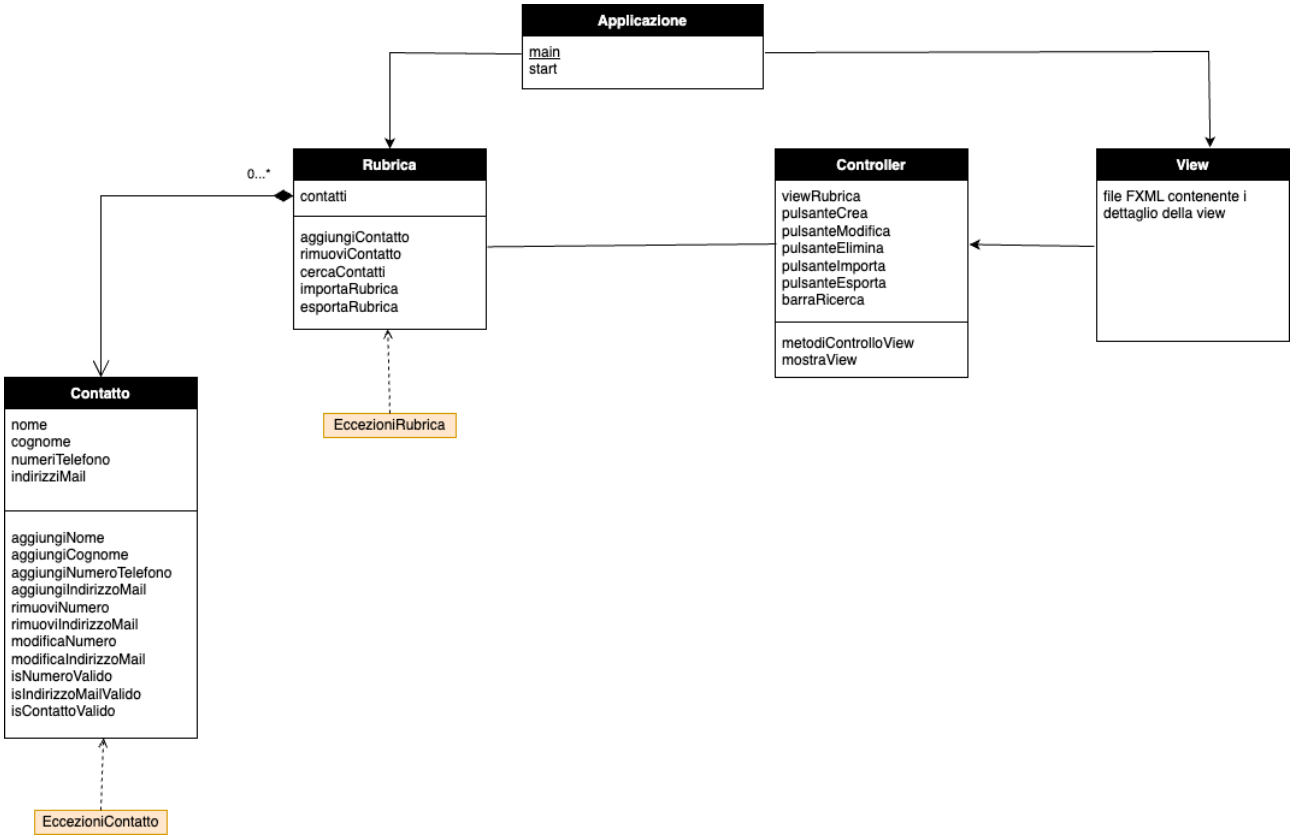
# Design del progetto

Nella suddivisione degli elementi caratterizzanti del nostro progetto si è scelto di seguire una *decomposizione object-oriented*, suddividendo il dominio del problema in classi aventi propri attributi e metodi, e decomponendo tali classi in altre classi via via più semplici da implementare.

Seguendo questo approccio è stato possibile ottenere un primo *diagramma delle classi concettuale*, da sfruttare come prototipo per quello che poi è diventato il *diagramma delle classi definitivo*.

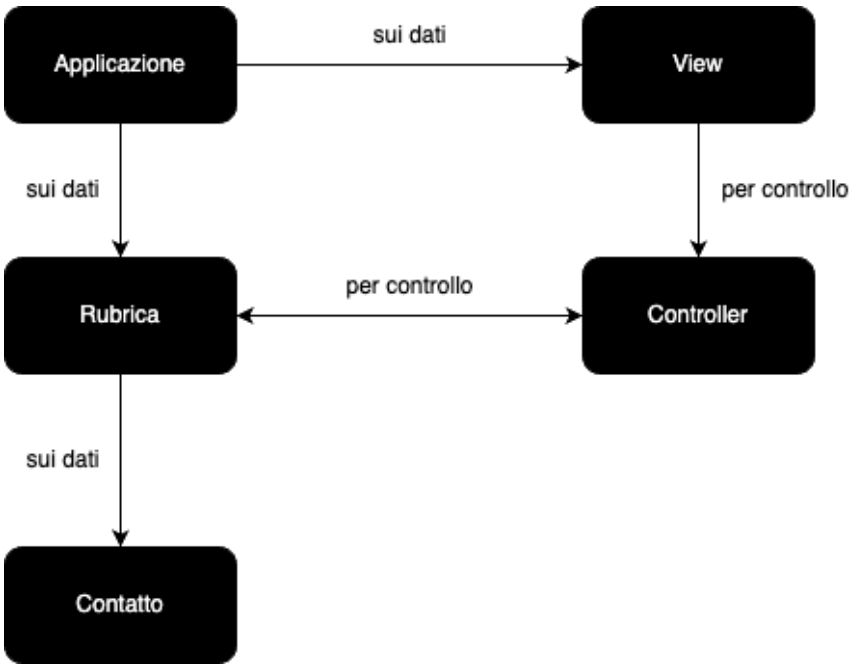



Osservando tale insieme ci è stato possibile fare delle valutazioni su *coesione e accoppiamento* tra classi.

*Valutazioni sulla coesione.*

| Modulo | Coesione | Descrizione |
|---|---|---|
| **Applicazione** | *funzionale* | La classe gestisce solo la funzione principale main |
| **Rubrica** | *procedurale* | La classe contiene funzionalità diverse ma che spesso vengono usate insieme (creazione/gestione contatti, importazione/esportazione) |
| **Contatto** | *funzionale* | La classe contiene solo informazioni che costituiscono un contatto e i metodi per gestirle e verificarle |
| **Controller** | *comunicazionale* | La classe contiene le funzioni inerenti a tutti i pulsanti mostrati nella GUI; essi lavorano tutti sulla rubrica, ma svolgono funzionalità diverse |

*Valutazioni sull'accoppiamento.*




L'accoppiamento tra Applicazione e Rubrica è *per dati*, visto che il main (in Applicazione) richiama solo i metodi di Rubrica strettamente necessari alle operazioni che desidera svolgere.

Analogamente, anche l'accoppiamento tra Applicazione e View è *per dati*, visto Applicazione sfrutta l'FXMLLoader per caricare il file fxml della view e basta.

L'accoppiamento tra Rubrica e Controller è *per controllo*, visto che le due classi si scambiano informazioni di controllo: l'interfaccia grafica mostra informazioni che cambiano a seconda delle operazioni svolte in Rubrica, e la lista contatti di Rubrica si modifica in base ai pulsanti premuti nell'interfaccia grafica (i cui metodi sono esplicitati in Controller).

L'accoppiamento tra View e Controller è *per controllo*, dato che View – ossia il file fxml – specifica direttamente la classe Controller come suo controllore appunto, e dipende dai metodi e dagli id definiti al suo interno.

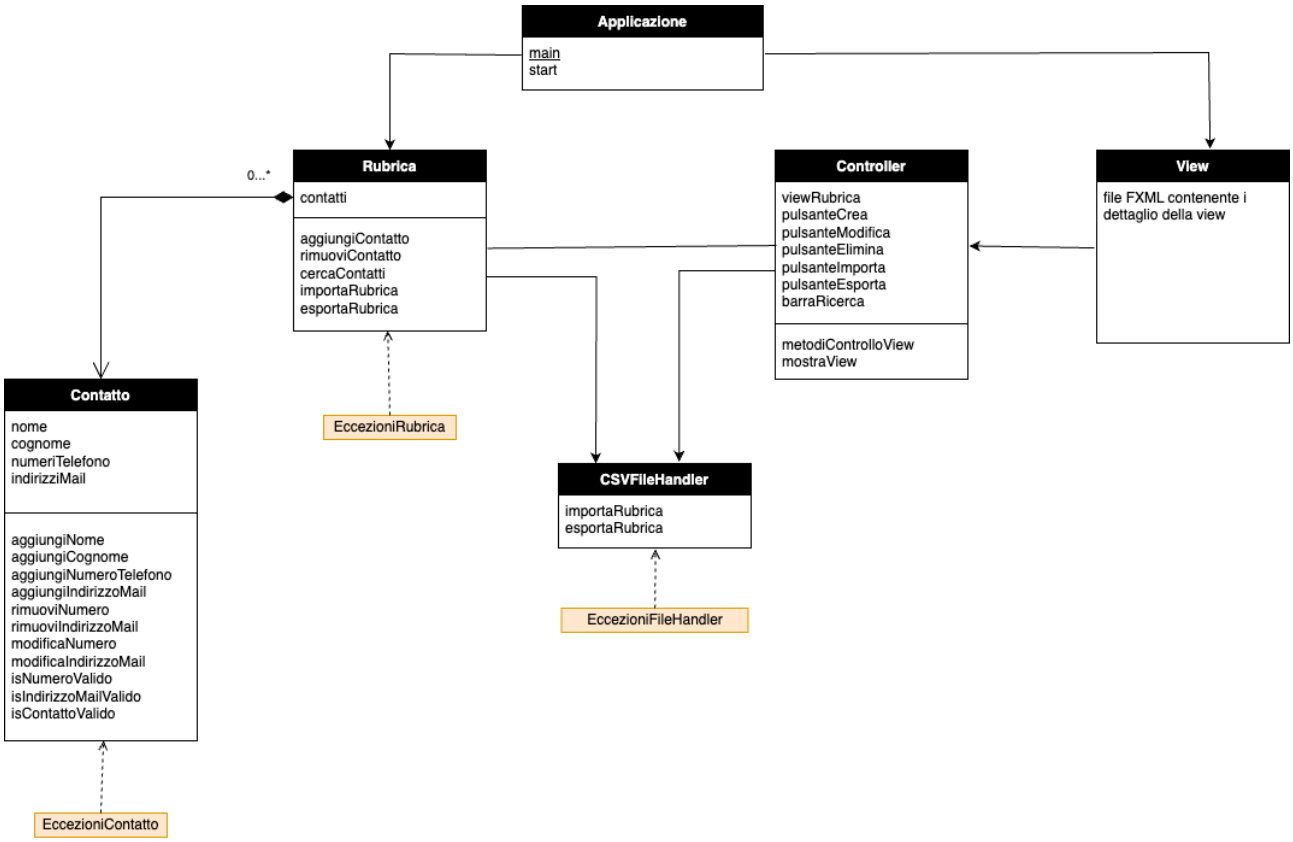
Infine, anche l'accoppiamento tra Rubrica e Contatto è *per dati*, visto che Rubrica invoca solo le operazioni di Contatto necessarie a gestire i contatti presenti in lista.

Alla luce di queste analisi, ci siamo considerati soddisfatti dei livelli di accoppiamento ottenuti, ma abbiamo ritenuto di poter lavorare meglio su quelli che erano i livelli di coesione, in modo da ottenere un maggior numero di coesioni funzionali.

| Livelli di accoppiamento |
|---|
| Per contenuti |
| Per aree comuni |
| Per controllo |
| Per timbro |
| Per dati |
| Nessuno |

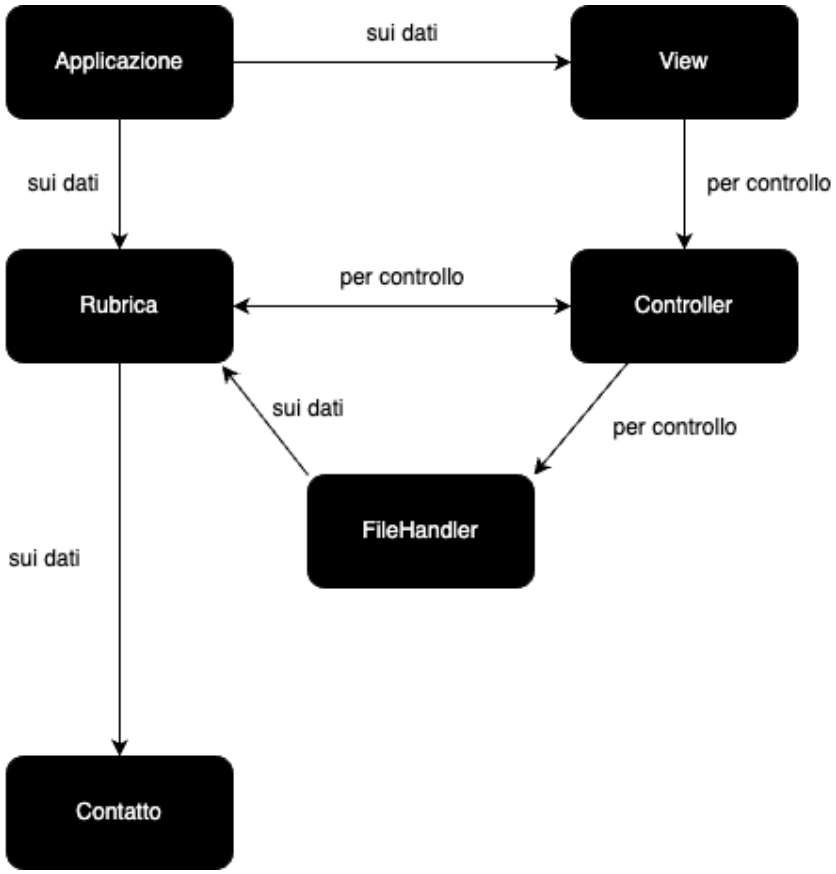
| Livelli di coesione |
|---|
| Funzionale |
| Sequenziale |
| Comunicazionale |
| Procedurale |
| Temporale |
| Logica |
| Coincidentale |

Per realizzare il nostro obiettivo abbiamo quindi deciso di aggiungere una classe *CSVFileHandler* che si occupasse unicamente delle operazioni di importazione/esportazione della lista contatti da/su un file .csv esterno.




Siamo così riusciti a migliorare la coesione di Rubrica, facendola diventare funzionale, così come anche la coesione della nuova classe CSVFileHandler, che per l'appunto racchiude unicamente funzioni responsabili delle operazioni di importazione ed esportazione.

La classe aggiunta CSVFileHandler è connessa sia alla classe Rubrica, con la quale ha un accoppiamento *per dati* (dato che la classe accede ai dati di Rubrica solo per salvarli o aggiornarli, senza modificare la struttura di Rubrica o il suo comportamento), sia alla classe Controller, con la quale ha un accoppiamento *per controllo* (dato che il Controller dirige le operazioni di FileHandler su importazione ed esportazione, senza dipendere dalla sua implementazione)




Successivamente, abbiamo ritenuto di poter creare un'interfaccia *FileHandler* - da far implementare a *CSVFileHandler* – dato che in un futuro potremmo essere interessati a trattare altri tipi di file oltre al CSV, e sarebbe dunque corretto separare la logica di trattamento dei file dalla loro effettiva implementazione.

In secondo luogo, abbiamo ritenuto opportuno effettuare un'ulteriore separazione logica all'interno della classe Contatto tra le operazioni di gestione e quelle di validazione.

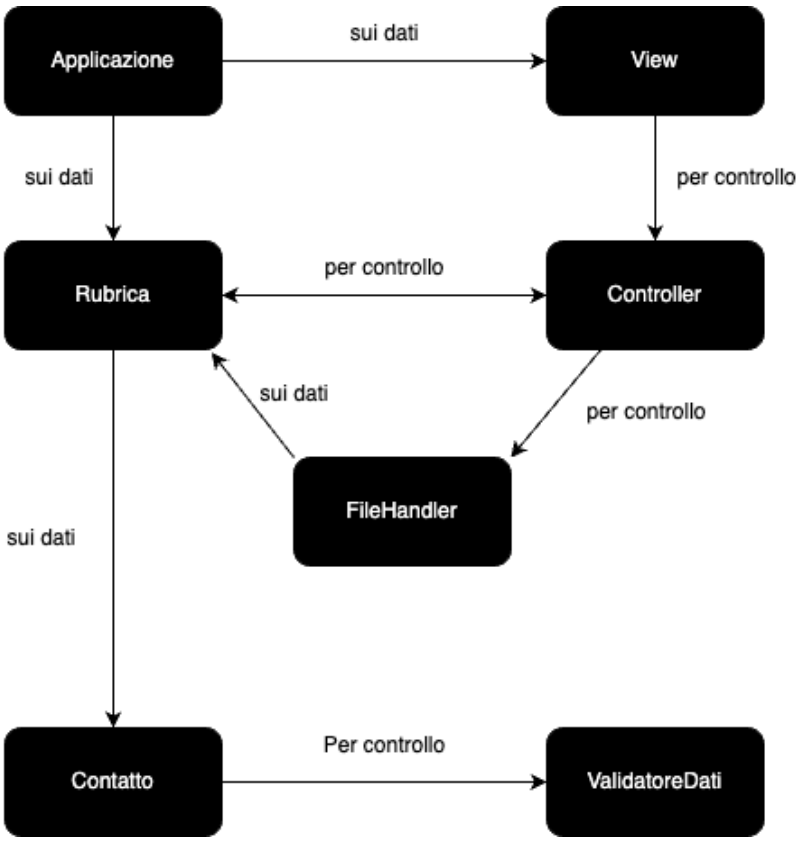
Tale scelta non ha influito tanto sulla coesione della classe, ma favorisce la leggibilità e la manutenzione del codice.

Si è quindi ritenuto di procedere alla creazione di una classe ValidatoreDati, che contenesse al suo interno i metodi per la verifica del formato di mail e numero, nonché il metodo per testare la validità dell'utente.

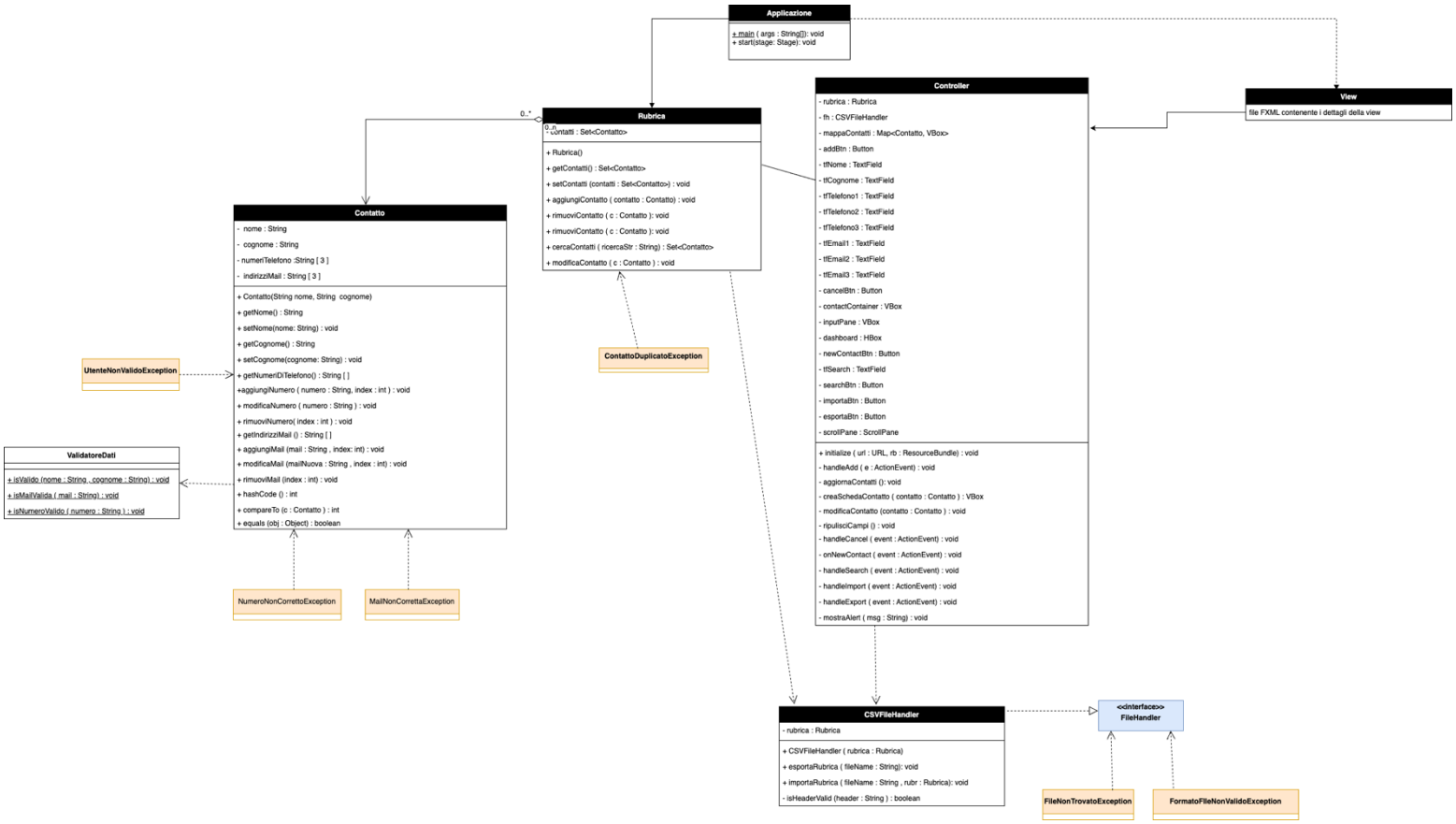
Tali metodi sono stati inseriti come statici, in modo tale da poter essere poi chiamati nella classe Contatto senza necessariamente istanziare un oggetto del

tipo ValidatoreDati (di fatti l'unica funzione della classe è raggruppare dei metodi con un fine comune).

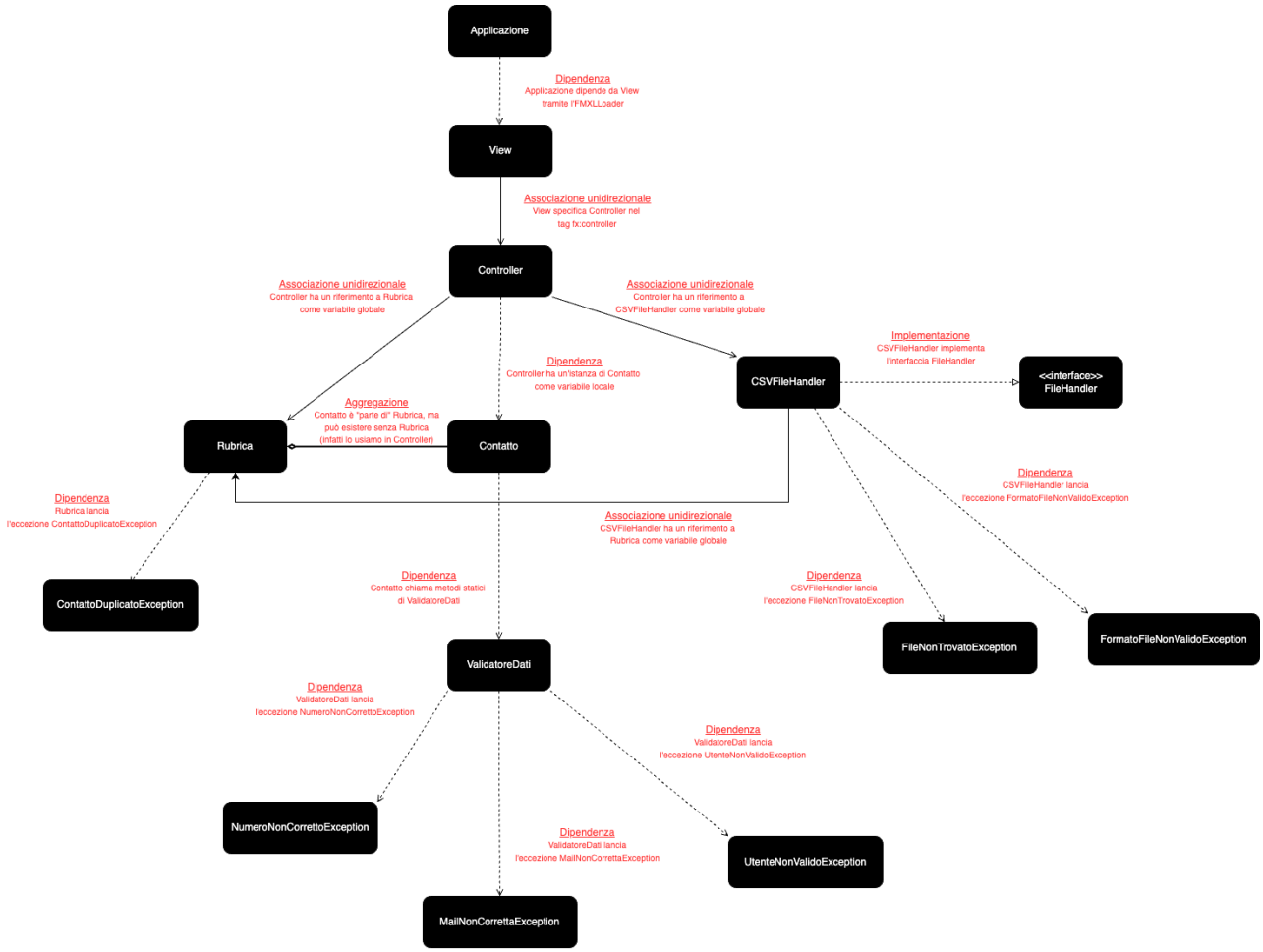
Precisiamo che la coesione della nuova classe è anch'essa funzionale, inoltre - comunicando solamente con la classe Contatto - ha con quest'ultima un accoppiamento *per controllo*, dato che - in base ai risultati riportati dai metodi di controllo della validità - le operazioni interne a contatto potranno o meno essere svolte.



A tal punto ci è stato possibile giungere al nostro effettivo *diagramma delle classi*.

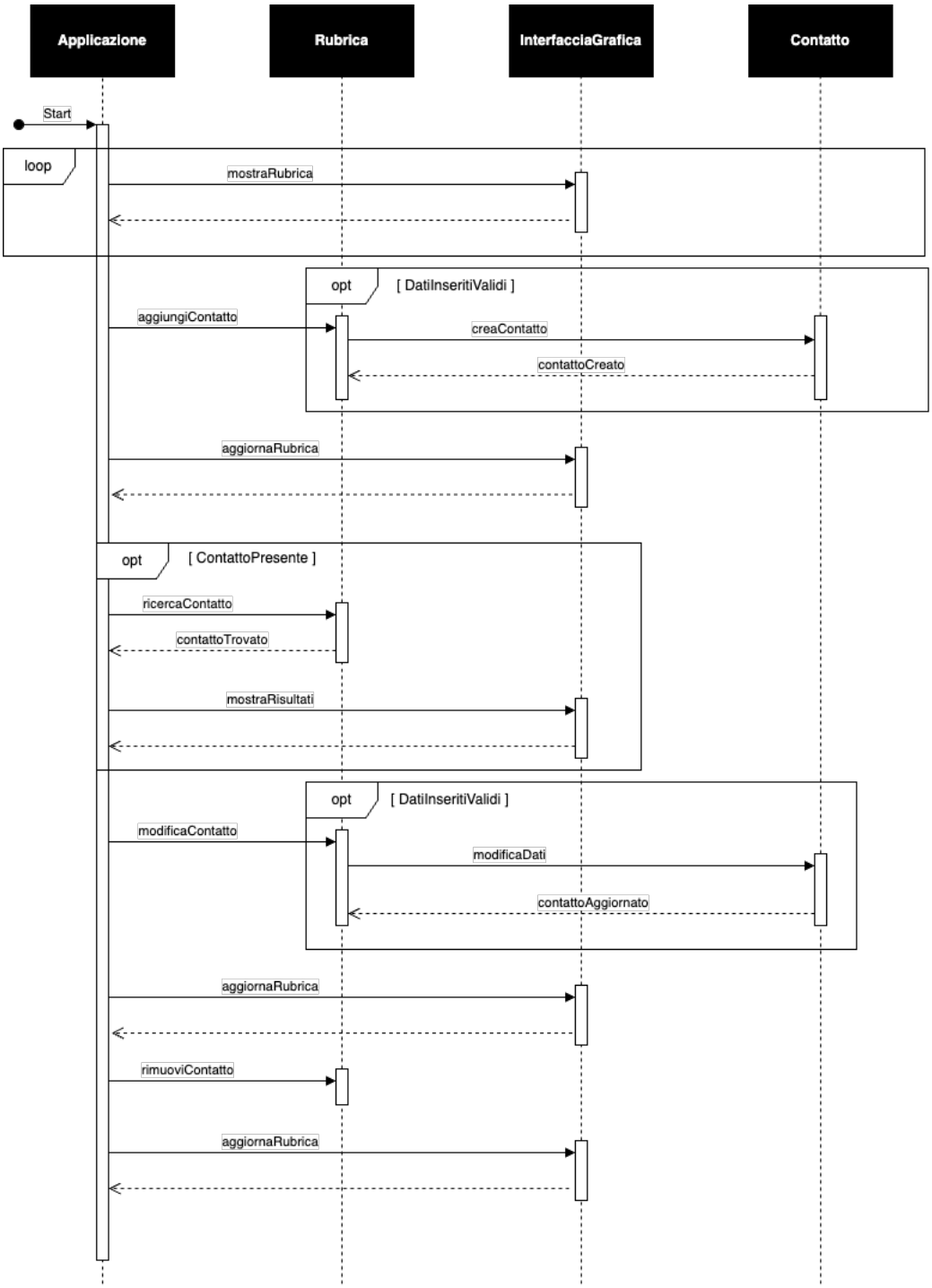


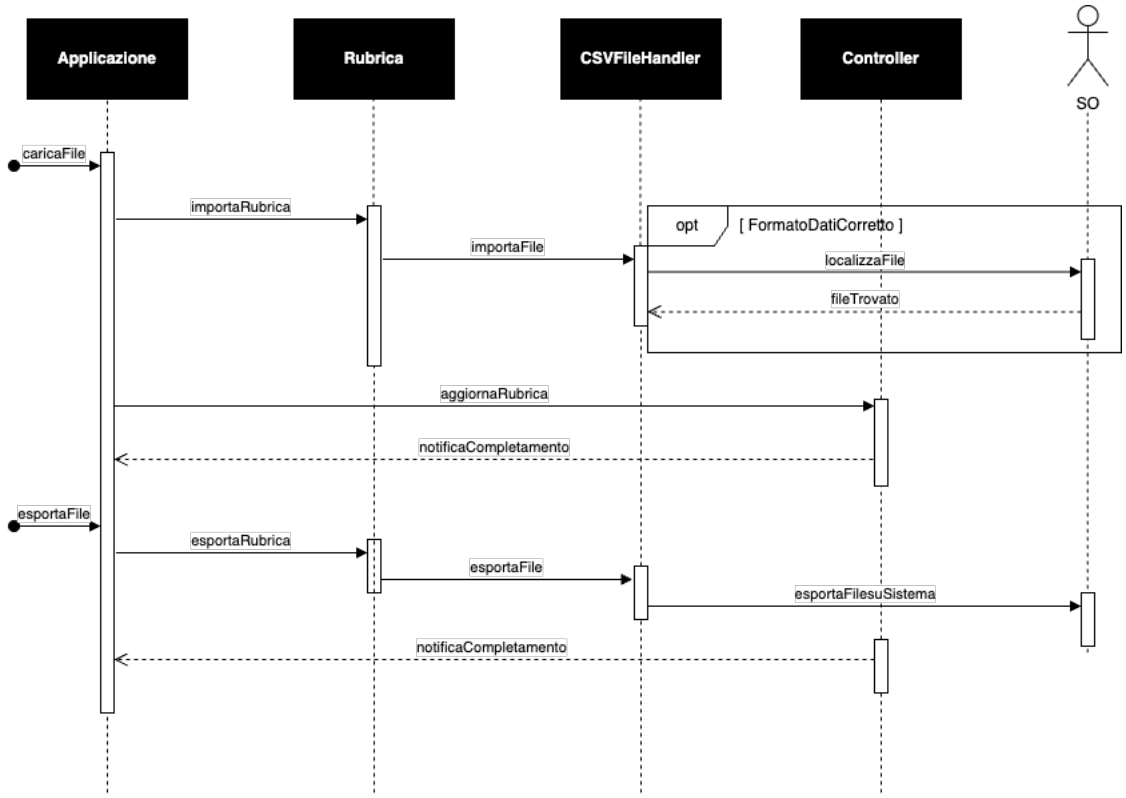
Le relazioni istaurate tra le classi hanno trovato le seguenti giustificazioni.




Alla luce del diagramma ottenuto, è stata chiarita la visione statica del nostro programma, ossia la sua organizzazione in classi e le relazioni tra queste ultime. Restava però da mettere in chiaro anche l'aspetto *dinamico*, chiarendo in che modo le classi comunicassero tra loro.

Per mettere in luce lo scambio di messaggi tra classi durante le varie operazioni di visualizzazione, creazione, modifica, rimozione, ricerca di un contatto e lo scambio di messaggi durante le operazioni di importazione/esportazione si è scelto di costruire due diagrammi di sequenza, ottenendo i risultati seguenti.


Applicazione
Rubrica
InterfacciaGrafica
Contatto
Start
loop
mostraRubrica
opt
[ DatiInseritiValidi ]
aggiungiContatto
creaContatto
contattoCreato
aggiornaRubrica
opt
[ ContattoPresente ]
ricercaContatto
contattoTrovato
mostraRisultati
opt
[ DatiInseritiValidi ]
modificaContatto
modificaDati
contattoAggiornato
aggiornaRubrica
rimuoviContatto
aggiornaRubrica




Ottenuti tali diagrammi si è potuta considerare fondamentalmente conclusa la fase di design del nostro programma, e dunque è stato possibile procedere con l'effettiva implementazione del codice.