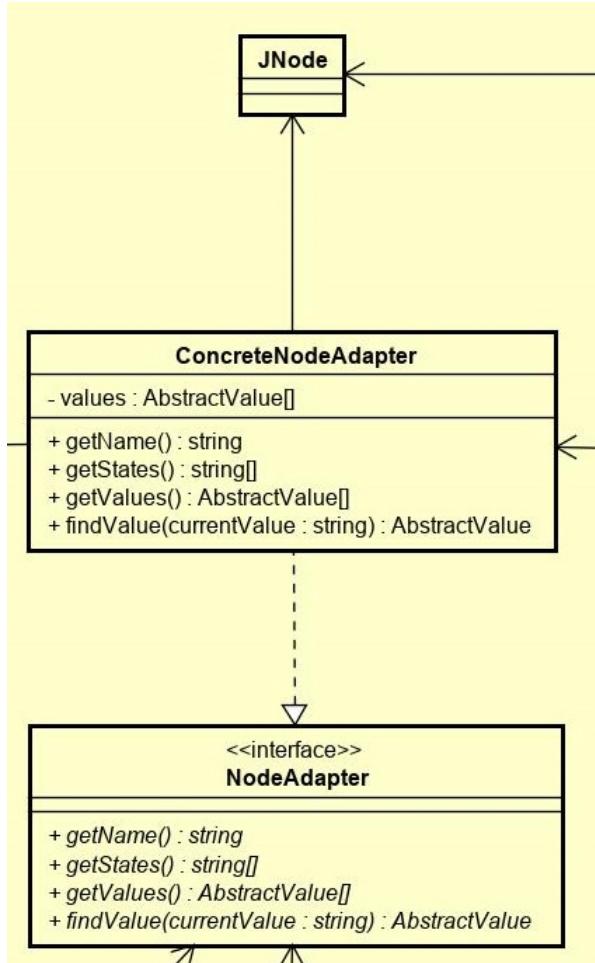


Figura 5: Diagramma UML NodeAdapter

6.1.3 AbstractValue

Per poter determinare in quale stato il nodo si trova sulla base dei dati prelevati da una qualsiasi fonte dati, come un database, il semplice nome che JSBayes associa ad ogni stato non era sufficiente. Il compito degli `AbstractValue` è proprio quello di colmare questa lacuna. Invocando su un oggetto `AbstractValue` il metodo `isValueType(value:string):bool` e passandogli come parametro il risultato di una lettura da una qualsiasi fonte dati l'`AbstractValue` ritorna un booleano indicando all'invocatore se lo stato associato è attualmente vero o meno.

Il team ha realizzato solamente tre implementazioni concrete di abstract value, rispettivamente `RangeValue`, `StringValue` e `BoolValue`, tuttavia è possibile realizzare altre implementazioni semplicemente estendendo dall'interfaccia `AbstractValue`.

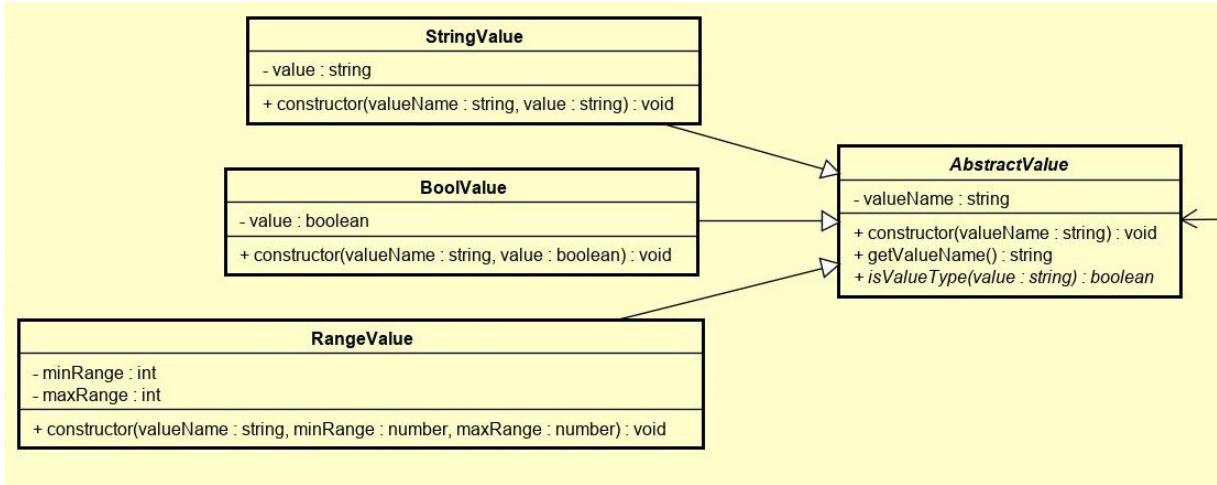


Figura 6: Diagramma UML AbstractValue

6.1.4 NetBuilder

Per garantire l'integrità dei dati abbiamo deciso di esternare il processo di creazione della rete JSBayes all'interno di un builder esterno il quale ha il compito di effettuare tutti i controlli necessari e ritornare un riferimento ad un NetworkAdapter solo al termine dell'intero processo di creazione invocando il metodo `build():NetworkAdapter`.

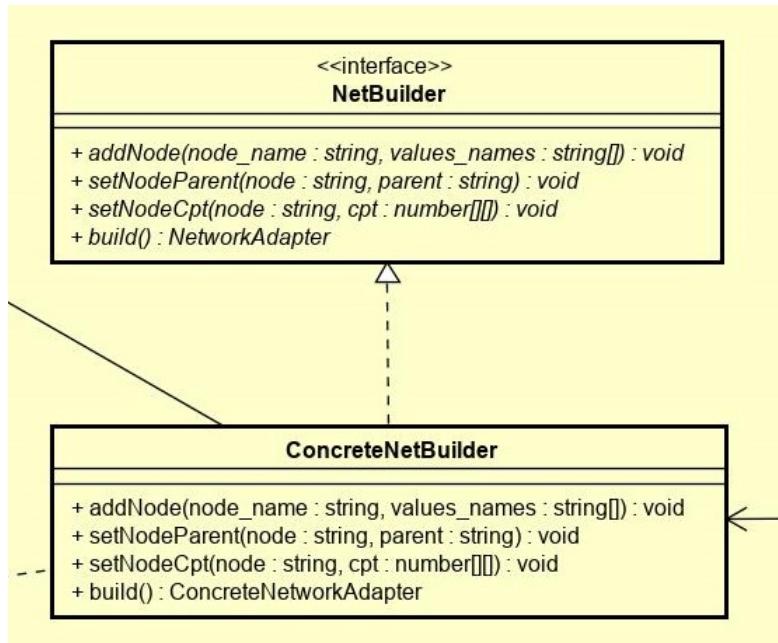


Figura 7: Diagramma UML NetBuilder

6.1.5 NetParser

Il NetParser ha il compito di realizzare un NetAdapter a partire dal contenuto di un file di testo, sfruttando le funzionalità esposte dal NetBuilder. Il team fino ad ora ha realizzato

un'unica implementazione, quella relativa a file di tipo JSON in quanto era l'unico formato richiesto esplicitamente dal proponente. Nel caso in cui un giorno fosse necessario utilizzare formati differenti sarà sufficiente realizzare una differente implementazione dell'interfaccia NetParser.

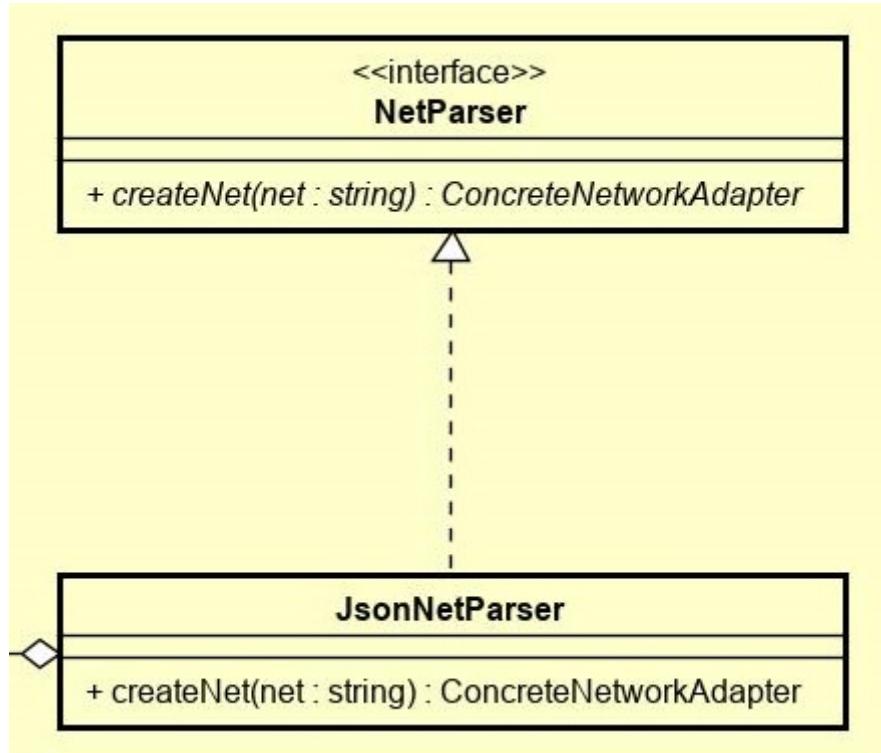


Figura 8: Diagramma UML NetParser

6.2 Buisness Layer

Il business layer ingloba all'interno di esso tutta la logica relativa al processo di calcolo delle probabilità. Dopo un'attenta analisi del problema il team è riuscito a scomporre il processo di calcolo delle probabilità in 3 macro-fasi principali:

- La lettura dei dati dai database associati ai nodi delle reti;
- Il processo di calcolo delle probabilità vero e proprio partendo dai dati letti precedentemente;
- La scrittura delle probabilità appena calcolate su un database che l'utente poi potrà utilizzare per fare la visualizzazione dei dati su grafana.

Ci è sembrato dunque abbastanza naturale modellare l'architettura del business layer in tre oggetti distinti ognuno dei quali si occupa di gestire una specifica fase del processo dall'inizio alla fine.

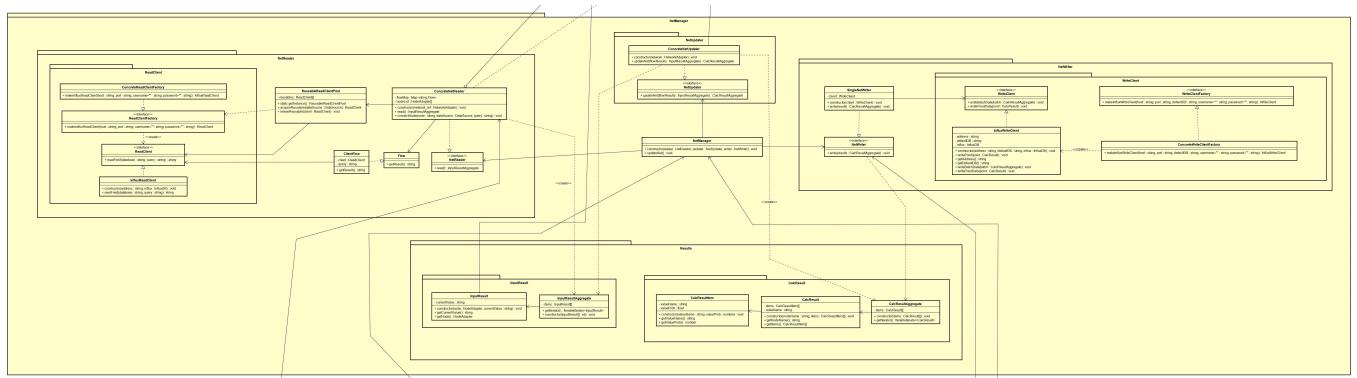


Figura 9: Diagramma UML Business Layer

Analizziamo ora più in dettaglio gli oggetti precedentemente citati.

6.2.1 NetReader

Il NetReader ha il compito di associare i nodi della rete a delle sorgenti dati scelte dall'utente, come per esempio dei database. Quest'ultime vengono rappresentate all'interno della nostra applicazione da oggetti di tipo Flow.

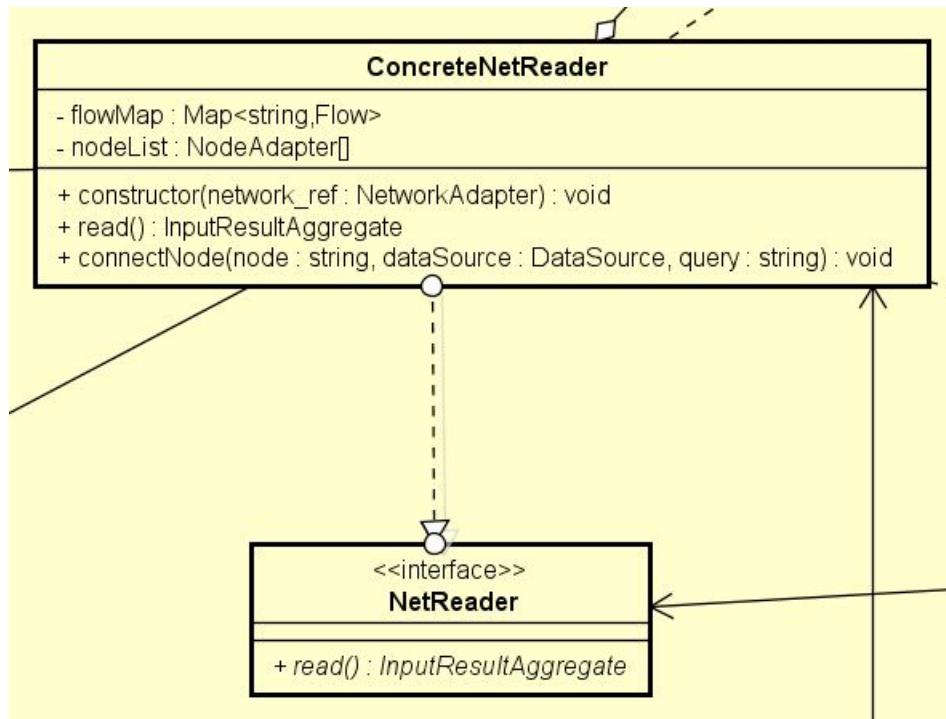


Figura 10: Diagramma UML NetReader

6.2.1.1 Flow

Un oggetto Flow ha il compito ritornare un risultato prelevato da una qualsiasi sorgente dati.

Il team attualmente ha realizzato un'unica implementazione di un Flow il ClientFlow il quale utilizza dei ReadClient, di cui parleremo più approfonditamente all'interno della prossima sezione, per interfacciarsi con dei database.

6.2.1.2 ClientPool

Per non dover istanziare più client di lettura per la stessa sorgente dati o database, abbiamo implementato una Object Pool per i client di lettura: quando si connette un nodo ad un flusso di dati, se il client richiesto esiste viene estratto dalla pool, altrimenti viene creato. Di seguito è possibile vedere un diagramma di sequenza il quale riassume il processo di associazione di un flusso dati ad un nodo nel caso in cui il client non sia già presente all'interno della pool

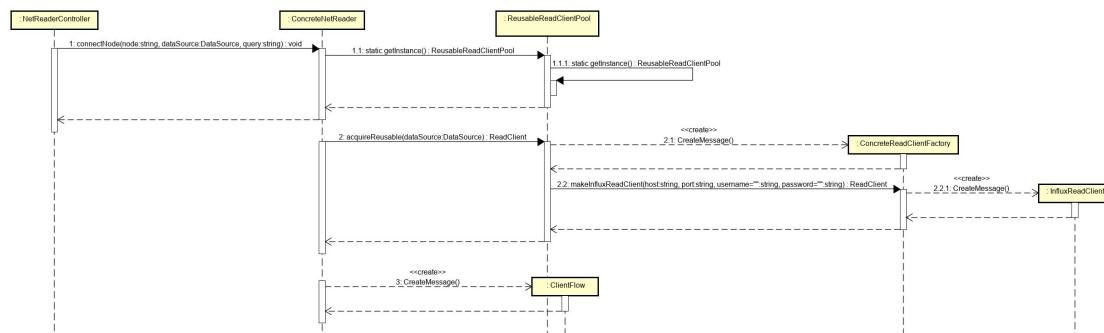


Figura 11: Diagramma di sequenza connessione nodi

6.2.2 NetUpdater

NetUpdater ha il compito di effettuare il calcolo delle probabilità vero e proprio utilizzando i dati letti da NetReader e le funzionalità esposte dal NetworkAdapter.

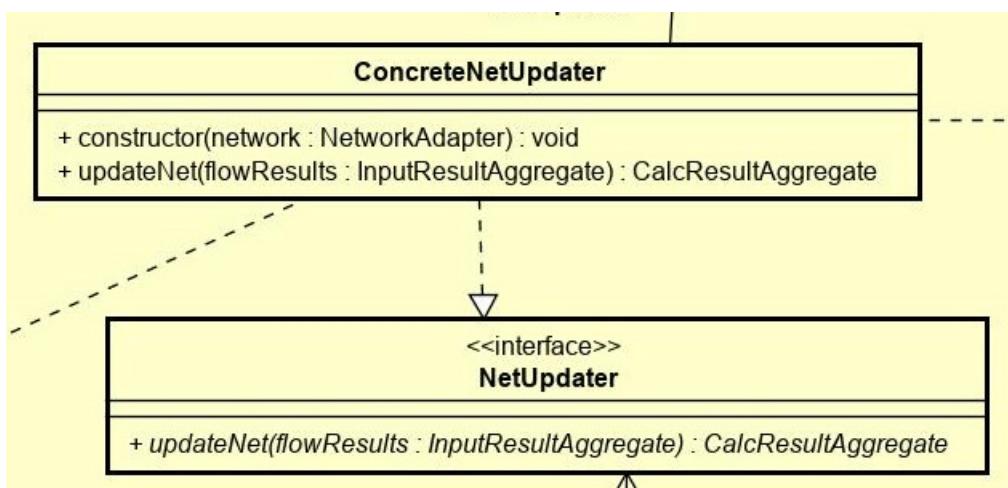


Figura 12: Diagramma UML NetUpdater

6.2.3 NetWriter

Il NetWriter ha il compito di scrivere all'interno di un apposito database scelto dall'utente tutti i dati calcolati dal NetUpdater. Per fare ciò esso si appoggia ad uno o più WriteClient classe che analizzeremo più approfonditamente all'interno del database layer. Il team, ai fini del progetto, attualmente ha realizzato un'unica implementazione del NetWriter, il SingleNetWriter il quale scrive tutti i dati ricevuti in input all'interno di un unico database. E' importante sottolineare tuttavia come, grazie all'interfaccia NetWriter, sia relativamente semplice integrare all'interno del plug-in una versione differente del NetWriter il quale per esempio permette la scrittura dei dati ricevuti in input all'interno di molteplici database differenti.

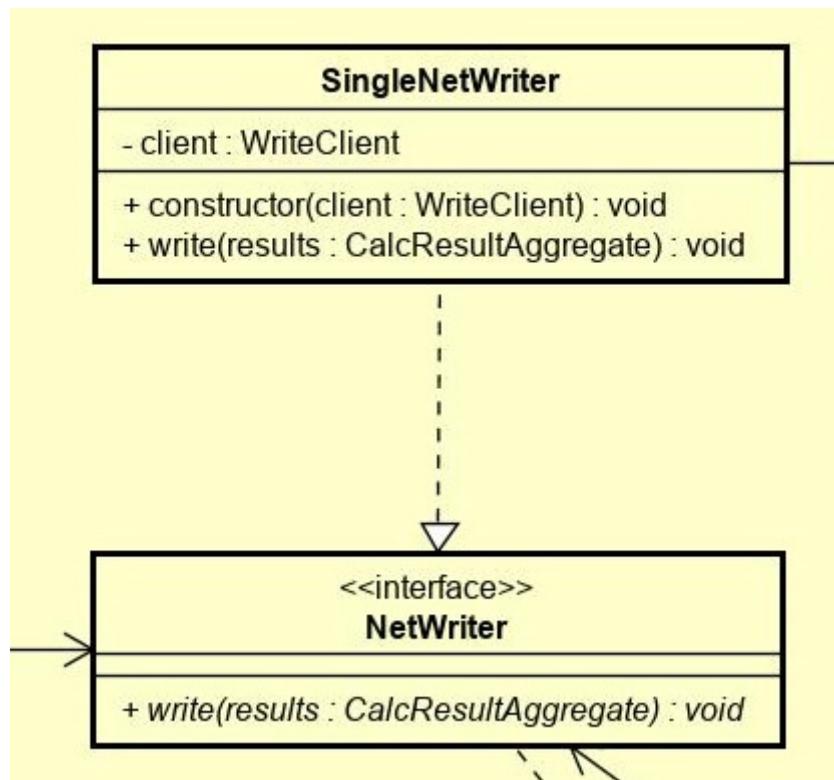


Figura 13: Diagramma UML NetWriter

6.2.4 NetManager

La classe NetManager ha un riferimento ad un'istanza di NetReader, NetUpdater e NetWriter. Esso ha il compito di coordinare l'intero processo di calcolo delle probabilità invocando sequenzialmente le operazioni giuste nell'ordine corretto.

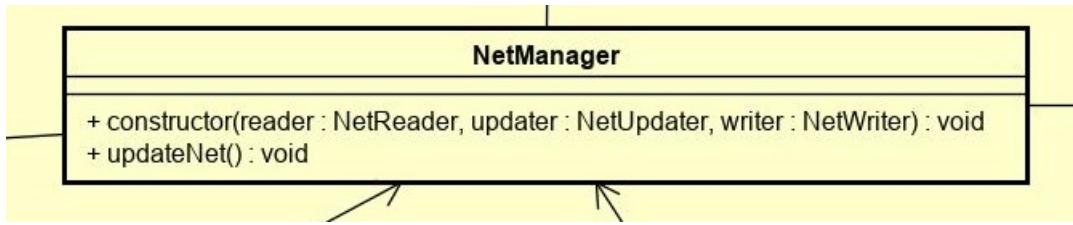


Figura 14: Diagramma UML NetManager

6.2.5 Results

NetReader, NetUpdater e NetWriter comunicano tra loro fornendo i risultati delle proprie operazioni all'interno di appositi oggetti “risultato” il cui cambiamento è altamente improbabile. In questo modo è possibile effettuare anche importanti modifiche alla struttura interna di queste macro-componenti senza che le interazioni con le altre ne risentano anche solo minimamente.

Inoltre abbiamo deciso di rendere questi oggetti risultato degli iterator per garantire un ordine nel loro scorrimento.

In particolare sono state realizzate due tipologie di result:

- Gli InputResult ritornati dal NetReader al termine delle sue operazioni ed utilizzati dal NetUpdater per effettuare il calcolo delle probabilità.

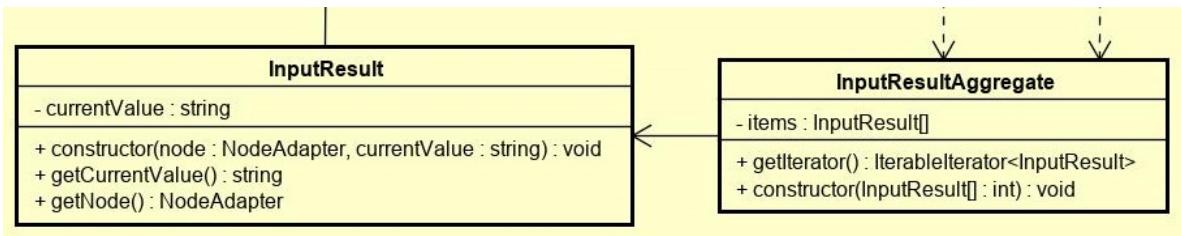


Figura 15: Diagramma UML InputResult

- I CalcResult ritornati dal NetUpdater al termine delle sue operazioni ed utilizzati dal NetWriter per effettuare la scrittura sul database di output indicato dall'utente;

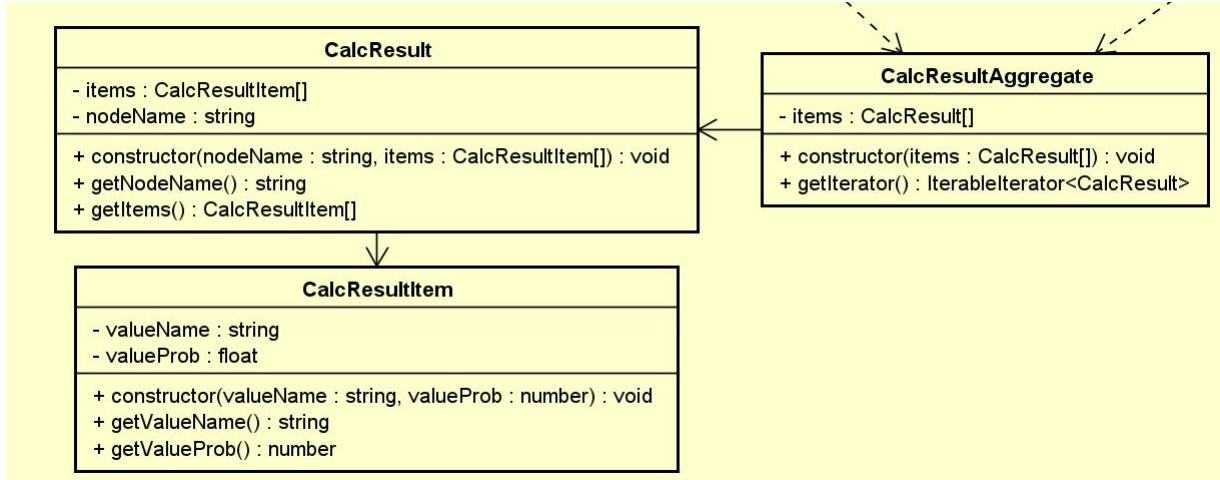


Figura 16: Diagramma UML CalcResult

Di seguito è possibile vedere un diagramma di sequenza il quale riassume l'intero processo di calcolo delle probabilità descritto all'interno di questa sezione.

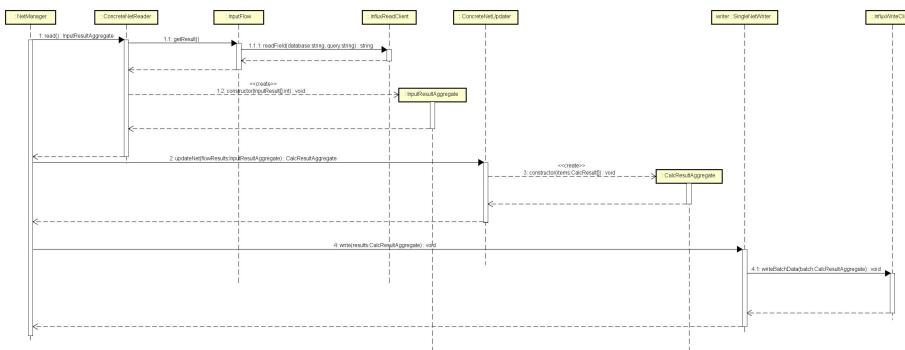


Figura 17: Diagramma sequenza calcolo probabilità

6.2.6 TimeBasedNetUpdater

Il TimeBasedNetUpdater possiede un riferimento al NetManager ed ha il compito di effettuare il processo di calcolo regolarmente secondo regole temporali definite dall'utente.

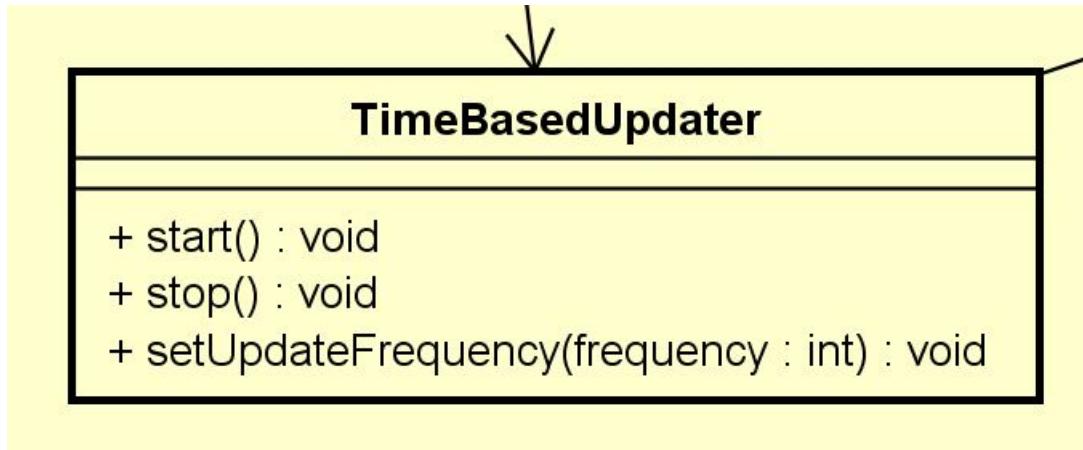


Figura 18: Diagramma UML TimeBasedNetUpdater

6.3 Database Layer

Il database layer è composto da tutto l'insieme di classi che permettono all'applicazione di interfacciarsi con sorgenti dati esterne come per l'appunto dei database. In particolare questo layer è suddivisibile in due tipologie di classi, i `ReadClient` ed i `WriteClient` i quali offrono rispettivamente funzionalità di scrittura e lettura su database.

Il team attualmente ha realizzato solo due implementazioni di `ReadClient` e `WriteClient`, quelle necessarie per interfacciarsi con database di tipo influx in quanto espressamente richiesto dal committente.

Tuttavia essendo i `ReadClient` e `WriteClient` degli strategy pattern è relativamente facile realizzare differenti implementazioni di quest'ultimi che permettono di interfacciarsi con database differenti.

Inoltre per semplificare la creazione dei client si è deciso di implementare un abstract factory.

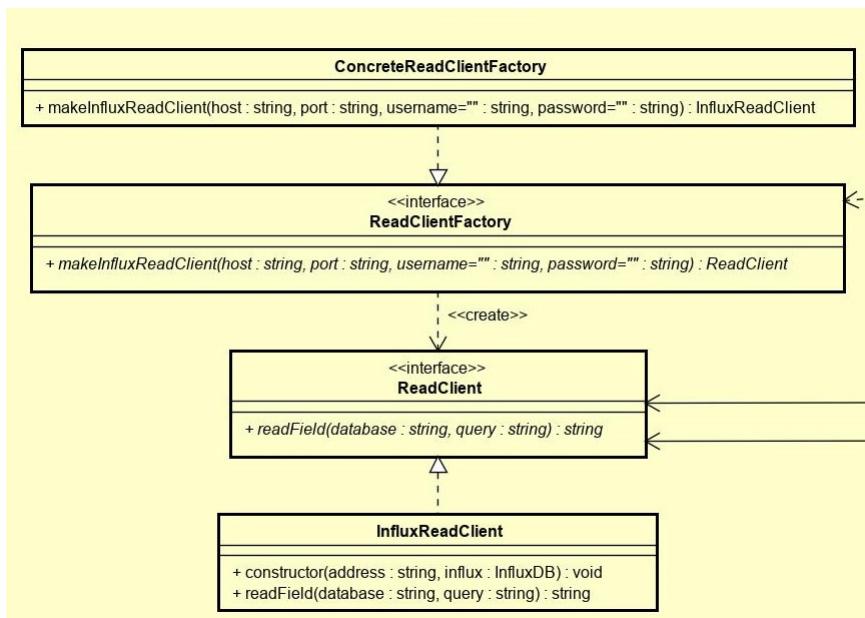


Figura 19: Diagramma UML ReadClient

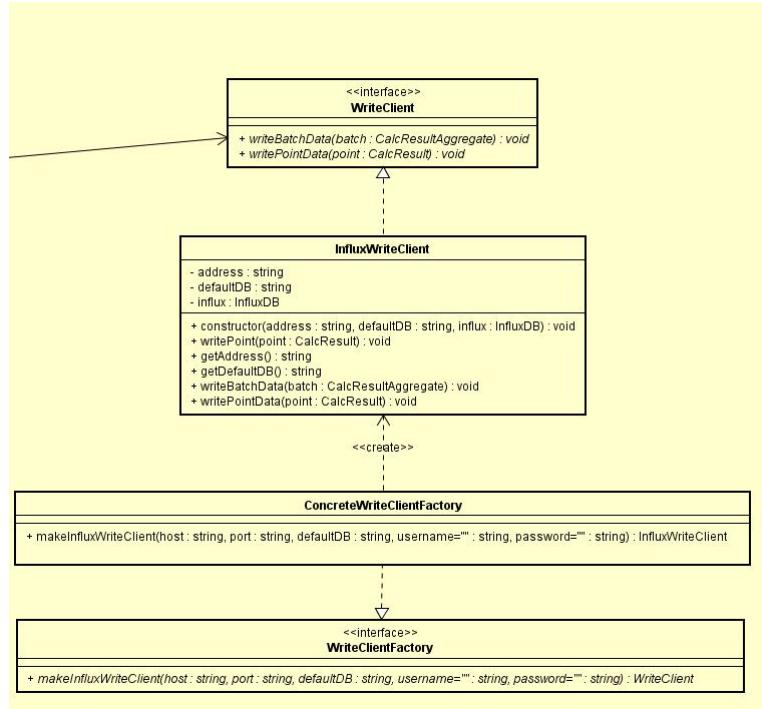


Figura 20: Diagramma UML WriteClient

6.4 Presentation Layer

Il presentation layer ha il compito di gestire l'interfaccia grafica del plug-in.

Ad ogni tab di edit del plug-in è associata una view ed un controller, in particolare abbiamo:

- Una tab per l'associazione dei database da cui prelevare i dati necessari per il processo di calcolo delle probabilità, la quale fa riferimento al NetReader come model;
- Una tab che permette di selezionare il database di output in cui si vogliono scrivere le probabilità calcolate, la quale fa riferimento al NetWriter come model;
- Una tab in cui l'utente può caricare un file JSON a partire dal quale verrà poi creata l'intera rete, il cui model è il NetParser;
- Una tab in cui l'utente può avviare e terminare il processo di ricalcolo continuo e scegliere la frequenza con cui quest'ultimo avverrà. Questa tab fa riferimento al Time-BasedUpdater come model.

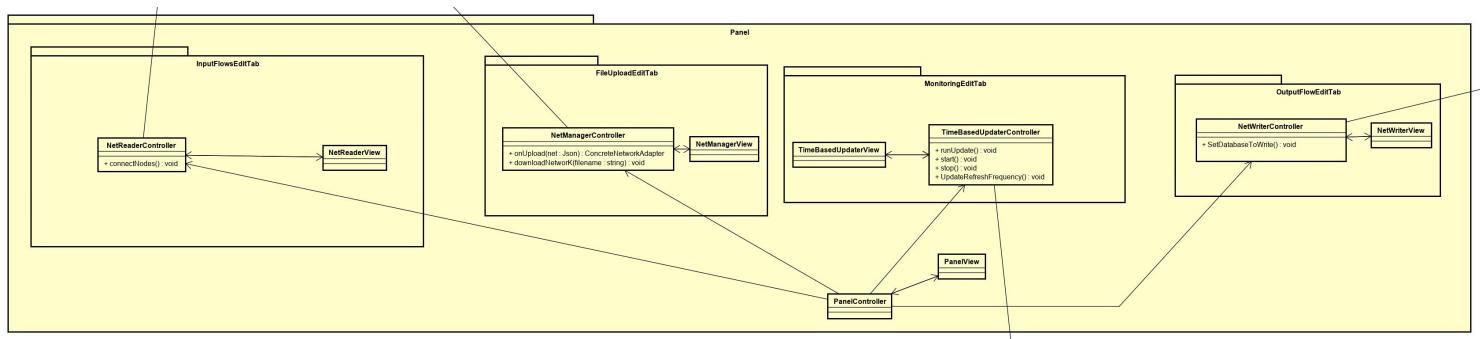


Figura 21: Diagramma UML presentation layer

Di seguito è possibile vedere un diagramma di sequenza il quale riassume l'intero processo di creazione delle classi che avviene una volta avviato il plug-in

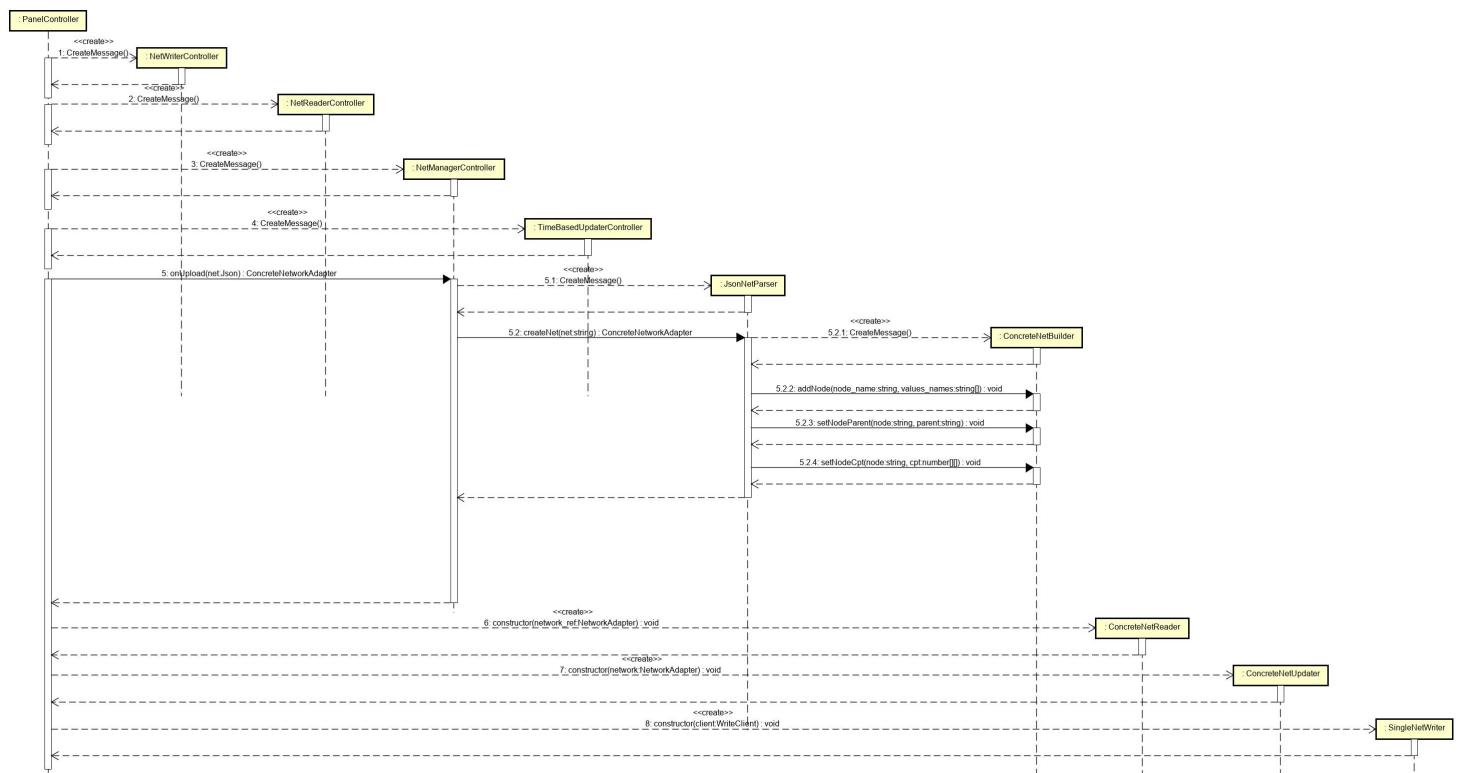


Figura 22: Diagramma sequenza creazione classi plug-in

7 Struttura del JSON

I file JSON che contengono la definizione di una rete Bayesiana sono strutturati nel seguente modo:

- **nodes**: contiene l'elenco dei nodi della rete;
 - **name**: nome del nodo;
 - **values**: contiene l'elenco delle values:
 - * **name**: nome della value;
 - * **type**: tipo della value;
 - * **value/rangeMin/rangeMax**: valore della value.
 - **parents**: contiene l'elenco dei nodi padri;
 - **cpt**: contiene l'elenco delle probabilità associate.

Ecco un esempio di rete con due nodi in formato JSON:

```
1 {
2     "nodes": [
3         {
4             "name": "Node1",
5             "values": [
6                 {
7                     "name": "Example of string value",
8                     "type": "string",
9                     "value": "Value1"
10                },
11                {
12                    "name": "Another example of string value",
13                    "type": "string",
14                    "value": "Value2"
15                }
16            ],
17            "parents": ["Node2"],
18            "cpt": [[0.2, 0.8], [0.5, 0.5], [0.4, 0.6]]
19        },
20        {
21            "name": "Node2",
22            "values": [
23                {
24                    "name": "Low Range",
25                    "type": "range",
26                    "rangeMin": 0,
27                    "rangeMax": 10
28                },
29                {
30                    "name": "Normal Range",
```

```
31         "type": "range",
32         "rangeMin": 11,
33         "rangeMax": 80
34     },
35     {
36         "name": "Alert Range",
37         "type": "range",
38         "rangeMin": 81,
39         "rangeMax": 100
40     }
41 ],
42 "parents": [],
43 "cpt": [[0.6, 0.2, 0.2]]
44 }
45 ]
46 }
```

A Glossario

D

Datasource

Una datasource è una sorgente di dati in Grafana. Solitamente si tratta di un database. Grafana supporta nativamente:

- CloudWatch;
- Elasticsearch;
- Graphite;
- InfluxDB;
- Microsoft SQL;
- MySql;
- OpenTSDB;
- PostgreSQL;
- Prometheus;
- Stackdriver.

F

Framework

Un framework è un architettura di supporto che ha lo scopo di ospitare e facilitare lo sviluppo di software.

I

IDE

IDE: Integradeted Development Environment è un applicazione software che fornisce tutta una serie di funzionalità, tool e features utili allo sviluppo software.

InfluxDB

InfluxDB è un database basato sul concetto di serie temporale. InfluxDB è specializzato e ottimizzato per il salvataggio e la lettura di serie temporali: record salvati in ordine temporale e caratterizzati dal timestamp, ovvero un campo che indica una data. InfluxDB è utilizzato per lo più in ambiti in cui è necessario salvare valori generati da sensori oppure analytics in tempo reale.

J

JSbayes

JSbayes è una libreria javascript che implementa un tool per l'inferenza di reti bayesiane.

N

Node

Node.js è un ambiente run-time che permette di eseguire codice JavaScript al di fuori del browser.

Npm

Npm: Node.js package manager è il package manager di default di Node.js. Consiste in un applicativo client a riga di comando e un database online pubblico chiamato npm registry. Npm registry può essere navigato tramite il client a riga di comando per cercare ed installare packages.

P

Package.json

Il package.json è un file, in formato json, utilizzato principalmente in progetti Javascript o Node.js che permette di configurare il package o l'applicazione. Permette inoltre a npm o yarn di salvare nomi e versioni dei package installati.

Panel

Un *panel* è una rappresentazione visiva di dati elaborati da un sistema mostrati sotto-forma di grafici.

R

Root directory

La root directory di una qualsiasi partizione, è la cartella più "alta" nella gerarchia delle cartelle.