Università degli studi di Salerno

Dipartimento dell'Ingegneria dell'Informazione ed Elettronica e Matematica applicata

Corso di Laurea in *Ingegneria Informatica* / L-8

Documentazione Design

Moscariello Davide
Quaranta Davide
Ronca Ciro
Sessa Domenico

Docente: *Capuano Nicola*
A.A. 2024-2025

Approccio di Scomposizione

La decisione sulla scomposizione del Sistema, è stata fatta tramite l'approccio di Scomposizione Object-Oriented, esso ci permette di poter suddividere il problema da dover risolvere in diversi componenti (classi/oggetti) che rappresentano singolarmente una parte del dominio del problema.

Diagramma delle Classi

Tramite il Diagramma delle Classi mostriamo, come nell'effettivo abbiamo scomposto il Sistema, con le annesse informazioni di Dipendenza tra i singoli componenti e le Interfacce fornite da ognuno di essi.

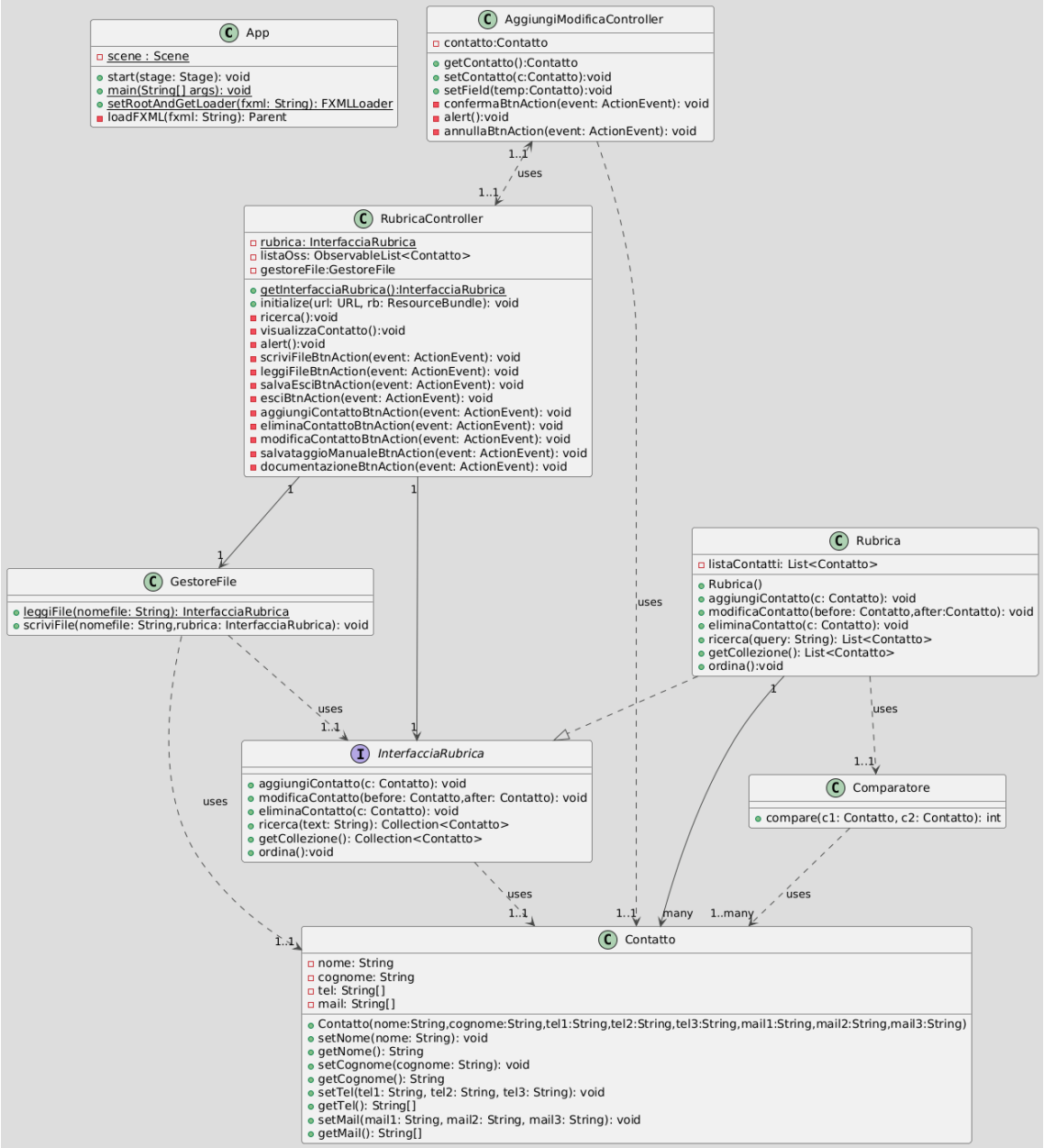

Figure 1: Diagramma delle classi **AddressBook**

Mockup AddressBook

Scena principale della Rubrica

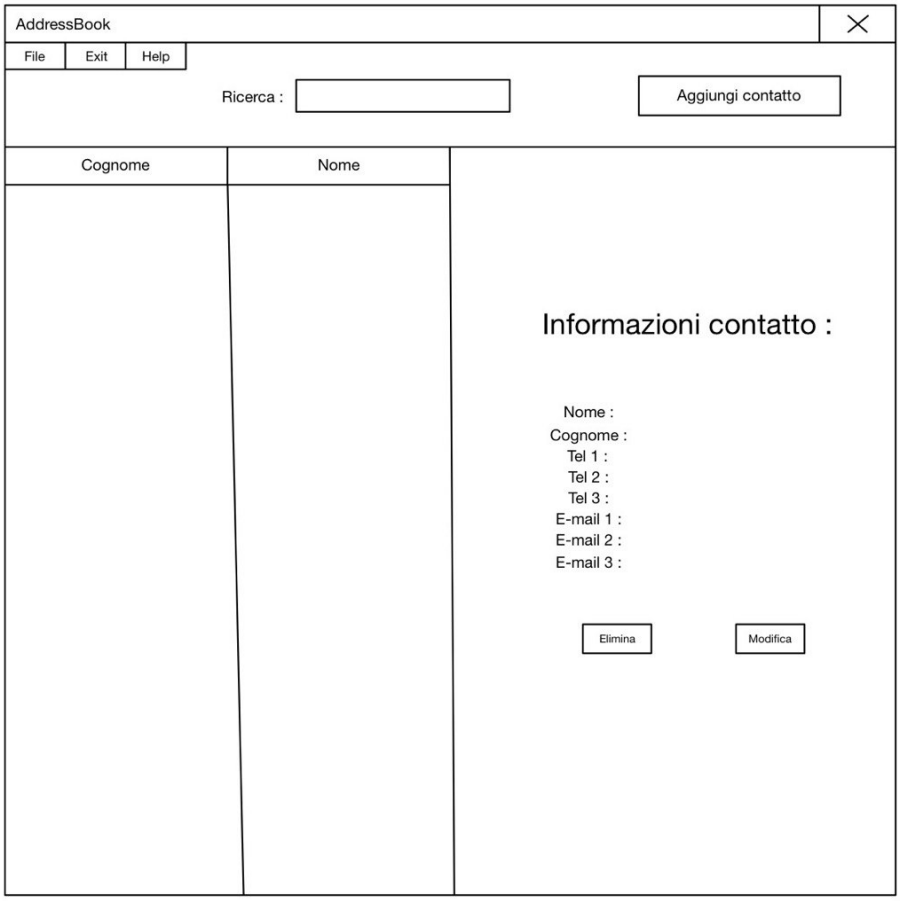

Figure 2: Mockup visualizzazione rubrica **AddressBook**

Scena di Aggiunta/Modifica Contatto

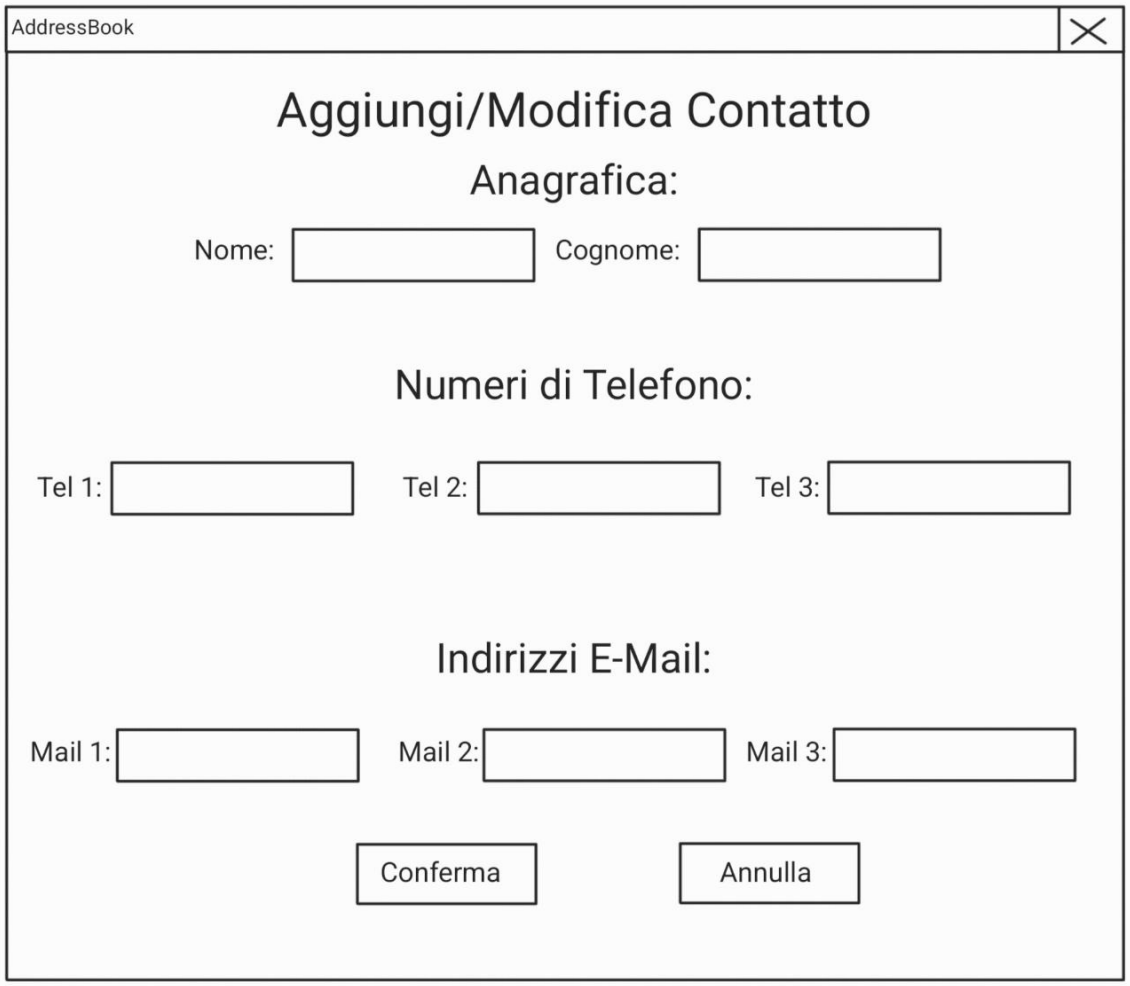

Figure 3: Mockup Aggiunta/Modifica Contatto **AddressBook**

Valutazione Scomposizione

Coesione

Classe	Livello di Coesione	Descrizione
AggiungiModificaController	Coesione Funzionale	Cattura le azioni dell'utente sulla vista, di conferma o annullamento delle operazioni di aggiunta o modifica di un contatto.
Comparatore	Coesione Funzionale	La classe Comparatore permette di confrontare due contatti in modo tale che si possano ordinare su cognome e nome.
Contatto	Coesione Funzionale	Tale classe gestisce tutte le informazioni di un contatto, in modo tale che si possano leggere e modificare singolarmente.
RubricaController	Coesione Funzionale	Cattura le azioni dell'utente sulla vista, in modo tale che sia possibile effettuare le operazioni di ricerca e eliminazione di un contatto e le attività di salvataggio, import/export e inizializzazione della Rubrica che vengono in parte affidate ad altre classi, così come le operazioni di aggiunta e modifica di un contatto.
GestoreFile	Coesione Funzionale	Gestisce le operazioni di lettura e scrittura di un file.
InterfacciaRubrica	Coesione Funzionale	Definisce le operazioni di: Aggiunta, Modifica, Eliminazione, Ordinamento e Ricerca dei contatti e l'operazione di restituzione di una collezione di contatti.
Main	Coesione Funzionale	Il suo compito è quello di richiamare e visualizzare la scena da un File FXML.
Rubrica	Coesione Funzionale	Gestisce una lista di contatti tramite l'implementazione delle diverse operazioni definite nell'InterfacciaRubrica.

Classi e Livello di Accoppiamento

Classe	**Livello di Coesione**	**Descrizione**
Contatto e Comparatore	Accoppiamento per Dati	La classe Comparatore utilizza gli oggetti della classe Contatto per confrontarli.
Contatto e InterfacciaRubrica	Accoppiamento per Dati	L'InterfacciaRubrica utilizza, come parametri dei metodi, degli oggetti della classe Contatto.
Contatto e Rubrica	Accoppiamento per Dati	La classe Rubrica richiama i soli metodi pubblici della classe contatto.
RubricaController e AggiuntaModificaController	Accoppiamento per Controllo	I Controller si scambiano un oggetto Contatto che permette di effettuare le operazioni di aggiunta e modifica di un contatto nella Rubrica. Inoltre AggiuntaModificaController utilizza attraverso i metodi pubblici una variabile privata statica di RubricaController.
GestoreFile e RubricaController	Accoppiamento per Dati	RubricaController istanzia un oggetto di GestoreFile, sul quale richiama i soli metodi pubblici.
InterfacciaRubrica e RubricaController	Accoppiamento per Dati	RubricaController istanzia un oggetto InterfacciaRubrica sul quale richiama i soli metodi pubblici, grazie all'implementazione in Rubrica.
InterfacciaRubrica e AggiuntaModificaController	Accoppiamento per Dati	AggiuntaModificaController utilizza i metodi pubblici di InterfacciaRubrica, implementati in Rubrica, per aggiungere e modificare un Contatto.
GestoreFile e Contatto	Accoppiamento per Dati	Il GestoreFile istanzia oggetti Contatto e usa i suoi metodi pubblici.
InterfacciaRubrica e GestoreFile	Accoppiamento per Dati	Il GestoreFile usa InterfacciaRubrica per poter effettuare le sue operazioni di lettura e scrittura da file.
Rubrica e Comparatore	Accoppiamento per Dati	Viene istanziato un oggetto Comparatore sul quale viene richiamato il suo unico metodo pubblico.
GestoreFile e Contatto	Accoppiamento per Dati	Viene istanziato un oggetto della classe Contatto e vengono richiamati i metodi pubblici.

Nota:

- La classe Main non presenta alcun accoppiamento con le altre classi.
- Le relazioni presenti tra InterfacciaRubrica e le altre classi sono garantite da InterfacciaRubrica, grazie all'UP-CAST effettuato dalla classe Rubrica.

Requisiti Di Buona Progettazione:

Con l'implementazione presentata, possiamo affermare di aver seguito diversi principi di buona progettazione e diversi attributi di qualità (QA). In particolare, abbiamo applicato i principi del **S.O.L.I.D.**. Come **il Principio della Singola Responsabilità** (Separation of Concerns). Questo è stato possibile grazie alla suddivisione delle diverse tipologie di problemi in più classi, un approccio che riesce anche a garantire due attributi di qualità interna, ovvero la **manutenibilità** e la **riusabilità** del sistema. Abbiamo inoltre rispettato il Principio Aperto/Chiuso, implementando l'interfacciaRubrica e usando l'incapsulamento. Questo perché ci consente di definire diverse implementazioni senza modificare il sistema esistente, e anche un limite alle modifiche del sistema. In questo modo, il sistema rimane chiuso alle modifiche, ma aperto all'estensione. **Il Principio di Sostituzione di Liskov** è stato rispettato poiché l'oggetto rubrica, presente all'interno di RubricaController può essere sostituito con diverse implementazioni dell'InterfacciaRubrica, perché viene usato un UP-Cast, e questo permette di mantenere un corretto funzionamento del sistema. Inoltre l'implementazione dell'interfacciaRubrica contribuisce anche all'introduzione del **QA Interno** per la **Modularità** del sistema. Poiché grazie ad essa possiamo ridefinire nuove Rubriche con modifiche o aggiunte di funzionalità senza avere alcun problema sul funzionamento complessivo del sistema. Oltre al S.O.L.I.D. sono rispettati anche altri Principi di buona progettazione come il **Dry - Don't Repeat Yourself** grazie alla definizione della classe Comparatore e al metodo ordina nell'InterfacciaRubrica, che ci permettono di evitare di ripetere in diversi punti del codice l'implementazione di una classe anonima per andare a comparare i contatti. **Il Principio della Minima Sorpresa** grazie all'utilizzo delle convenzioni CamelCase (per metodi e attributi) e Upper CamelCase (per classi e interfacce) e all'utilizzo di nomi per classi,metodi e attributi abbastanza esplicativi. Infine abbiamo favorito l'utilizzo delle **relazioni di associazione rispetto alla Ereditarietà** tra le classi.

Diagramma delle interazioni

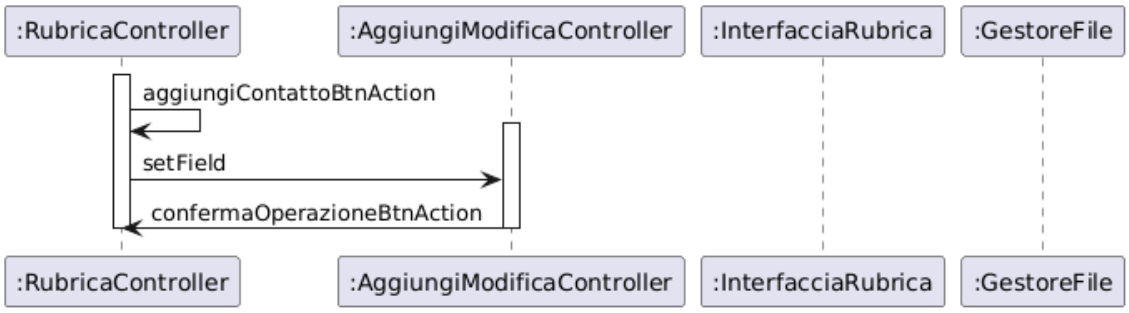

Figure 4: Operazione Aggiungi Contatto

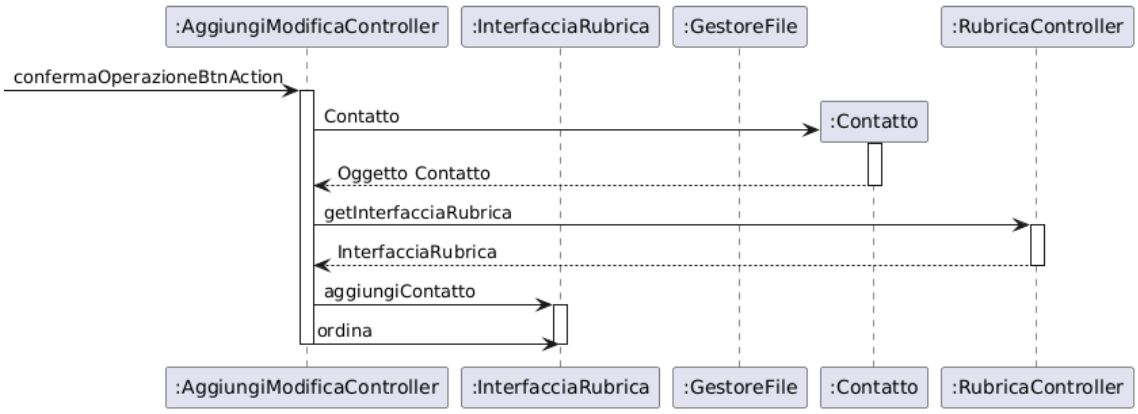

Figure 5: Dettaglio Operazione di Aggiunta

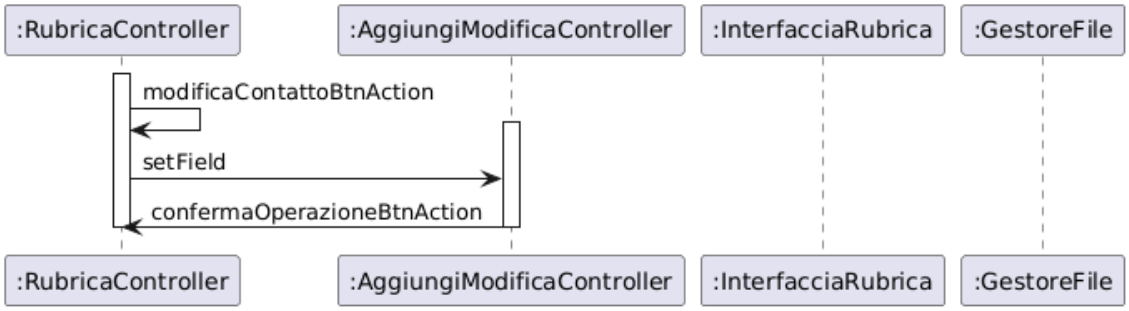

Figure 6: Operazione Modifica Contatto

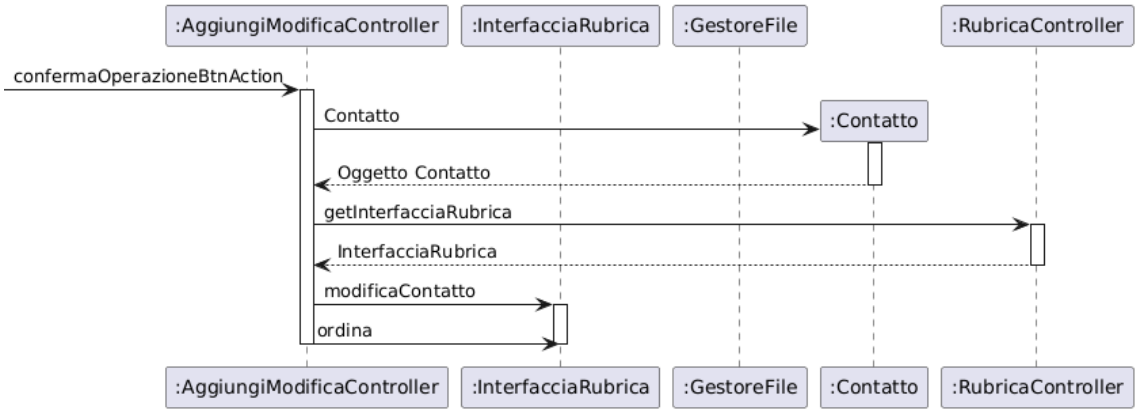

Figure 7: Dettaglio Operazione di Modifica

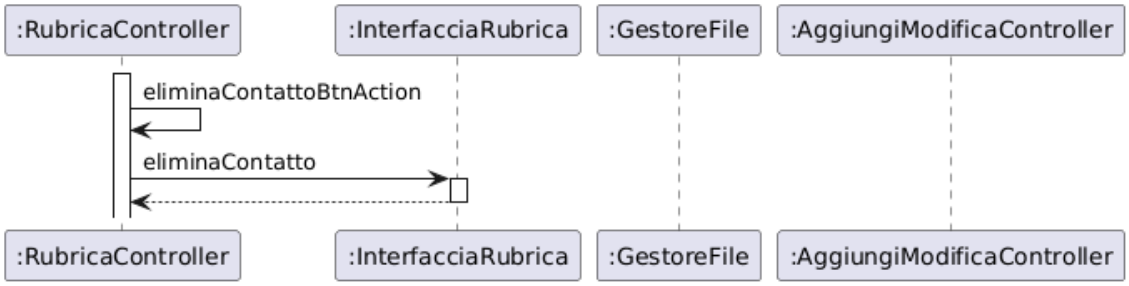

Figure 8: Elimina Contatto

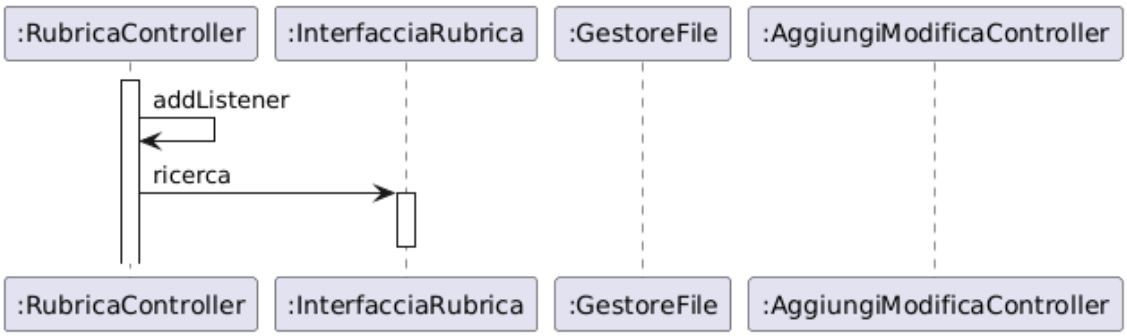

Figure 9: Ricerca Contatto

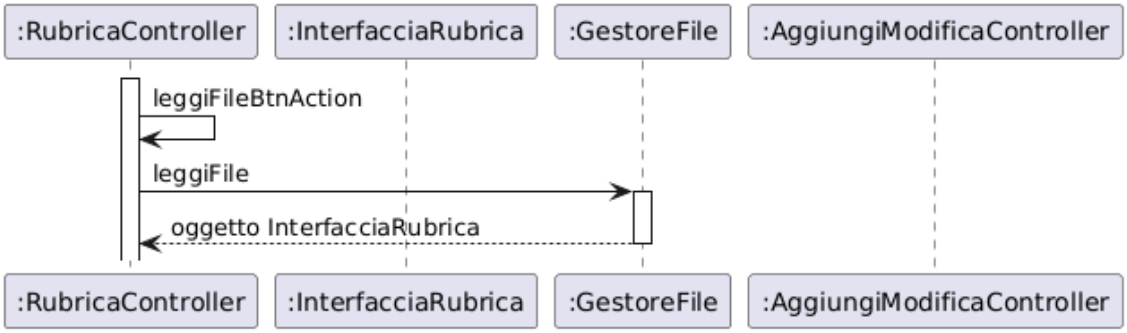

Figure 10: LeggiFile

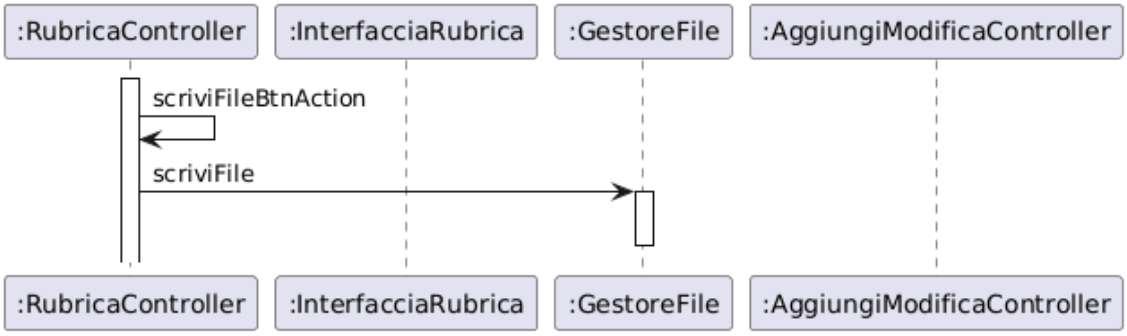

Figure 11: ScriviFile

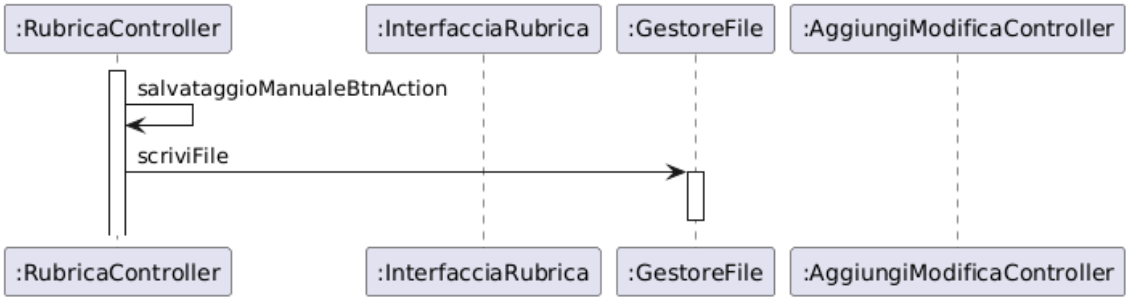

Figure 12: Salvataggio Manuale

Diagramma Dei Package

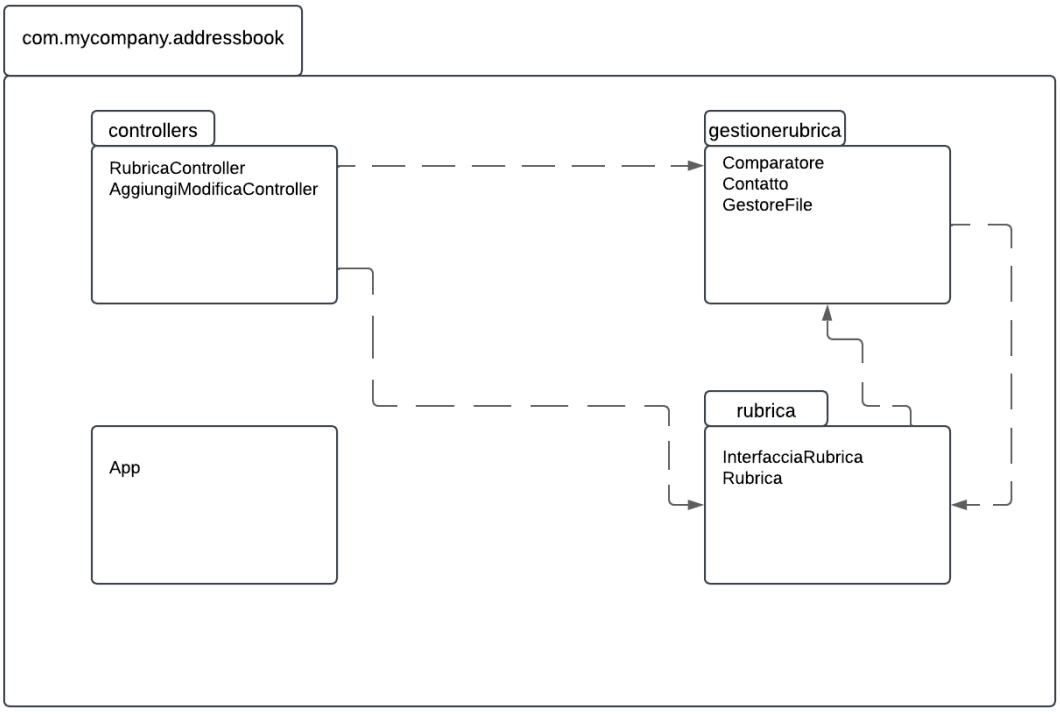

Figure 13: Diagramma Package **AddressBook**