Università degli studi di Salerno
Terzo anno Ingegneria Informatica
2024/2025

Design

Rubrica telefonica

A cura di :
Amato Salvatore
Bruno Andrea
Cerra Claudia
Finamore Francesco

Indice

1) Obiettivo del documento

Questo documento ha lo scopo di fornire una descrizione precisa e approfondita della progettazione della rubrica, includendo il diagramma delle classi, i diagrammi di sequenza ed ulteriori diagrammi, quali il diagramma dei package e il diagramma delle attività, necessari a chiarire il funzionamento di alcune parti del software. Ogni sezione è accompagnata da una breve introduzione per facilitarne la comprensione e una parte di commenti successiva ai diagrammi per chiarire le scelte di design, tra cui coesione, accoppiamento e principi di buona progettazione.

2) Diagramma dei package

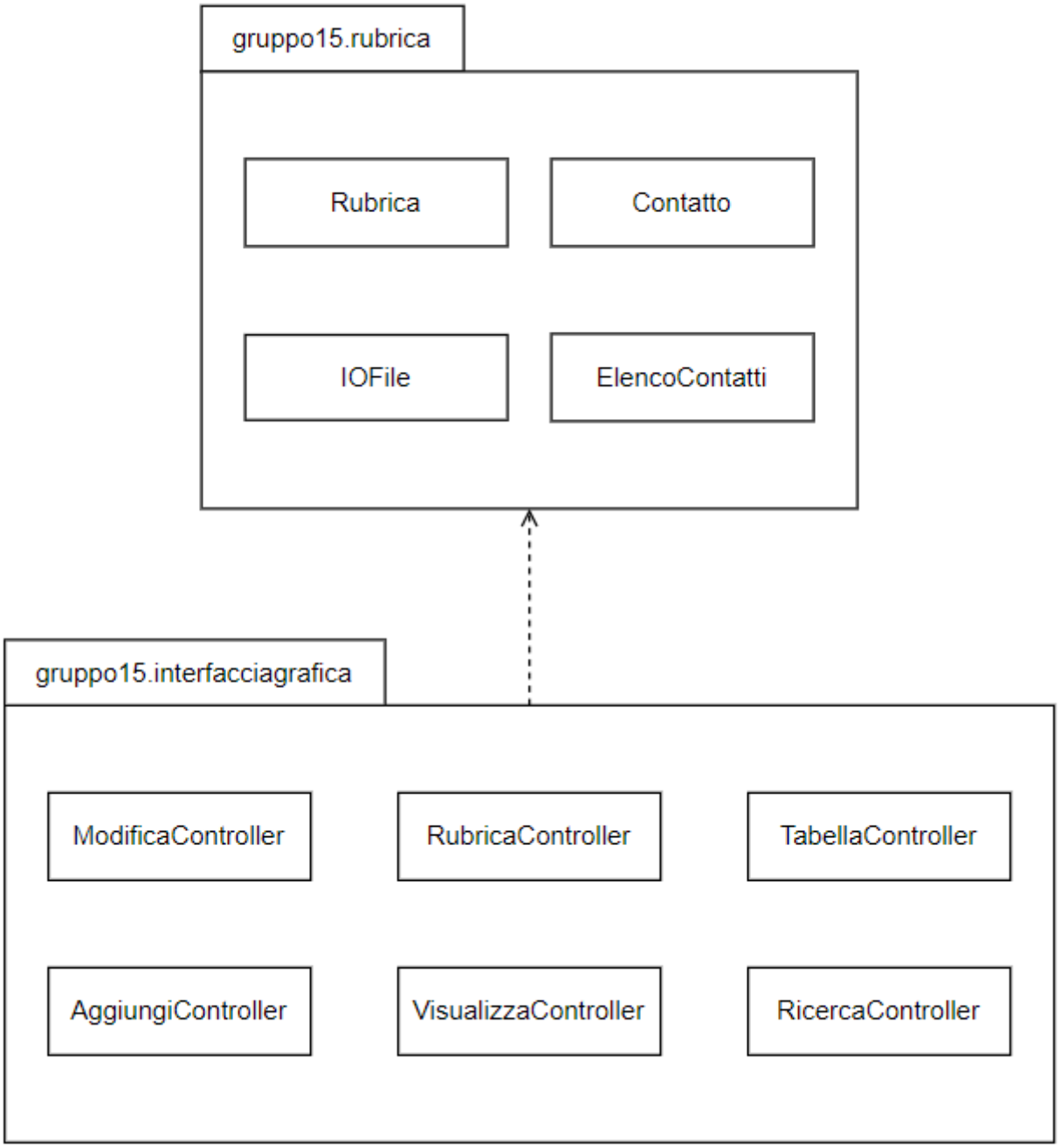
Un diagramma dei package illustra come le classi sono organizzate in package.
Il software è diviso in due package :

- **gruppo15.rubrica**

Si occupa della logica dell'applicazione, gestisce i dati e include tutte le classi che si occupano delle funzionalità base della rubrica.

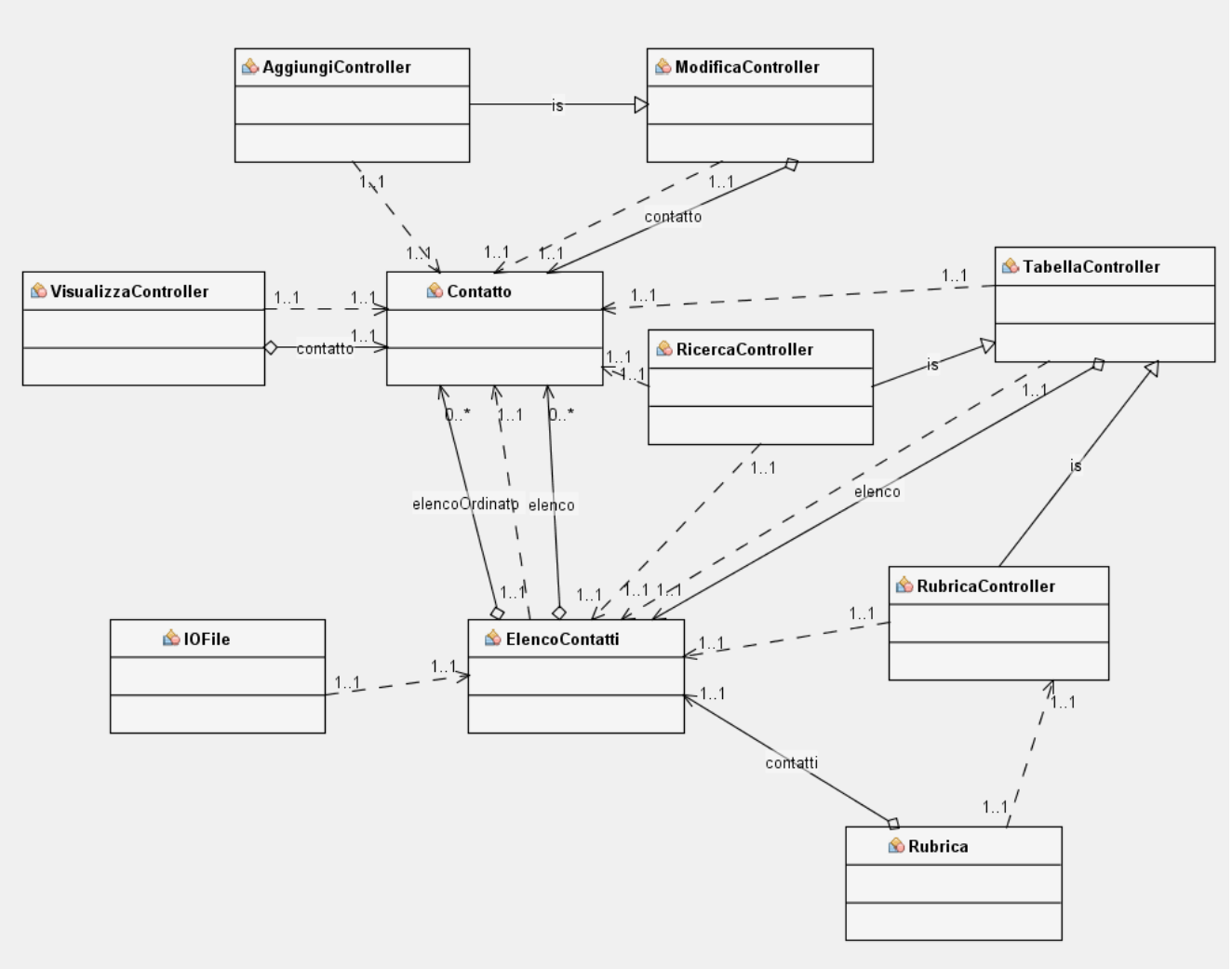
- **gruppo15.interfacciagrafica**

Include tutte le classi che lavorano sull'interfaccia grafica. I file FXML si trovano nelle risorse del progetto e le classi di questo package vi hanno accesso e ne costituiscono la logica applicativa.

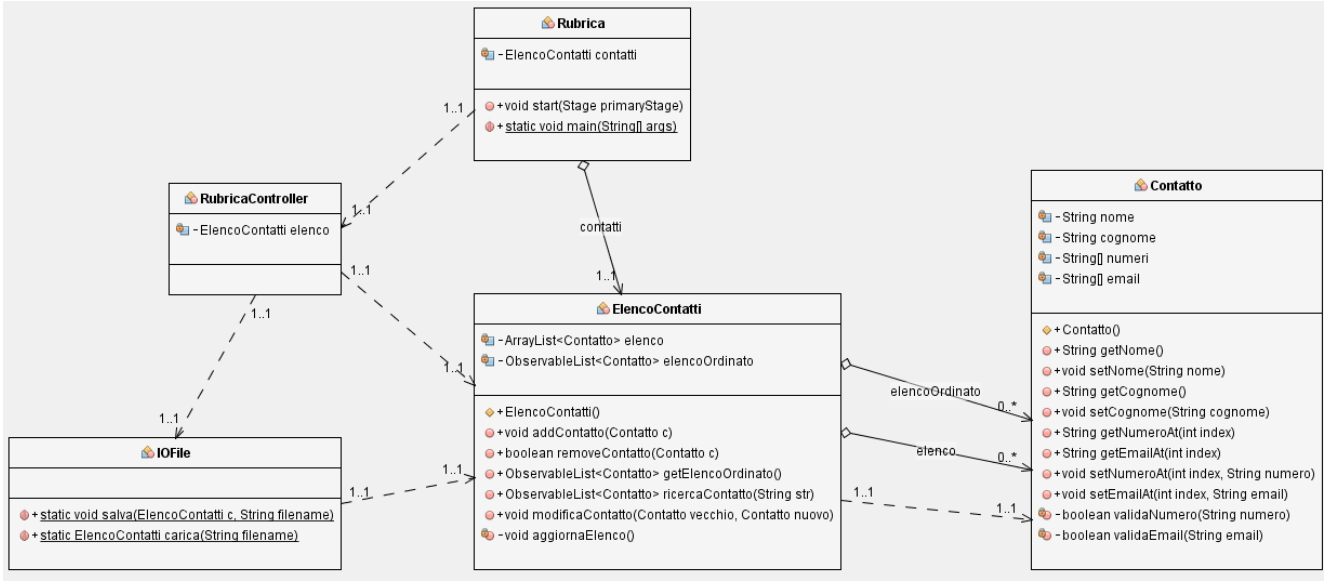


3) Diagramma delle classi

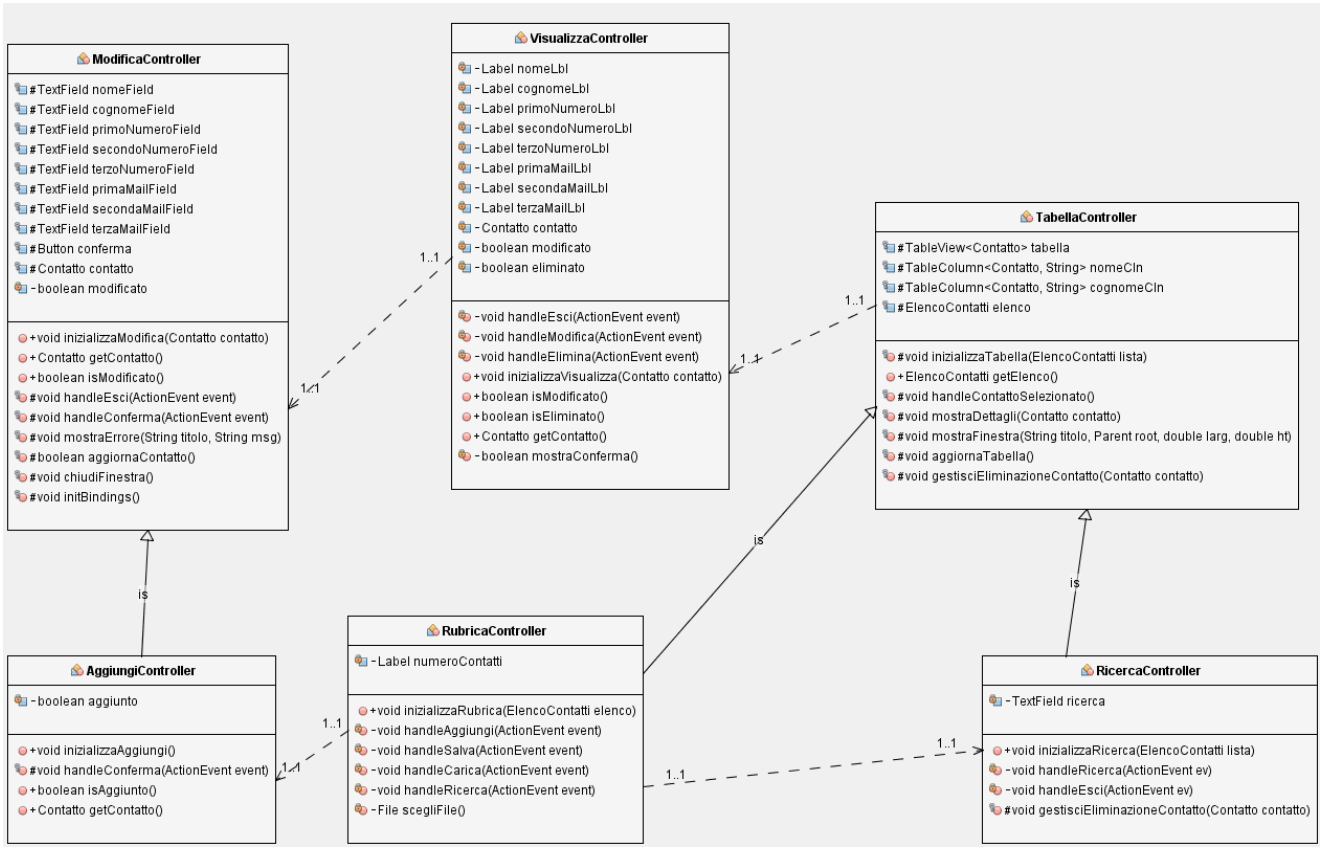
Un diagramma delle classi è un diagramma che permette di descrivere le caratteristiche delle classi e le relazioni tra le stesse. Abbiamo creato tre diagrammi utilizzando EasyUML per illustrare le dipendenze tra le classi.



Questo diagramma descrive le principali dipendenze tra tutte le classi del progetto, indipendentemente dal loro package, tralasciando gli attributi e i metodi delle classi in favore di una migliore leggibilità.



Il secondo diagramma descrive le relazioni tra le classi del package “**gruppo15.rubrica**”, contente la logica fondamentale dell’applicazione. Viene evidenziata anche la relazione che queste classi hanno con RubricaController, la classe che viene istanziata per avviare l’interfaccia grafica.



Il terzo diagramma descrive le relazioni tra le classi del package “**gruppo15.interfacciagrafica**”, contenente tutte le classi relative all’interfaccia grafica. Sono tralasciate le classi dell’altro package per esplicitare meglio le relazioni tra i vari controller.

3.1) Coesione e Accoppiamento

L'organizzazione del software è a finestre. Ogni controller gestisce la propria view, che a sua volta rappresenta una finestra. Questa scelta ci permette di raggiungere una coesione funzionale per la maggior parte delle classi relative all'interfaccia grafica: infatti, in ogni controller è implementata esclusivamente la logica per gestire la sua finestra. Inoltre, come già detto in precedenza, la logica relativa alle funzionalità chiave dell'applicazione è gestita in un altro package ed è separata dall'interfaccia grafica: questo permette anche a quelle classi di raggiungere una coesione funzionale, in quanto ogni classe del package Rubrica assolve un unico compito.
L'accoppiamento che abbiamo raggiunto è quello per dati, infatti, come si evince anche dal diagramma delle classi, tra un modulo e un altro vengono scambiate solo le informazioni strettamente necessarie.

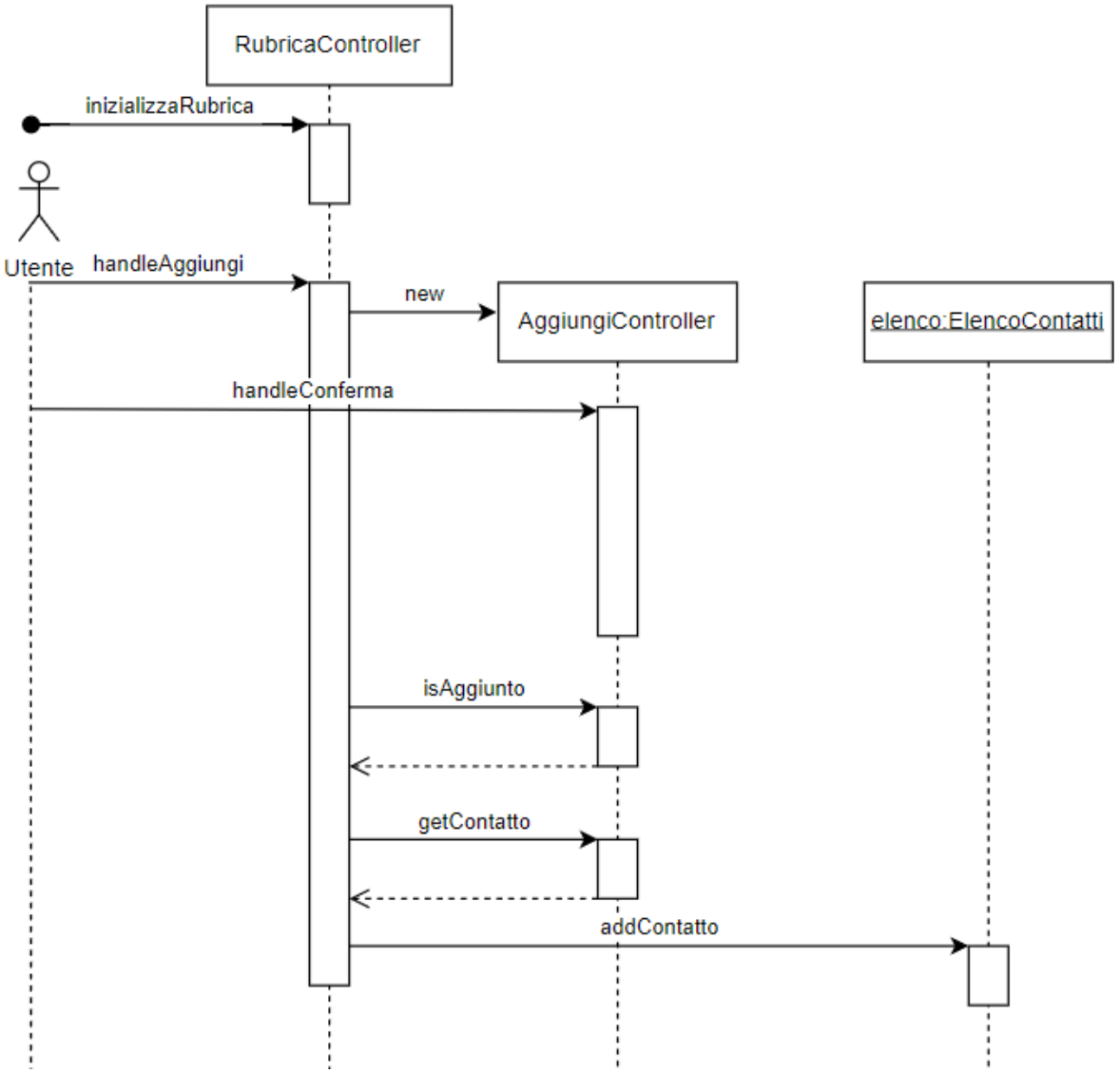
4) Diagramma di sequenza

Un diagramma di sequenza descrive il comportamento di un sottoinsieme del sistema in uno scenario. Grazie allo stesso è possibile visualizzare gli oggetti coinvolti e i messaggi scambiati tra di loro. Le interazioni che verranno trattate, considerate più significative, sono:

1. **Aggiungi:** la sequenza di azioni relativa a quando un utente vuole aggiungere un contatto alla rubrica;
2. **Visualizza:** la sequenza di azioni relativa a quando l'utente vuole vedere tutte le informazioni di un contatto;
3. **Ricerca:** la sequenza di azioni relativa a quando l'utente vuole cercare un contatto;
4. **Modifica:** la sequenza di azioni relativa a quando l'utente vuole modificare un contatto.

La quinta interazione relativa alla **creazione della finestra** è una vista dettagliata: siccome la creazione della finestra è un'operazione onerosa, renderebbe gli altri diagrammi di sequenza poco leggibili. Per questo motivo, abbiamo preferito ometterla dai diagrammi e l'abbiamo presa in esame separatamente.

4.1) Aggiungi

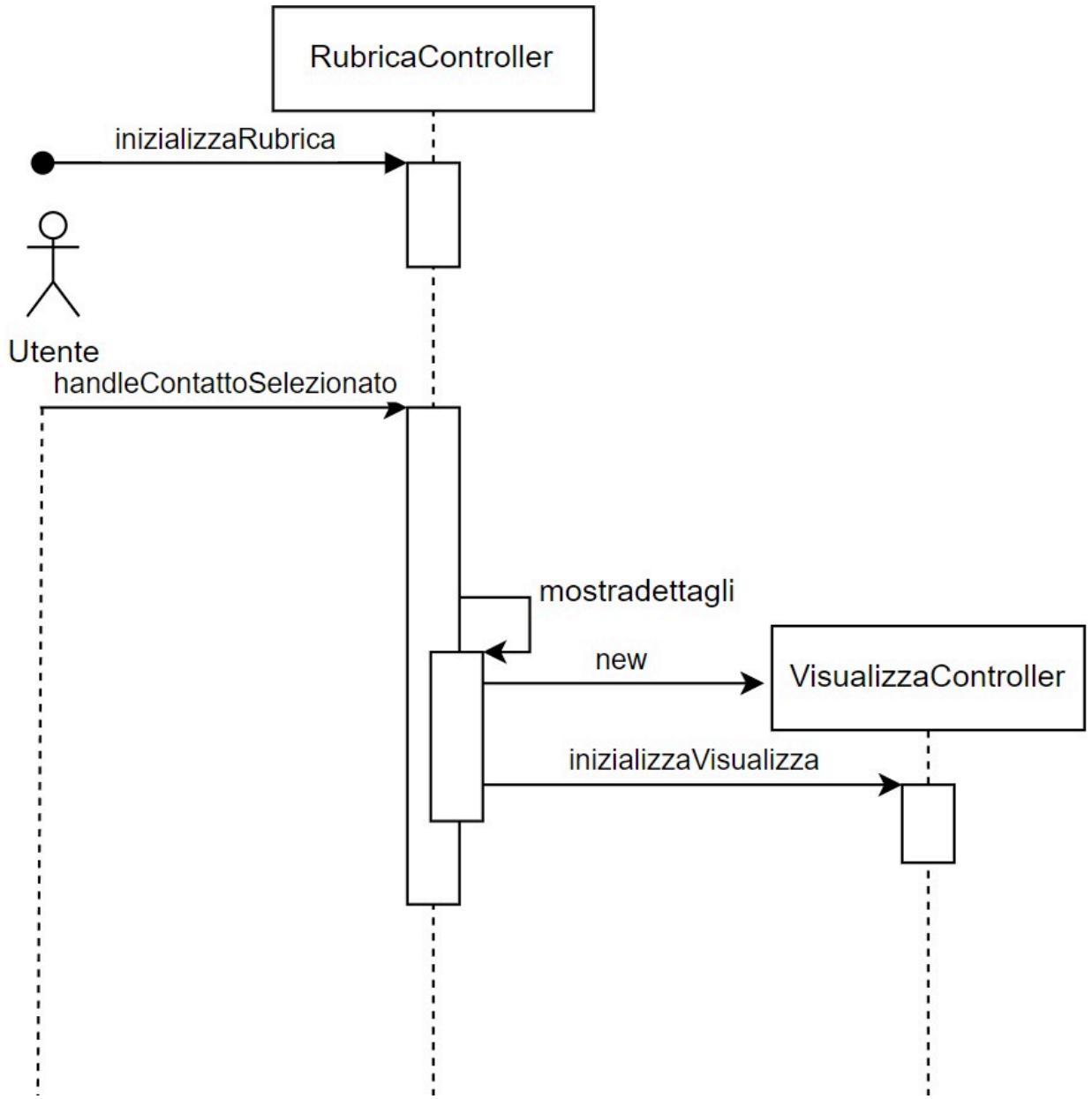


Questo diagramma mostra l'operazione "Aggiungi" proposta all'interno della nostra Rubrica. Sono presenti quattro attori:

- **Utente**, il cui compito principale è quello di interagire con l'interfaccia mediante i bottoni;
- **RubricaController**, che gestisce la finestra principale della rubrica;
- **AggiungiController**, che gestisce la finestra relativa all'aggiunta di un contatto nella rubrica;
- **elenco**, oggetto della classe **ElencoContatti**, che implementa la lista dei contatti.

È opportuno precisare che la creazione dell'oggetto AggiungiConferma incapsula la creazione di una nuova finestra. Dunque, in questo diagramma, l'utente clicca sul bottone "Aggiungi" nella finestra principale, inserisce i dati del contatto nella nuova finestra e clicca sul bottone "Conferma". I metodi handleConferma e handleAggiungi vengono quindi chiamati quando l'utente clicca sui bottoni "Conferma" e "Aggiungi". In questo diagramma consideriamo l'ipotesi che il contatto debba essere aggiunto (potrebbe non esserlo se l'utente ha chiuso la relativa finestra), dunque il risultato del metodo isAggiunto sarà true e l'oggetto RubricaController potrà fare la get del contatto per inserirlo nella rubrica.

4.2) Visualizza

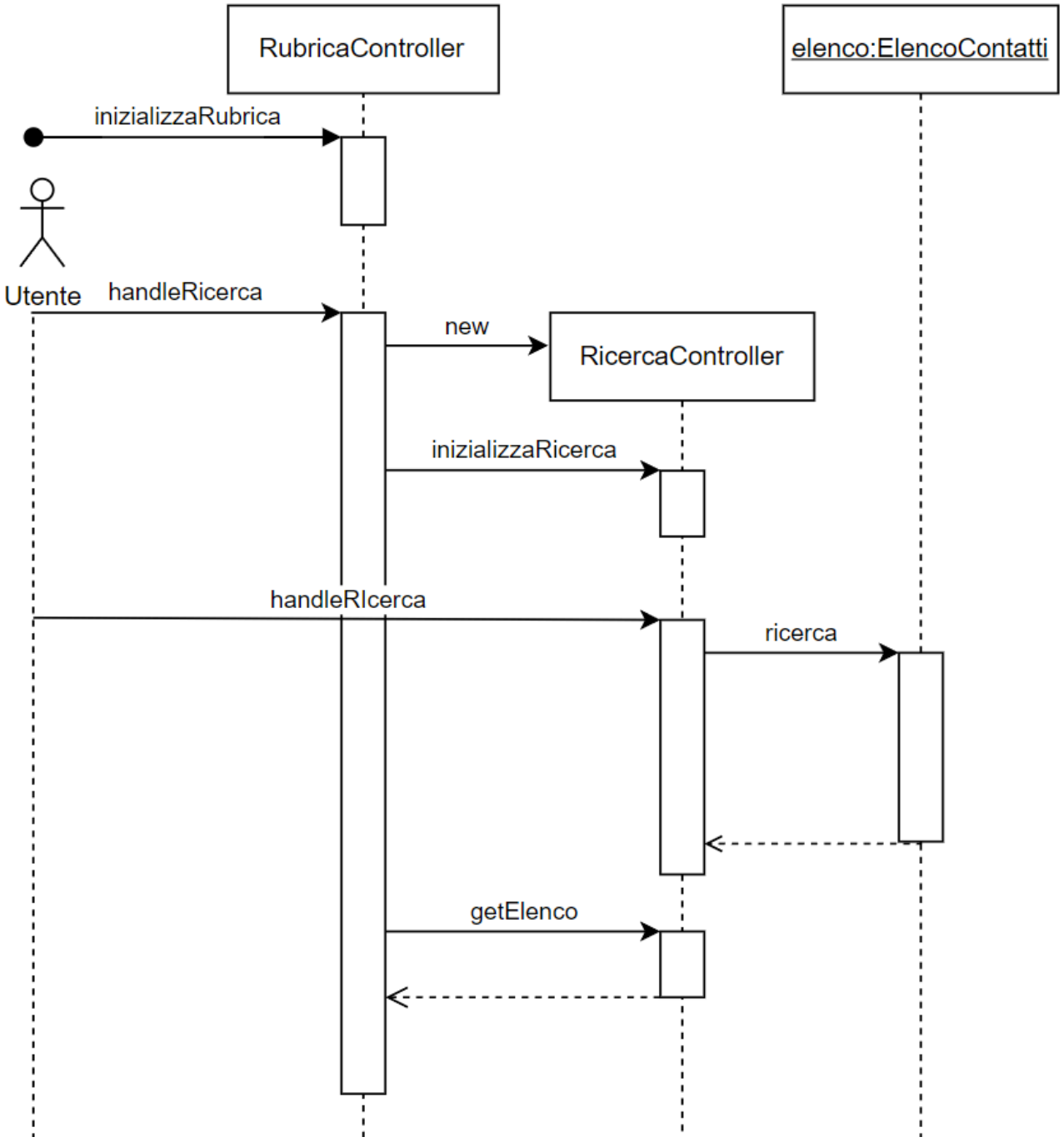


Questo diagramma mostra l'operazione "Visualizza" proposta all'interno della nostra Rubrica. Sono presenti tre attori:

- **Utente**, che clicca su un contatto da visualizzare nella tabella dei contatti;
- **RubricaController**, che gestisce la finestra principale della rubrica;
- **VisualizzaController**, che gestisce la finestra relativa alla visualizzazione delle informazioni di un contatto.

Quando viene istanziato l'oggetto VisualizzaController viene anche creata la nuova finestra e a quell'oggetto viene passato il contatto da visualizzare tramite inizializzaVisualizzazione.

4.3) Ricerca



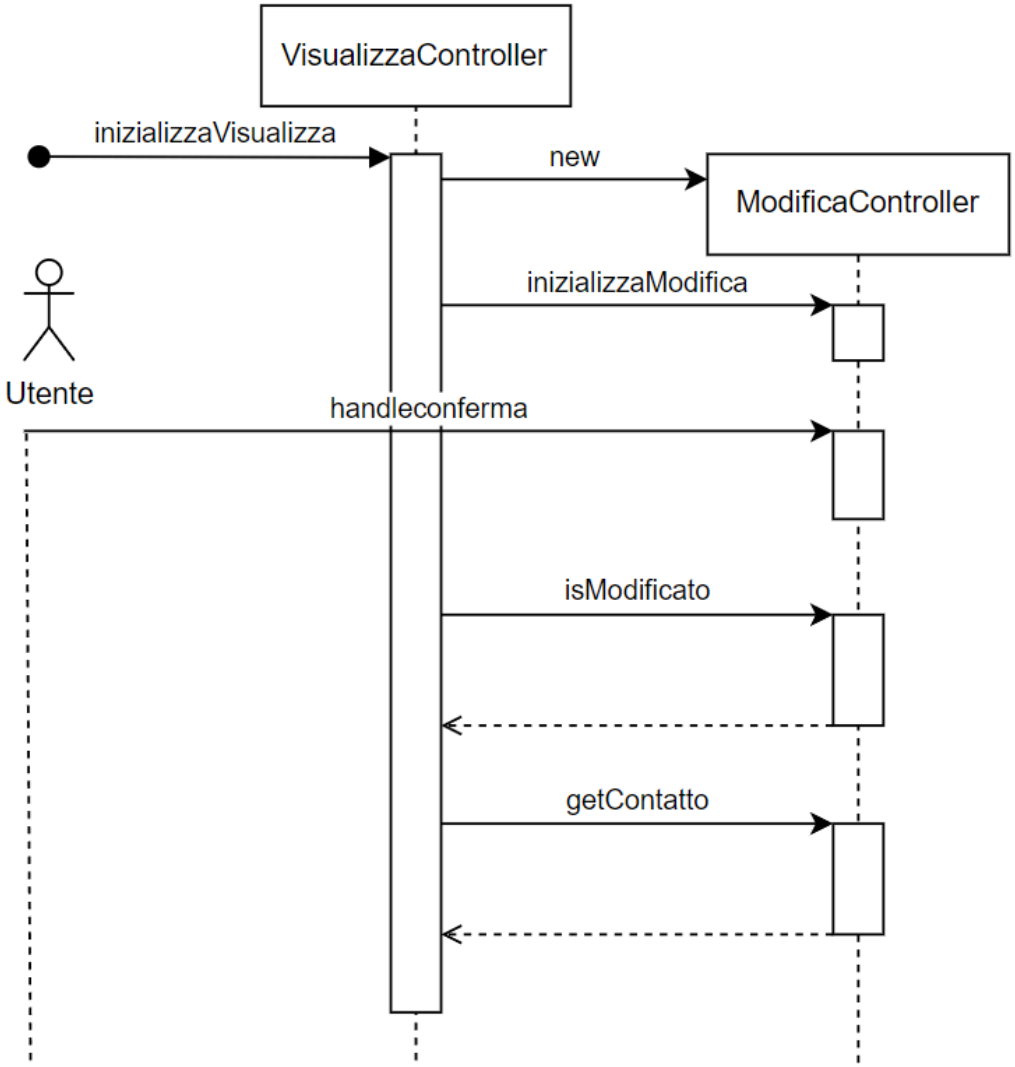
Questo diagramma mostra l'operazione "Ricerca" proposta all'interno della nostra Rubrica. Sono presenti quattro attori:

- **Utente**, che clicca sul bottone "Ricerca" nella finestra principale;
- **RubricaController**, che gestisce la finestra principale della rubrica;
- **RicercaController**, che gestisce la finestra relativa alla ricerca di un contatto.
- **elenco**, oggetto della classe **ElencoContatti**, mantenuto dalla classe RicercaController, che gestisce l'elenco dei contatti che vengono trovati dalla ricerca.

Quando viene istanziato l'oggetto VisualizzaController viene anche creata la nuova finestra e a quell'oggetto viene passato l'elenco in cui cercare. Quello stesso elenco verrà modificato da

RicercaController in caso di modifiche o rimozioni di singoli contatti, per cui RubricaController dovrà effettuare una getElenco per ottenere l'elenco aggiornato da mostrare nella tabella principale.
Il metodo handleRicerca assume due significati: quando è chiamato su RubricaController rappresenta la pressione del bottone "Ricerca" nella finestra principale, mentre quando è chiamato su RicercaController rappresenta l'inizio dell'effettiva ricerca (l'utente scrive nel textfield e preme il tasto "Ricerca").

4.4) Modifica



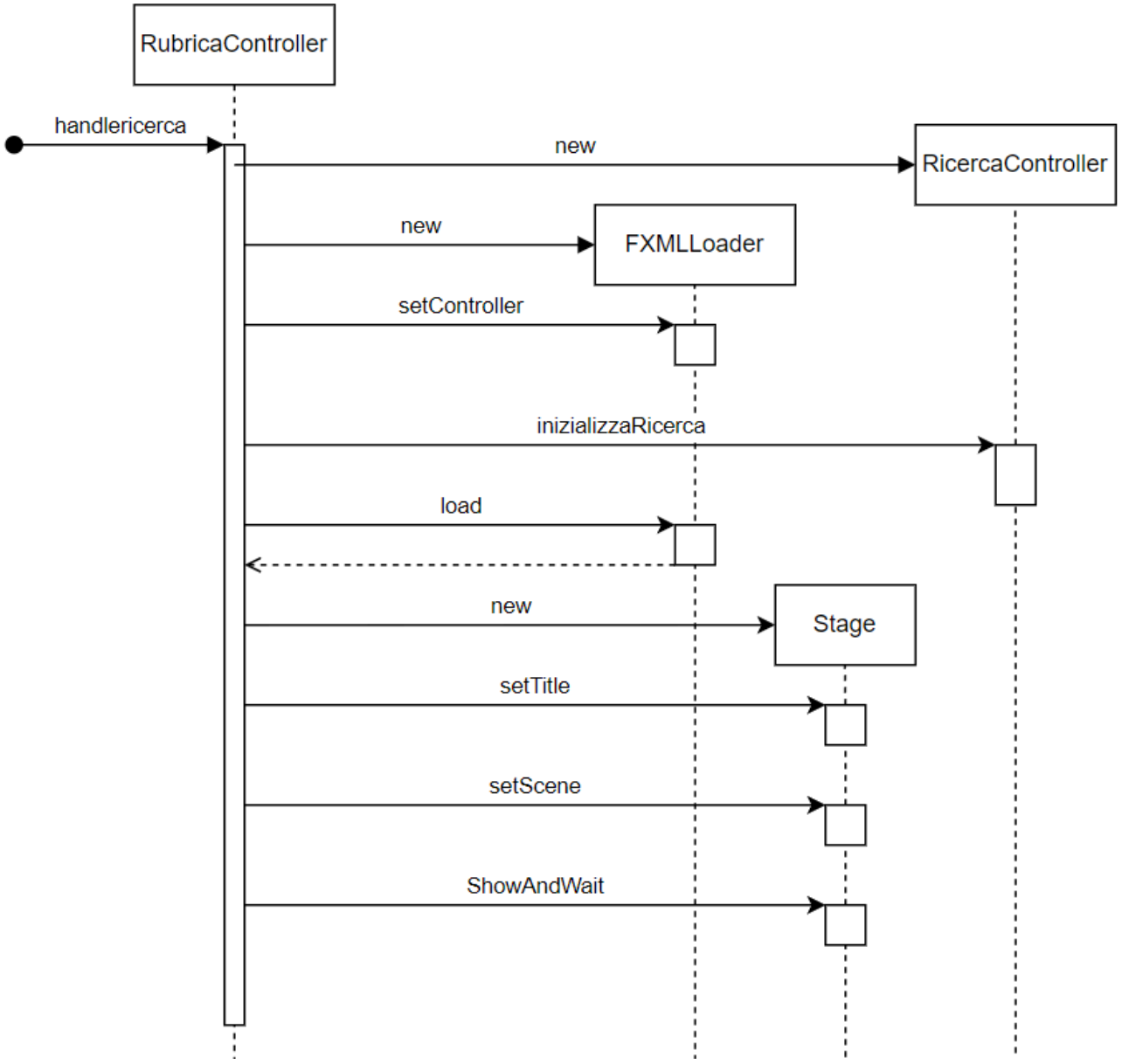
Questo diagramma mostra l'operazione "Modifica" proposta all'interno della nostra Rubrica. Sono presenti tre attori :

- **Utente**, che clicca sul contatto da modificare e effettua le modifiche che preferisce;
- **VisualizzaController**, che gestisce la finestra relativa alla visualizzazione delle informazioni di un contatto;
- **ModificaController**, che gestisce la finestra relativa alla modifica.

Quando viene istanziato l'oggetto ModificaController viene anche creata la nuova finestra e a quell'oggetto viene passato il contatto da modificare. Nel caso illustrato dal diagramma il contatto viene effettivamente modificato (l'utente potrebbe chiudere la finestra di modifica senza confermare): VisualizzaController fa quindi una get del contatto in modo da aggiornare le sue informazioni.

4.5) Versione dettagliata: Creazione della finestra

In questa vista dettagliata abbiamo voluto approfondire la creazione di una finestra. Siccome il processo è molto simile per la maggior parte dei controller, prenderemo in esame solo il caso in cui viene creata la finestra relativa alla ricerca. Illustreremo sia la creazione che la chiusura di una finestra attraverso due diagrammi:

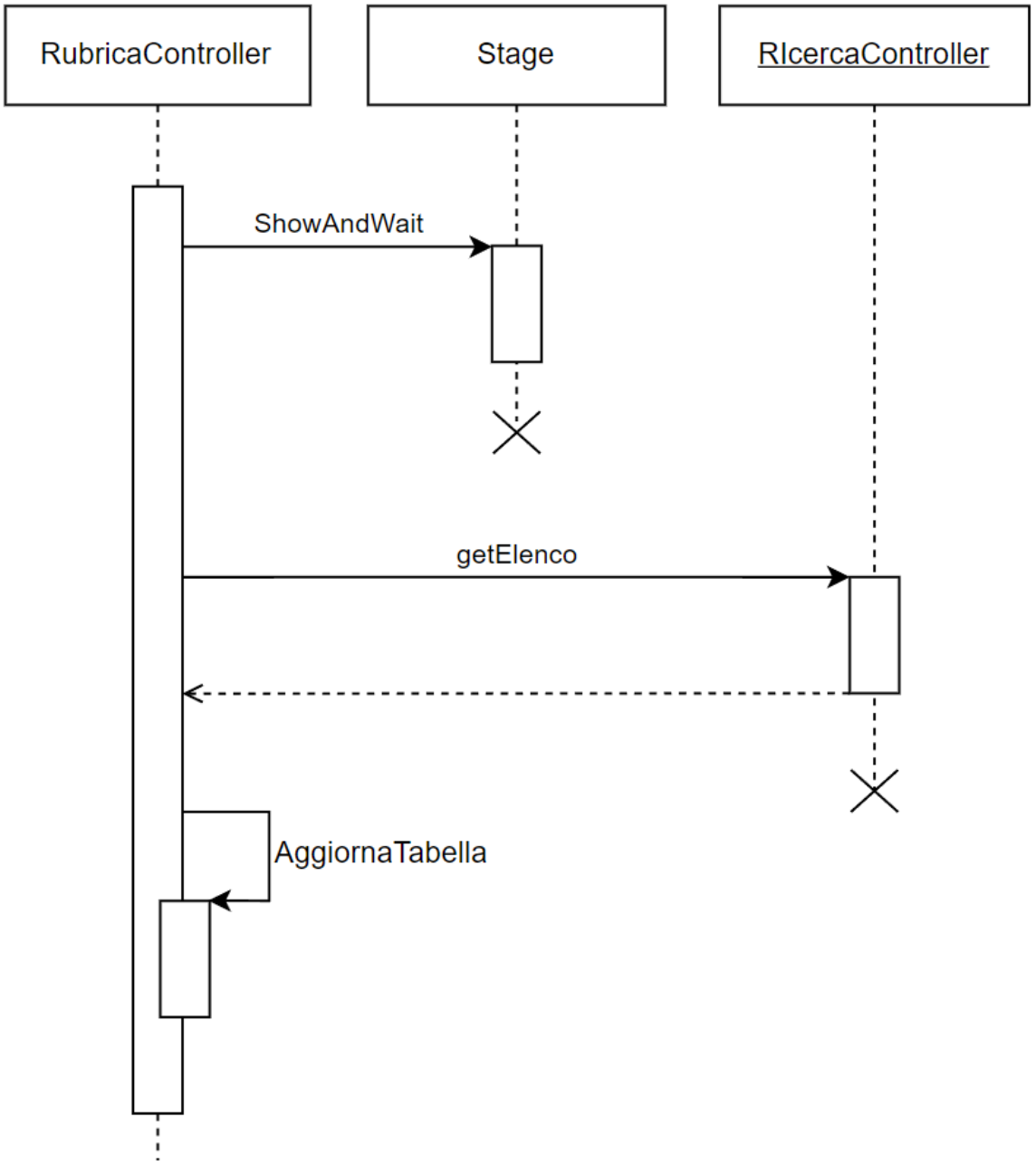


Questo diagramma mostra la creazione della finestra di ricerca proposta all'interno della nostra Rubrica. Sono presenti quattro attori :

- **RubricaController**, che gestisce l'intera operazione dopo che l'utente ha cliccato sul bottone "Ricerca";
- **FXMLLoader**, che gestisce le informazioni relative al caricamento del file fxml associato alla finestra di ricerca;
- **Stage**, che rappresenta la nuova finestra;
- **RicercaController**, che gestisce il file fxml caricato in quanto argomento del metodo setController chiamato sull'oggetto FXMLLoader.

L'esecuzione del metodo handleRicerca viene messa "in pausa" dopo l'esecuzione del metodo showAndWait sull'oggetto Stage. Il metodo ,quindi, si mette in attesa della chiusura della finestra, in

modo tale da aggiornare le informazioni sulla finestra principale gestita da RubricaController solo dopo che l'utente ha ultimato l'operazione sulla seconda finestra (in questo caso, quella di ricerca).



Questo diagramma mostra la chiusura della stessa finestra e può essere considerato un'estensione del precedente diagramma: anche gli attori presenti svolgono gli stessi ruoli.
In questo caso supponiamo che l'utente abbia chiuso la finestra di ricerca. L'esecuzione del metodo handleRicerca viene ripresa e:

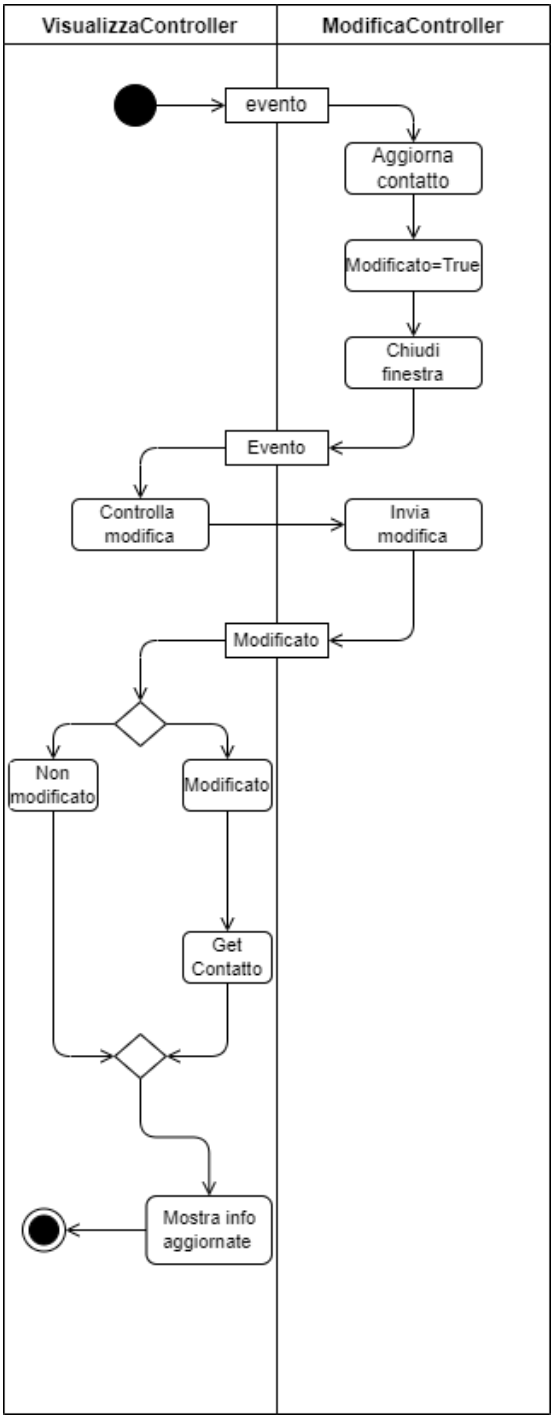
- l'oggetto Stage non ha più un utilizzo, in quanto rappresentava la finestra che ormai è stata chiusa
- RubricaController chiama una getElenco sull'oggetto RicercaController per ottenere il nuovo elenco (vedi diagramma 4.3) e poi aggiorna la tabella della finestra principale.

5) Altri diagrammi

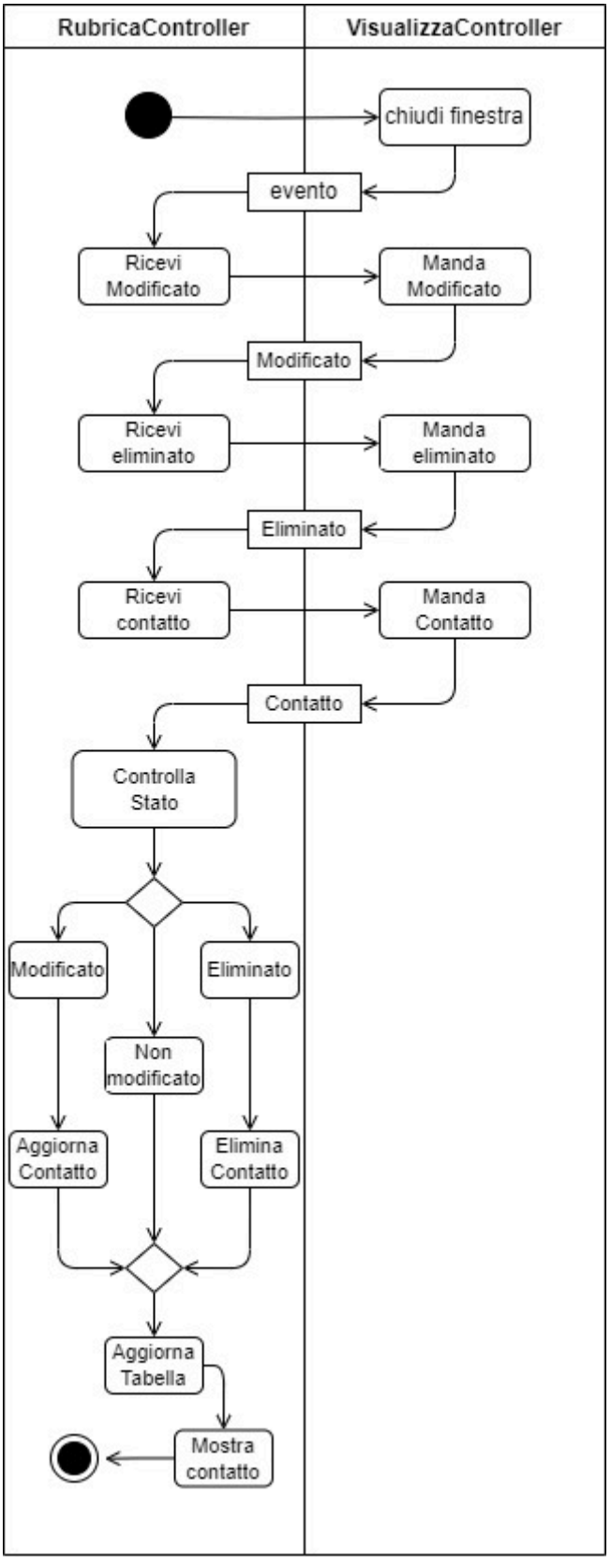
5.1) Diagramma delle attività

Un diagramma delle attività consente di visualizzare il flusso di azioni coinvolte in un processo dando maggior importanza alla sequenza e alle condizioni di flusso.

Sono presenti due diagrammi delle attività il cui compito principale è quello di descrivere la “modifica”, l’operazione più complessa del nostro programma.
Nel primo diagramma, azionato dalla conferma di una modifica, viene descritto il processo di modifica del contatto.



Il secondo diagramma, azionato dalla chiusura della finestra di visualizzazione, descrive il processo di modifica del contatto all'interno dell'elenco dei contatti. Questo diagramma descrive l'interazione tra RubricaController e VisualizzaController. L'attività porta alla verifica e al rispettivo aggiornamento del valore di flag di "modifica" ed "eliminato". Il contatto viene poi passato a RubricaController che controlla lo stato delle flag e sceglie se eliminare o modificare il contatto per poi aggiornare la tabella.



6) Scelte progettuali

Durante la prima fase progettuale abbiamo riflettuto su come la nostra rubrica dovesse presentarsi all'utente. Il nostro obiettivo era quello di volerla gestire attraverso finestre per evitare di, seppur più facilmente implementabile, dover interagire con un'unica finestra che contenesse tutti i dati, dando l'impressione di star guardando ad una grossa tabella statica. All'inizio il design era molto complesso: la prima proposta prevedeva dei riferimenti diretti tra i controller creando un pessimo accoppiamento e una pessima coesione. Questa soluzione non rispettava inoltre in alcun modo il principio **K.I.S.S. (Keep It Simple, Stupid!)** e quindi siamo stati portati a rivalutarla. La successiva idea era quella di utilizzare una variabile globale "ElencoContatti" che sarebbe stata facile da usare e da gestire per i controller, ma questo, pur mantenendo comunque un'alta coesione, portava ad un alto accoppiamento. L'idea finale è stata poi quella di usare delle flag che ci hanno permesso di abbassare notevolmente l'accoppiamento, rendendo il codice più leggibile e manutenibile. Abbiamo poi notato un grave errore che andava contro il principio **D.R.Y. (Don't Repeat Yourself)**: sia la rubrica che la ricerca (RubricaController e RicercaController) utilizzavano una TableView con metodi molto simili, onde evitare problemi di classi quasi identiche è stata creata una classe generica TabellaController che gestisce tutte le operazioni comuni. Tutto questo tenendo comunque conto della Separation of Concerns e annesso principio di Ortogonalità, difatti ogni controller è stato pensato per gestire un compito specifico e ben delimitato, come la rubrica controller per la visualizzazione dei contatti o la ricercaController per le operazioni di ricerca e filtraggio.

6.1) Considerazione sui principi di buona progettazione

Abbiamo poi rivisto tutto rimanendo saldi sui principi di progettazione orientati agli oggetti, i 5 principi sono noti come: **S.O.L.I.D.**

- **Singola responsabilità (SRP) :** Abbiamo mantenuto ogni classe focalizzata su un singolo compito, come aggiungiController che gestisce solo l'interfaccia e la logica legata all'aggiunta o alla modifica del contatto.

- **Principio aperto/chiuso (OCP) :** Il design consente di aggiungere nuove funzionalità senza modificare il codice esistente, la TabellaController è estendibile senza alterare la logica del progetto.

- **Sostituzione di Liskov (LSP) :** Abbiamo garantito che le classi derivate potessero sostituire le relative classi base senza problemi. Ad esempio, la generalizzazione tra AggiungiController e ModificaController in una classe comune ha evitato duplicazioni inutili. Non abbiamo utilizzato interfacce troppo grandi e complesse. Ogni classe implementa solo le funzionalità necessarie.

- **Inversione delle dipendenze (DIP) :** Abbiamo fatto in modo che i controller del sistema dipendessero da interfacce comuni. Ad esempio, invece di accedere direttamente all'elenco contatti come variabile globale o legare strettamente i controller tra loro (come nella nostra idea iniziale), abbiamo separato i moduli e fatto in modo che interagissero tramite un livello di astrazione condiviso, minimizzando la dipendenza diretta. Questo approccio permette di cambiare la gestione dell'elenco contatti o altre componenti senza dover riscrivere i controller.