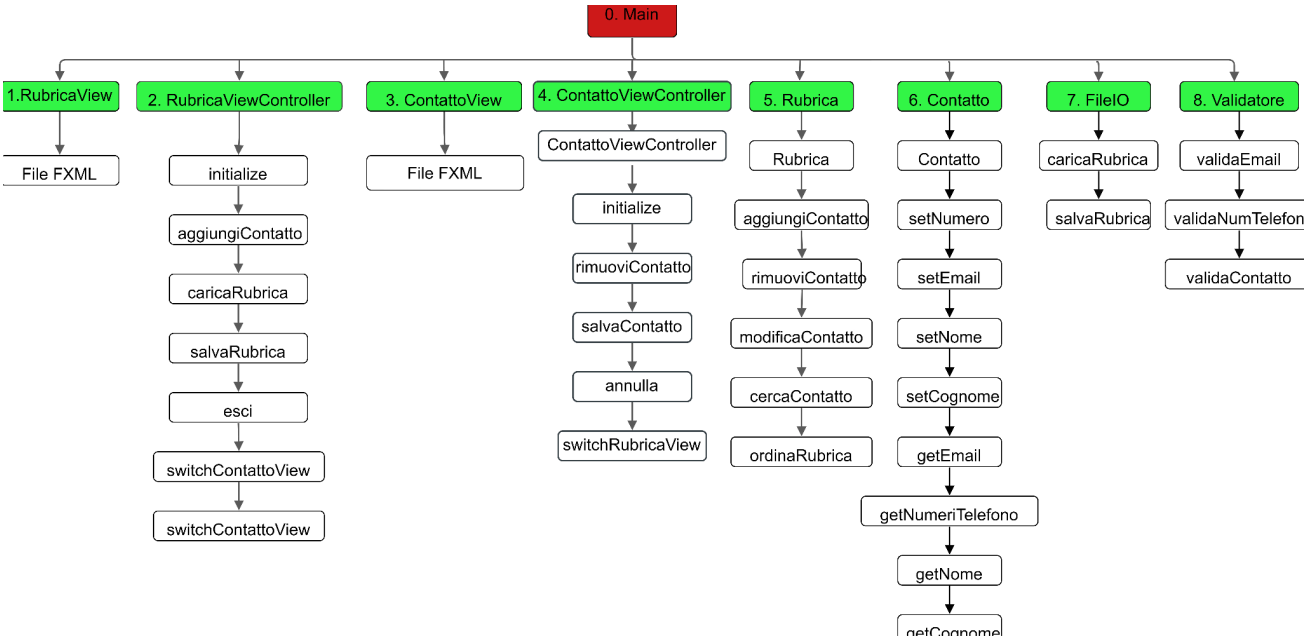
DESIGN

1 Preview

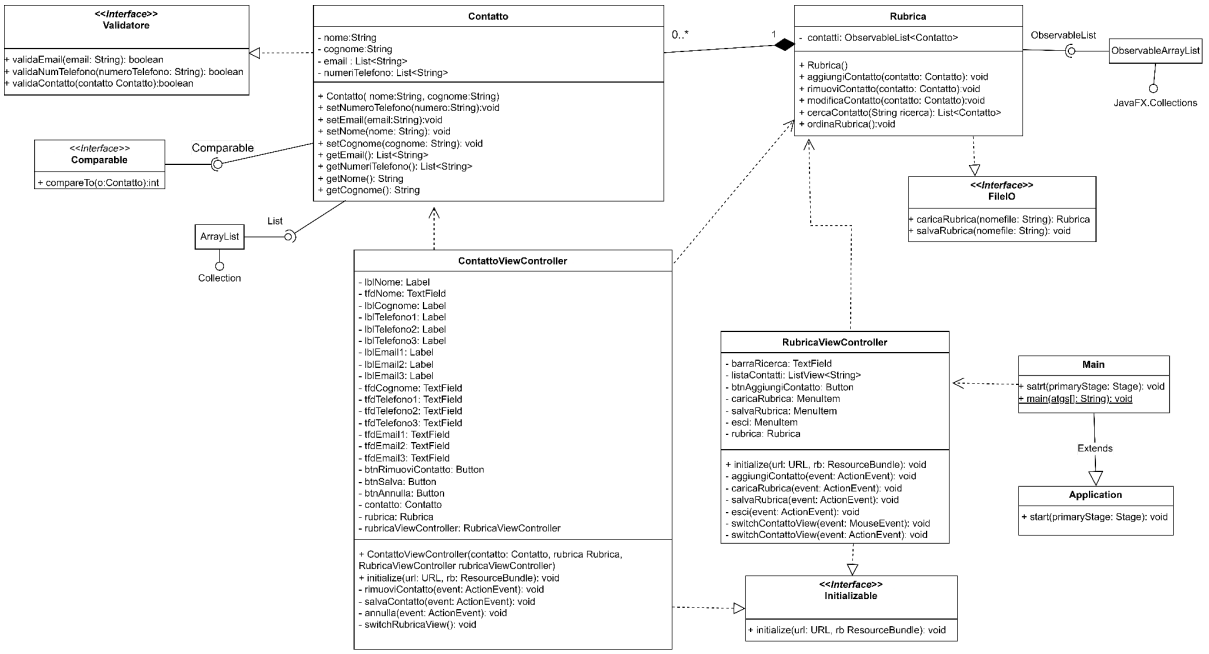
Il documento descrive le scelte progettuali intraprese per la strutturazione della rubrica. Per massimizzare quanto più la comprensione, saranno utilizzati più diagrammi, aggiungendo una sezione di chiarimento per i passaggi meno intuitivi. Il diagramma delle classi sarà utile per comprendere le relazioni tra le classi, evidenziando coesione e accoppiamento. Infine sarà presente un diagramma delle sequenze per presentare il funzionamento effettivo del sistema, attraverso l'interazione tra l'utente ed il sistema in casi d'uso specifici.



2 Diagramma delle classi

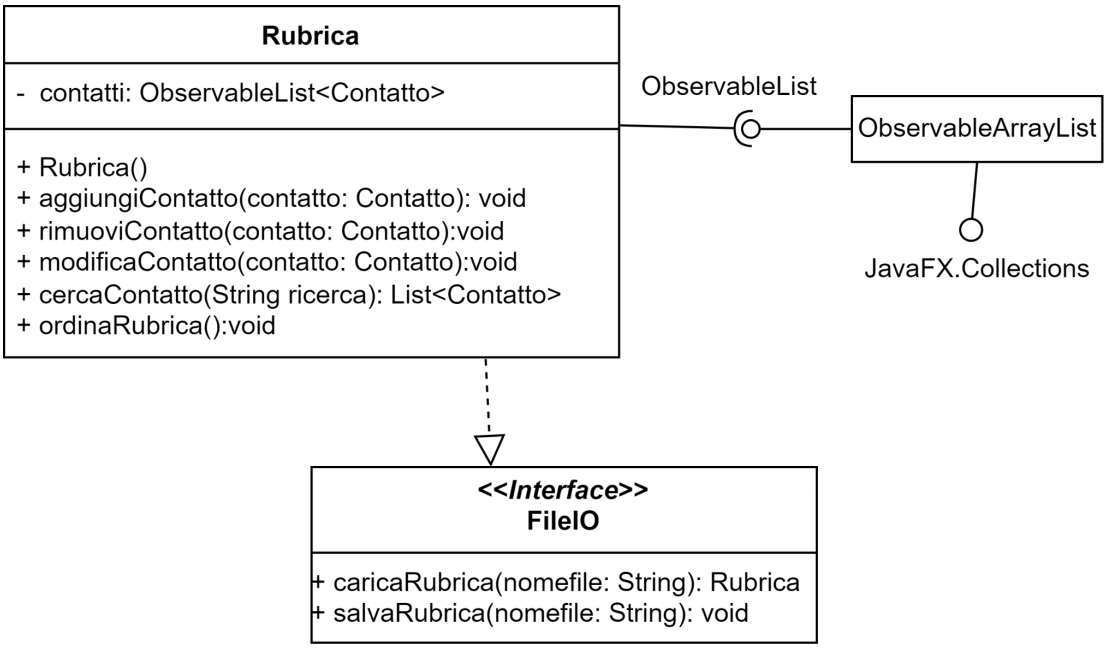
Per facilitare la visione, è stato deciso di frammentare il diagramma totale in sotto-diagrammi descrittivi delle più significative relazioni tra classi. Per maggiori informazioni sulla singola classe, i loro metodi e attributi, si rimanda il lettore al seguente link per visionare la documentazione doxygen del progetto.

2.1 Diagramma complessivo



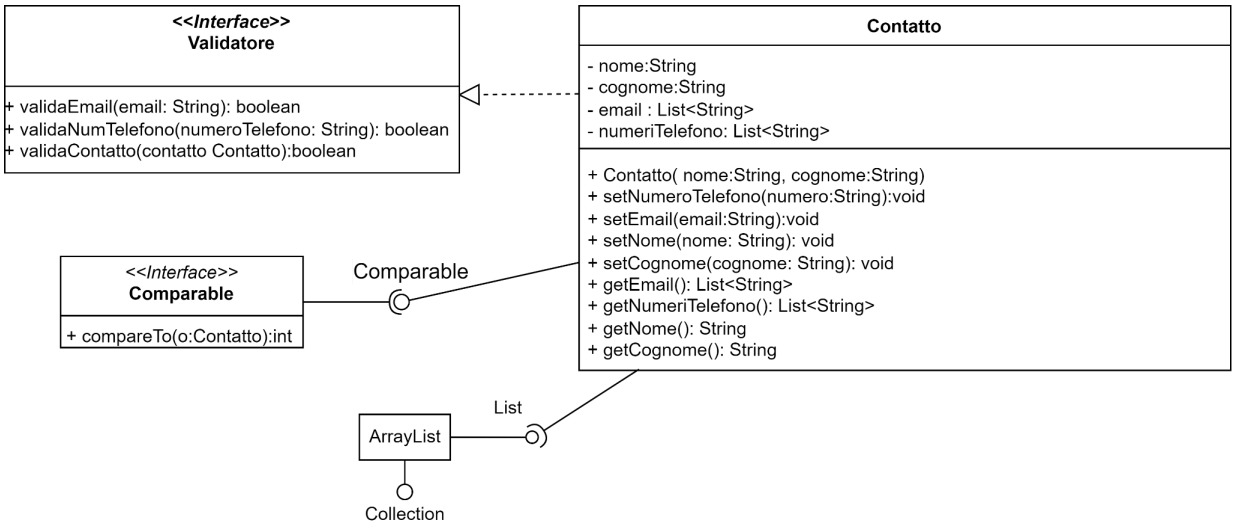
2.2 Diagramma della Rubrica

Il seguente diagramma mostra il funzionamento della classe Rubrica e di come questa implementi l'interfaccia funzionale FileIO, per salvare la rubrica su file o caricarla da questo.

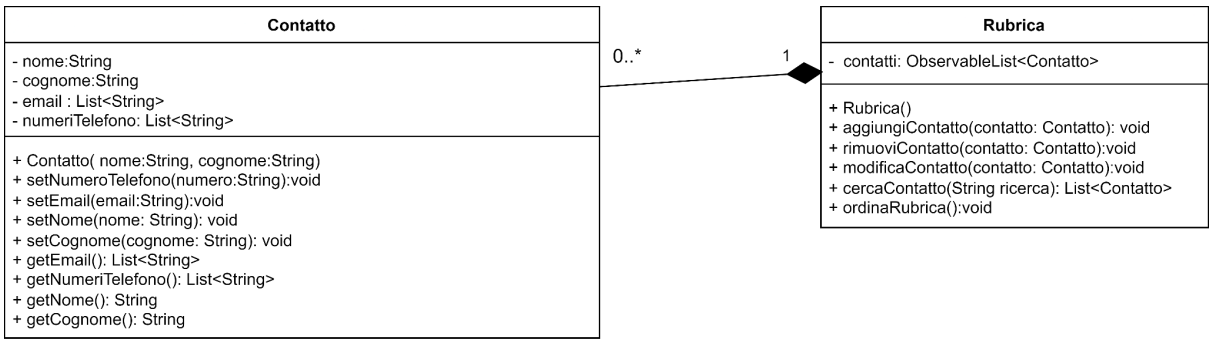


2.3 Diagramma del Contatto

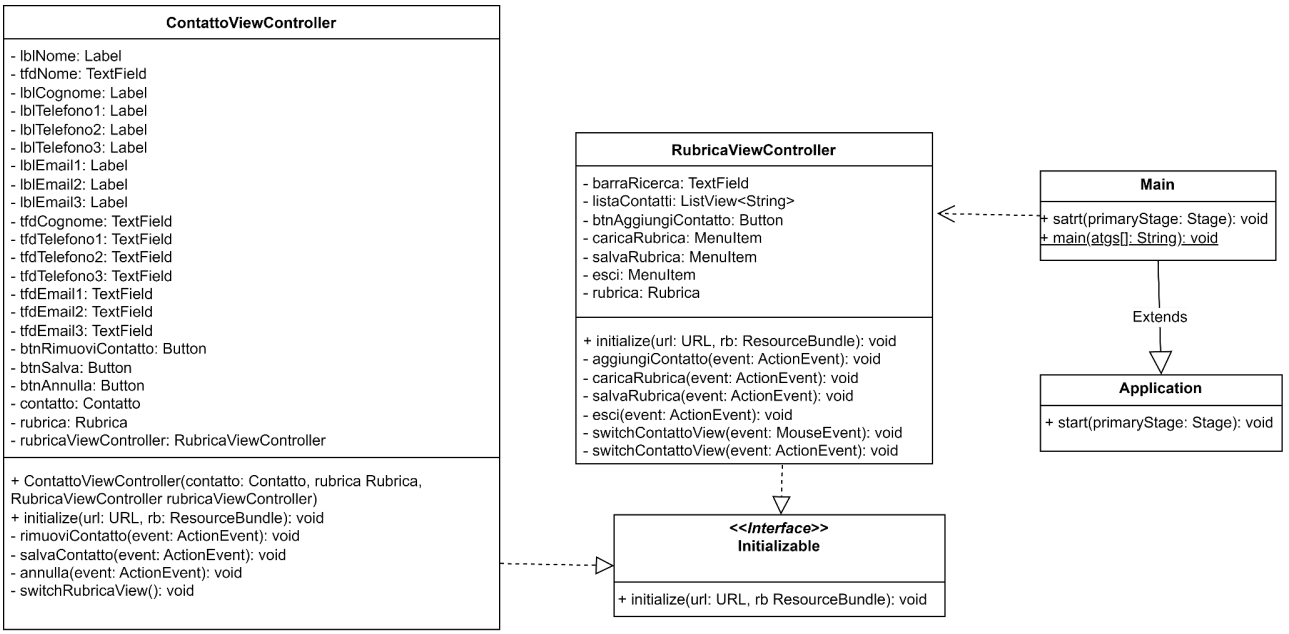
Il seguente diagramma mostra il funzionamento della classe Contatto, che implementa le interfacce List, Comparable e Validatore.



2.4 Diagramma Rubrica-Contatto

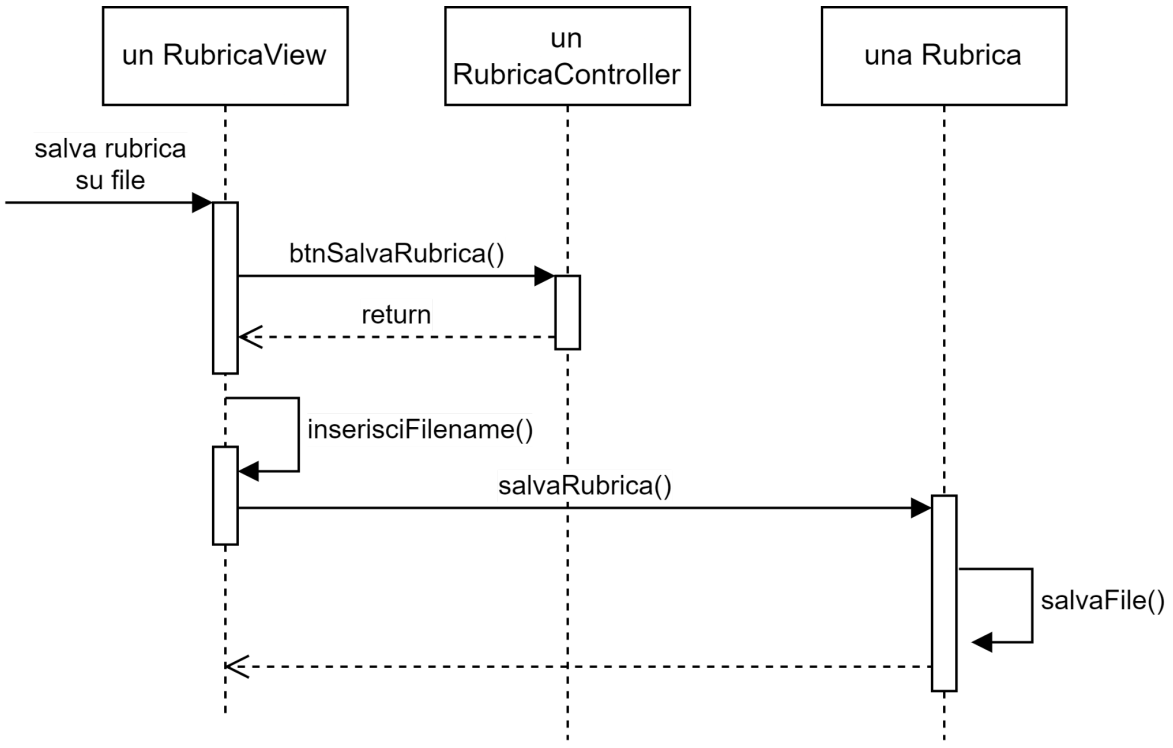


2.5 Diagramma Interfaccia Grafica e Main

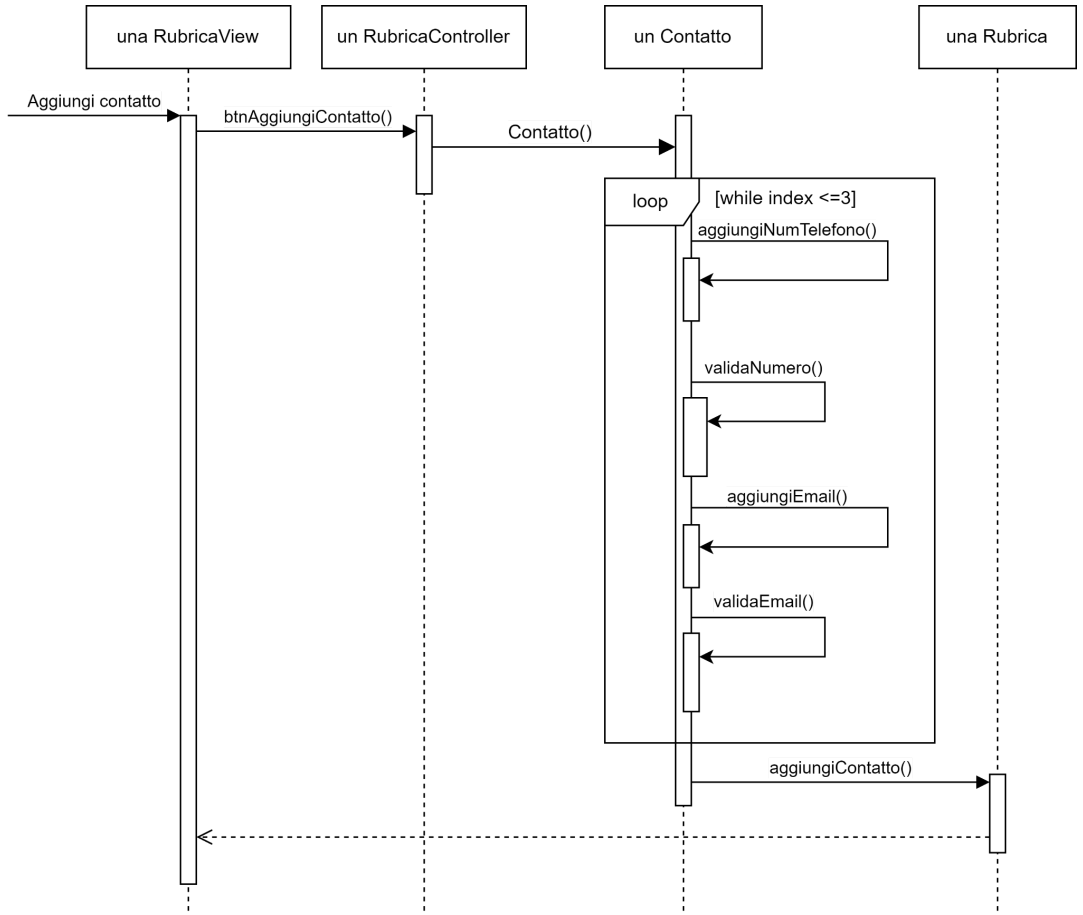


3. Diagrammi delle sequenze

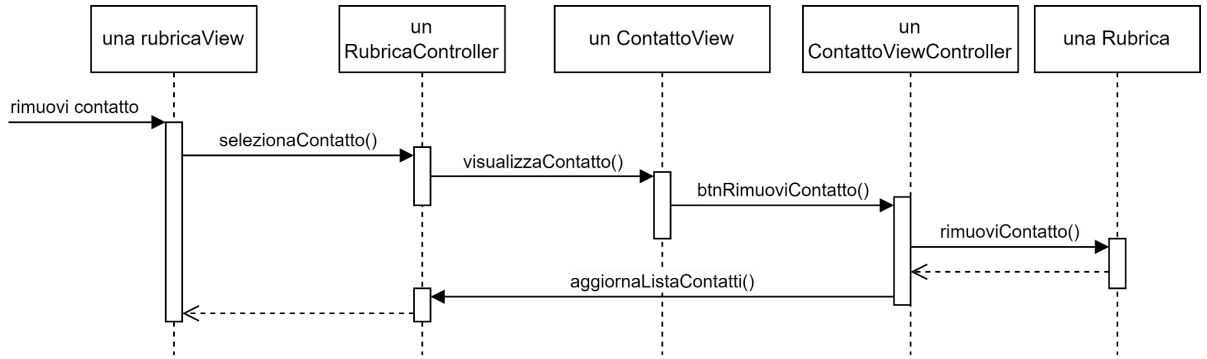
3.1 Salva rubrica su file



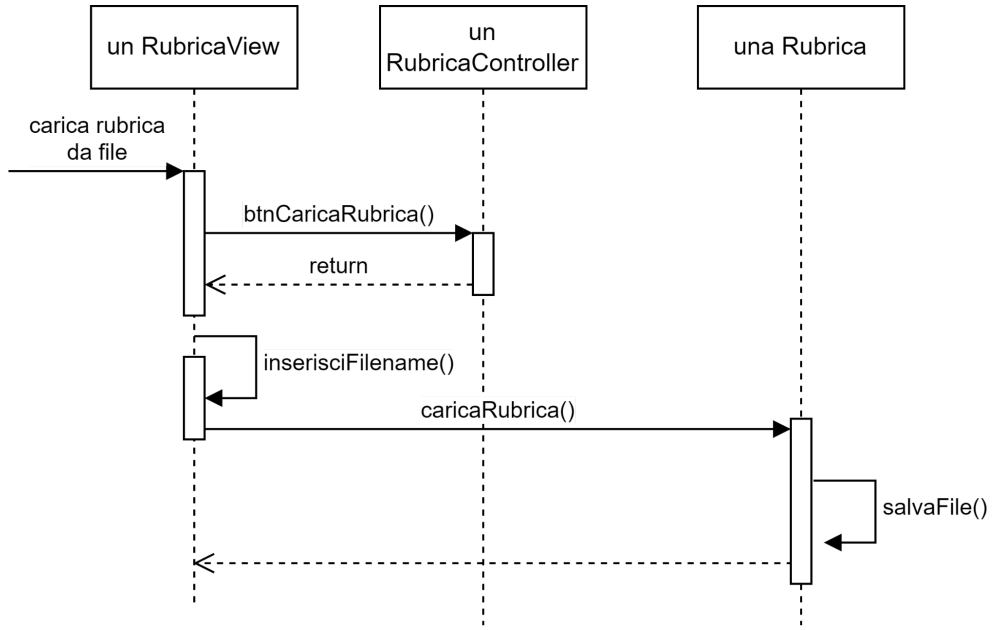
3.2 Aggiungi contatto



3.3 Rimuovi contatto



3.4 Carica rubrica da file



4. Considerazioni su coesione, accoppiamento e princìpi di buona progettazione

4.1 Coesione

La classe Contatto si occupa di rappresentare un contatto, includendo un nome, un cognome, più numeri di telefono ed email. Ogni metodo contribuisce direttamente a tale mansione, perciò la coesione è funzionale.

L'interfaccia Validatore svolge il compito di fornire metodi per verificare la correttezza sintattica dei dati presenti all'interno di un contatto. Attraverso i metodi validaEmail, validaNumero e validaContatto, l'interfaccia si presenta coesa, svolgendo operazioni su dati strettamente correlati tra loro. Questo fa sì che la coesione sia funzionale.

Rubrica è la classe che si occupa della gestione di una lista di contatti. Le operazioni di aggiungiContatto, rimuoviContatto, modificaContatto e cercaContatto hanno una stretta correlazione, in quanto deputate alla gestione di tale lista. Quindi, il livello di coesione è funzionale. La scelta di utilizzare una lista osservabile (ObservableList) è giustificata dalla necessità di rappresentare graficamente la rubrica, favorendo la semplicità d'uso.

FileIO gestisce il caricamento ed il salvataggio dei dati della rubrica su/da file, mantenendo una singola responsabilità focalizzata sulle operazioni di I/O. Ragion per cui, il grado di coesione di tale interfaccia è funzionale.

Per l'interfaccia grafica è stato deciso di suddividere la visualizzazione per **Rubrica** e **Contatto**. Inoltre, avendo utilizzato SceneBuilder (che fornisce in output un file FXML come classe Viewer) per curare la veste grafica, la coesione di tali interfacce ha uno stretto legame con le operazioni svolte da ognuno dei suoi blocchi. Risulta, dunque, una coesione **procedurale**.

Analogamente, per le classi **RubricaViewController** e **ContattoViewController,** la coesione risulta **comunicazionale**. Lavorando su dati definiti rispettivamente da **Rubrica** e **Contatto**, tali classi lavorano in simbiosi con le rispettive interfacce grafiche (descritte dai rispettivi file FXML). Ciò riduce notevolmente la coesione, ma aumenta la semplicità d'uso.

4.2 Accoppiamento

Nel diagramma delle classi è possibile notare un accoppiamento **per dati** nei seguenti casi:

- La classe **Rubrica** necessita di informazioni della classe **Contatto**, tuttavia questa non accede direttamente alle strutture dati, né modifica il loro contenuto. Lo scambio di informazioni è ridotto a quelle strettamente necessarie (oggetto Contatto) a Rubrica per il suo corretto funzionamento;
- Per i motivi sopraelencati è possibile confermare lo stesso tipo di accoppiamento per le seguenti coppie di classi (classe che usa i dati -> classe che fornisce i dati)
    - **ContattoViewController -> Contatto**
    - **ContattoViewController -> Rubrica**
    - **RubricaViewController -> Rubrica**
    - **Main -> RubricaViewController**
    - **Validatore -> Contatto**
    - **FileIO -> Rubrica**

4.3 Princìpi di buona progettazione

I principi di progettazione utilizzati sono stati il **KISS** (Keep It Simple, Stupid) e **SINE** (Simple Is Not Easy). Si è scelto di adoperare un principio KISS per elevare al massimo la semplicità d'uso, evitando di incorporare funzionalità considerate superflue (You Aren't Going to Need It, **YAGNI**) o che avrebbero ecceduto le abilità in ambito di programmazione di ogni membro del gruppo (**Ortogonalità**). Infine, sono stati adoperati il principio di **programmazione ad oggetti** per lavorare con la massima astrazione possibile ed il principio **SOLID** (Single Responsibility Principle, Open-Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, Dependency Inversion Principle) nella scelta dell'interfaccia delle classi proposte.

5. Matrice di tracciabilità

Nome requisito	ID requisito	Design	Codice	Test	Requisiti correlati
Dati contatto	1	Modulo 6			2,3,4,5,6,8
Creazione contatto	2	Modulo 3 Modulo 4 Modulo 5 Modulo 6 Modulo 8			1,8
Modifica contatto	3	Modulo 3 Modulo 4 Modulo 5 Modulo 6 Modulo 8			1,8
Eliminazione contatto	4	Modulo 3 Modulo 4 Modulo 5 Modulo 6			8
Salvataggio rubrica su file	5	Modulo 1 Modulo 2 Modulo 5 Modulo 7			1
Carica rubrica da file	6	Modulo 1 Modulo 2 Modulo 5 Modulo 7			1
Cerca contatto	7	Modulo 1 Modulo 2 Modulo 5 Modulo 6			8,9
Interfaccia utente	8	Modulo 1 Modulo 2 Modulo 3 Modulo 4			1,2,3,4,5,6,7,9,10
Ordine rubrica	9	Modulo 1 Modulo 2 Modulo 5			7,8
Semplicità d’uso	10	Modulo 1 Modulo 3			8