Gruppo 25

Documento del Software
Sistema informativo per la gestione di una rubrica telefonica

Francesca De Pascale, Luisa Crivo, Antonio Manuel Fedullo

Indice del Documento

1 Analisi delle scelte progettuali

Per garantire una struttura di design chiara e intuitiva per il software in sviluppo, è stato adottato un approccio che distingue tra la **visione logica** del sistema (Logical View), la **visione del processo** (Process View) e la **visione implementativa** del sistema (Implementation View).
Questo approccio si ispira al modello *"4+1 View Model"*, tradizionalmente utilizzato per descrivere sistemi software complessi e articolati. Sebbene il progetto in esame non richieda tutte le cinque visioni del modello completo, questa suddivisione consente di organizzare il design in modo strutturato e mirato.
Ogni suddivisione è progettata per uno scopo specifico e rivolta a un pubblico mirato, contribuendo a una rappresentazione completa della struttura, del comportamento e dell'implementazione del sistema. L'adozione di questo metodo garantisce che il sistema sia coerente con i requisiti richiesti, migliorandone la manutenibilità e l'estensibilità.

Logical View

La *Logical View* fornisce una rappresentazione delle funzionalità principali del sistema a un livello alto di astrazione, concentrandosi sull'identificazione e sulla descrizione dei componenti essenziali, quali classi, oggetti e relazioni.
In questa suddivisione, è di fondamentale importanza il **diagramma delle classi**, per illustrare le entità principali individuate e il modo in cui queste interagiscono per supportare le funzionalità del sistema.
Questa *view* fornisce una visione chiara del flusso principale del sistema, evidenziando il design e l'organizzazione delle sue funzionalità.

Process View

La *Process View* si focalizza sugli aspetti dinamici del sistema, descrivendo il comportamento del sistema e il flusso di controllo tra le varie attività o processi.
Particolarmente importanti sono i **diagrammi di sequenza**, che rappresentano i processi, gli oggetti coinvolti e la sequenza di messaggi scambiati per implementare specifiche funzionalità. Questa *view* evidenzia come i componenti interagiscono nel tempo per realizzare le funzionalità richieste.

Implementation View

L'*Implementation View* offre una prospettiva del sistema orientata alla programmazione e alla gestione del software. In questa *view*, viene posta attenzione all'organizzazione interna dei componenti del sistema e alla loro interazione.

I **diagrammi dei componenti** vengono utilizzati per rappresentare la struttura del software, evidenziando i moduli principali, le loro dipendenze e le interfacce attraverso cui comunicano.

I **diagrammi dei package** suddividono il sistema in macro-funzionalità. Ogni package rappresenta una funzionalità e ciascun componente è incapsulato in uno specifico package in base al proprio scopo.

Questa visione aiuta a definire chiaramente i confini tra i diversi moduli, semplificando la gestione del codice e favorendo la modularità e il riutilizzo delle componenti.

MVC Design Pattern

La scelta architetturale di suddividere in più view le funzionalità del sistema, ha portato i progettisti ad optare per un approccio implementativo di tipo **Model-View-Controller (MVC)**. Questo approccio permette di suddividere il sistema in tre componenti principali, ognuna con una responsabilità chiara e distinta.

Model
Il componente Model del design pattern MVC si occupa della gestione dei dati e della logica di business, assicurando l'organizzazione delle informazioni.

View
Si occupa di mostrare i dati del Model all'utente e di inviare gli input dell'utente al Controller. Offre un'interfaccia chiara ed intuitiva, favorendo l'interazione con l'utente.

Controller
Il Controller ha il ruolo di intermediario tra il Model e la View e di gestione degli input dell'utente. Coordina le operazioni del sistema, aggiornando i dati del Model e riflettendo le modifiche nella View.

Questo pattern consente di separare chiaramente le responsabilità delle diverse componenti del software, garantendo modularità, facilità di manutenzione e scalabilità.
Inoltre, con questa suddivisione, si rende più facile lavorare su ciascuna parte separatamente, in modo che eventuali modifiche o aggiunte future non coinvolgano l'intero sistema.

2 Diagramma concettuale delle Classi

Sulla base dei casi d'uso descritti nel Documento dei Requisiti, si è progettato il diagramma concettuale delle classi, visibile in *Figura 1*, per rappresentare le entità principali del software **Rubrica** e le loro relazioni. Le classi individuate sono:

- **Applicazione:** rappresenta il punto di avvio del sistema e coordina le operazioni principali. Si occupa di inizializzare le componenti e gestire il flusso generale dell'applicazione.
- **Rubrica**: gestisce la logica dei contatti. Funzioni principali includono l'aggiunta, la visualizzazione e la ricerca di contatti.
- **Contatto**: rappresenta l'entità base della rubrica, contenente le informazioni individuali di un contatto.
- **InterfacciaElenco:** implementa il pattern MVC suddividendo le responsabilità in:
    - **View:** si occupa di visualizzare la lista completa dei contatti e fornire un'interfaccia grafica per interagire con essa.
    - **Controller:** gestisce le operazioni richieste dall'utente, come la ricerca, l'importazione/esportazione della rubrica, l'eliminazione di un contatto e l'aggiunta ai preferiti. Il Controller utilizza la classe **Rubrica** per accedere ai dati e aggiorna la View in base ai risultati.
- **InterfacciaContatto:** anch'essa segue il modello MVC:
    - **View:** consente di visualizzare i dettagli di un singolo contatto in maniera leggibile e user-friendly.
    - **Controller:** gestisce gli input dell'utente per operazioni come la modifica e la creazione di un contatto. Lavora con la classe **Contatto** per effettuare le operazioni necessarie e aggiorna la View per riflettere i cambiamenti.
- **GestoreFile:** è responsabile della gestione delle operazioni di Input/Output per la rubrica, interagendo direttamente con il file system. Offre funzionalità come il salvataggio e il caricamento dei contatti da file.

Nella fase attuale, il design delle interfacce semplifica la distinzione tra View e Controller, combinandoli in un’unica classe. Questa unificazione temporanea riduce la complessità iniziale, concentrandosi sull'implementazione funzionale del flusso principale del sistema. Tuttavia, è previsto che nel **diagramma dei componenti** View e Controller saranno separati in entità distinte per migliorare modularità, manutenibilità e flessibilità, aderendo al pattern MVC completo.

Le classi sono state scelte e progettate affinché le responsabilità fossero ben definite e conformi al **principio di singola responsabilità (SRP)**. Questo garantisce un'architettura modulare e facilmente estendibile, con un basso accoppiamento tra le componenti e alta coesione all'interno di ciascuna classe.

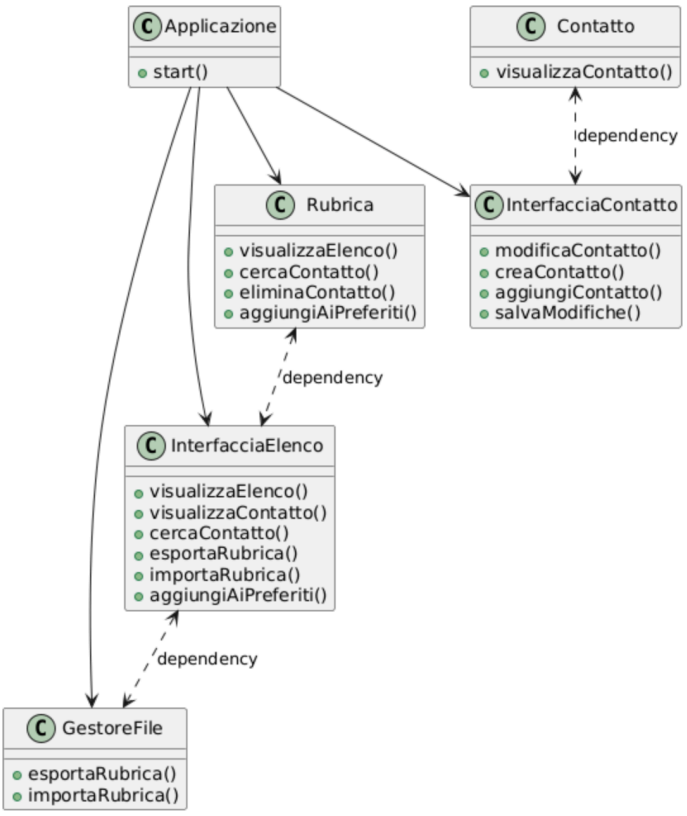


Figura 1: Diagramma concettuale delle classi.

3 Valutazione del livello di coesione

Tra i principi fondamentali per il software design vi è la coesione. Ai fini di una buona progettazione, è necessario che vi sia una **forte coesione** che garantisca che i metodi e le funzionalità all'interno di una classe collaborino strettamente per raggiungere un obiettivo comune, evitando la dispersione tra responsabilità diverse.

CLASSE APPLICAZIONE

La classe Applicazione presenta un livello di **coesione funzionale**, in quanto prevede un unico metodo che ha il ruolo di avviare il sistema. Questa scelta ci soddisfa, in quanto viene rispettato il criterio di alta coesione ricercato.

CLASSE RUBRICA

La classe Rubrica presenta un livello di **coesione funzionale**, poiché tutti i suoi metodi collaborano per raggiungere uno scopo specifico: gestire e organizzare i contatti.
Pertanto viene rispettato il principio di alta coesione ricercato, infatti la classe Rubrica può essere utilizzata in diversi contesti senza che vi sia necessario apportare delle modifiche.

CLASSE CONTATTO

La classe Contatto presenta un livello di **coesione funzionale**, tutti i metodi e attributi sono strettamente connessi alla gestione di un singolo contatto, includendo solo attributi e metodi

strettamente pertinenti a questa responsabilità senza dipendere da funzionalità non pertinenti (come la gestione della rubrica, che è separata).

CLASSE INTERFACCIAELENCO

La parte View di InterfacciaElenco presenta un livello di coesione funzionale, si occupa esclusivamente di visualizzare l'elenco completo dei contatti e le informazioni relative ad un contatto. I metodi implementati nella View riguardano esclusivamente la rappresentazione (es. mostrare l'elenco, mostrare le informazioni dei contatti, visualizzare messaggi di errore o conferma).

La parte Controller di InterfacciaElenco presenta un livello di **coesione funzionale,** si concentra unicamente sulla gestione delle azioni dell'utente, come la visualizzazione, l'eliminazione o l'aggiunta ai preferiti, la ricerca di contatti. Tutti i metodi del Controller servono per tradurre gli input dell'utente in operazioni sul Model (la classe Rubrica) e aggiornare la View, mantenendo la responsabilità ben delimitata.

CLASSE GESTIONEFILE

La classe GestioneFile presenta un livello di **coesione funzionale**, poiché i suoi metodi condividono un contesto comune (gestione dei file della rubrica) e lavorano per realizzare un unico obiettivo: trasferire i dati tra i file esterni e la rubrica.

CLASSE INTERFACCIACONTATTO

La View di InterfacciaContatto presenta un livello di **coesione funzionale,** è responsabile della creazione e modifica di un singolo contatto, garantendo una rappresentazione chiara e leggibile dei dati. La coesione è elevata, poiché include solo metodi strettamente legati alla presentazione delle informazioni (es. mostrare nome, numero di telefono, ecc.).

Il Controller di InterfacciaContatto presenta un livello di **coesione funzionale,** è dedicato alla gestione delle azioni legate a un singolo contatto, come la modifica, l'aggiornamento. Le sue funzionalità sono strettamente connesse alla logica di gestione di un contatto specifico, garantendo una chiara separazione dalle altre componenti.

4 Valutazione del livello di accoppiamento

CLASSE APPLICAZIONE - CLASSE RUBRICA

L'Applicazione interagisce con Rubrica per importare ed esportare i contatti, nonché per accedere ai metodi di gestione della rubrica.Tra queste due classi vi è un grado di **accoppiamento per dati**, dato che la comunicazione avviene tramite lo scambio di dati strutturati.

CLASSE APPLICAZIONE - CLASSE INTERFACCIAELENCO

La classe Applicazione è direttamente connessa alla classe InterfacciaElenco. L'Applicazione comunica con InterfacciaElenco per gestire la visualizzazione dei contatti. Tra queste due classi vi è un grado di **accoppiamento per dati**, poiché la comunicazione è basata su richieste di esecuzione di azioni e dati strettamente necessari.

CLASSE APPLICAZIONE - CLASSE INTERFACCIACONTATTO

L'interazione tra Applicazione e InterfacciaContatto avviene tramite il passaggio di dati definiti (es. nome, cognome, email). L'accoppiamento è per **dati**, senza accesso diretto all'implementazione interna di InterfacciaContatto.

CLASSE APPLICAZIONE - CLASSE GESTOREFILE

La classe Applicazione è direttamente connessa alla classe Rubrica. Tra queste due classi vi è un grado di **accoppiamento per dati**, dato che la classe Applicazione passa alla classe GestoreFile solo i dati necessari (come un file o un elenco di contatti), senza accedere alla sua logica interna o struttura.

CLASSE RUBRICA - CLASSE CONTATTO

La classe Rubrica è direttamente connessa alla classe Contatto. La classe Rubrica contiene e gestisce una collezione di oggetti di tipo Contatto. La relazione è rappresentata da un'associazione di tipo "uno-a-molti" (1 a *), dove una Rubrica può avere più Contatti. Tra le due classi vi è un **accoppiamento per dati**, dato che Rubrica non accede direttamente agli attributi interni di Contatto (i dettagli dell'implementazione di Contatto sono incapsulati nella classe stessa).

CLASSE INTERFACCIACONTATTO - CLASSE CONTATTO

Accoppiamento tra Contatto e View (di InterfacciaContatto)

La View di `InterfacciaContatto` accede ai dati del modello Contatto attraverso metodi specifici (es. i metodi getter come `getNome()`, ecc.), senza manipolare direttamente gli attributi interni della classe Contatto. Tra le due classi vi è un **accoppiamento per dati,** dato che la struttura interna della classe Contatto può cambiare senza influenzare la View, a condizione che i metodi pubblici rimangano invariati. Inoltre, essendo mediate dal **Controller,** la dipendenza tra View e **Contatto** è ulteriormente ridotta.

Accoppiamento tra Contatto e Controller (di InterfacciaContatto)

La classe Contatto è direttamente connessa al Controller di InterfacciaContatto. Tra le due classi vi è un **accoppiamento per controllo**, dato che il Controller funge da intermediario tra la View e il Model deve tradurre le azioni dell'utente (input) in operazioni sui dati del Model. Per farlo, il Controller deve inevitabilmente chiamare i metodi del Model per aggiornare, modificare o leggere i dati.
L'accoppiamento per controllo tra il Controller di InterfacciaContatto e il modello Contatto è inevitabile per la corretta implementazione del pattern MVC, ma è accettabile e ben gestibile finchè il Controller e il Model restano moduli distinti, con il Controller che utilizza solo i metodi pubblici di **Contatto**, e il Controller può essere modificato o sostituito senza cambiare la logica interna del Model.

CLASSE INTERFACCIAELENCO - CLASSE RUBRICA

Accoppiamento tra Rubrica e View (di InterfacciaElenco)

La View di InterfacciaElenco non è direttamente connessa alla classe Rubrica, in quanto vi è il Controller. La View di InterfacciaElenco accede ai dati dell'oggetto Rubrica tramite metodi pubblici che restituiscono i dati in un formato leggibile (ad esempio, una lista di contatti). Tra le due classi vi è un **accoppiamento per dati,** dato che la View non accede direttamente agli attributi interni della classe Rubrica.

Accoppiamento tra Rubrica e Controller (di InterfacciaElenco)

La classe Rubrica è direttamente connessa al Controller di InterfacciaElenco.
Tra le due classi vi è un **accoppiamento per controllo,** dato che il Controller funge da intermediario tra la View e il Model deve tradurre le azioni dell'utente (input) in operazioni sui dati del Model. Per

farlo, il Controller deve inevitabilmente chiamare i metodi del model per aggiornare, modificare o leggere i dati. Il Controller può essere modificato o sostituito senza influire sul model, a meno che non cambino le interfacce pubbliche.

CLASSE INTERFACCIAELENCO - CLASSE GESTOREFILE

Accoppiamento tra GestoreFile e View (di InterfacciaElenco)

La View di InterfacciaElenco non è direttamente connessa alla classe GestoreFile, in quanto la comunicazione avviene tramite il Controller di InterfacciaElenco. Tuttavia, la view può invocare indirettamente i metodi pubblici di GestoreFile (importaRubrica() ed esportaRubrica()) attraverso il Controller per visualizzare i risultati o notificare l'utente sullo stato delle operazioni. Tra la View di InterfacciaElenco e GestoreFile vi è un **accoppiamento per dati**, poiché la View si limita a ricevere i dati elaborati senza accedere direttamente alla logica interna o agli attributi di GestoreFile.

Accoppiamento tra GestoreFile e Controller (di InterfacciaElenco)

Il Controller di InterfacciaElenco è direttamente connesso alla classe GestoreFile. Tra il Controller di InterfacciaElenco e GestoreFile vi è un **accoppiamento per controllo**, dato che il Controller chiama esplicitamente i metodi pubblici di GestoreFile per realizzare le operazioni necessarie. Tuttavia, GestoreFile rimane incapsulato e non esiste una dipendenza diretta dalla View o da altre parti dell'InterfacciaElenco.

CLASSE INTERFACCIAELENCO - CLASSE INTERFACCIACONTATTO
Accoppiamento tra View (di InterfacciaElenco) e View (di InterfacciaContatto)

La View di InterfacciaElenco non è direttamente connessa alla View di InterfacciaContatto. Ogni View è responsabile della propria presentazione dei dati e non interagiscono direttamente. Eventuali dati visualizzati in InterfacciaElenco o InterfacciaContatto passano sempre tramite il rispettivo Controller, mantenendo le View isolate l'una dall'altra. Non c'è accoppiamento diretto tra le View.

Accoppiamento tra Controller (di InterfacciaElenco) e Controller (di InterfacciaContatto)

Il Controller di InterfacciaElenco interagisce con il Controller di InterfacciaContatto in modo indiretto. Tra le due classi vi è un **accoppiamento per controllo**, in quanto il Controller di InterfacciaElenco può delegare operazioni specifiche sui contatti al Controller di InterfacciaContatto. Tuttavia, questa relazione è mediata e ben incapsulata, riducendo il rischio di dipendenze eccessive.

5 Diagrammi di Sequenza

I diagrammi di sequenza sono uno strumento essenziale nella progettazione del software, poiché evidenziano l'ordine cronologico delle operazioni, mostrando come i messaggi vengono scambiati tra le diverse parti del sistema per realizzare una specifica funzionalità.
L'inserimento dei diagrammi di sequenza in questa fase del progetto ha l'obiettivo di:

- Chiarire il flusso delle operazioni.
- Mostrare come gli oggetti e le componenti del sistema collaborano per soddisfare i requisiti dei principali casi d'uso.
- Verificare la progettazione.

5.1 Sequenza di eventi per l’azione “start”

Il diagramma di sequenza per l’azione “start” rappresentato in *Figura 2*, illustra il flusso principale delle interazioni tra i componenti del sistema durante l'esecuzione dei casi d'uso più rilevanti. L'obiettivo è rappresentare come l'utente, attraverso l'interfaccia principale dell'applicazione, possa effettuare operazioni come:

- Visualizzare l'elenco dei contatti.
- Aggiungere, modificare o eliminare contatti.
- Importare o esportare la rubrica.

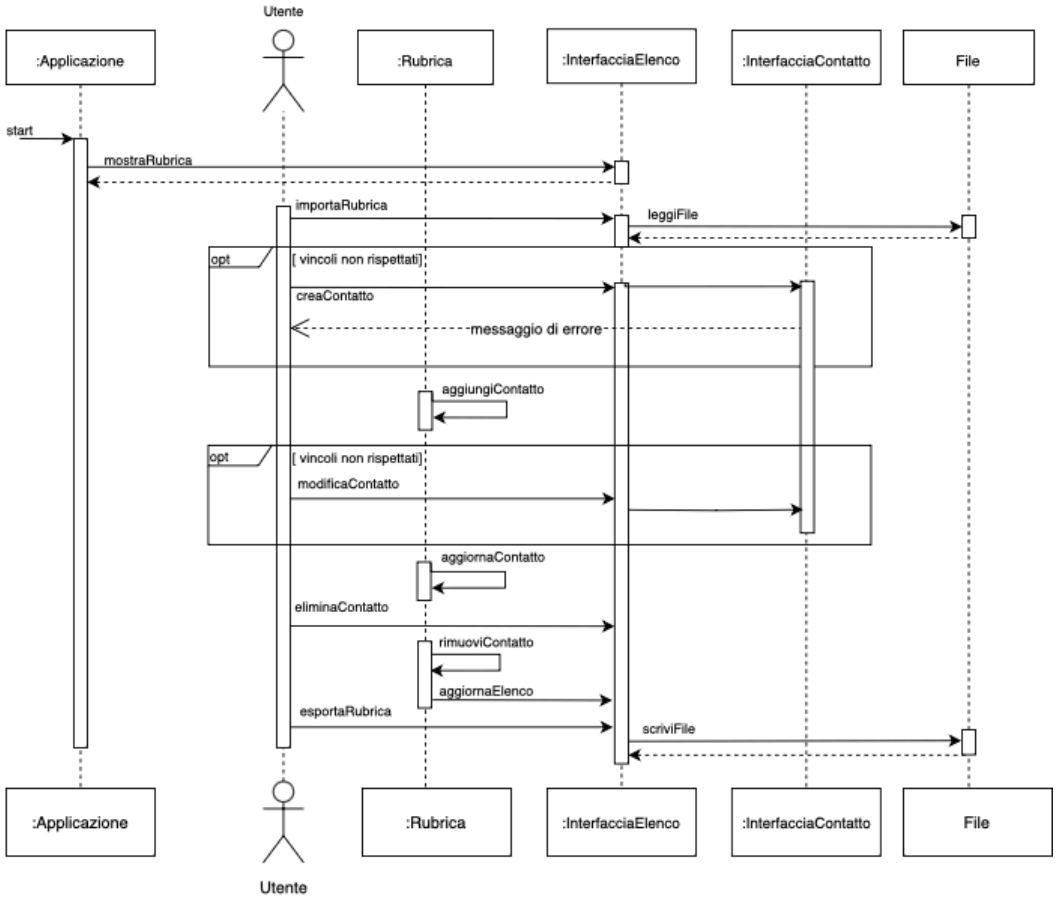


Figura 2: Diagramma di sequenza per l’azione “start”.

Nel diagramma si evidenziano i ruoli principali dei componenti del sistema: la classe *Rubrica* gestisce i dati, le *Interfacce* (Elenco e Contatto) facilitano l'interazione tra il modello e il resto del sistema, mentre il *GestoreFile* si occupa delle operazioni di lettura e scrittura su file. Gli eventi sono disposti cronologicamente per mostrare il flusso delle operazioni e l'ordine delle interazioni tra gli attori coinvolti.

5.2 Sequenza di eventi per l’azione “creaContatto”

Il diagramma in *Figura 3* mostra la sequenza di eventi dettagliata, necessaria per aggiungere un nuovo contatto alla rubrica.

Questo diagramma di sequenza descrive il flusso di interazioni tra i componenti durante il processo di creazione di un nuovo contatto nella rubrica. Esso mostra le fasi principali del caso d'uso, evidenziando:

- L'inserimento delle informazioni da parte dell'utente (nome,cognome, numero di telefono, e-mail).
- La verifica che il numero di telefono e l’indirizzo email siano in un formato valido
- La creazione dell'oggetto *Contatto* e l'aggiornamento del sistema tramite la classe *Rubrica*.
- Il ritorno alla schermata principale al termine dell'operazione.

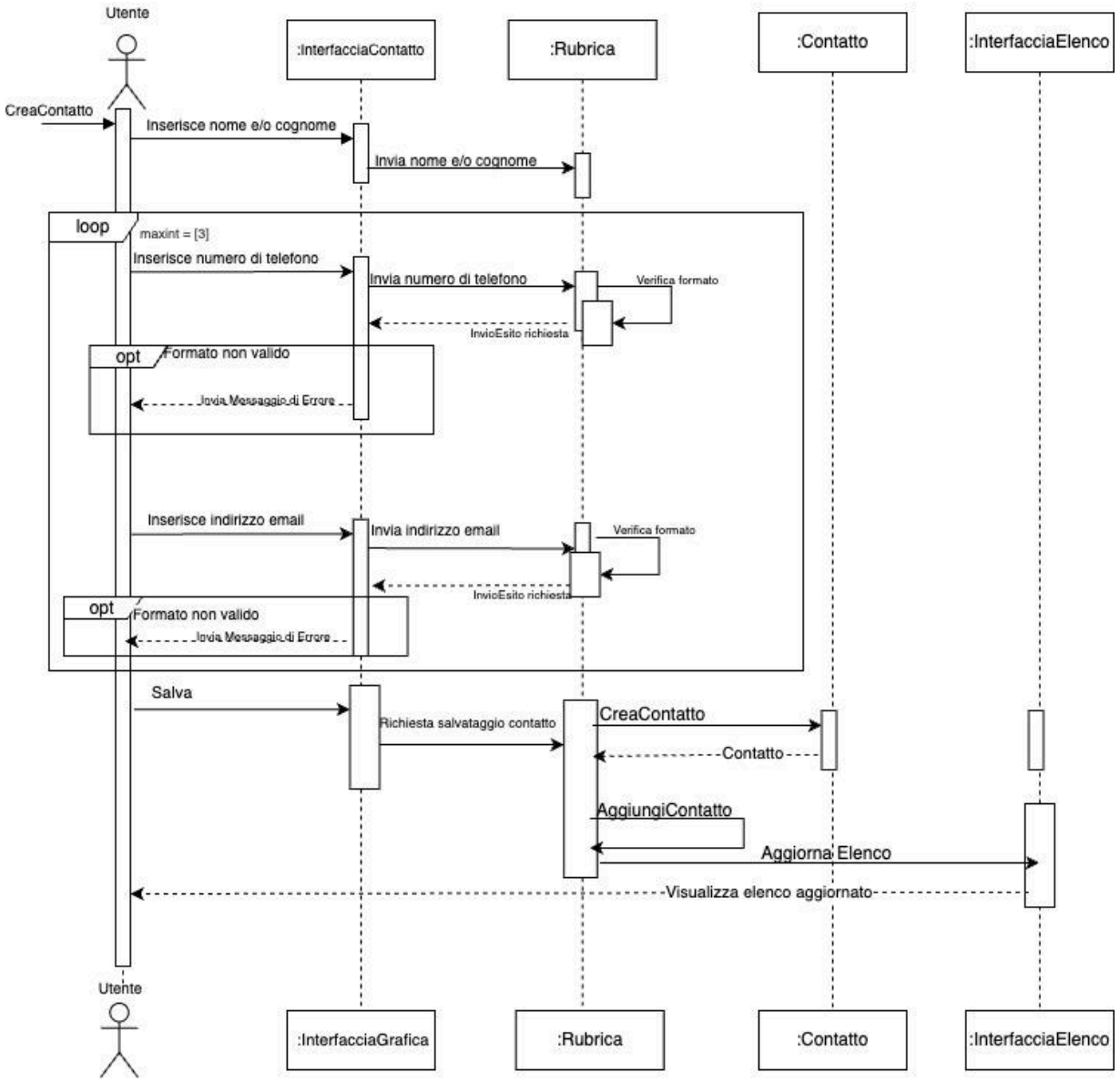
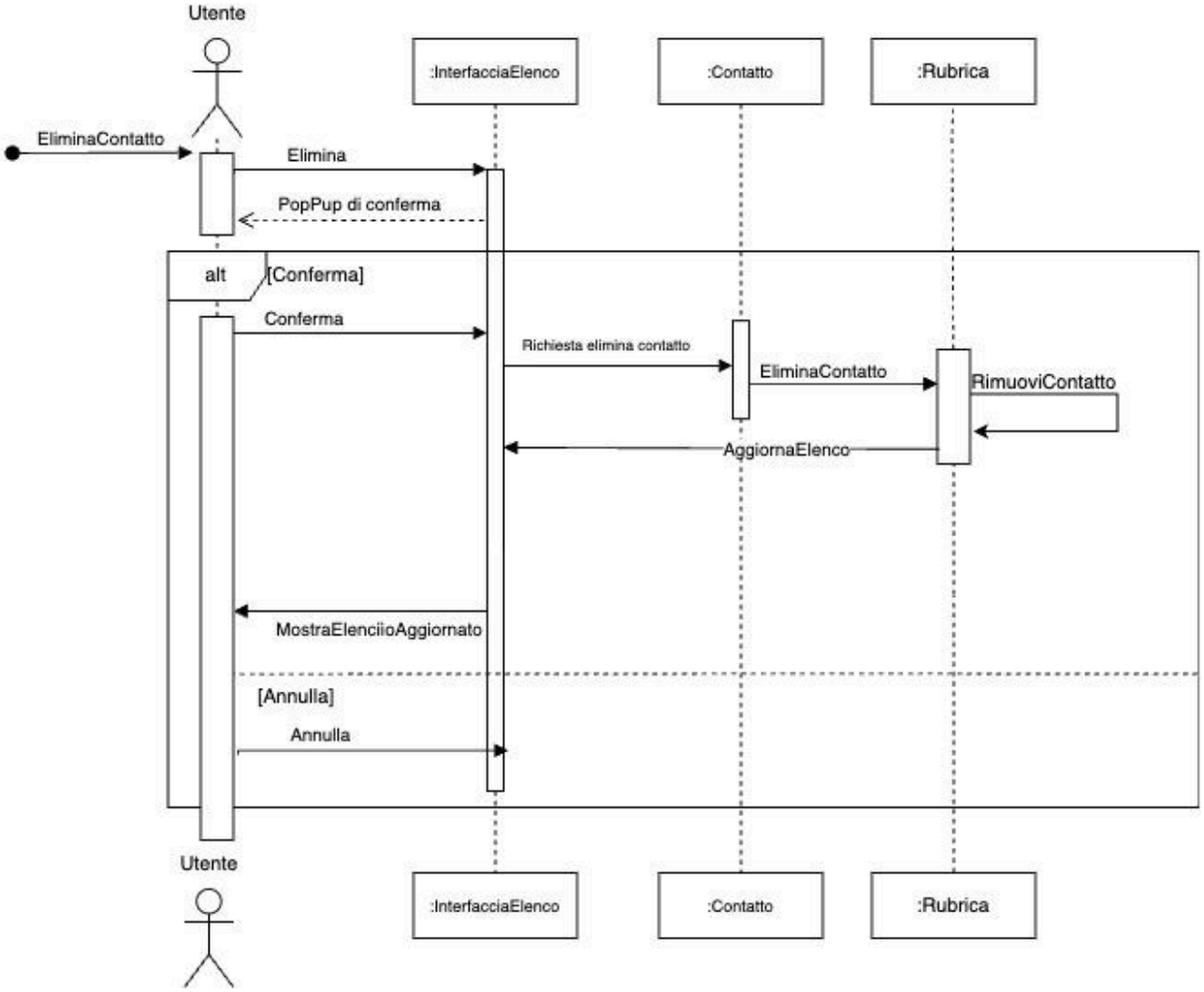


Figura 3: Diagramma di sequenza per l'azione "creaContatto".

5.3 Sequenza di eventi per l'azione "eliminaContatto"

Il diagramma in *Figura 4* mostra il flusso dettagliato delle interazioni necessarie per eliminare un contatto dalla rubrica. Questo diagramma di sequenza descrive la sequenza di eventi tra i componenti durante il processo di eliminazione di un contatto dalla rubrica, evidenziando:

- L'eliminazione del contatto selezionato da parte dell'utente.
- L'invio del messaggio di conferma dell'operazione.
- La rimozione del contatto dall'elenco dei contatti.
- La visualizzazione dell'elenco aggiornato al termine dell'operazione.



- L'annullamento dell'operazione di eliminazione di un contatto.

Figura 4: Diagramma di sequenza per l'azione "eliminaContatto".

5.4 Sequenza di eventi per l'azione "modificaContatto"

Il diagramma in *Figura 5* rappresenta il flusso dettagliato di interazioni tra l'utente e le principali componenti del sistema durante l'operazione di modifica di un contatto. Il flusso è costituito dalla seguente sequenza di eventi:

- La verifica dell'esistenza del contatto tramite il numero di telefono fornito.
- La modifica di uno o più attributi del contatto tramite *l'InterfacciaContatto.*
- L'aggiornamento dell'elenco dei contatti per riflettere i cambiamenti effettuati.

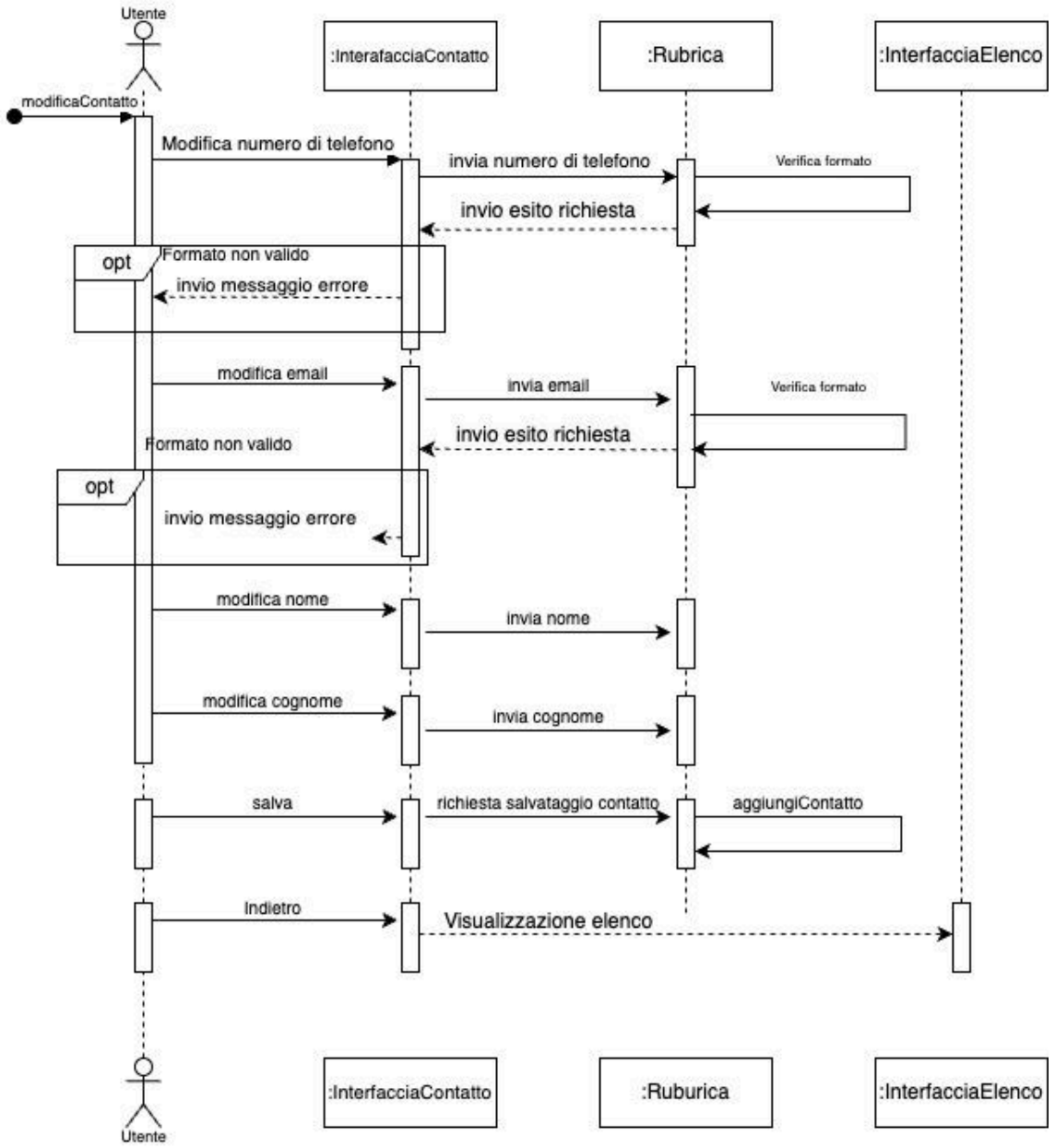


Figura 5: Diagramma di sequenza per l'azione "modificaContatto".

5.5 Sequenza di eventi per l'azione "cercaContatto"

Il diagramma illustrato in *Figura 6*, evidenzia in maniera dettagliata il flusso di eventi per l'azione "cercaContatto". La sequenza di eventi si struttura come segue:

- *InterfacciaElenco* invia una richiesta di ricerca a *Rubrica* con la stringa/sottostringa..
- *Rubrica* verifica se ci sono contatti corrispondenti:
    - Se ci sono contatti, ritorna una lista filtrata e viene visualizzata.
    - Altrimenti, ritorna nessuna corrispondenza e si visualizza il messaggio "nessun contatto trovato"
- Quando la stringa/sottostringa viene cancellata, il sistema mostra la lista completa.

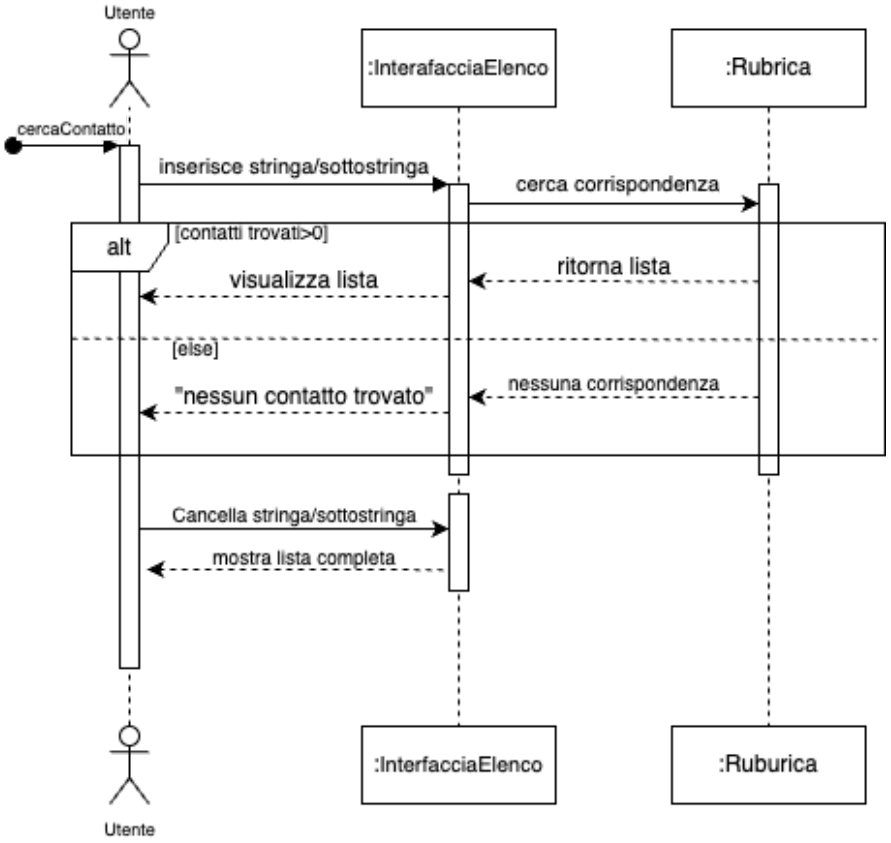


Figura 6: Diagramma di sequenza per l'azione "cercaContatto".

5.6 Sequenza di eventi per l'azione "importa"

Il diagramma in *Figura 7*, mostra la sequenza dettagliata di eventi necessari per l'azione "importa". Il flusso è volto ad evidenziare le seguenti interazioni:

- L'utente avvia il processo, premendo il pulsante "importa" da *InterfacciaElenco.*
- *InterfacciaElenco* mostra una finestra per selezionare il percorso e comunica alla classe *Rubrica* il file scelto.
- *Rubrica* inoltra una richiesta di lettura a *GestoreFile*.
- La classe *GestoreFile* legge il file e verifica il formato.
    - Se il formato del file è **valido**, la lista dei contatti viene ritornata dalla classe *Rubrica* e mostrata nell'interfaccia.

- Se il formato **non è valido**, viene mostrato un messaggio di errore all'utente tramite l'interfaccia.

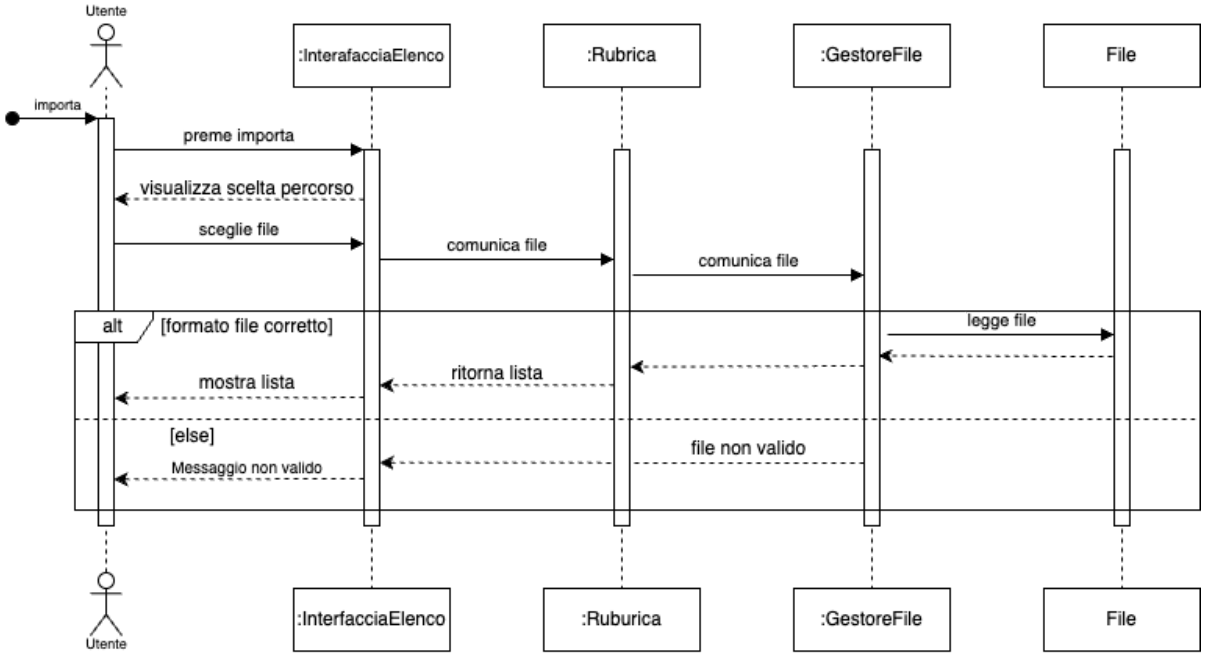


Figura 7: Diagramma di sequenza per l'azione "importa".

5.7 Sequenza di eventi per l'azione "esporta"

Il diagramma in *Figura 8* mostra il flusso delle interazioni per l'azione "importa", evidenziando i seguenti passaggi principali:

- L'utente avvia l'operazione, premendo il pulsante "esporta" da *InterfacciaElenco*.
- *InterfacciaElenco* mostra una finestra per selezionare il percorso e comunica alla classe *Rubrica* il file scelto.
- *Rubrica* inoltra una richiesta di scrittura a *GestoreFile*.
- GestoreFile scrive il file e termina l'operazione.

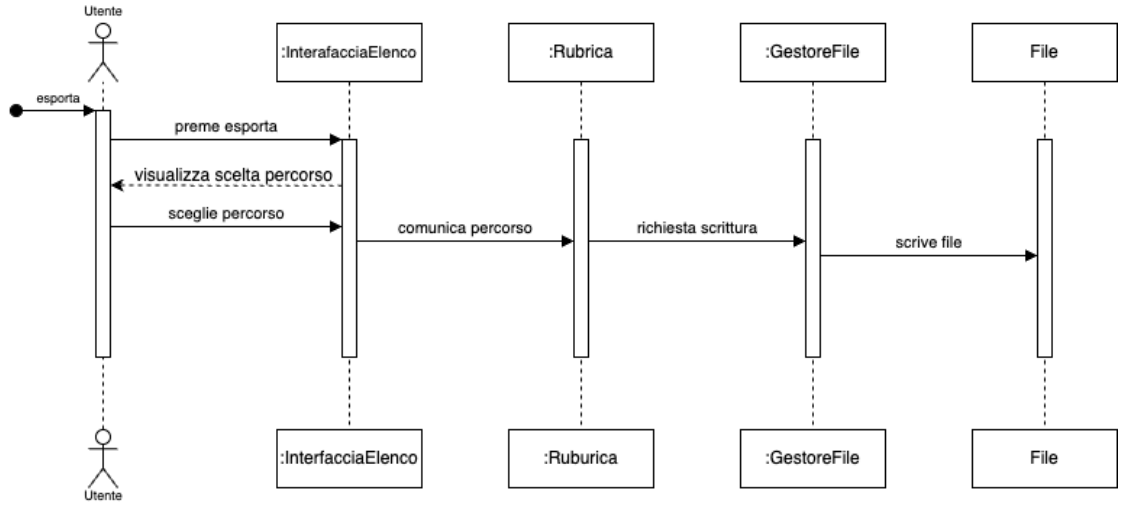


Figura 8: Diagramma di sequenza per l'azione "esporta".

5.8 Sequenza di eventi per l'azione "marcaContattoComePreferito"

Il diagramma illustrato in *Figura 9* rappresenta la sequenza di eventi per l'azione **"marcaContattoComePreferito"**, che coinvolge i seguenti passi principali:

- L'utente interagisce con la visualizzazione del dettaglio di un contatto nell' *InterfacciaElenco* per avviare l'azione.
- *InterfacciaContatto* invia una richiesta alla classe *Contatto* per marcare il contatto come preferito.
- *Contatto* registra il contatto come preferito e notifica *InterfacciaElenco* della modifica.
- *InterfacciaElenco* aggiorna la visualizzazione, riflettendo il cambiamento dello stato del contatto.

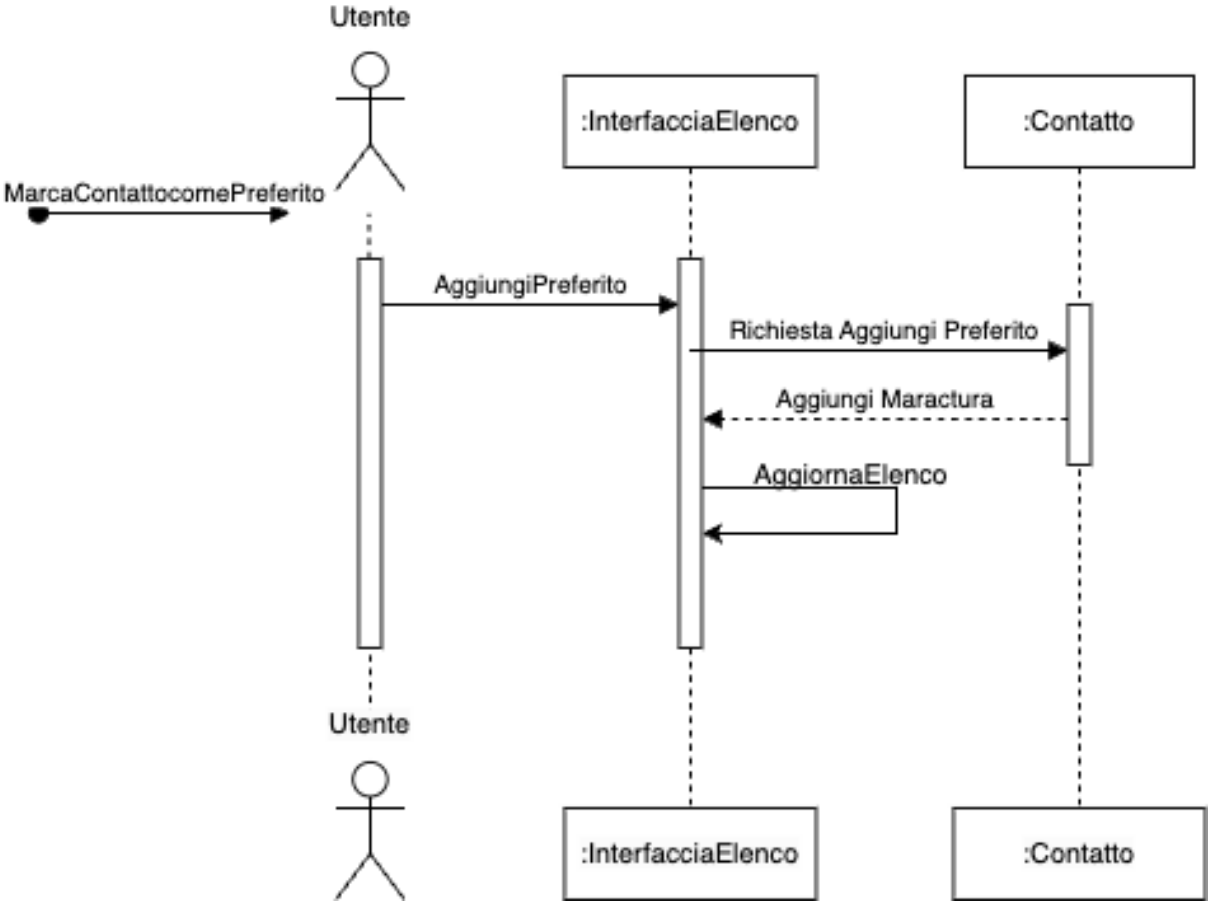


Figura 9: Diagramma di sequenza per l’azione "marcaContattoComePreferito".

6 Diagramma dei Componenti

Fin dalle prime fasi della progettazione, durante la definizione del diagramma delle classi, è stato chiarito che le classi InterfacciaElenco e InterfacciaContatto sarebbero state implementate seguendo il pattern MVC (Model-View-Controller). Queste classi, nella loro descrizione iniziale, combinavano i ruoli di View e Controller per semplificare la verifica delle funzionalità principali del sistema.

In questo diagramma, le precedenti interfacce sono state suddivise nei rispettivi componenti View e Controller. Questo approccio consente di rappresentare in modo più fedele l'architettura del sistema e i flussi di comunicazione tra i diversi moduli. In particolare, le View si occupano di interagire con l'utente, mostrando i dati recuperati dal Model, mentre i Controller elaborano le richieste dell'utente e interagiscono con il Model per aggiornare o manipolare i dati. Tale suddivisione garantisce una progettazione più robusta, flessibile e conforme ai principi del pattern MVC.

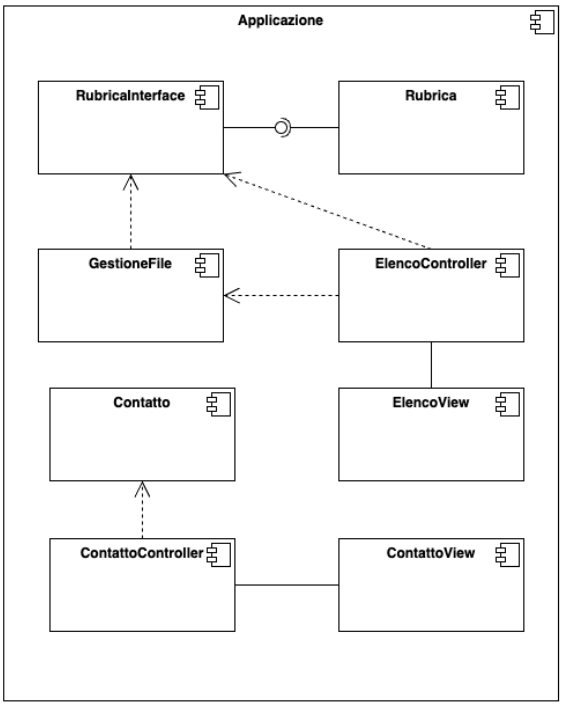


Figura 10: Diagramma dei componenti.

Con l'introduzione della suddivisione in View e Controller nel diagramma dei componenti, i seguenti componenti sono stati aggiunti per riflettere meglio la separazione delle responsabilità e la gestione delle operazioni secondo il pattern MVC. Ogni componente ha un ruolo specifico nel sistema, come descritto di seguito:

RubricaInterface

La RubricaInterface funge da interfaccia tra l'utente e la logica sottostante della rubrica. La RubricaInterface comunica con la classe Rubrica, che gestisce la logica operativa dei contatti, e con altre componenti tramite i controller e le view. L'implementazione di tale classe consente una maggiore flessibilità nel sistema, poiché è possibile cambiare o aggiornare la logica sottostante (come la gestione dei contatti) senza modificare l'interfaccia utente o la gestione dell'interazione con l'utente.

ElencoController

ElencoController implementa la logica che coordina le operazioni sul modello Rubrica e aggiorna la vista ElencoView. La sua funzione è centralizzare la logica di interazione, mantenendo separata la gestione dei dati (che spetta alla Rubrica) dalla logica di visualizzazione (che è delegata alla ElencoView); funge da intermediario tra la view e il model, attuando le azioni richieste dall'utente e assicurando che i dati siano correttamente aggiornati nel sistema.

ElencoView

La classe ElencoView è responsabile della visualizzazione dell'elenco dei contatti e dell'interfaccia grafica per l'utente. Gestisce la presentazione dei dati relativi ai contatti, come il loro nome, numero di telefono, email, ecc. Inoltre, ElencoView si occupa della gestione degli eventi di interazione da parte dell'utente , ma non contiene la logica per l'elaborazione o la modifica dei dati.

ContattoController

La classe ContattoController è responsabile di ricevere gli input dell'utente per aggiornare i dati di un contatto esistente, ad esempio modificando il nome, il numero di telefono o l'email. Il controller si occupa di invocare i metodi adeguati nel modello Contatto per eseguire le operazioni richieste, e di inviare i dati aggiornati alla vista ContattoView. Dunque funge da intermediario tra l'utente e il modello Contatto.

ContattoView

La classe ContattoView è responsabile della presentazione delle informazioni di un singolo contatto all'utente. Mostra i dettagli di un contatto specifico (nome, cognome, numero di telefono, email) e permette all'utente di visualizzare e modificare questi dati tramite l'interfaccia grafica. Si occupa esclusivamente della presentazione dei dati del contatto.

7 Diagramma dei Package

Le classi con funzioni simili o correlate sono collocate nello stesso package, favorendo così un'organizzazione chiara e strutturata del codice. Il diagramma dei package in *Figura 11* mette in evidenza le relazioni e le dipendenze tra i diversi package, facilitando la comprensione dell'architettura del sistema.

- **Package**: `it.unisa.diem.progettoinf.gruppo25.app`
  Contiene la classe principale dell'applicazione, responsabile dell'avvio e della gestione globale del sistema.

  Dipendenze:
  1) `it.unisa.diem.progettoinf.gruppo25.model`: Per accedere alle classi di dominio durante l'inizializzazione.
  2) `it.unisa.diem.progettoinf.gruppo25.controller`: Per coordinare la logica applicativa iniziale e invocare i controller.

- **Package**: `it.unisa.diem.progettoinf.gruppo25.model`
  Include le classi che rappresentano il dominio dell'applicazione, come la gestione della rubrica, i contatti e i file.
  Dipendenze: Non dipende da nessun altro package.

- **Package**: `it.unisa.diem.progettoinf.gruppo25.controller`
  Contiene le classi che gestiscono la logica applicativa e coordinano le interazioni tra il model e la view.
  Dipendenze:
  1) `it.unisa.diem.progettoinf.gruppo25.model`: Per eseguire operazioni sui dati (es. aggiunta, modifica, eliminazione di contatti).

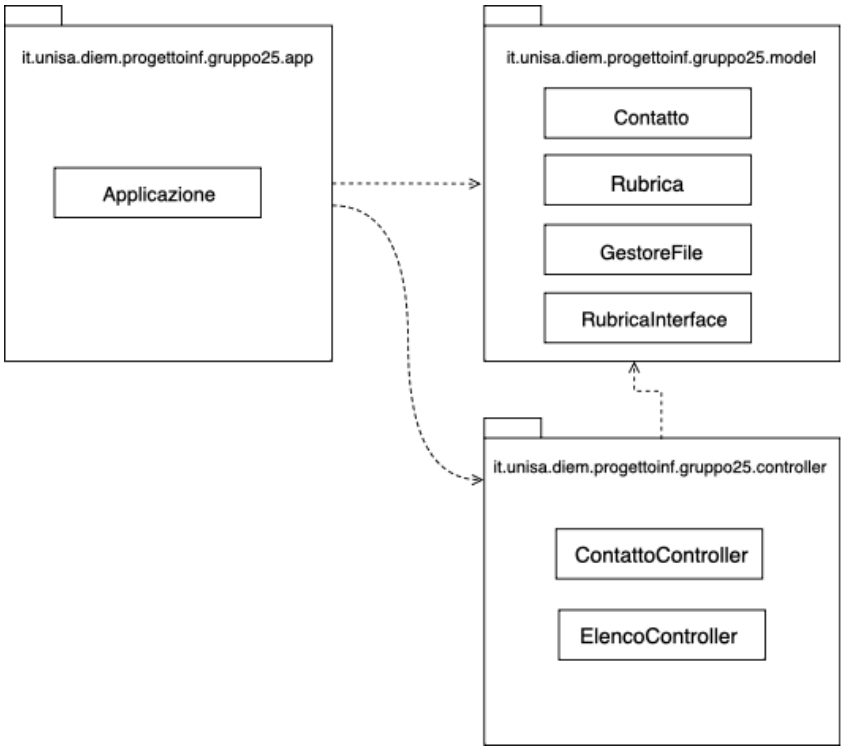


Figura 11: Diagramma dei package.

8 Diagramma implementativo delle classi

Sulla base delle elaborazioni effettuate nei precedenti step di progettazione si è elaborato il seguente diagramma implementativo delle classi.
Il diagramma in *Figura 12* rappresenta la struttura delle classi che compongono il sistema. Esso evidenzia le relazioni tra le classi, come l'ereditarietà, l'associazione, la dipendenza e l'aggregazione, delineando le responsabilità di ciascuna e come interagiranno tra loro.

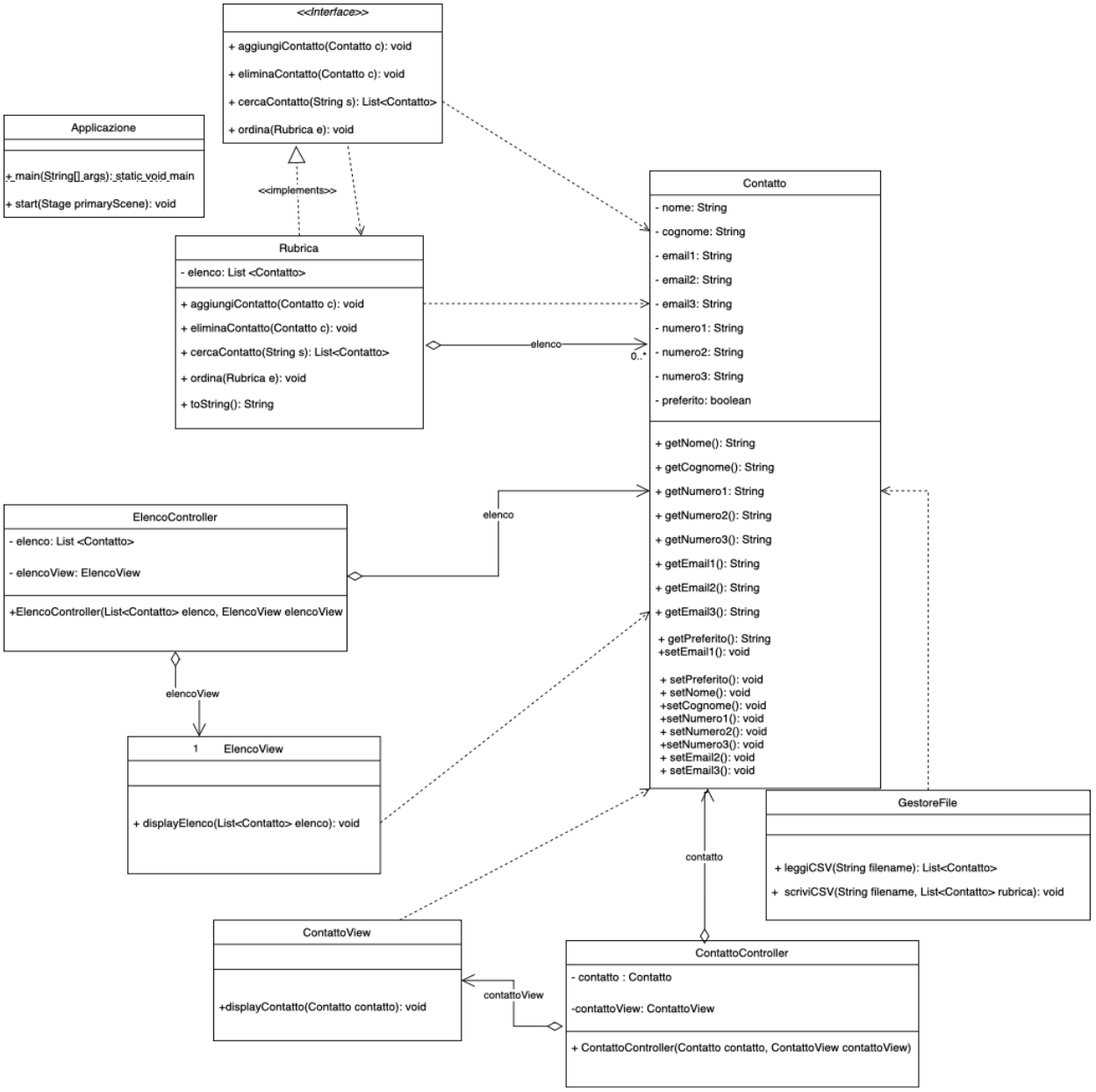


Figura 12: Diagramma implementativo delle classi.