

DESIGN

PROJECT REPORT

Gruppo 6

Avallone Emanuele
Barbato Emanuele
Barberio Gregorio

1 Diagramma delle classi

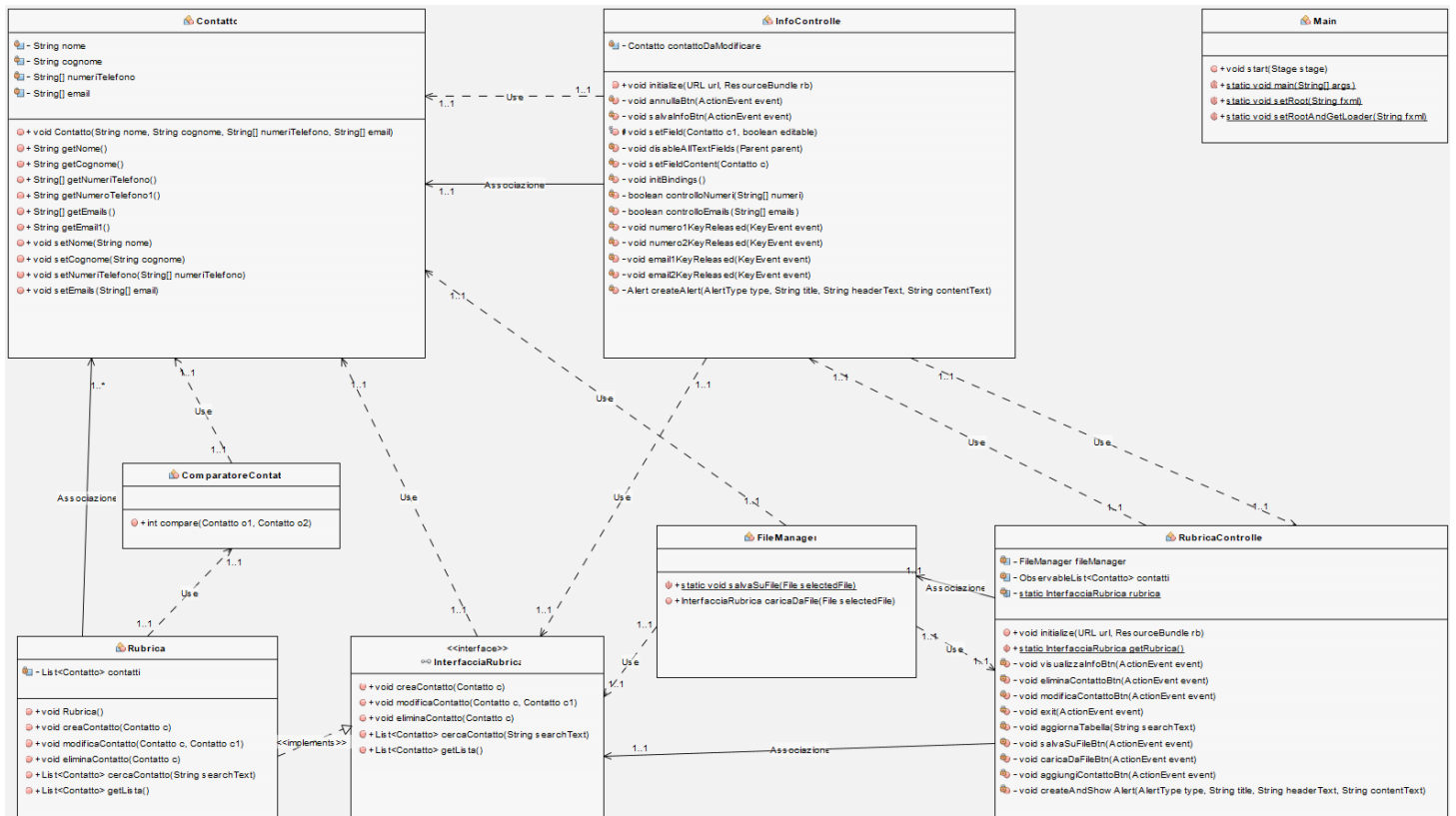


Figure 1: Diagramma delle classi

1.1 Chiarimenti sul diagramma delle classi

Per la progettazione è stato seguito un approccio alla scomposizione di tipo ObjectOriented. Il sistema è stato diviso in classi, ciascuna delle quali rappresenta un'entità del dominio del problema.

Le scelte progettuali sono state effettuate in accordo ai principi di buona progettazione orientata agli oggetti (**S.O.L.I.D.**) ed in modo da garantire al sistema un certo numero di Attributi di Qualità(**Q.A.**). Coerentemente con quanto detto poc'anzi, la scelta di inserire, tra i moduli in cui è decomposto il sistema, l'interfaccia "*InterfacciaRubrica*" è finalizzata a fornire maggiore **Modularità**, **Riusabilità** e **Manutenibilità** al sistema stesso.

La **Modularità** è garantita dal fatto che, con l'aggiunta di un'interfaccia, si possono modificare in qualunque modo le classi (o la classe) che la implementano e tali modifiche non avranno alcun effetto sugli altri componenti del sistema. Ciò è realizzabile poichè il contratto pubblico dei metodi definiti dall'interfaccia rimarrà invariato.

Per quanto riguarda la **Riusabilità**, possiamo affermare che è garantita. L'interfaccia potrebbe essere implementata in qualsiasi altra applicazione senza alcun problema, gli unici vincoli da dover rispettare sono i contratti pubblici dei metodi dell'interfaccia stessa.

E' garantita anche una buona **Manutenibilit **. E' possibile correggere o modificare facilmente una delle classi che implementa l'interfaccia senza dover necessariamente modificare anche le altre classi che la implementano.

Oltre a tali attributi di qualità possiamo notare che la scelta di definire un'interfaccia nel sistema garantisce il rispetto dell'**Open-Closed Principle**, poiché la classe "Rubrica" risulta chiusa alla modifica (modifiche dei metodi della classe non hanno impatto sui client della stessa) ed aperta all'estensione (la classe può essere modificata ed adattata a scenari differenti da quello attuale).

Altra scelta progettuale dettata dai principi di buona progettazione è stata la definizione delle classi "*ComparatoreContatti*" e "*FileManager*". Queste due classi sono state progettate per garantire il rispetto del **Single Responsibility Principle**. In questo modo, ogni modulo in cui il sistema è stato decomposto si occupa di un singolo compito ben definito, favorendo una maggiore chiarezza e manutenibilità del codice.

Il sistema rispetta anche altri principi fondamentali per una corretta progettazione, come il **K.I.S.S.** (Keep It Simple Stupid) ed il **D.R.Y.** (Don't Repeat Yourself). In particolare si noti come l'aggiunta della classe "*ComparatoreContatti*" sia stata finalizzata proprio ad evitare inutili ripetizioni nel codice, che ne avrebbero per'altro diminuito la leggibilità.

2 Valutazione del livello di coesione:

Classe	Livello di coesione	Descrizione
<i>Contatto</i>	Funzionale	La classe implementa le operazioni necessarie alla gestione della struttura dati che rappresenta un contatto, in particolare i metodi getter e setter
<i>Rubrica</i>	Funzionale	La classe presenta tutti i metodi necessari per la creazione e la gestione di una lista di contatti. Implementa tutti i metodi di <i>InterfacciaRubrica</i>
<i>RubricaController</i>	Comunicazionale	La classe implementa i metodi necessari alla gestione dell'interfaccia grafica, i quali lavorano sugli stessi dati di input
<i>InfoController</i>	Comunicazionale	La classe contiene i metodi necessari per la gestione delle funzionalità della seconda scena, i quali lavorano sugli stessi dati di input
<i>FileManager</i>	Funzionale	La classe presenta i metodi per salvare o caricare la rubrica su/da un file
<i>ComparatoreContatti</i>	Funzionale	La classe ridefinisce il metodo dell'interfaccia <i>Comparator</i> che consente di comparare i contatti in accordo con il criterio di ordinamento richiesto
<i>Main</i>	Funzionale	La classe contiene i metodi che consentono l'esecuzione dell'applicazione

Table 1: Valutazione del livello di coesione

3 Valutazione del livello di accoppiamento:

Classi	Livello di accoppiamento	Descrizione
<i>Rubrica-Contatto</i>	Per dati	La classe Rubrica chiama i metodi pubblici di Contatto, scambiando solamente le informazioni necessarie alla gestione della rubrica
<i>Rubrica-ComparatoreContatti</i>	Per dati	Rubrica chiama il costruttore della classe ComparatoreContatti per istanziare un nuovo comparatore da passare al metodo <i>sort()</i>
<i>InterfacciaRubrica-Contatto</i>	Per dati	InterfacciaRubrica prende come parametri di input dei propri metodi astratti degli oggetti di Contatto. Il modo in cui verranno utilizzati tali contatti dipenderà dalla specifica implementazione di tale interfaccia
<i>InfoController-InterfacciaRubrica</i>	Per dati	InfoController accede ai metodi pubblici di InterfacciaRubrica, scambiando solamente i dati necessari per le operazioni di modifica e creazione
<i>InfoController-Contatto</i>	Per dati	InfoController accede ai metodi pubblici di Contatto per salvare le modifiche effettuate
<i>FileManager-InterfacciaRubrica</i>	Per dati	FileManager utilizza i metodi pubblici di InterfacciaRubrica per inserire la lista dei contatti letta dal file selezionato dall'utente. Inoltre istanzia un oggetto InterfacciaRubrica per l'esecuzione del metodo <i>salvaSuFile</i>
<i>FileManager-Contatto</i>	Per dati	FileManager usa i metodi pubblici di Contatto nelle sue operazioni, sia per creare un nuovo contatto con i dati letti da un file, sia per accedere agli attributi dei contatti da scrivere su un file
<i>RubricaController-FileManager</i>	Per dati	RubricaController accede solo ai metodi pubblici di FileManager, per garantire le operazioni di lettura e scrittura della rubrica dal/sul file
<i>RubricaController-InfoController</i>	Per controllo	RubricaController invoca i metodi pubblici di InfoController passando come parametro di input un valore di controllo, dal quale dipende la logica di funzionamento di tale classe
<i>RubricaController-InterfacciaRubrica</i>	Per dati	RubricaController chiama i metodi pubblici di InterfacciaRubrica per eseguire le operazioni di base sulla lista dei contatti
<i>ComparatoreContatti-Contatto</i>	Per dati	ComparatoreContatti usa i metodi getter pubblici messi a disposizione dalla classe Contatto per ottenere i cognomi ed i nomi da comparare

Table 2: Valutazione del livello di accoppiamento

4 Diagrammi di sequenza

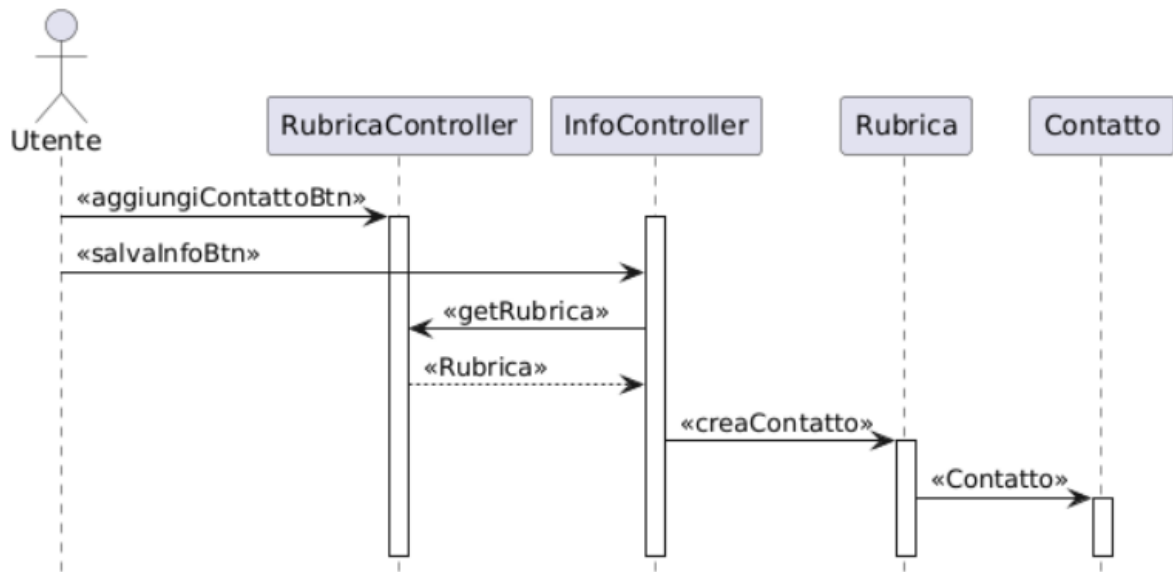


Figure 2: Interazione 1 - Crea contatto

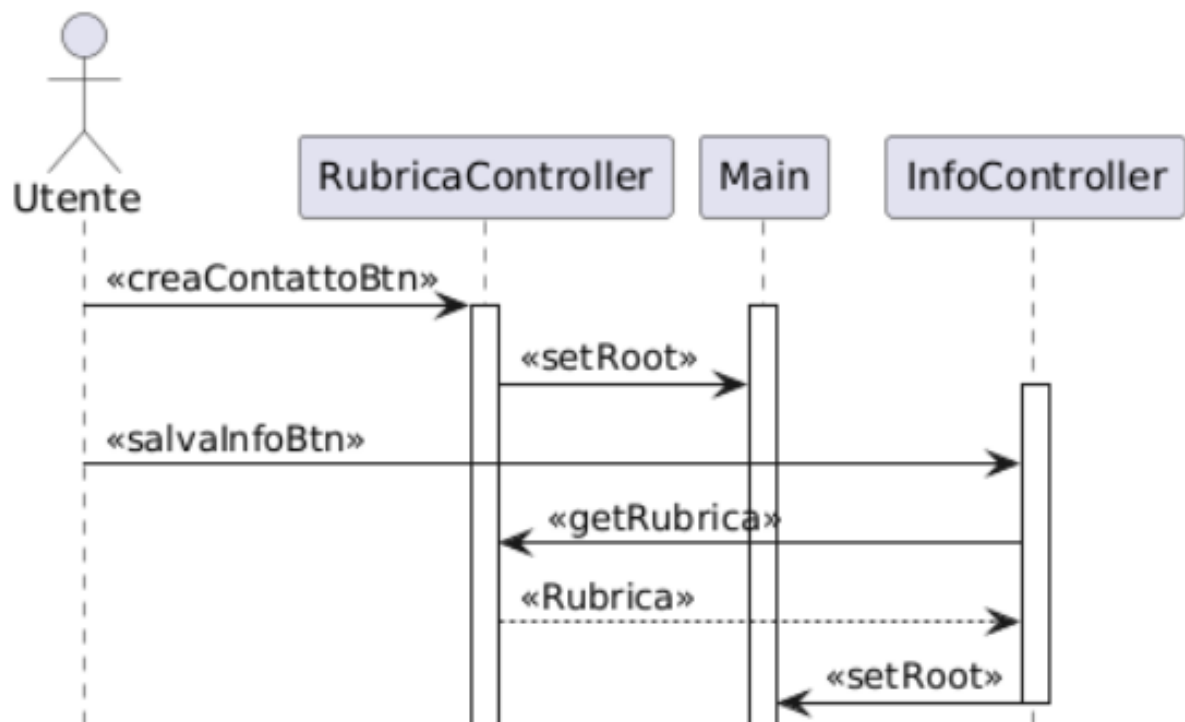


Figure 3: Diagramma di dettaglio della creazione

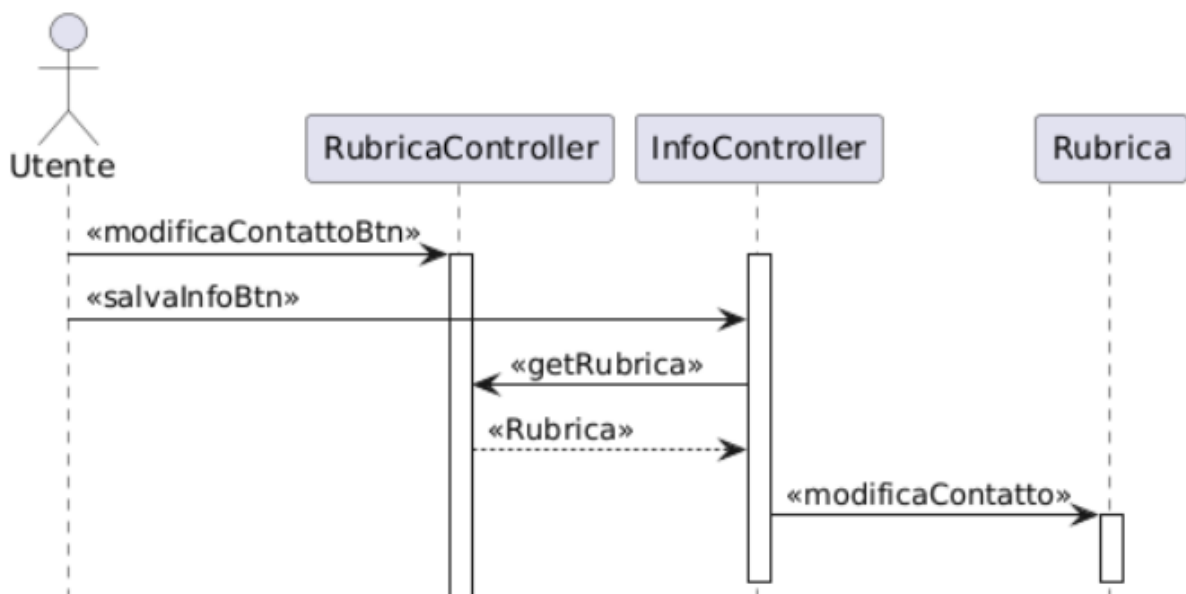


Figure 4: Interazione 2 - Modifica

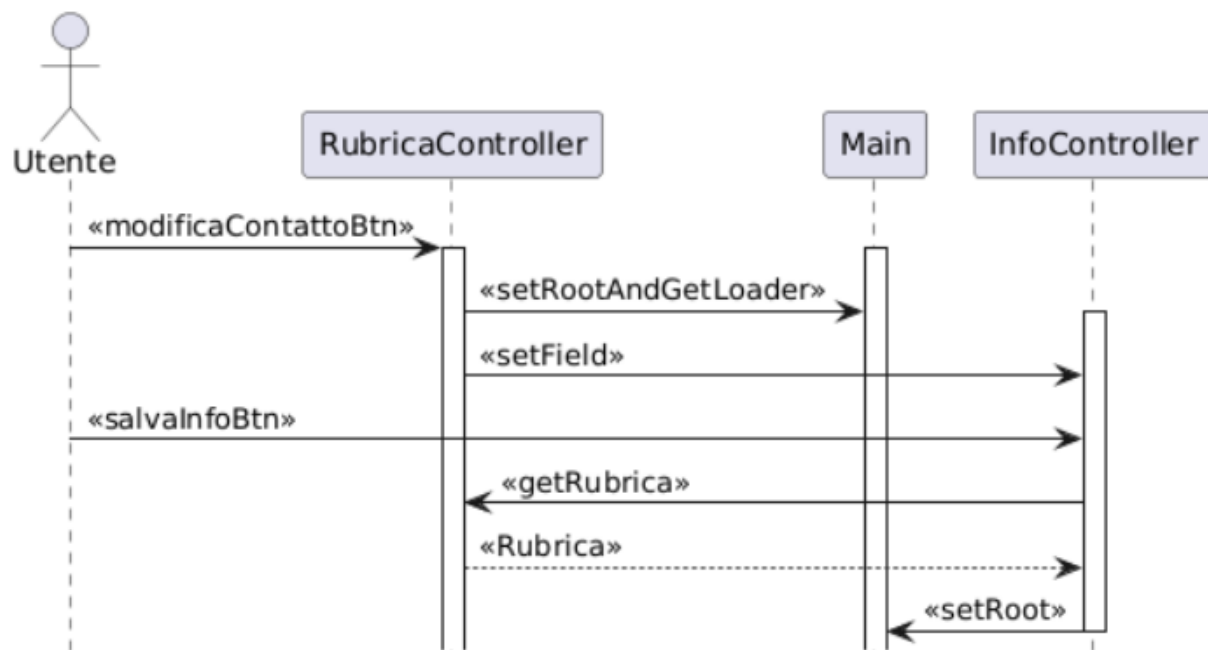


Figure 5: Diagramma di dettaglio della modifica

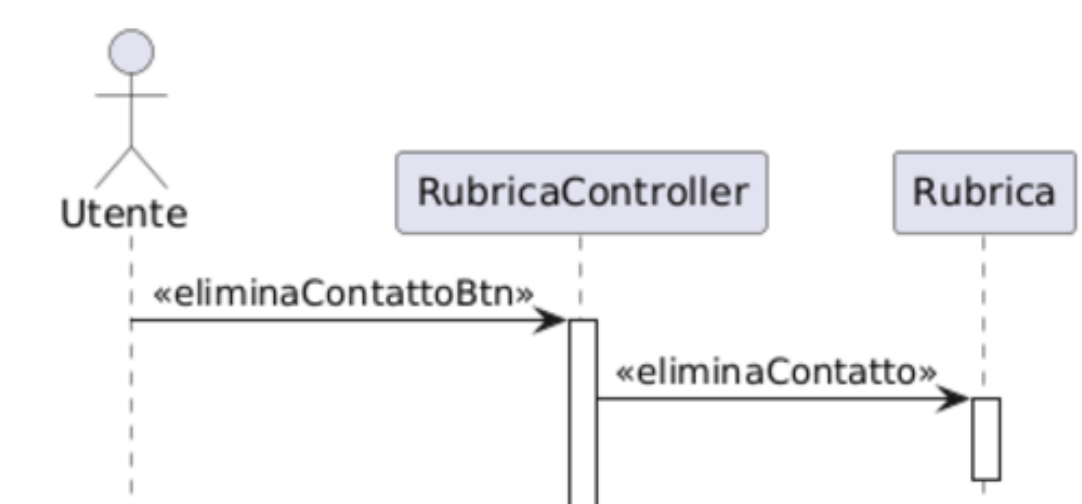


Figure 6: Interazione 3 - Elimina

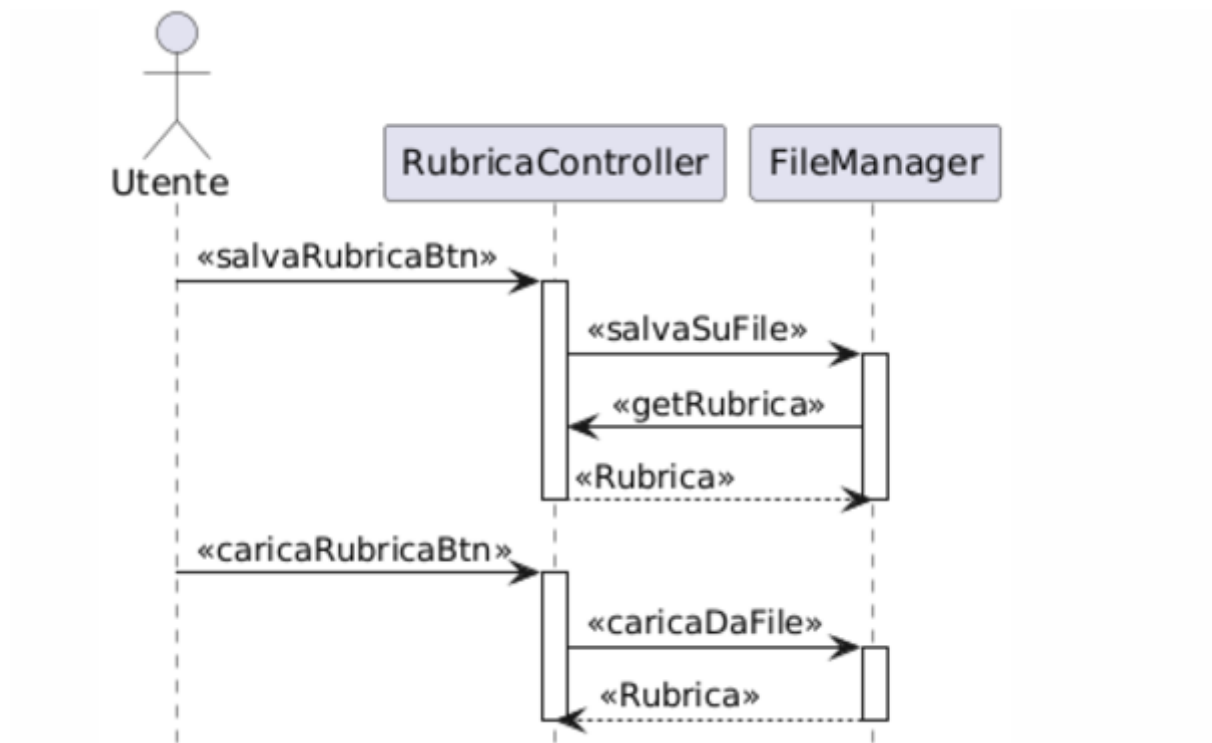


Figure 7: Interazione 4 - Operazioni su file

5 Diagramma dei Package

