Allegato Tecnico

Gruppo TeamAFK - Progetto "Predire in Grafana"

gruppoafk15@gmail.com

Informazioni sul documento

Versione	1.0.0
Approvatore	Alessandro Canesso
Redattori	Victor Dutca Simone Meneghin Olivier Utshudi Davide Zilio
Verificatori	Simone Federico Bergamin Fouad Farid Simone Meneghin
Uso	Esterno
Distribuzione	Prof. Cardin Riccardo TeamAFK

Descrizione

Allegato Tecnico contenente le scelte architetturali che il *TeamAFK* ha effettuato ai fini realizzativi del progetto *Predire in Grafana*. Comprende i design pattern utilizzati e i diagrammi di attività, sequenza, classi e package.

Indice

1 Introduzione **4**
1.1 Scopo del documento 4
1.2 Scopo del prodotto 4
1.3 Glossario 4
1.4 Riferimenti 4
1.4.1 Riferimenti normativi 4
1.4.2 Riferimenti informativi 4

2 Architettura del prodotto **6**
2.1 Descrizione generale 6
2.1.1 Diagrammi delle attività 7
2.2 Architettura Prediction Tool 12
2.2.1 Descrizione 12
2.2.2 Diagrammi dei package 13
2.2.3 Diagrammi delle classi 13
2.2.4 Diagrammi di sequenza 14
2.2.5 Design pattern comportamentali utilizzati 14
2.3 Architettura Prediction Plug-in 15
2.3.1 Descrizione 15
2.3.2 Diagrammi dei package 16
2.3.3 Diagrammi delle classi 16
2.3.4 Diagrammi di sequenza 18
2.3.5 Design pattern comportamentali utilizzati 19

3 Requisiti soddisfatti **20**
3.1 Tabella del soddisfacimento dei requisiti 20
3.2 Grafici del soddisfacimento dei requisiti 22

Elenco delle figure

2.1.1 Diagramma delle attività dello UC1 . . . 7
2.1.2 Diagramma delle attività dello UC2 . . . 8
2.1.3 Diagramma delle attività dello UC3 . . . 9
2.1.4 Diagramma delle attività dello UC4 . . . 10
2.1.5 Diagramma delle attività dello UC5 . . . 11
2.1.6 Diagramma delle attività dello UC6 . . . 12
2.2.1 Diagramma dei package del Prediction Tool . . . 13
2.2.2 Diagramma delle classi del Prediction Tool . . . 13
2.2.3 Diagramma di sequenza del TrainSVM . . . 14
2.3.1 Diagramma dei package del Prediction Plug-in . . . 16
2.3.2 Diagramma delle classi del Model del Prediction Plug-in . . . 16
2.3.3 Diagramma delle classi della View del Prediction Plug-in . . . 17
2.3.4 Diagramma delle classi del Controller del Prediction Plug-in . . . 17
2.3.5 Diagramma di sequenza del Panel . . . 18
3.2.1 Totale requisiti soddisfatti . . . 22
3.2.2 Requisiti obbligatori soddisfatti . . . 23
3.2.3 Requisiti desiderabili e opzionali soddisfatti . . . 23

Elenco delle tabelle

3.1.1 Tabella del soddisfacimento dei requisiti 20

1 Introduzione

1.1 Scopo del documento

Lo scopo del documento è una descrizione esaustiva delle capacità del software *Predire in Grafana* sviluppato dal team AFK. Il documento si concluderà con un resoconto su quanto sia stato soddisfatto dei vari requisiti.

1.2 Scopo del prodotto

Predire in Grafana$_G$ soddisfa le necessità di monitorare costantemente applicazioni e informazioni contenute in esse. Con questo scopo il team AFK si propone per la realizzazione per l'azienda *Zucchetti SPA* di un tool$_G$ di addestramento e di un plug-in$_G$ di monitoraggio per Grafana che utilizzi algoritmi di SVM$_G$ e Regressione Lineare$_G$ sui dati in ingresso.

1.3 Glossario

Per evitare ambiguità nei documenti formali, viene fornito il documento *Glossario*, contenente tutti i termini considerati di difficile comprensione. Perciò nella documentazione fornita ogni vocabolo contenuto nel Glossario è contrassegnato dalla lettera G a pedice.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- **Capitolato d'appalto C4**:
  https://www.math.unipd.it/~tullio/IS-1/2019/Progetto/C4.pdf;
- *norme_di_progetto_v3.0.0*;
- *VI_2020-06-01*.

1.4.2 Riferimenti informativi

- *analisi_dei_requisiti_v3.0.0*;
- **Model-View Patterns - Materiale didattico del corso di Ingegneria del Software**:
  https://www.math.unipd.it/~rcardin/sweb/2020/L02.pdf;
- **Diagrammi delle classi - Materiale didattico del corso di Ingegneria del Software**:
  https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E01b.pdf;
  - proprietà e operazioni, slide 11 - 34;
  - caratteristiche, slide 35 - 38.

- **Diagrammi dei package - Materiale didattico del corso di Ingegneria del Software**:
  https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E01c.pdf;
  - package e dipendenze, slide 12 - 15.
- **Diagrammi di sequenza - Materiale didattico del corso di Ingegneria del Software**:
  https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E02a.pdf;
  - partecipanti e segnali, slide 8 - 16;
  - modellazione, slide 18 - 21.
- **Design Pattern Comportamentali - Materiale didattico del corso di Ingegneria del Software**:
  https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E08.pdf
  - strategy, slide 6 - 8.

2 Architettura del prodotto

2.1 Descrizione generale

Il progetto *Predire in Grafana* prevede la realizzazione di due moduli: un plug-in per la piattaforma Grafana e un tool esterno di supporto, rispettivamente chiamati **Prediction Plug-in** e **Prediction Tool**.
Il Prediction Tool si occupa di addestrare un algoritmo di *SVM* o *Regressione Lineare* utilizzando un dataset inserito dall'utente, per poi generare un file json contenente le informazioni necessarie per poter effettuare un calcolo di predizione. Questo modulo è stato sviluppato seguendo il pattern *Model-View-ViewModel (MVVM)*.
Il Prediction Plug-in invece si occuperà di ricevere in input il json e una volta collegati i predittori, contenuti nel file, ad un flusso dati, permetterà di iniziare ad effettuare i calcoli di previsione. Questo modulo è stato sviluppato seguendo il pattern *Model-View-Controller (MVC)*.
Le motivazioni principali che hanno portato alla scelta del design pattern MVVM per il Prediction Tool sono:

- per la realizzazione del componente è stato utilizzato *React* e abbiamo ritenuto che questo pattern si accoppiasse bene con la struttura di quest'ultimo.

Le motivazioni principali che hanno portato alla scelta del design pattern MVC per il Prediction Plug-in sono:

- abbiamo ritenuto che questo pattern si accoppiasse meglio con la struttura dei plug-in di Grafana.

Inoltre entrambi i pattern permettono:

- di disaccoppiare la parte di *presentation logic* da quella di *business logic*;
- il riutilizzo di alcune componenti in altri contesti, senza doverli modificare.

Di seguito sono mostrati i diagrammi delle attività corrispondenti agli Use Cases descritti nell'*Analisi dei Requisiti*.

2.1.1 Diagrammi delle attività

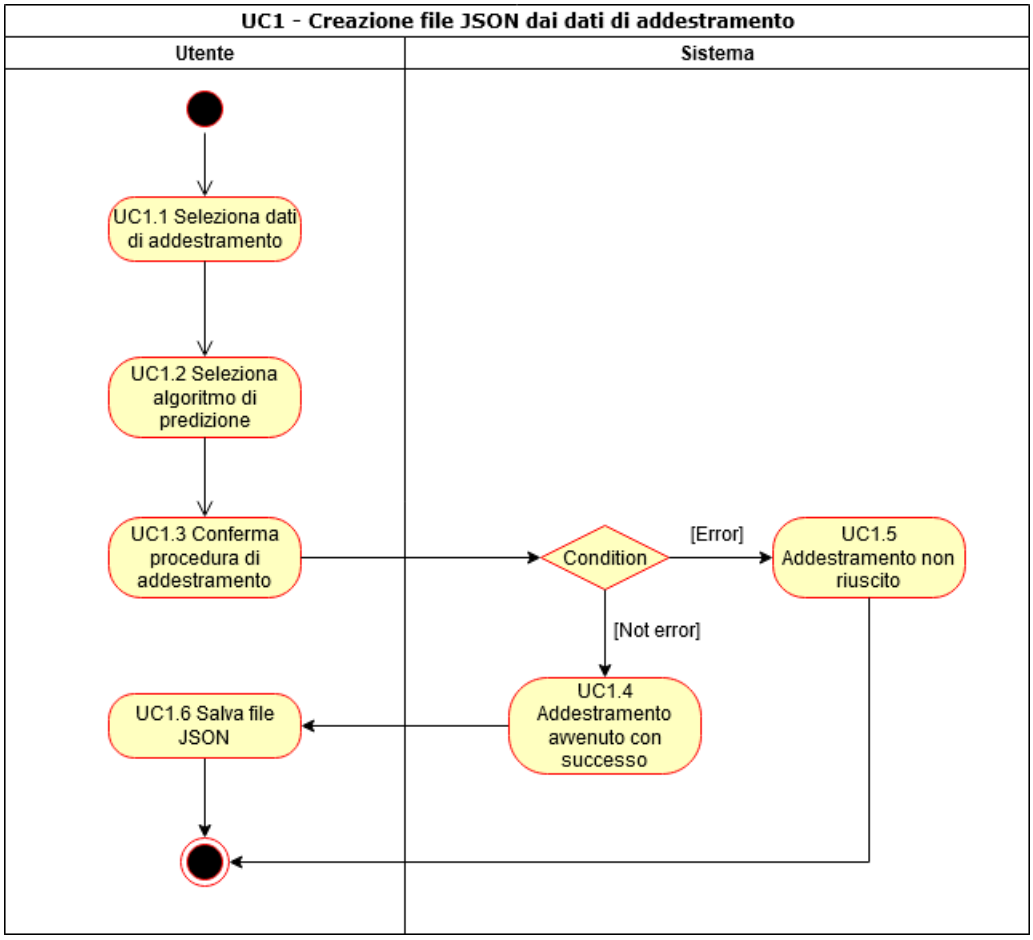

Figura 2.1.1: Diagramma delle attività dello UC1

Specifica: lo UC1.5 racchiude i seguenti tipi di errori:

- **UC7**: Nessun file CSV caricato;
- **UC8**: Nessun algoritmo selezionato;
- **UC9**: File CSV incompatibile.

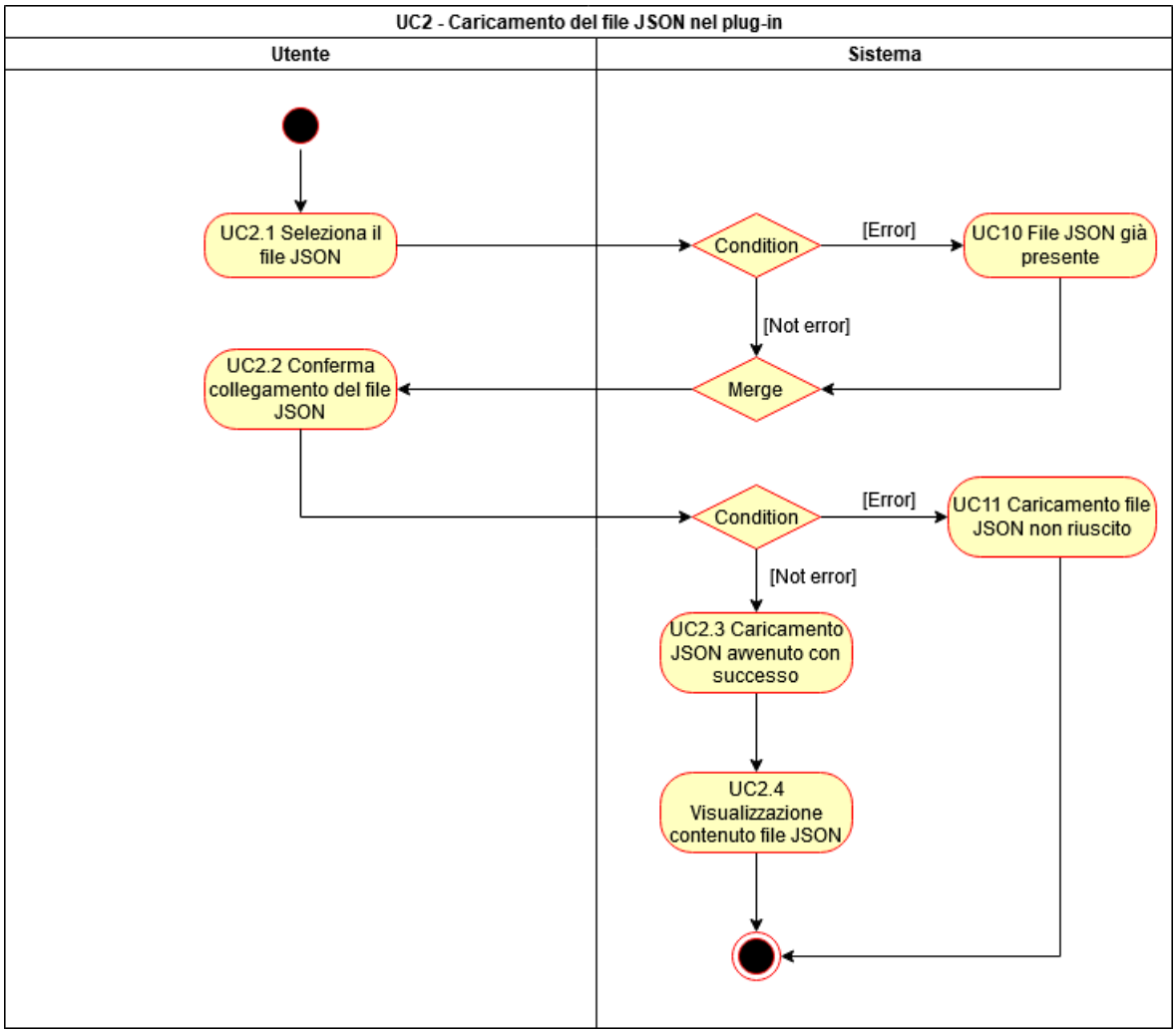

Figura 2.1.2: Diagramma delle attività dello UC2

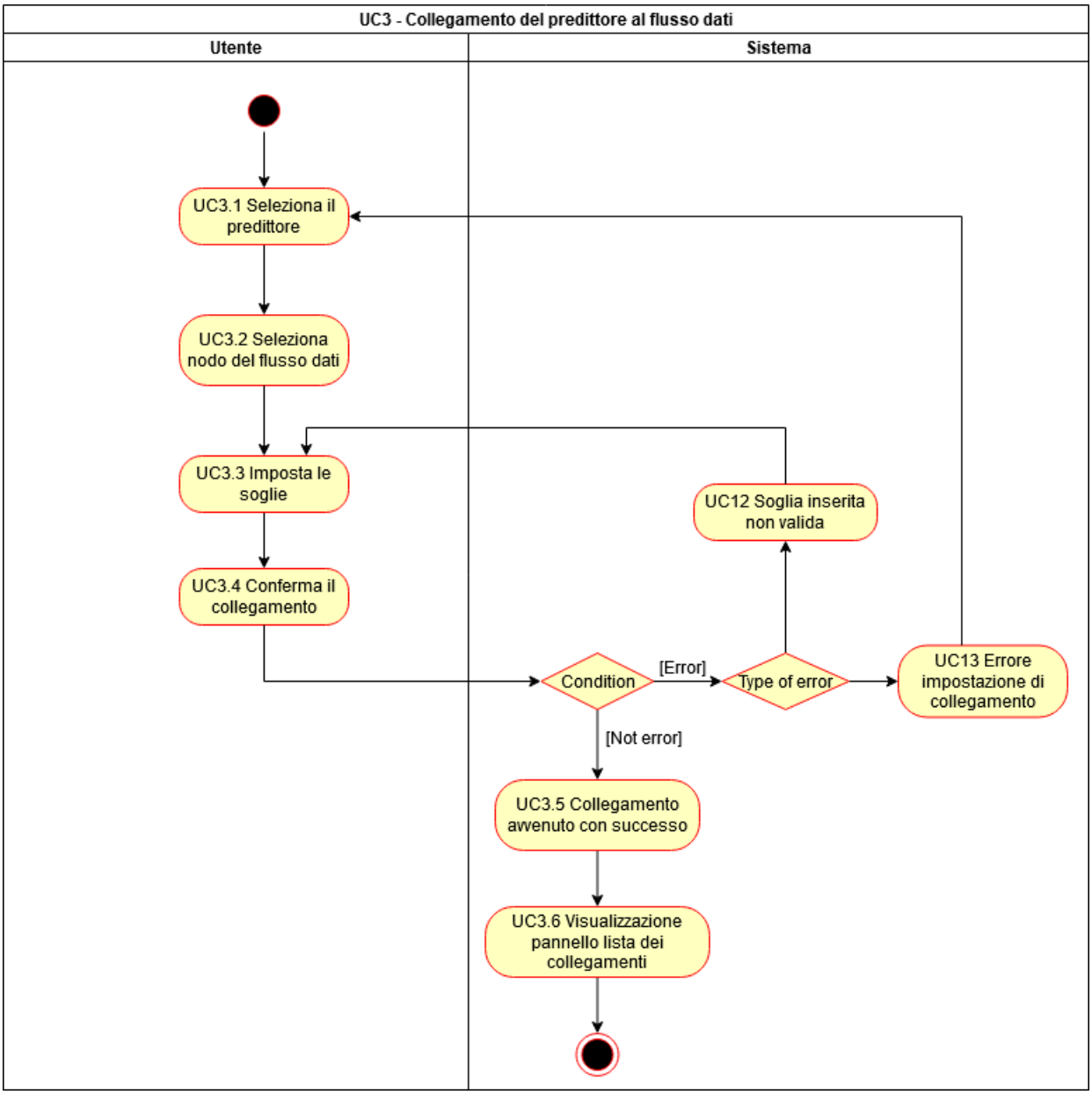

Figura 2.1.3: Diagramma delle attività dello UC3

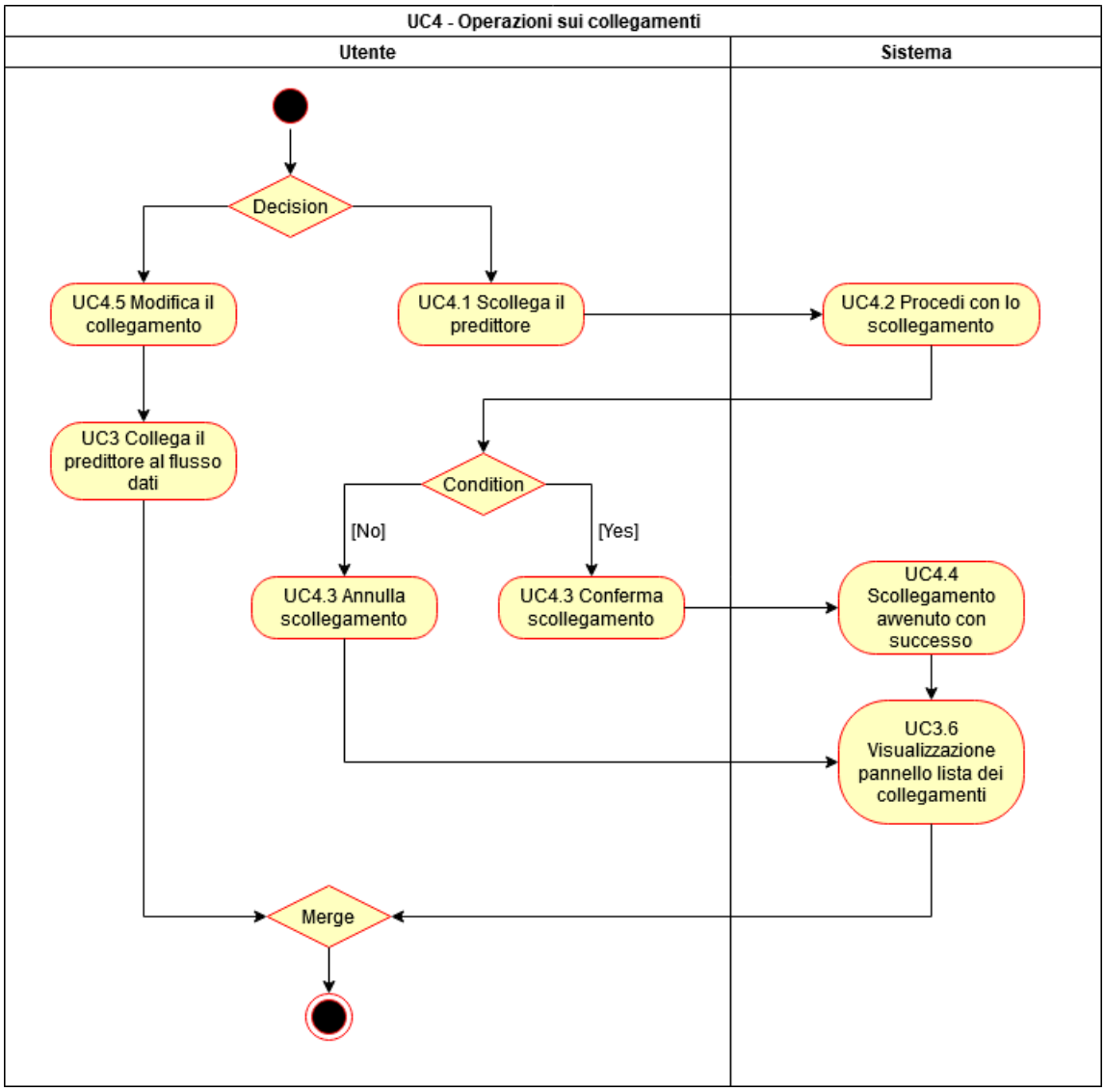

Figura 2.1.4: Diagramma delle attività dello UC4

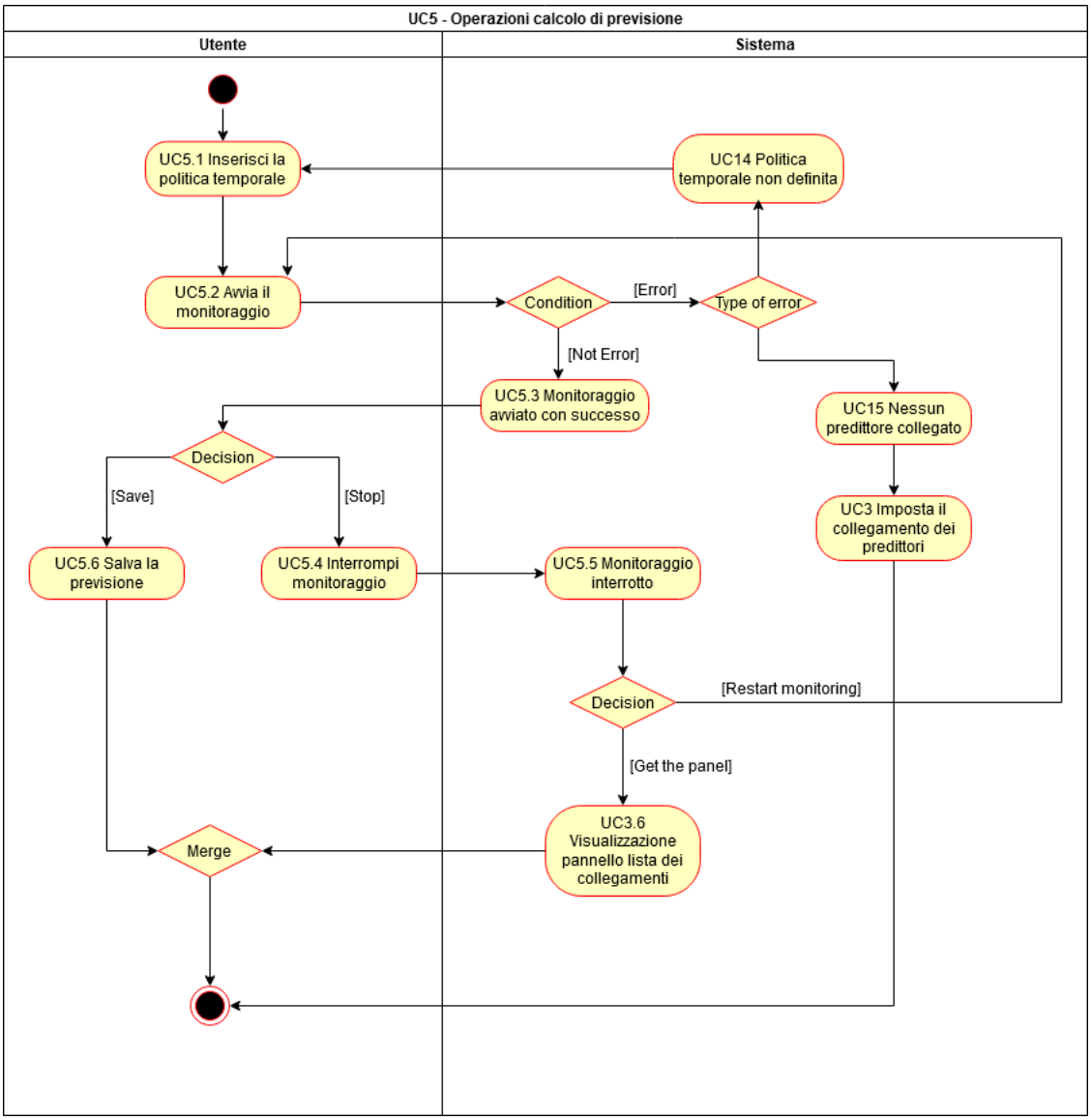

Figura 2.1.5: Diagramma delle attività dello UC5

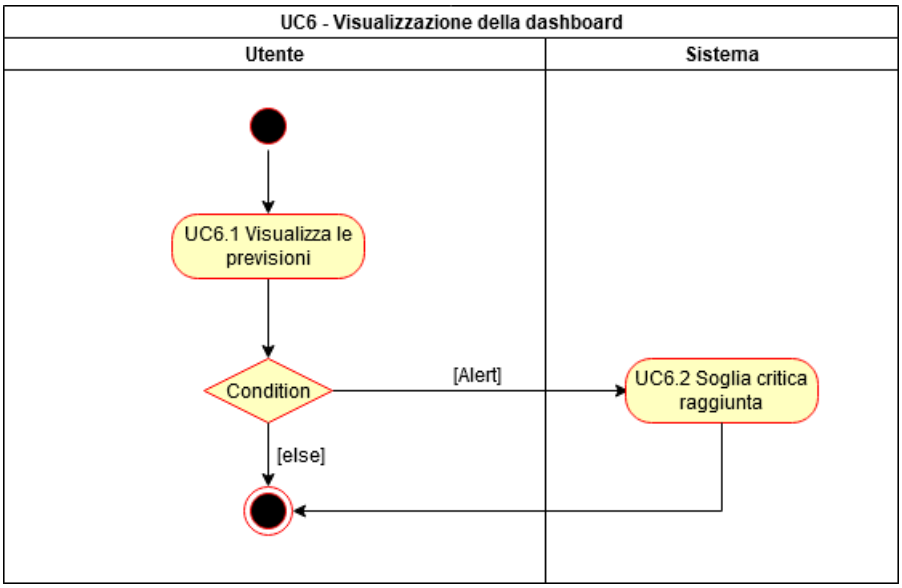

Figura 2.1.6: Diagramma delle attività dello UC6

2.2 Architettura Prediction Tool

2.2.1 Descrizione

L'implemetenazione del tool è stata realizzata utilizzando il design pattern MVVM. Il passaggio di dati dalle view al model avviene attraverso la modifica di un campo dati *props* immesso dal *ViewModel*. Attraverso queste *props* il *ViewModel* chiama le funzioni corrette quando l'utente interagisce con la vista. La divisione tra Business Logic$_G$ e Presentation Logic è rafforzato da questo utilizzo delle `props`. Nel modello viene fornita funzionalità per la gestione degli algoritmi tramite le classi *SVMtrain* e *RLtrain* che verranno utilizzate dal *ViewModel*.

2.2.2 Diagrammi dei package

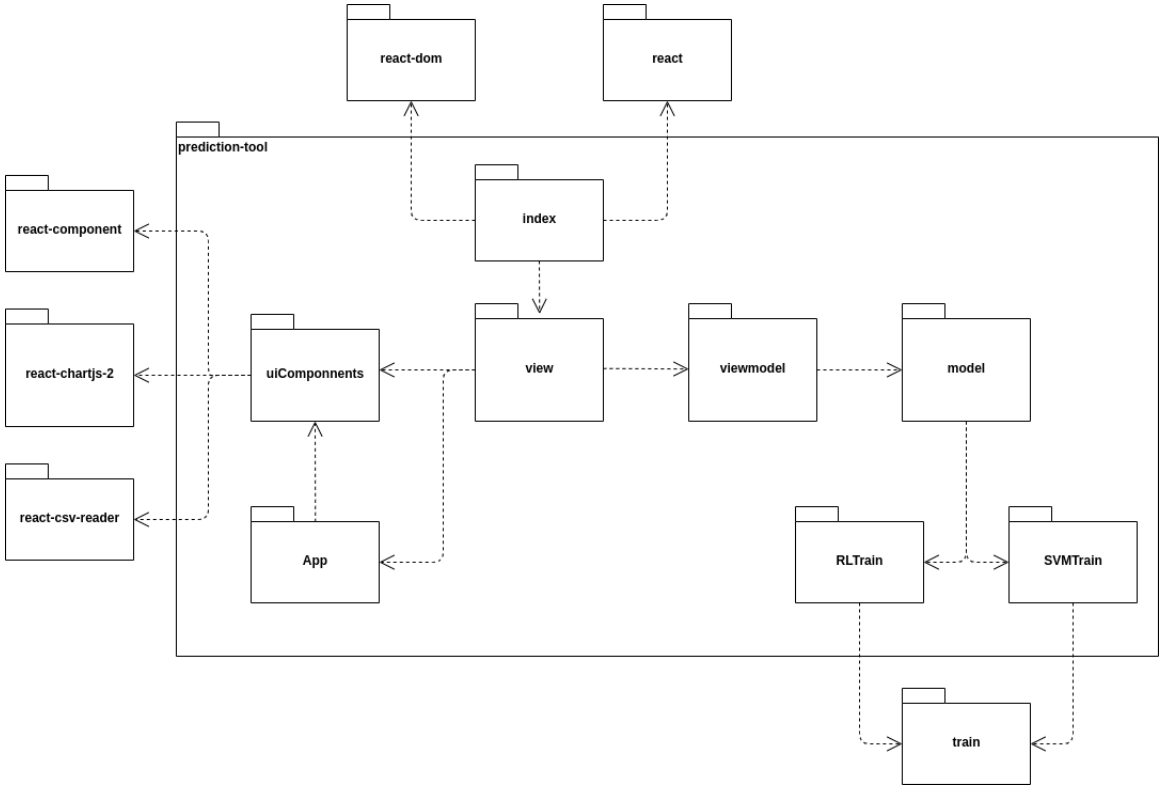

Figura 2.2.1: Diagramma dei package del Prediction Tool

2.2.3 Diagrammi delle classi

Quello che segue è il diagramma delle classi del Prediction Tool.

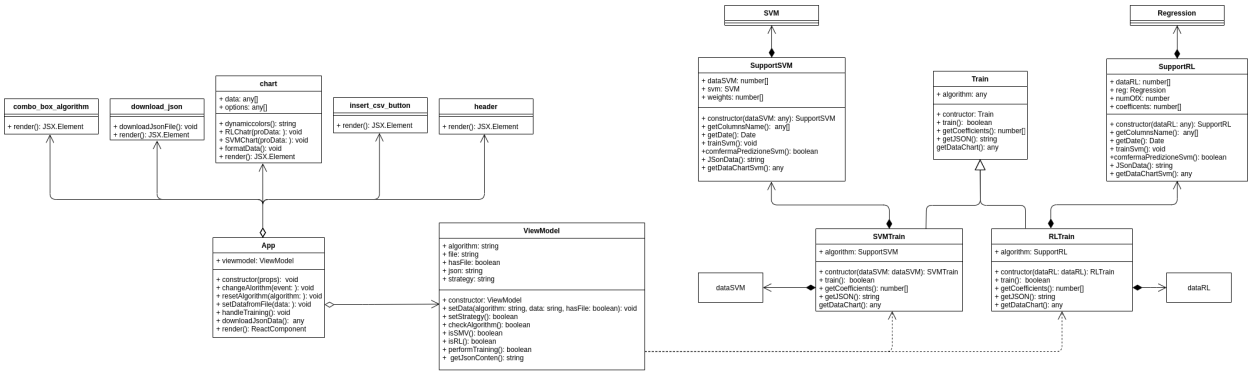

Figura 2.2.2: Diagramma delle classi del Prediction Tool

2.2.4 Diagrammi di sequenza

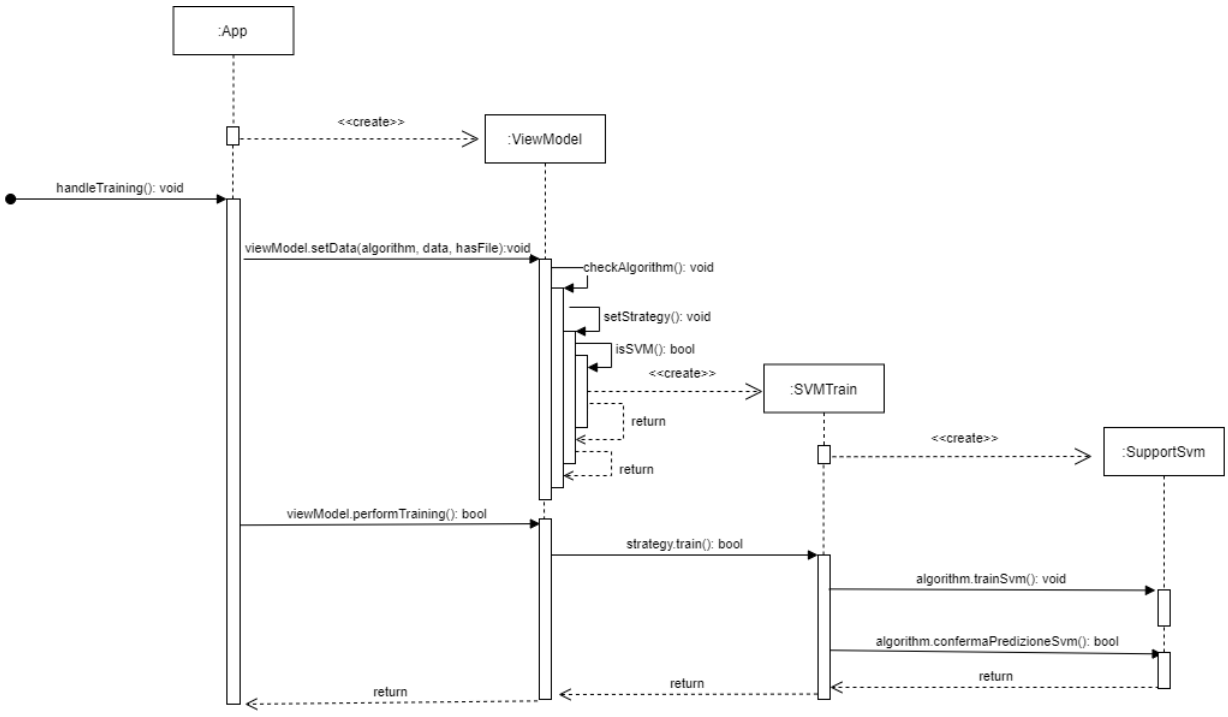

Figura 2.2.3: Diagramma di sequenza del TrainSVM

2.2.5 Design pattern comportamentali utilizzati

Per quanto riguarda i design pattern comportamentali utilizzati, durante la progettazione delle classi del Prediction Tool ci siamo accorti che alcune classi correlate tra loro (quelle adibite a richiamare le librerie per gli algoritmi di Machine Learning RL e SVM) differivano soltanto per il comportamento effettivo, dato dalla libreria richiamata. I metodi utilizzati dalle due classi erano in effetti più varianti di uno stesso algoritmo (quello di addestramento). Per queste ragioni abbiamo pensato di utilizzare il design pattern **Strategy**, così da definire una famiglia di algoritmi, incapsularli e renderli interscambiabili tra di loro, a seconda della scelta effettuata dall'utente. Nel nostro caso abbiamo identificato come *Strategy* l'oggetto contenente un riferimento all'algoritmo di previsione. Quest'ultimo rappresenta il modello della nostra applicazione, dove è contenuta effettivamente la business logic.

2.3 Architettura Prediction Plug-in

2.3.1 Descrizione

In questo modulo, composto da più pannelli all'interno di Grafana, verrà associato il predittore prodotto dal tool esterno al flusso dati monitorato appunto in Grafana. Per questo modulo è stato utilizzato il pattern architetturale *MVC*. In questa architettura l'Editor e il Panel condividono la variabile *props*, istanziata da Grafana. All'importazione del file JSON, il controller si occuperà di:

- leggere il suo contenuto attraverso la funzione `setJson(file)`;
- salvare una copia dei predittori;
- applicare l'algoritmo di previsione corretto attraverso il metodo `setStrategy()`;
- inviare una notifica alla viste di successo caricamento file JSON.

Per quanto riguarda la comunicazione che avviene tra *View* e *Model* viene gestita dal *Controller*. Questo è stato reso possibile per la presenza in *Editor* di un istanza della classe *Controller*. In questo modo, ad ogni caricamento del file JSON, la vista comunica al *Controller* l'avvenuto caricamento (tramite il metodo `setJson(file)`) passando come parametro il file appena ricevuto. Una volta che il *Controller* termina la lettura del file notifica la vista di aggiornarsi (tramite il metodo `update()`), mostrando le nuove informazioni ottenute dal file.

A seguito di una discussione col proponente, abbiamo deciso di creare una componente Panel come punto di partenza del nostro plug-in, la quale ha il compito di creare una dashboard per permettere all'utente di prendere familiarità con le funzionalità dei vari pannelli.

2.3.2 Diagrammi dei package

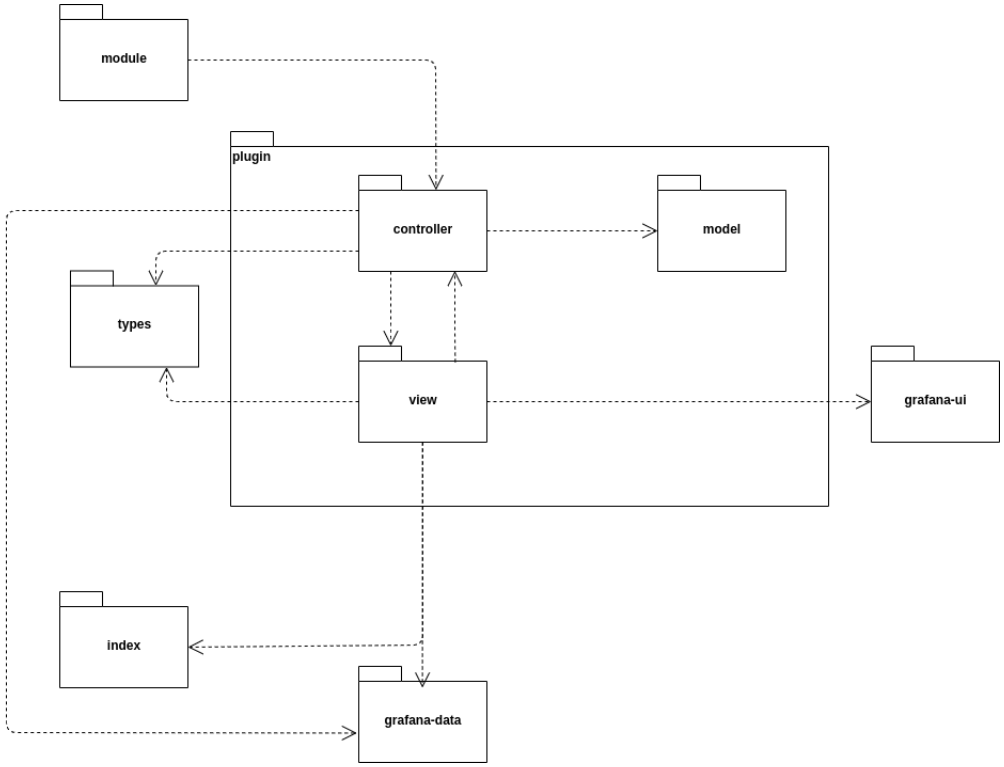

Figura 2.3.1: Diagramma dei package del Prediction Plug-in

2.3.3 Diagrammi delle classi

Model

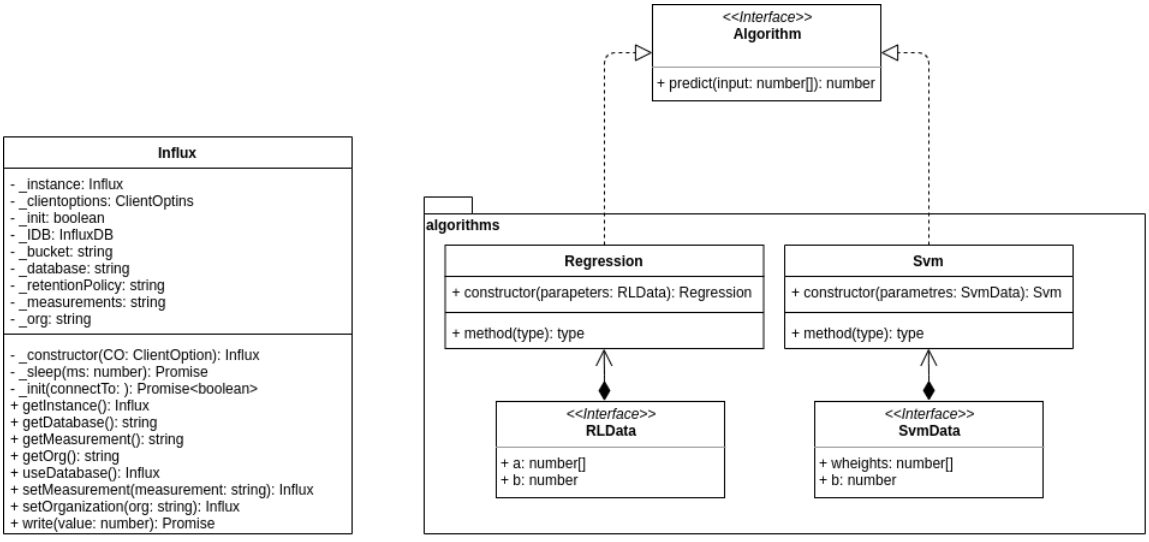

Figura 2.3.2: Diagramma delle classi del Model del Prediction Plug-in

View

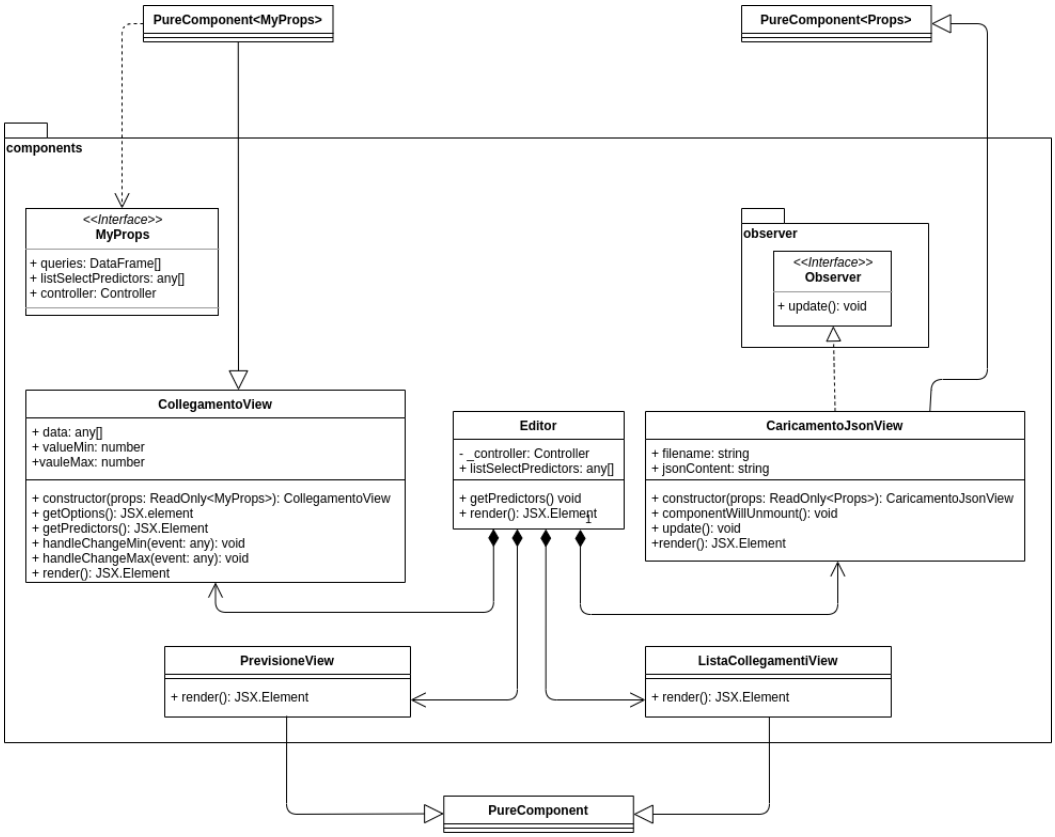

Figura 2.3.3: Diagramma delle classi della View del Prediction Plug-in

Controller

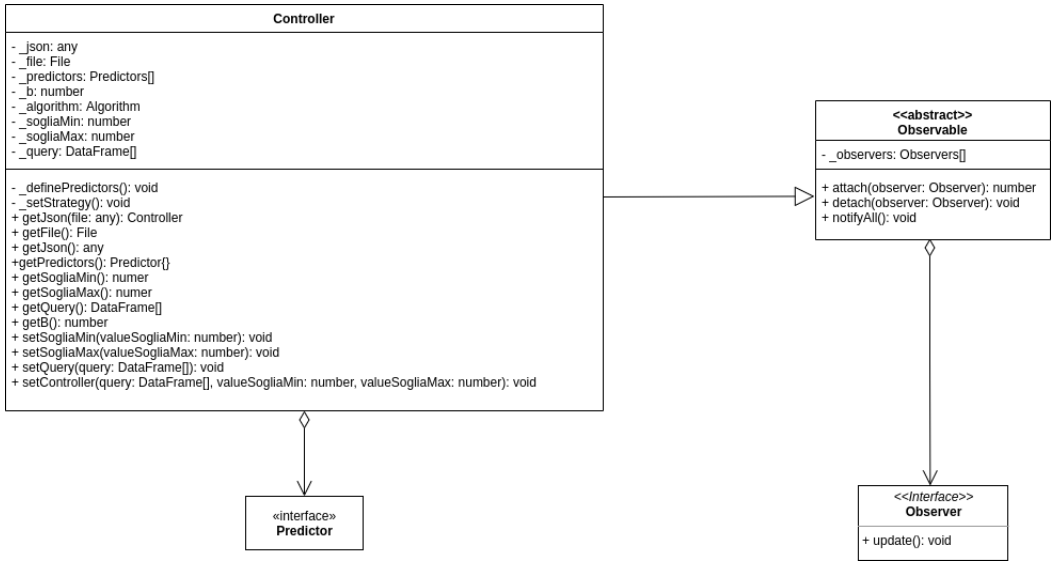

Figura 2.3.4: Diagramma delle classi del Controller del Prediction Plug-in

2.3.4 Diagrammi di sequenza

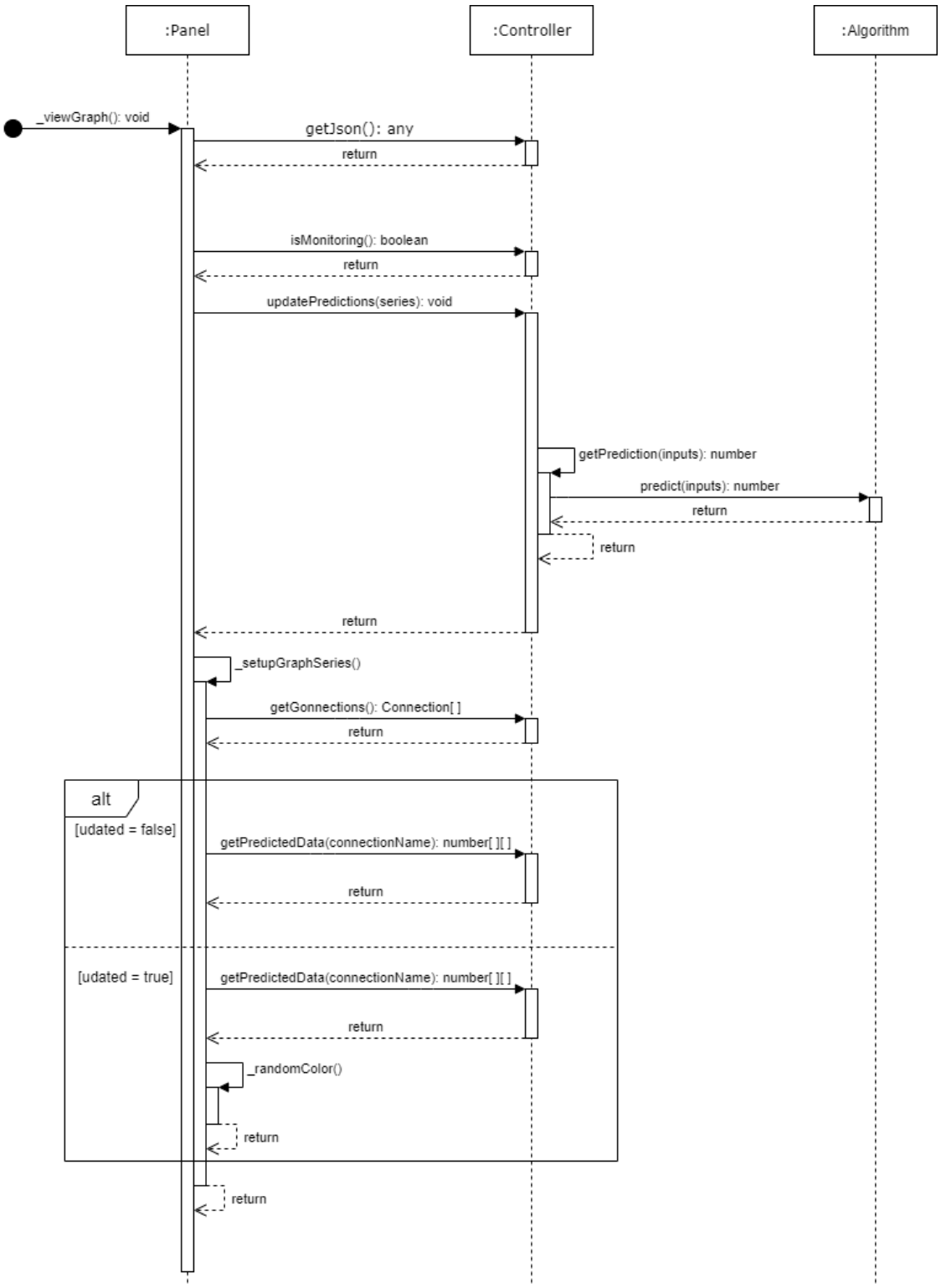

Figura 2.3.5: Diagramma di sequenza del Panel

2.3.5 Design pattern comportamentali utilizzati

Tra i design pattern che sono stati adottati per l'implementazione del plugin, rientrano l'**Observer Pattern** e lo **Strategy Pattern**.
L'Observer pattern è stato utilizzato per semplificare la comunicazione tra controller e le componenti della vista che necessitano di essere aggiornate a seguito della modifica dello stato del controler. Infatti nel nostro caso la classe *subject*, ovvero la classe di cui monitorarne lo stato, è la componente controller mentre la componente *Observer*, ovvero tutte le classi che necessitano di essere aggiornate a seguito di un cambiamento dello stato del subject, sono le componenti grafiche utilizzate dalla classe *Editor*.
Lo Strategy pattern è stato utilizzato per gli algoritmi di predizione, infatti il controller ha un riferimento all'interfaccia *Algorithm* che espone il metodo `predict(inputs)`. Nel model sono quindi presenti le classi concrete (*Regression* e *Svm*) che implementano l'algoritmo di predizione e il controller si occupa di istanziare l'algoritmo corretto rispetto quanto letto dal file json.

3 Requisiti soddisfatti

3.1 Tabella del soddisfacimento dei requisiti

Tabella 3.1.1: Tabella del soddisfacimento dei requisiti

Codice	Esito
Re1F1	Soddisfatto
Re1F1.1	Soddisfatto
Re1F1.2	Soddisfatto
Re1F1.3	Soddisfatto
Re2F1.4	Soddisfatto
Re2F1.5	Soddisfatto
Re1F1.6	Soddisfatto
Re1F2	Soddisfatto
Re1F2.1	Soddisfatto
Re1F2.2	Soddisfatto
Re2F2.3	Soddisfatto
Re1F2.4	Soddisfatto
Re1F3	Non soddisfatto
Re1F3.1	Soddisfatto
Re1F3.2	Soddisfatto
Re3F3.3	Non soddisfatto
Re1F3.4	Soddisfatto
Re2F3.5	Non soddisfatto
Re1F3.6	Soddisfatto
Re1F4	Soddisfatto
Re1F4.1	Soddisfatto
Re1F4.2	Soddisfatto
Re1F4.3	Soddisfatto
Re2F4.4	Soddisfatto
Re1F4.5	Soddisfatto

Tabella 3.1.1: (continua)

Codice	Esito
Re1F5	Non soddisfatto
Re1F5.1	Soddisfatto
Re1F5.2	Non soddisfatto
Re2F5.3	Soddisfatto
Re1F5.4	Non soddisfatto
Re2F5.5	Non soddisfatto
Re1F5.6	Non soddisfatto
Re1F6	Soddisfatto
Re1F6.1	Soddisfatto
Re3F6.2	Non soddisfatto
Re1F7	Soddisfatto
Re1F8	Soddisfatto
Re1F9	Soddisfatto
Re1F10	Non soddisfatto
Re1F11	Non soddisfatto
Re1F12	Non soddisfatto
Re1F13	Non soddisfatto
Re1F14	Non soddisfatto
Re1Q1	Soddisfatto
Re1Q2	Soddisfatto
Re1Q2.1	Soddisfatto
Re2Q2.2	Non soddisfatto
Re1Q3	Soddisfatto
Re1Q4	Soddisfatto
Re2Q5	Soddisfatto
Re2Q6	Soddisfatto
Re2Q7	Soddisfatto
Re1V1	Soddisfatto

Tabella 3.1.1: (continua)

Codice	Esito
Re1V1.1	Soddisfatto
Re1V1.2	Soddisfatto
Re1V1.3	Soddisfatto
Re1V1.4	Soddisfatto
Re1V2	Soddisfatto
Re1V3	Soddisfatto
Re1V4	Soddisfatto
Re1V5	Soddisfatto

3.2 Grafici del soddisfacimento dei requisiti

Dei 61 requisiti individuati, 46 sono stati stati soddisfatti, risultando quindi in una copertura del 75%.

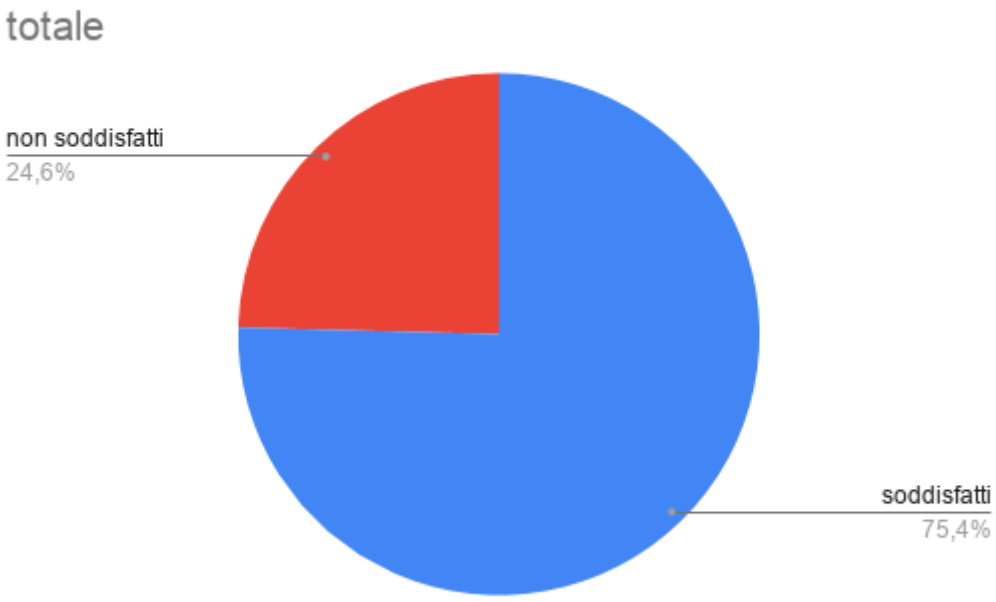

Figura 3.2.1: Totale requisiti soddisfatti

I requisiti obbligatori sono 48 sui 61 totali. Di questi 48 requisiti 38 sono stati soddisfatti completamente, risultando in una copertura del 79%; i restanti 10 requisiti sono messaggi d'errore o d'avviso che non sono quindi necessari al corretto funzionamento del prodotto. Alcuni di questi requisiti sono però già presenti in una implementazione parziale.

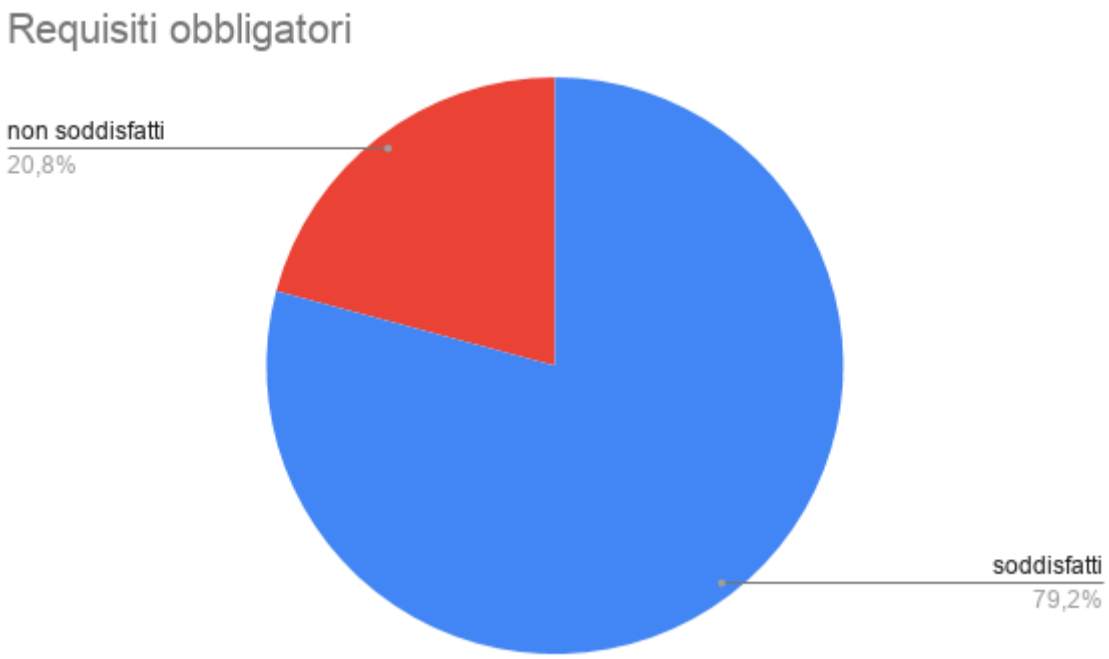

Figura 3.2.2: Requisiti obbligatori soddisfatti

I requisiti desiderabili e opzionali sono 13 sui 61 totali. di questi 13 requisiti 8 sono stati soddisfatti completamente, risultando quindi in una copertura del 61%.

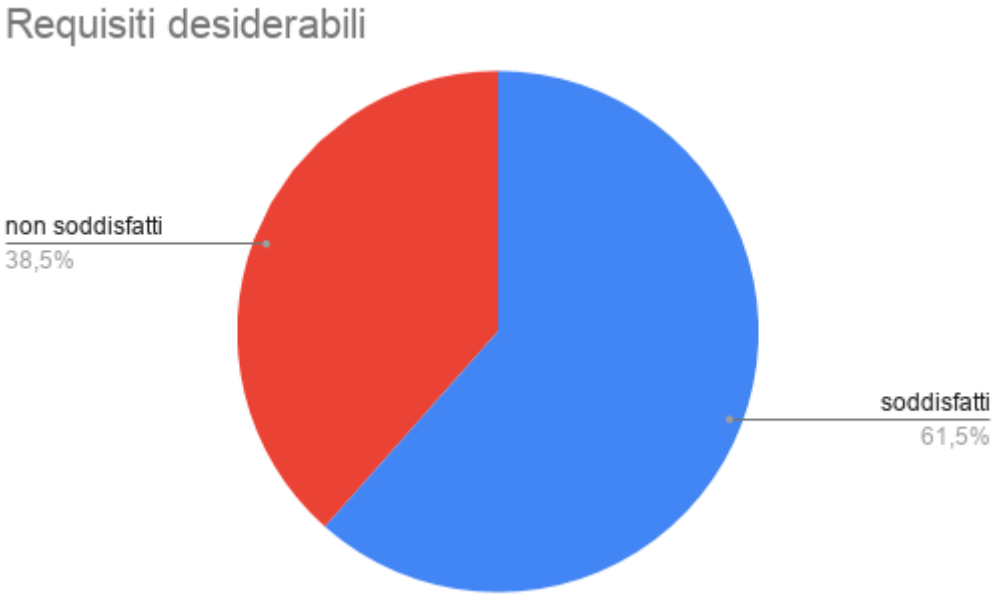

Figura 3.2.3: Requisiti desiderabili e opzionali soddisfatti