

Applicazione Software di una Rubrica

Andrea Savastano
Annamaria Scermino
Alessandro Monte
Francesco Musto

December 15, 2024

Contents

Descrizione introduttiva	4
Mockup	4
1 Ingegneria dei Requisiti - SRS	5
1.1 Tabella di categorizzazione dei requisiti	5
1.2 Requisiti Funzionali	6
1.3 Requisiti Non Funzionali	8
1.4 Diagramma dei Casi d'Uso	9
1.5 Casi d'Uso formato testuale	10
2 Design	14
2.1 Diagrammi di classi	14
2.1.1 Livello di dettaglio basso	15
2.1.2 Livello di dettaglio alto	16
2.2 Diagrammi di sequenza	17
2.2.1 C1 - Aggiungere Contatto	17
2.2.2 C2 C3 - Eliminare Modificare Contatto	18
2.2.3 C5 - Importare rubrica	19
2.2.4 C6 - Esportare rubrica	20
2.2.5 C8 - Salvare rubrica	21
2.2.6 Aggiunta DB	22
2.2.7 Inizializzazione rubrica	23
2.3 Principi di buona progettazione	24
2.3.1 Livelli di coesione	24
2.3.2 Livelli di accoppiamento	25
2.3.3 Relazioni tra le Classi	25
2.3.4 Riduzione dell'Accoppiamento	25
2.3.5 Relazione con l'Interfaccia Initializable	26
2.3.6 Livelli di dettaglio	26
2.3.7 Diagrammi di Sequenza	26
3 Implementazione e Testing	27
3.1 Diagramma dei Package	27
3.2 UTC 1 - AddressBook	29
3.3 UTC 2 - Contact	32
3.4 UTC 3 - Converter	32
3.5 UTC 4 - Database	35
3.6 UTC 5 - Tag	37

List of Figures

1	Mockup	4
2	Tabella di categorizzazione dei requisiti	5
3	Diagramma UML dei Casi d'uso	9
4	Diagramma classi essenziale	15
5	Diagramma classi completo	16
6	Diagramma sequenza C1 - Aggiungere Contatto	17
7	Diagramma sequenza C2 C3 - Eliminare Modificare Contatto	18
8	Diagramma sequenza C5 - Importare rubrica	19

9	Diagramma sequenza C6 - Esportare rubrica	20
10	Diagramma sequenza C8 - Salvare rubrica	21
11	Diagramma sequenza Aggiunta DB	22
12	Diagramma sequenza Inizializzazione rubrica	23
13	Livelli di coesione classi	24
14	Livelli di accoppiamento classi	25
15	Diagramma package implementazione	27
16	Diagramma package testing	27

Descrizione introduttiva

Si vuole realizzare un software destinato a gestire un insieme di contatti, associando ad ognuno di essi le informazioni personali quali nome, cognome, numero di telefono, e-mail.

Si rende disponibile un'interfaccia grafica che permette all'utente di usufruire delle funzionalità offerte dal sistema.

Mockup

Si inseriscono alcuni mockup dell'interfaccia grafica del software per scopo illustrativo, finalizzati ad una stesura dei requisiti più dettagliata e ad un supporto alla comprensione dei requisiti.

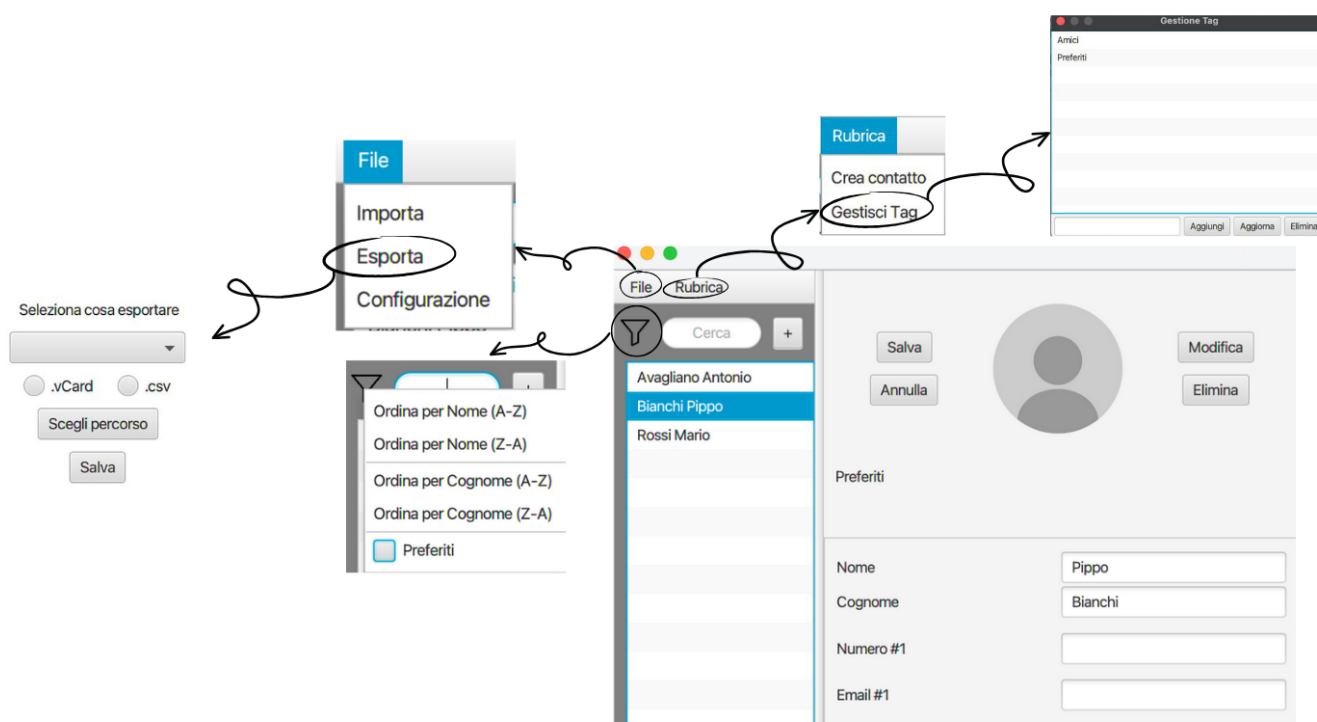


Figure 1: Mockup

1 Ingegneria dei Requisiti - SRS

1.1 Tabella di categorizzazione dei requisiti


Prefisso	T _T	Requisito	⌚	Priorità	⌚	Categoria	T _T	Note
IF-1		Visualizzazione in ordine alfabetico		Alta		Funzionalità individuali		La visualizzazione dell'elenco dei contatti è sempre in ordine alfabetico, per cognome e per nome.
IF-2		Ricerca del contatto		Alta		Funzionalità individuali		Ricerca un contatto nella rubrica specificando una sottostringa del nome o del cognome del contatto interessato.
IF-3		Ricerca del contatto per numero di telefono o mail		Medio		Funzionalità individuali		Ricerca un contatto nella rubrica specificando un prefisso del numero di telefono o una sottostringa della mail del contatto interessato.
IF-4		Aggiunta del contatto		Alta		Funzionalità individuali		Aggiunta di un contatto nella rubrica, specificandone i dati rispettando i vincoli di validità indicati dal requisito "Campi del contatto".
IF-5		Modifica di un contatto		Alta		Funzionalità individuali		Modifica di 1 o più campi di un contatto presente in rubrica.
IF-6		Eliminazione di un contatto		Alta		Funzionalità individuali		Rimuovere un contatto presente in rubrica.
IF-7		Importazione della rubrica da file		Alta		Funzionalità individuali		Possibilità di importare una rubrica da file con formato .csv o .vcf i cui contatti verranno aggiunti alla rubrica dell'utente.
IF-8		Esportazione della rubrica su file		Alta		Funzionalità individuali		Possibilità di esportare la rubrica su file con formato .csv o .vcf.
IF-9		Visualizzazione per tag		Bassa		Funzionalità individuali		Selezionando uno dei tag creati verrà visualizzato l'elenco dei contatti associati a tale tag.
DF-1		Campi del contatto		Alta		Esigenze di dati e informazioni		Conservare una serie di informazioni per ogni contatto.
DF-2		Categorizzazione del contatto		Bassa		Esigenze di dati e informazioni		Durante la fase di aggiunta/modifica di un contatto è possibile creare dei tag per categorizzarli.
DF-3		Immagine del contatto		Bassa		Esigenze di dati e informazioni		Possibilità di associare un'immagine ad un contatto in fase di aggiunta/modifica di un contatto.
DF-4		Salvataggio in locale		Alta		Esigenze di dati e informazioni		Salvataggio in locale di ciascun contatto inserito in rubrica.
UI-1		Interfaccia utente di tipo grafico		Alta		Interfaccia Utente		Utilizzare la rubrica dei contatti tramite un'interfaccia grafica.
UI-2		Visualizzazione specifica del contatto		Medio		Interfaccia Utente		Possibilità di visualizzare i campi associati ad un singolo contatto.
IS-1		Sincronizzazione con database		Bassa		Interfaccia con Sistemi Esterni		Il database deve conservare i dati dell'utente, ossia l'insieme dei contatti aggiunti nella rubrica.

Figure 2: Tabella di categorizzazione dei requisiti

1.2 Requisiti Funzionali

Funzionalità individuali IF

IF-1 Visualizzazione in ordine alfabetico

I contatti sono sempre visualizzati in ordine alfabetico, ordinandoli per cognome e nome. Si offre la possibilità di poter scegliere, cliccando sull'icona dell'imbuto , tra l'ordinamento A-Z oppure Z-A rispetto al nome o al cognome.

Anche usufruendo della visualizzazione per Tag l'ordine dei contatti è sempre alfabetico (vedi IF-9).

IF-2 Ricerca del contatto

L'utente può cercare il contatto desiderato inserendo nella casella di ricerca una sottostringa del nome o del cognome del contatto desiderato.

Verranno visualizzati, secondo l'ordine alfabetico stabilito in IF-1, i contatti che rispettano il requisito di ricerca.

IF-3 Ricerca per numero di telefono o mail

L'utente può cercare il contatto desiderato inserendo nella casella di ricerca un suffisso del numero di telefono o una sottostringa della mail associati al contatto.

Verranno visualizzati, secondo l'ordine alfabetico stabilito in IF-1, i contatti che rispettano il requisito di ricerca.

IF-4 Aggiunta del contatto

L'utente ha la possibilità di aggiungere un contatto alla sua rubrica.

In particolare cliccando sul simbolo “+” oppure attraverso il menù a tendina “Rubrica”, compare una schermata in cui è possibile inserire i campi del contatto (definiti in DF-1, DF-2, DF-3). Successivamente si può salvare il nuovo contatto o annullare l'operazione con i rispettivi pulsanti “Salva” o “Annulla”.

E' possibile aggiungere un nuovo contatto contenente gli stessi valori dei campi di 1 o più contatti già esistenti.

IF-5 Modifica di un contatto

L'utente ha la possibilità di modificare un contatto dalla sua rubrica.

Cliccando su un contatto e premendo sul pulsante “Modifica” è possibile modificare i suoi campi (definiti in DF-1, DF-2, DF-3).

Dopo aver effettuato modifiche è possibile annullare o salvare l'operazione con i pulsanti “Salva” o “Annulla”.

IF-6 Eliminazione di un contatto

L'utente ha la possibilità di eliminare un contatto dalla sua rubrica.

In particolare cliccando su un contatto e premendo sul pulsante “Elimina”, l'utente potrà confermare o meno l'operazione attraverso una schermata di conferma.

IF-7 Importazione della rubrica da file

L'utente può scegliere, tramite la sezione del menù a tendina “File”, di importare una rubrica con 0 o più contatti da un file .csv o .vcf.

Gli eventuali contatti presenti nella rubrica importata verranno aggiunti alla rubrica dell'utente.


L'utente può interrompere l'operazione in qualsiasi momento.

IF-8 Esportazione della rubrica su file

L'utente può scegliere, tramite la sezione del menù a tendina “File”, di esportare la sua rubrica, includendo tutti i contatti oppure quelli associati ad un particolare Tag selezionato da una lista che uscirà al momento dell'esportazione.

L'utente può scegliere di esportare la rubrica in un file `.csv` o `.vcf` e specificare il nome del file.

IF-9 Visualizzazione per tag

L'utente cliccando sull'icona dell'imbuto  può scegliere 1 o più Tag tra quelli presenti che comporta la visualizzazione dei soli contatti associati al/ai Tag selezionato/i.

La visualizzazione segue i criteri definiti in [IF-1](#).

Esigenze dei dati e informazioni DF

DF-1 Campi del contatto

Per ogni contatto bisogna conservare i seguenti dati:

- cognome e/o nome;
- da 0 a 3 numeri di telefono;
- da 0 a 3 mail.

Tali dati possono essere aggiunti in fase di creazione del contatto e modificati in qualsiasi momento.

DF-2 Categorizzazione contatto

L'utente può aggiungere facoltativamente un ulteriore campo ossia un Tag.

In particolare l'utente può inserire da 1 a 3 Tag a piacere (ad esempio preferiti, famiglia, lavoro, ...) ad ogni contatto in fase di aggiunta/modifica del contatto.

Un Tag è una particolare proprietà che si può associare ad 1 o più contatti.

Attraverso il menù a tendina "Rubrica", cliccando su "Gestisci Tag", l'utente può visualizzare tutti i Tag inseriti precedentemente, e può aggiungerli, rimuoverli o modificarli.

Per la visualizzazione dei contatti appartenenti ai Tag vedi [IF-9](#).

DF-3 Immagine contatto

Ad ogni contatto è associato un ulteriore campo ossia un'immagine. In particolare una sagoma grigia che si vedrà nella visione dettagliata del contatto.

Ma in fase di aggiunta/modifica di un contatto, l'utente può aggiungere un'immagine personalizzata, la quale può essere selezionata tra quelle suggerite dall'applicazione stessa oppure importata dall'esterno.

DF-4 Salvataggio in locale

Salvataggio in locale dei contatti inseriti in rubrica nel file binario `Data.bin`. Quest'ultimo file offre la possibilità all'utente di conservare i contatti, e i relativi campi (vedi [DF-1](#), [DF-2](#), [DF-3](#)), in modo che ogniqualvolta che riapre l'applicazione della rubrica visualizza gli stessi contatti della sessione precedente.

Tale salvataggio dei contatti in locale è quello di default se non viene specificata la preferenza di usare un database (vedi [IS-1](#)).

Oltre al file `Data.bin`, viene salvato in locale il file binario `Config.bin` il quale mantiene le impostazioni dell'applicazione, come ad esempio il link del database se questo viene fornito dall'utente.

Interfaccia Utente UI

UI-1 Avere interfaccia utente di tipo grafico

Tramite l'utilizzo di JavaFX l'utente interagisce con il programma tramite interfaccia grafica, permettendone un utilizzo facilitato e maggiormente intuitivo.

UI-2 Visualizzazione specifica del contatto

Dopo aver selezionato dalla rubrica un contatto, l'utente vedrà la visualizzazione dettagliata del contatto scelto.

In questa sezione sono visibili i campi definiti in [DF-1](#), [DF-2](#), [DF-3](#).

Interfacce con sistemi esterni IS

IS-1 Sincronizzazione con database

Cliccando sul menù a tendina "File" e poi su "Configurazione" è possibile esprimere la propria preferenza riguardo la possibilità di salvare la propria rubrica su un database esterno, fornendo il link.

In questo caso, il salvataggio dati non avverrà più in locale, come avviene di default tramite file `Data.bin` (vedi [DF-4](#)), ma sul database.

1.3 Requisiti Non Funzionali

Il software non prevede requisiti non funzionali

1.4 Diagramma dei Casi d'Uso

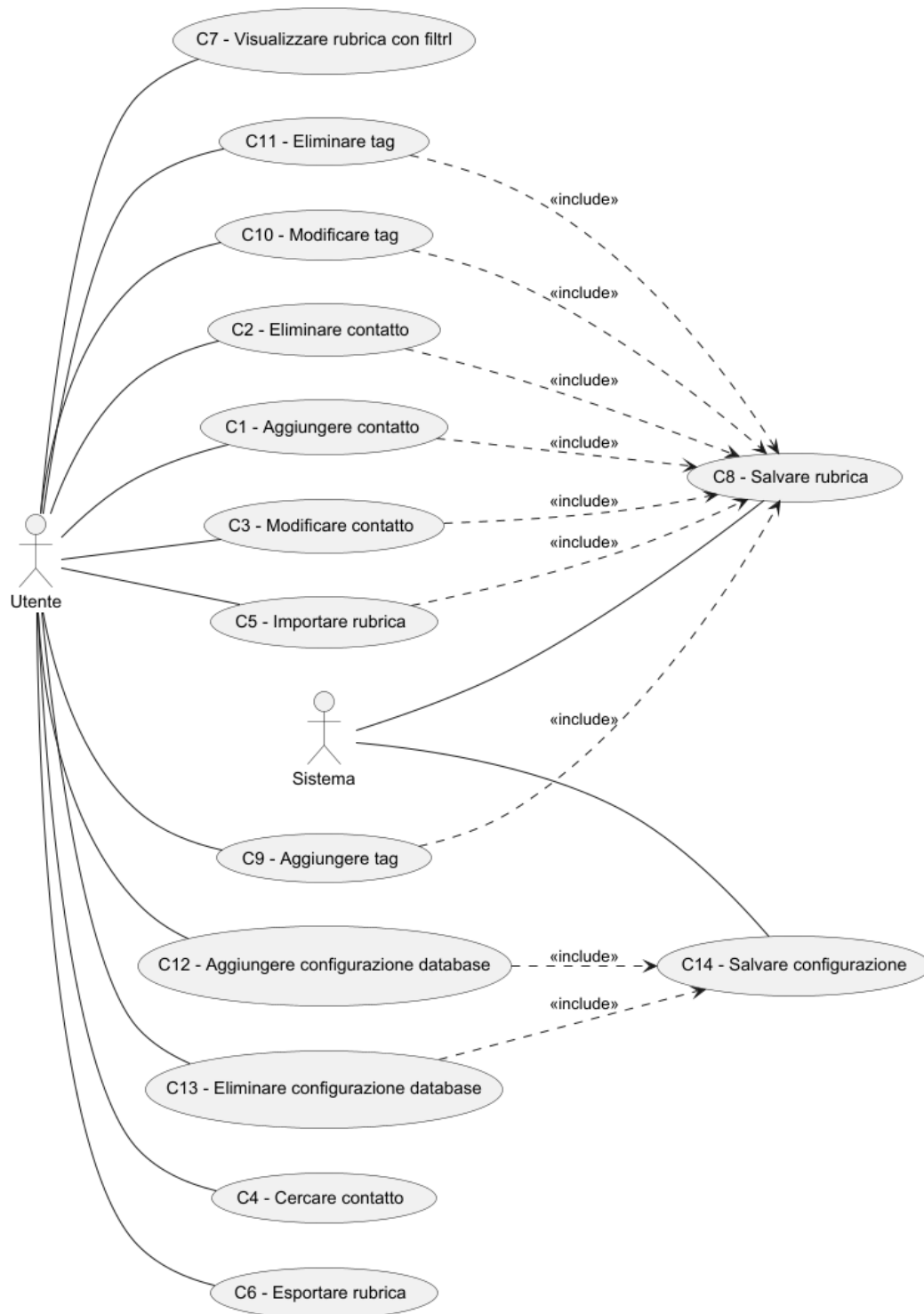


Figure 3: Diagramma UML dei Casi d'uso

1.5 Casi d'Uso formato testuale

C1 - Aggiungere contatto

Attore partecipante: Utente

Precondizioni: Utente ha la schermata della rubrica aperta.

Postcondizioni: Nella rubrica viene aggiunto un contatto.

Flusso di eventi normale:

1. Utente clicca il pulsante “+”;
2. Utente inserisce il nome e/o il cognome;
3. Utente inserisce da 0 a 3 numeri di telefono;
4. Utente inserisce da 0 a 3 mail ;
5. Utente inserisce da 0 a 3 Tag;
6. Utente inserisce un'immagine;
7. Utente salva l'operazione;

Flusso di eventi alternativo:

- 1a. Utente clicca il menù a tendina “Rubrica”;
- 1a.1 Utente crea il contatto;
- 6a. Utente non inserisce alcuna immagine, quindi rimane quella di default;
- 7a. Utente annulla l'operazione;
- 7a.1 L'esecuzione riprende dal passo 1;

C2 - Eliminare contatto

Attore partecipante: Utente

Precondizioni: Esiste almeno un contatto nella rubrica.

Postcondizioni: Viene rimosso dalla rubrica il contatto selezionato.

Flusso di eventi normale:

1. Utente seleziona il contatto da eliminare;
2. Utente clicca il pulsante “Elimina”;
3. Utente conferma l'operazione dalla schermata di conferma;

Flusso di eventi alternativo:

- 3a. Utente annulla l'operazione dalla schermata di conferma;
- 3a.1 L'esecuzione riprende dal passo 1;

C3 - Modificare contatto

Attore partecipante: Utente

Precondizioni: Esiste almeno un contatto nella rubrica.

Postcondizioni: Viene modificato il contatto selezionato.

Flusso di eventi normale:

1. Utente seleziona il contatto da modificare;
2. Utente clicca il pulsante “Modifica”;
3. Utente modifica uno o più campi del contatto;
4. Utente salva l'operazione;

Flusso di eventi alternativo:

- 4a. Utente annulla l'operazione;
- 4a.1 L'esecuzione riprende dal passo 1;

C4 - Cercare contatto

Attore partecipante: Utente

Precondizioni: Utente ha la schermata della rubrica aperta.

Postcondizioni: I contatti che rispettano il criterio di ricerca vengono visualizzati.

Flusso di eventi normale:

1. Utente scrive nella casella di ricerca una sottostringa del nome o del cognome del contatto da cercare;

Flusso di eventi alternativo:

- 1a. Utente scrive nella casella di ricerca un prefisso del numero di tel. del contatto da cercare;
- 1b. Utente scrive nella casella di ricerca una sottostringa della mail del contatto da cercare;

C5 - Importare rubrica

Attore partecipante: Utente

Precondizioni: Utente possiede un file .csv o .vcf

Postcondizioni: Vengono caricati nella rubrica dell'utente i contatti contenuti nel file fornito.

Flusso di eventi normale:

1. Utente clicca il menù a tendina "File";
2. Utente clicca il pulsante "Importa";
3. Utente fornisce il file con estensione .csv o .vcf;
4. Utente seleziona il file;
5. Utente importa il file;

Flusso di eventi alternativo:

- 3a. Utente annulla l'operazione;
- 3a.1 L'esecuzione riprende dal passo 1;
- 4a. Utente fornisce un file con contenuto non interpretabile dalla rubrica;
- 4a.1 L'esecuzione riprende dal passo 3;
- 5a. Utente annulla l'operazione;
- 5a.1 L'esecuzione riprende dal passo 1;

C6 - Esportare rubrica

Attore partecipante: Utente

Precondizioni: Utente ha la schermata della rubrica aperta.

Postcondizioni: Viene prodotto un file .csv o .vcf con i contatti selezionati.

Flusso di eventi normale:

1. Utente clicca il menù a tendina "File";
2. Utente clicca il pulsante "Esporta";
3. Utente sceglie la categoria dei contatti da esportare
4. Utente sceglie l'estensione del file tra l'opzione .csv e .vcf;
5. Utente sceglie il percorso dove salvare il file;
6. Utente salva l'operazione;

Flusso di eventi alternativo:

- 3a. Utente annulla l'operazione;
- 3a.1 L'esecuzione riprende al passo 1;
- 4a. Utente annulla l'operazione;
- 4a.1 L'esecuzione riprende al passo 1;
- 5a. Utente annulla l'operazione;
- 5a.1 L'esecuzione riprende al passo 1;
- 6a. Utente annulla l'operazione;
- 6a.1 L'esecuzione riprende al passo 1;

C7 - Visualizzare rubrica con filtri

Attore partecipante: Utente

Precondizioni: Utente ha la schermata della rubrica aperta.

Postcondizioni: Utente visualizza i contatti associati al filtro selezionato.

Flusso di eventi normale:

1. Utente clicca il pulsante dell'imbuto ∇ ;
2. Utente sceglie 1 o più Tag tra quelli presenti.

Flusso di eventi alternativo:

- 2a. Utente sceglie l'ordine alfabetico inverso.
- 2b. Utente sceglie l'ordine per nome.

C8 - Salvare rubrica

Attore partecipante: Sistema

Precondizioni: Utente ha effettuato una qualsiasi modifica (aggiunge contatto, elimina Tag, importa un file di una rubrica, ...).

Postcondizioni: Il Sistema salva la modifica.

Flusso di eventi normale:

1. La modifica viene salvata in locale nel file binario `Data.bin`.

Flusso di eventi alternativo:

- 1a. La modifica viene salvata sul database.

C9 - Aggiungere Tag

Attore partecipante: Utente

Precondizioni: Utente ha la schermata della rubrica aperta.

Postcondizioni: 1 Tag è stato aggiunto.

Flusso di eventi normale:

1. Utente clicca il menù a tendina "Rubrica";
2. Utente clicca il pulsante "Gestisci Tag";
3. Utente aggiunge un nuovo Tag;
4. Utente salva l'operazione.

Flusso di eventi alternativo:

- 4a. Utente annulla l'operazione;
- 4a.1 L'esecuzione riprende al passo 1;

C10 - Modificare Tag

Attore partecipante: Utente

Precondizioni: Utente ha la schermata della rubrica aperta.

Postcondizioni: 1 Tag è stato modificato.

Flusso di eventi normale:

1. Utente clicca il menù a tendina "Rubrica";
2. Utente clicca il pulsante "Gestisci Tag";
3. Utente modifica un Tag precedentemente aggiunto;
4. Utente salva l'operazione.

Flusso di eventi alternativo:

- 4a. Utente annulla l'operazione;
- 4a.1 L'esecuzione riprende al passo 1;

C11 - Eliminare Tag

Attore partecipante: Utente

Precondizioni: Utente ha la schermata della rubrica aperta.

Postcondizioni: 1 Tag è stato eliminato.

Flusso di eventi normale:

1. Utente clicca il menù a tendina "Rubrica";
2. Utente clicca il pulsante "Gestisci Tag";
3. Utente seleziona il Tag che vuole eliminare;
4. Utente conferma l'operazione dalla schermata di conferma;

Flusso di eventi alternativo:

- 4a. Utente annulla l'operazione dalla schermata di conferma;
4a.1 L'esecuzione riprende dal passo 1;

C12 - Aggiungere configurazione database

Attore partecipante: Utente

Precondizioni: Utente ha la schermata della rubrica aperta.

Postcondizioni: Utente ha aggiunto il link del database.

Flusso di eventi normale:

1. Utente clicca il menù a tendina "File";
2. Utente clicca il pulsante "Configurazione";
3. Utente fornisce il link del database scelto.

Flusso di eventi alternativo:

- 3a. Utente aggiunge un link non valido;
3a.1 L'esecuzione riprende al passo 3;

C13 - Eliminare configurazione database

Attore partecipante: Utente

Precondizioni: Utente ha inserito precedentemente il link del database.

Postcondizioni: Utente ha eliminato il link del database.

Flusso di eventi normale:

1. Utente clicca il menù a tendina "File";
2. Utente clicca il pulsante "Configurazione";
3. L'utente sceglie di rimuovere il link del database;
4. Utente conferma l'operazione dalla schermata di conferma.

Flusso di eventi alternativo:

- 3a. Utente annulla l'operazione dalla schermata di conferma;
3a.1 L'esecuzione riprende dal passo 3.

C14 - Salvare configurazione

Attore partecipante: Sistema

Precondizioni: Utente ha modificato le configurazioni dell'applicazione (ad esempio aggiunta o eliminazione del link del database)

Postcondizioni: Il sistema salva la modifica.

Flusso di eventi normale:

1. Sistema salva la modifica in locale in un file di configurazione `Config.bin`.

Flusso di eventi alternativo: /

2 Design

2.1 Diagrammi di classi

I diagrammi di classi rappresentano il sistema della rubrica con le funzionalità di gestione dei contatti, dei tag, importazione/esportazione di dati, configurazioni e salvataggio dei dati in locale o su DB.

- **Contact**: Modella un contatto con attributi come nome, cognome, numeri di telefono, e-mail, immagine del profilo, e tag associati. Ha metodi per gestire i contatti, aggiungere e rimuovere numeri e e-mail, e per gestire i tag associati.
- **Tag**: Modella un'etichetta associata ai contatti, con attributi come id, descrizione, e un indice statico. Include metodi per ottenere e impostare descrizioni e id, e per confrontare gli oggetti.
- **AddressBook**: Contiene contatti e tag. È una classe singleton, quindi ha un'istanza unica, e gestisce il salvataggio dei dati tramite un database o la serializzazione. Ha metodi per aggiungere, rimuovere e ottenere contatti e tag, e per caricare e salvare i dati.
- **MainController**: Gestisce l'interfaccia utente principale, con metodi per inizializzare i componenti dell'interfaccia, aggiungere, modificare, rimuovere contatti, per visualizzare pop-up per l'importazione/esportazione e la gestione dei tag.
- **ImportPopupController, ExportPopupController, ManageTagsPopupController, ConfigPopupController, ImagePopupController**: Gestiscono i pop-up per importare/esportare contatti, gestire i tag, gestire la configurazione (link del DB) e gestire le immagini di profilo.
- **ConfirmPopupController**: Gestisce anch'essa un pop-up di conferma di un'eventuale operazione di eliminazione di un contatto o di un tag richiesta dall'utente.
- **Database**: Gestisce l'interazione con un database MongoDB per l'archiviazione di contatti e tag. Ha metodi per inserire, aggiornare, rimuovere, recuperare contatti e tag dal database.
- **Converter**: Classe di utilità finalizzata a determinate conversioni per Import ed Export.

2.1.1 Livello di dettaglio basso

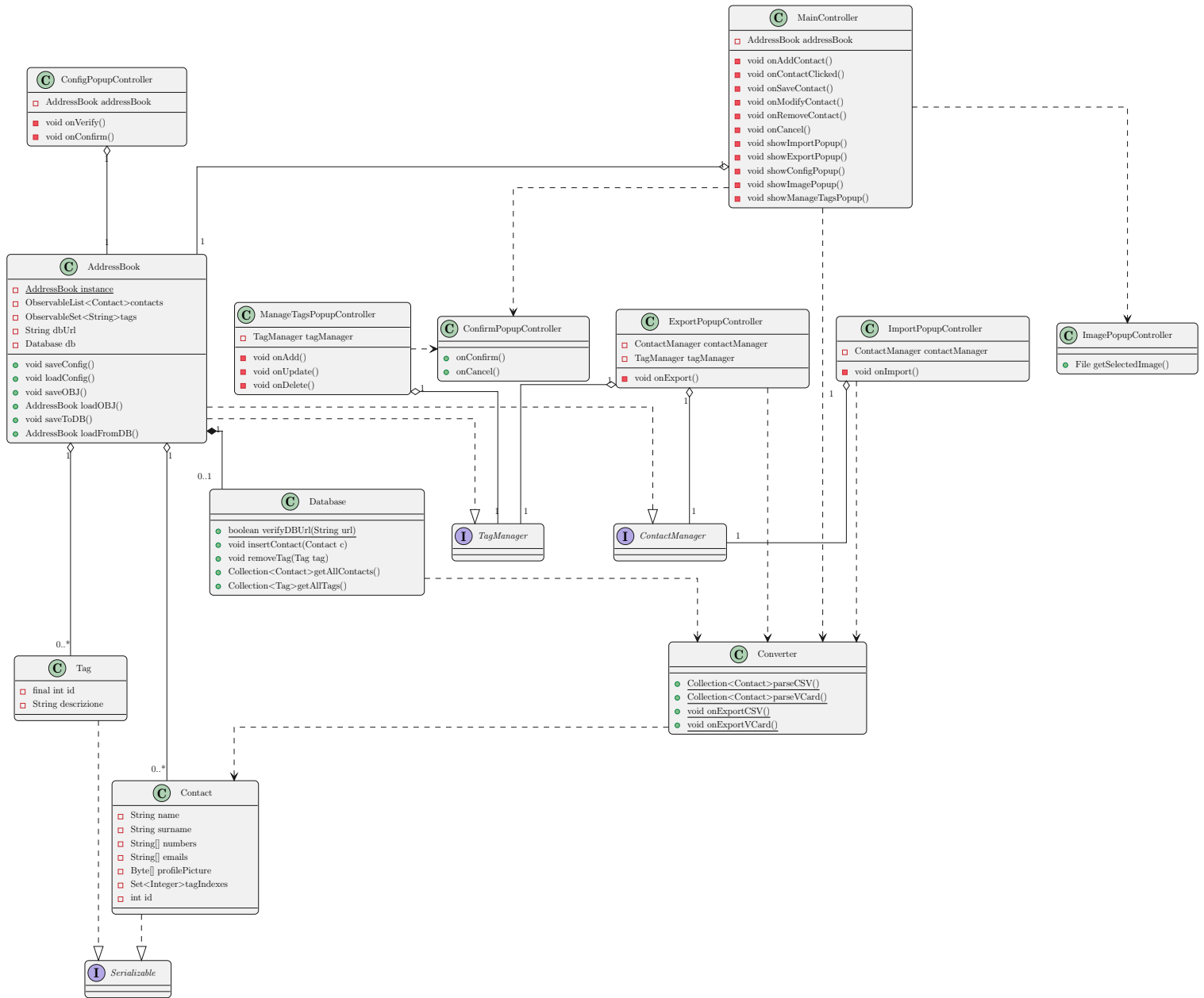


Figure 4: Diagramma classi essenziale

2.2 Diagrammi di sequenza

Si presentano una serie di diagrammi di sequenza che descrivono alcuni casi d'uso definiti nel sistema, descrivendo le interazioni tra l'attore *Utente* e gli oggetti del sistema coinvolti, al fine di soddisfare i requisiti specificati.

Oltre ai casi d'uso, vengono inclusi 2 diagrammi che rappresentano flussi operativi rilevanti per la comprensione del funzionamento complessivo del sistema.

2.2.1 C1 - Aggiungere Contatto

Il seguente diagramma di sequenza illustra l'esecuzione del caso d'uso C1:

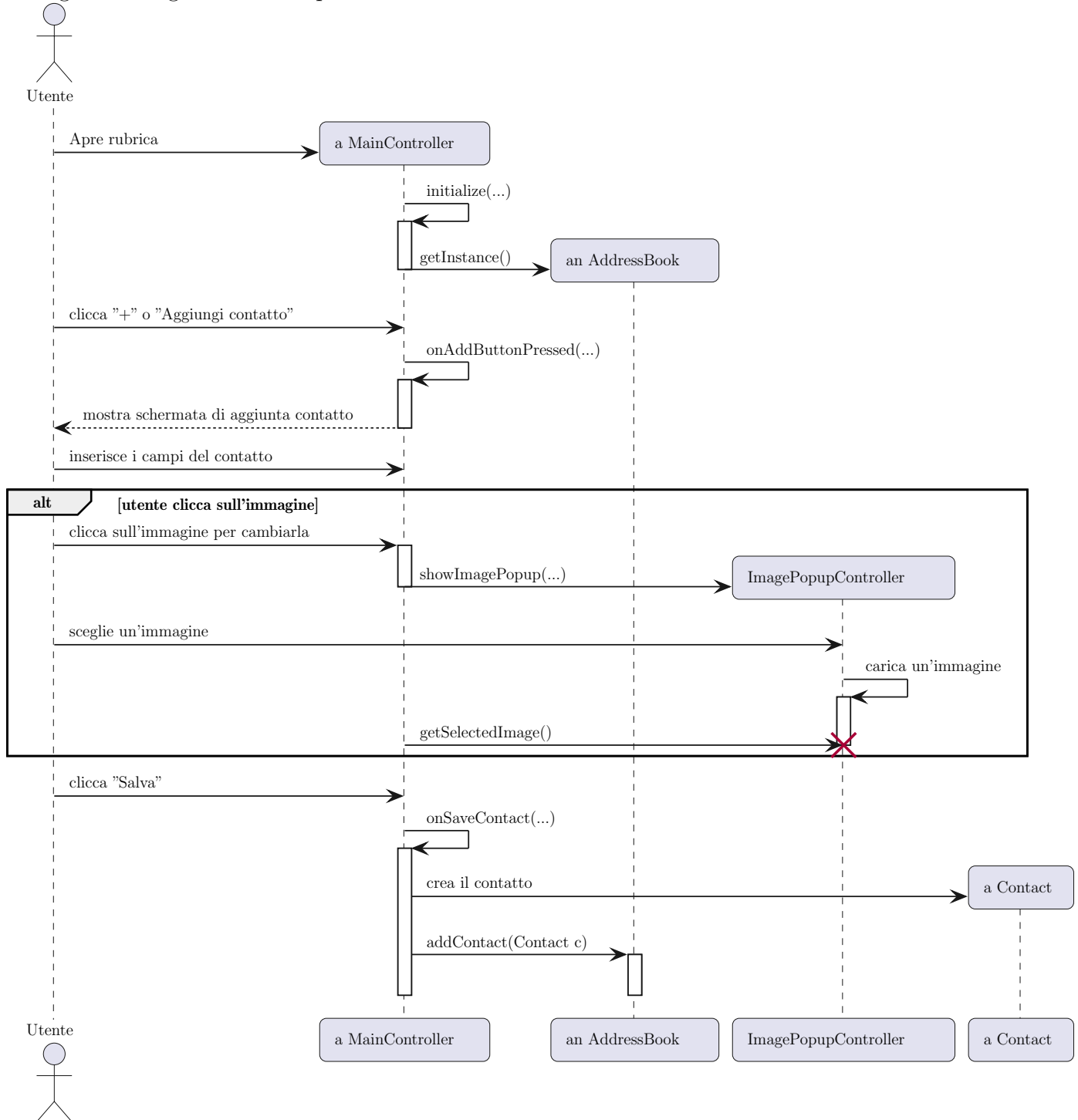


Figure 6: Diagramma sequenza C1 - Aggiungere Contatto

2.2.2 C2 C3 - Eliminare Modificare Contatto

Il seguente diagramma di sequenza illustra l'esecuzione di entrambi i casi d'uso C2 e C3, tramite la sintassi loop, break e alt:

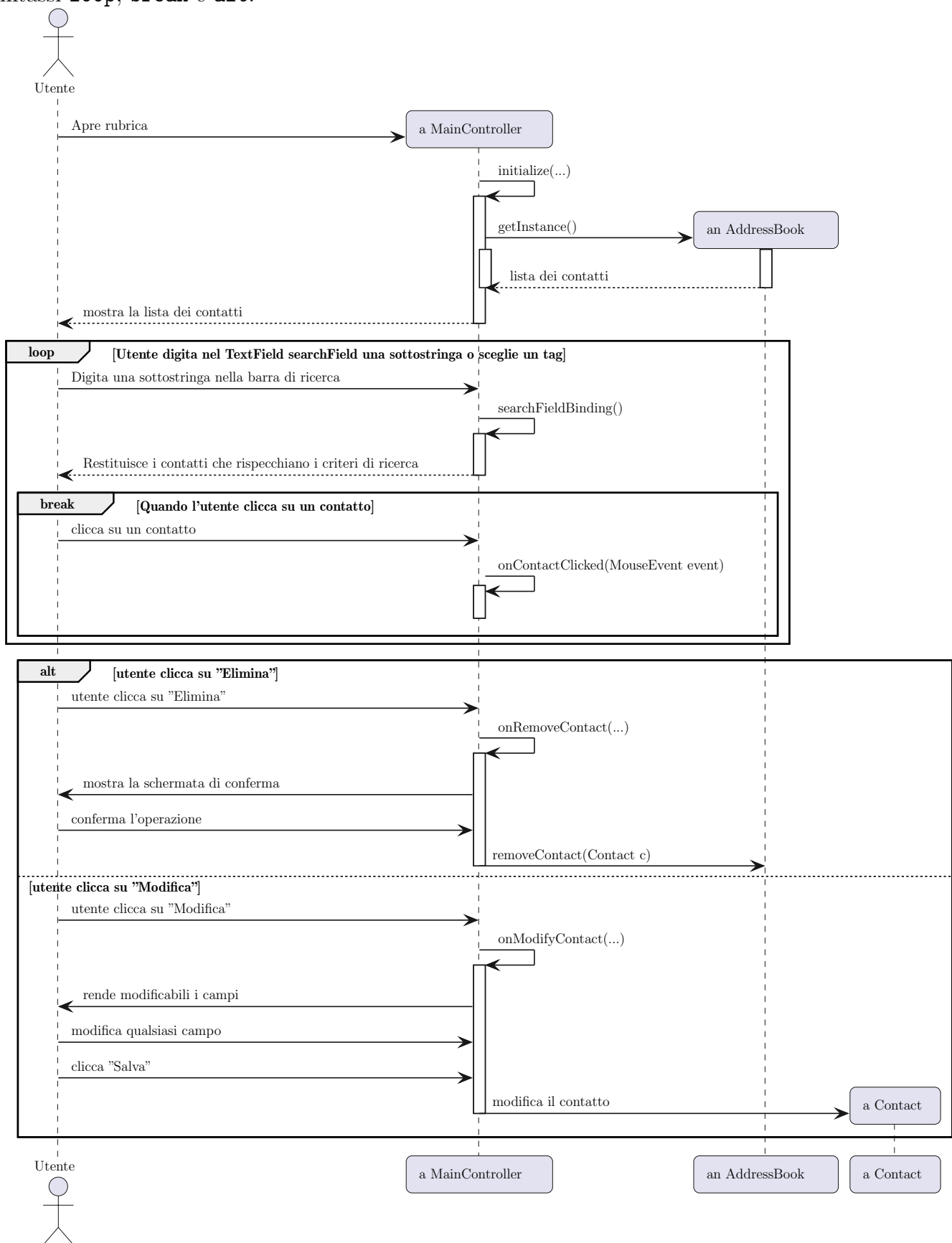


Figure 7: Diagramma sequenza C2 C3 - Eliminare Modificare Contatto

2.2.3 C5 - Importare rubrica

Il seguente diagramma di sequenza illustra l'esecuzione del caso d'uso C5:

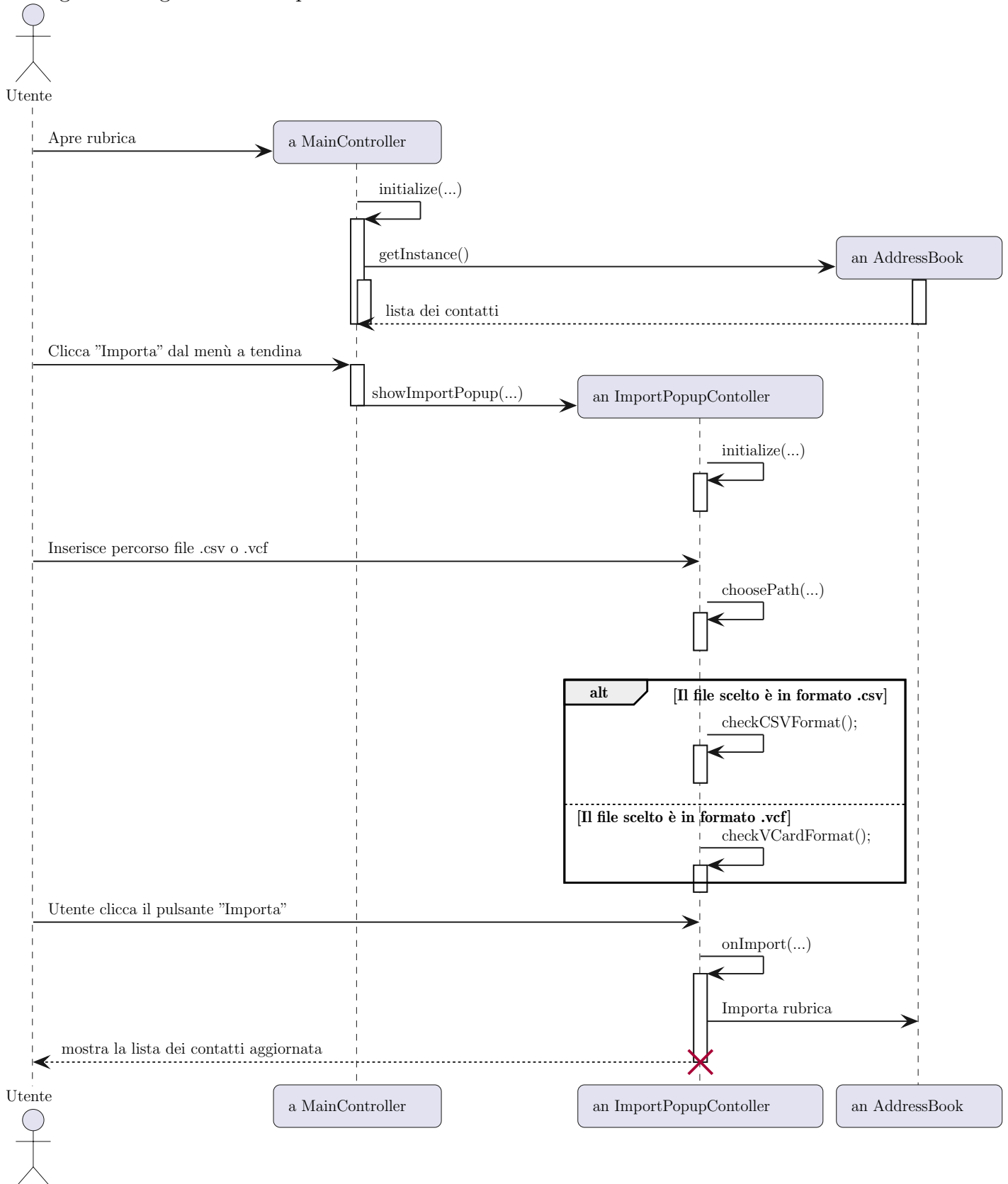


Figure 8: Diagramma sequenza C5 - Importare rubrica

2.2.4 C6 - Esportare rubrica

Il seguente diagramma di sequenza illustra l'esecuzione del caso d'uso C6:

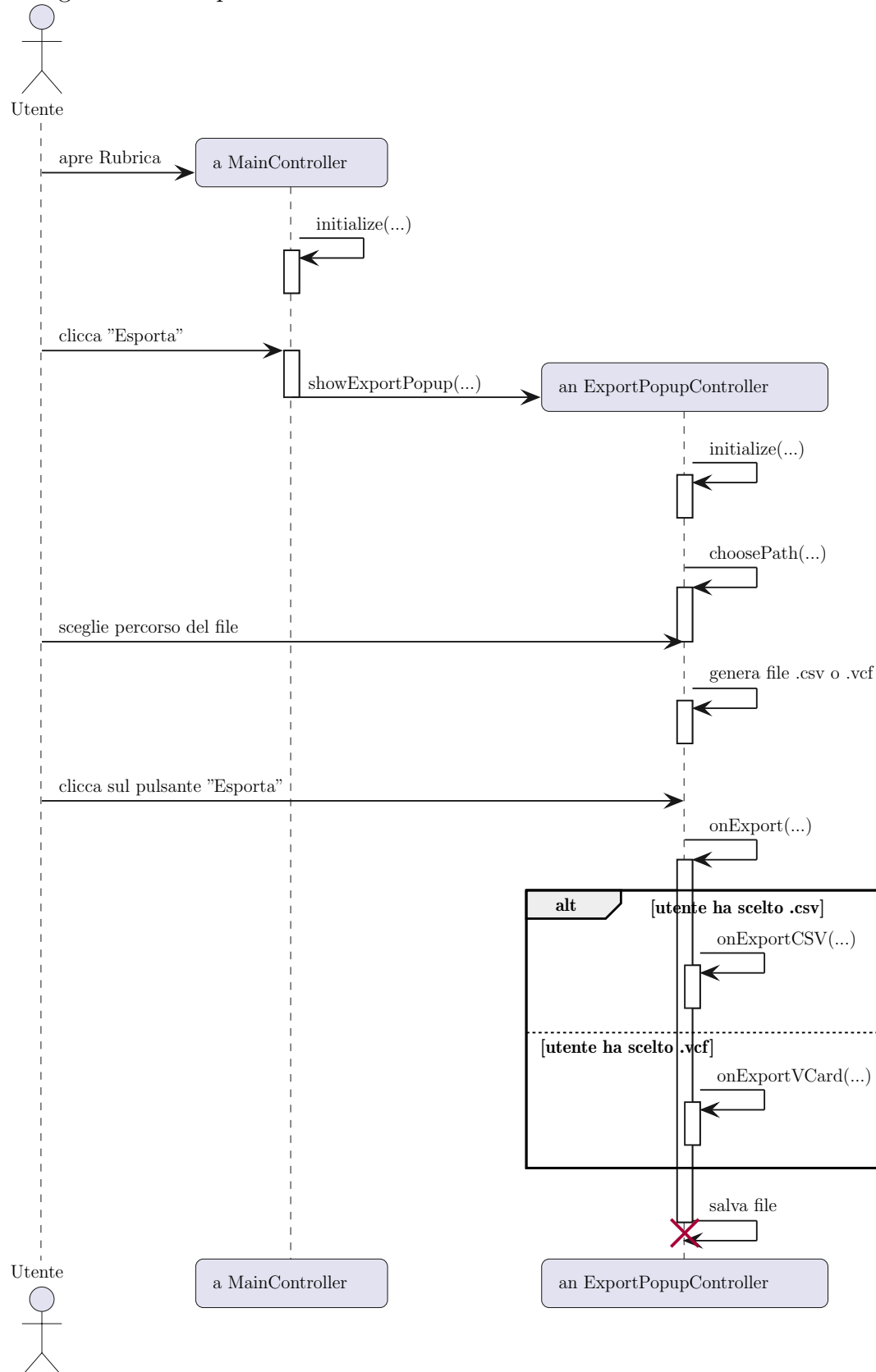


Figure 9: Diagramma sequenza C6 - Esportare rubrica

2.2.5 C8 - Salvare rubrica

Il seguente diagramma di sequenza illustra l'esecuzione del caso d'uso C8:

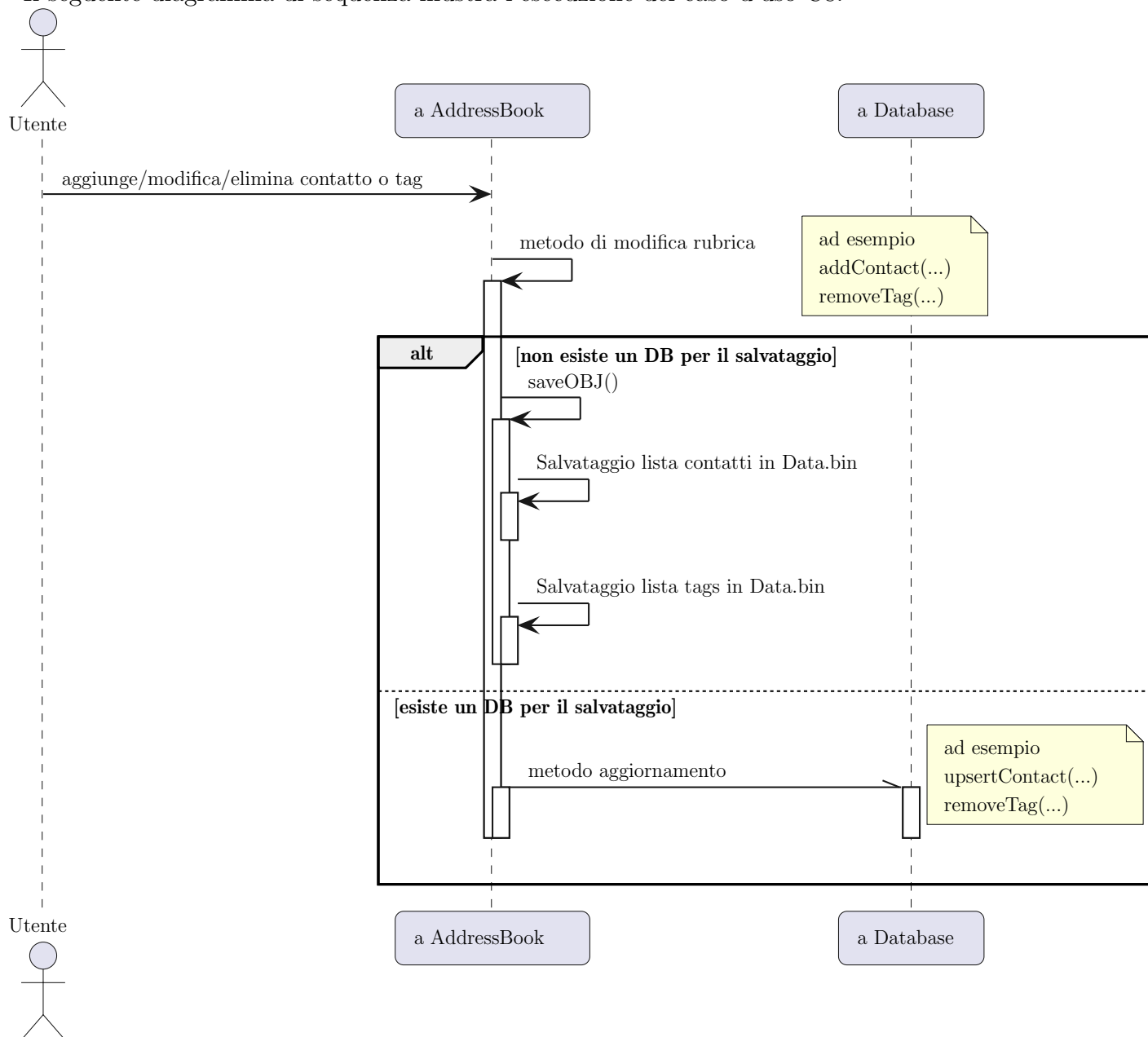


Figure 10: Diagramma sequenza C8 - Salvare rubrica

2.2.6 Aggiunta DB

Il seguente diagramma di sequenza illustra il flusso di operazioni attraverso cui l'utente aggiunge un link a un Database, consentendo il salvataggio dei dati nel DB anziché in locale.:

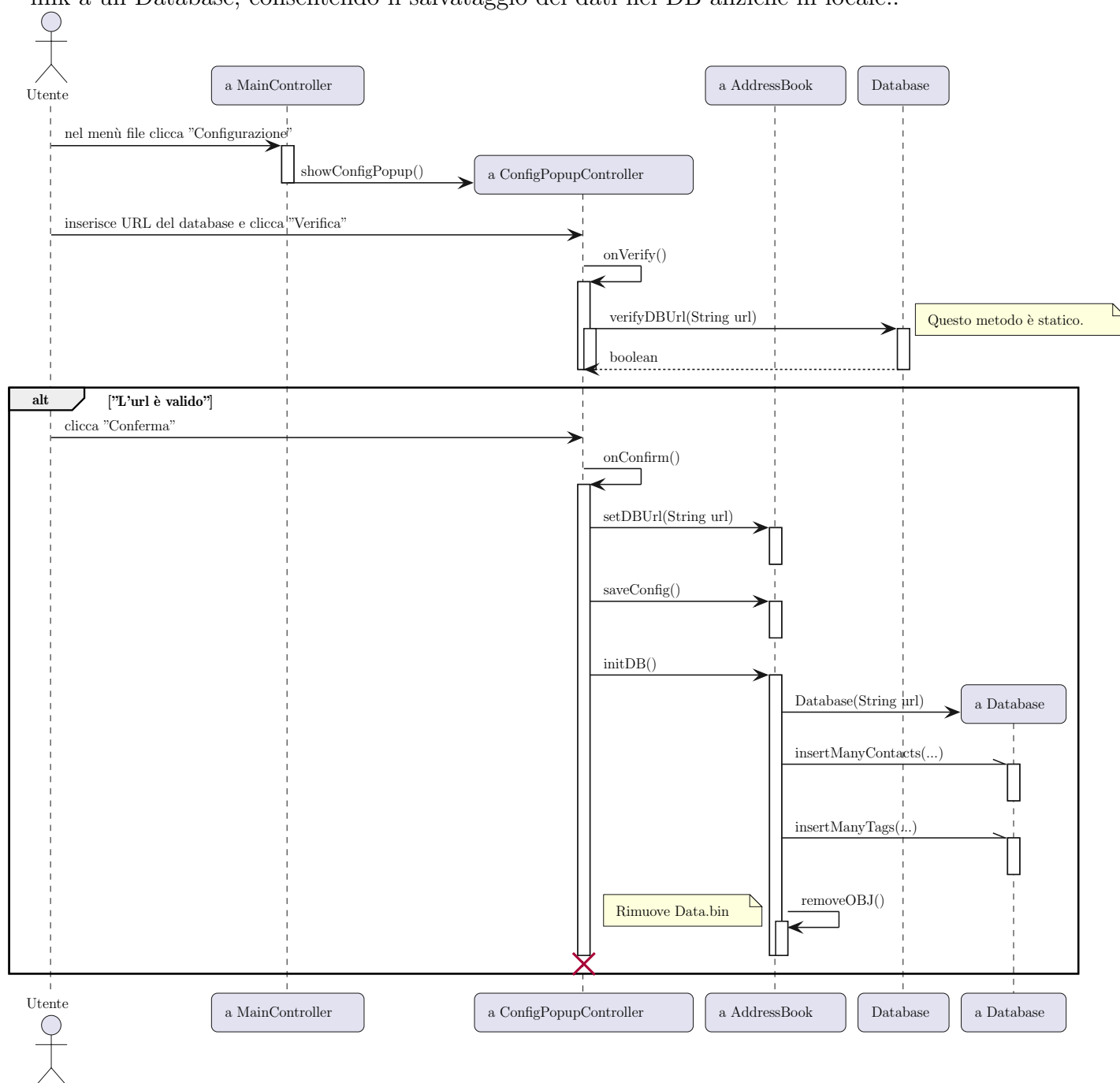


Figure 11: Diagramma sequenza Aggiunta DB

2.2.7 Inizializzazione rubrica

Il seguente diagramma di sequenza illustra il flusso di operazioni eseguito quando l'utente apre l'applicazione, durante la fase di inizializzazione, permettendogli di recuperare tutti i contatti della sessione precedente, attraverso il DB o il file `Data.bin`:

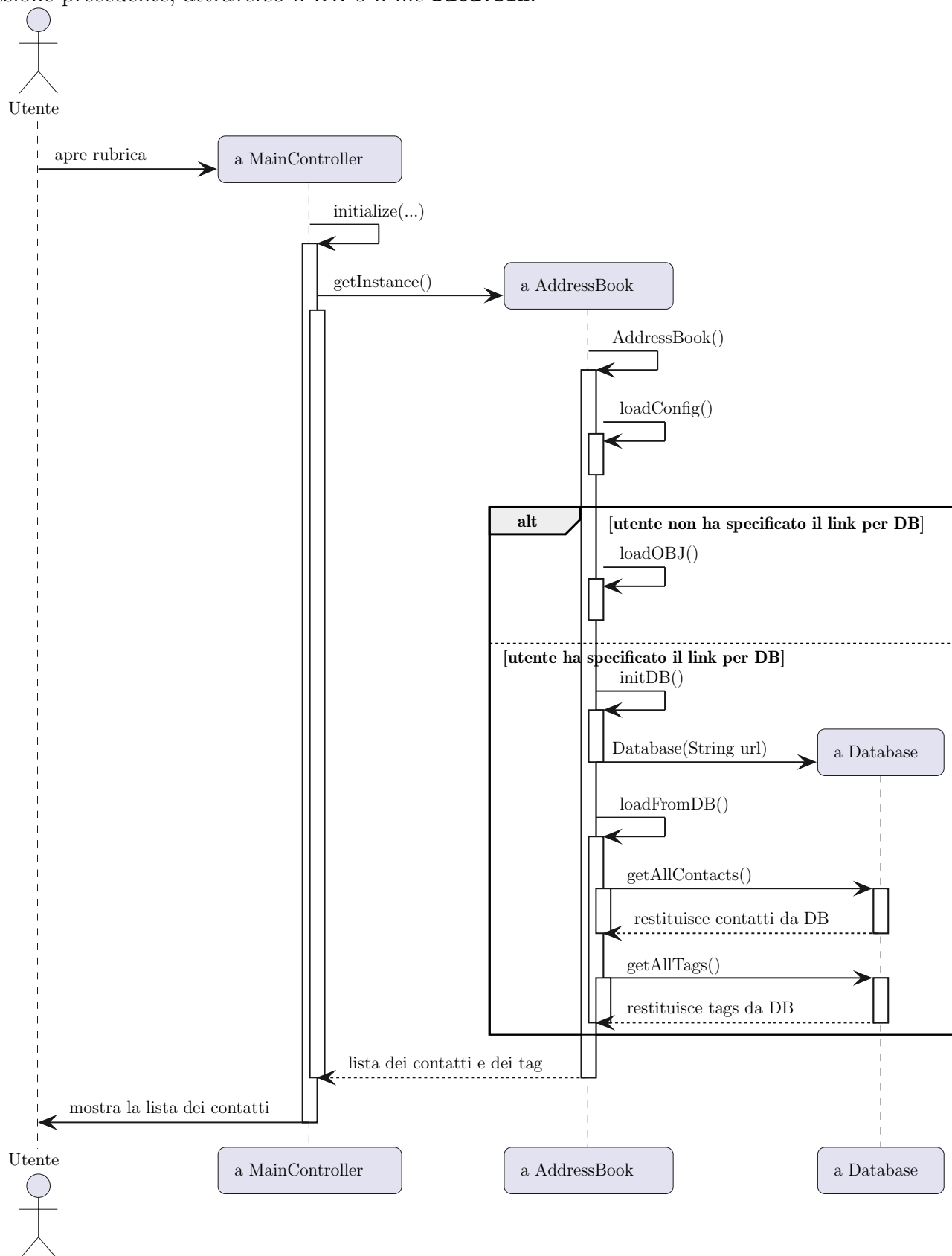


Figure 12: Diagramma sequenza Inizializzazione rubrica

2.3 Principi di buona progettazione

Si garantisce l'aderenza ai principi di buona progettazione del software, migliorando modularità, manutenibilità e chiarezza del sistema.

2.3.1 Livelli di coesione


Nome classe	 Livello Coesione	Nota
AddressBook	Comunicazionale	Tutti i metodi della classe utilizzano la lista dei contatti o dei tags, ad eccezione del metodo removeOBJ() che deve solo eliminare il file Data.bin
ConfigPopupController	Funzionale	Include i 2 metodi finalizzati alla verifica del link per il DB inserito dall'utente e alla conferma dell'operazione che prevede la creazione del file Config.bin
ConfirmPopupController	Funzionale	Include i 2 metodi finalizzati alla conferma dell'operazione di eliminazione di un contatto o di un tag, da parte dell'utente
Contact	Funzionale	Tutti i metodi della classe sono adibiti alla visione/modifica degli attributi
Database	Comunicazionale	I metodi della classe lavorano su dati (contatti e tag) che sono correlati tra loro perchè appartengono alla stessa rubrica, pertanto lavorano sempre sugli stessi dati.
ExportPopupController	Funzionale	I metodi lavorano insieme per realizzare il compito di esportare il file in uno dei 2 formati
ImagePopupController	Funzionale	I metodi che definisce sono finalizzati a scaricare l'immagine scelta dall'utente e a prelevarla dall'esterno tramite dei metodi get. Quindi sono funzionalità che lavorano insieme.
ImportPopupController	Funzionale	I metodi lavorano insieme per realizzare il compito di importare il file in uno dei 2 formati
MainController	Funzionale	I metodi definiti lavorano per la gestione di tutte le operazioni che l'utente può richiedere sulla rubrica dall'interfaccia grafica principale.
ManageTagsPopupController	Funzionale	I metodi che definisce sono finalizzati alla gestione dei tag. Quindi sono funzionalità che lavorano insieme.
Tag	Funzionale	Tutti i metodi della classe sono adibiti alla visione/modifica degli attributi

Figure 13: Livelli di coesione classi

2.3.2 Livelli di accoppiamento

Nome Classe 1	Nome Classe 2	Livello Accoppiamento	Nota
AddressBook	Contact	Per dati	Essendoci una relazione di aggregazione 1 a molti, la classe AddressBook contiene solo i riferimenti di Contact, che servono per gestire la relazione.
AddressBook	Tag	Per dati	Essendoci una relazione di aggregazione 1 a molti, la classe AddressBook contiene solo i riferimenti di Tag, che servono per gestire la relazione.
AddressBook	Database	Per dati	La classe AddressBook contiene un'istanza di Database per garantire le funzioni relative al database.
MainController	AddressBook	Per dati	La classe MainController ha il riferimento all'AddressBook, per garantire le funzionalità attribuite all'interfaccia grafica.
MainController	ImagePopupController	Per dati	La classe MainController usufruisce dei metodi get necessari al suo funzionamento.
MainController	ConfirmPopupController	Per controllo	L'operazione di eliminazione di un contatto, nel MainController, dipende dalla scelta dell'utente effettuata nel ConfirmPopup.
ManageTagsPopupController	ConfirmPopupController	Per controllo	L'operazione di eliminazione di un tag, nel MainController, dipende dalla scelta dell'utente effettuata nel ConfirmPopup.
ConfigPopupController	AddressBook	Per timbro (Stamp)	La classe ConfigPopupController contiene il riferimento all'AddressBook ma utilizza soltanto alcuni dei suoi metodi.

Figure 14: Livelli di accoppiamento classi

2.3.3 Relazioni tra le Classi

Nel progetto si è prediletta l'associazione rispetto alla specializzazione (ereditarietà).

In particolare:

- Esiste una relazione di aggregazione 1 a molti tra *AddressBook* e *Contact*, tra *AddressBook* e *Tag*.
- È presente una relazione di composizione 1 a 0..1 tra *AddressBook* e *Database*, poiché la rubrica può prevedere il salvataggio su database o in locale.

2.3.4 Riduzione dell'Accoppiamento

Per seguire il principio di riduzione dell'accoppiamento tra classi, sono state create le interfacce *TagManager* e *ContactManager*.

Questo consente alle classi *Export*, *Import* e *ManageTagsPopupController* di accedere solo ai metodi strettamente necessari, favorendo anche il principio di segregazione delle interfacce.

Tuttavia:

- *MainController* ha un riferimento diretto a *AddressBook*, poiché dipende da esso per la maggior parte dei metodi, esclusi quelli relativi ai tag.
- *ConfigPopupController* ha anch'esso un riferimento diretto a *AddressBook*, ma con un accoppiamento per timbro, dato che utilizza solo alcuni metodi.

2.3.5 Relazione con l'Interfaccia Initializable

Tutti i controller implementano l'interfaccia *Initializable*, ma nei diagrammi di classe tale relazione viene omessa per ridurre il livello di dettaglio e garantire una maggiore visibilità.

2.3.6 Livelli di dettaglio

Sono forniti due diagrammi di classi con livelli di dettaglio differenti:

- Diagramma con relazioni evidenziate: Mostra solo i metodi e attributi più rilevanti, evidenziando meglio le relazioni.
- Diagramma completo: Descrive in dettaglio tutti gli attributi e metodi delle classi, evidenziandone più concretamente il ruolo nella realizzazione del sistema.

2.3.7 Diagrammi di Sequenza

Sono forniti diagrammi di sequenza per descrivere i flussi di eventi relativi all'interazione tra l'utente e la rubrica, tra cui:

- Alcuni casi d'uso specifici (C1, C2, C3, C5, C6, C8).
- L'operazione di aggiunta del database tramite link inserito dall'utente.
- Lo scenario di apertura e inizializzazione della rubrica, che può avvenire attraverso il database o il file locale `Data.bin` in assenza del database.

3 Implementazione e Testing

Il progetto è stato sviluppato utilizzando **JavaFX** per la realizzazione dell'interfaccia grafica, adottando una struttura modulare che organizza le classi in package specifici in base al loro ruolo all'interno del progetto. Il funzionamento del progetto, con le relative dipendente, è affidato a **Maven**.

Inoltre, per garantire l'affidabilità e il corretto funzionamento delle principali classi del progetto, sono stati implementati test automatizzati utilizzando la libreria **JUnit** e i relativi metodi **assert**. Questo approccio ha consentito di identificare e risolvere diversi problemi in fase di implementazioni delle classi.

3.1 Diagramma dei Package

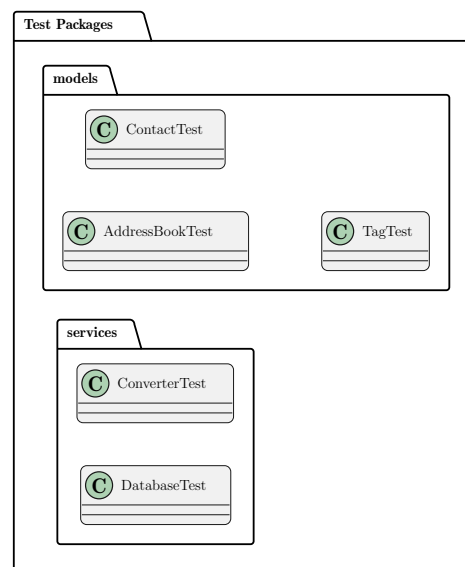
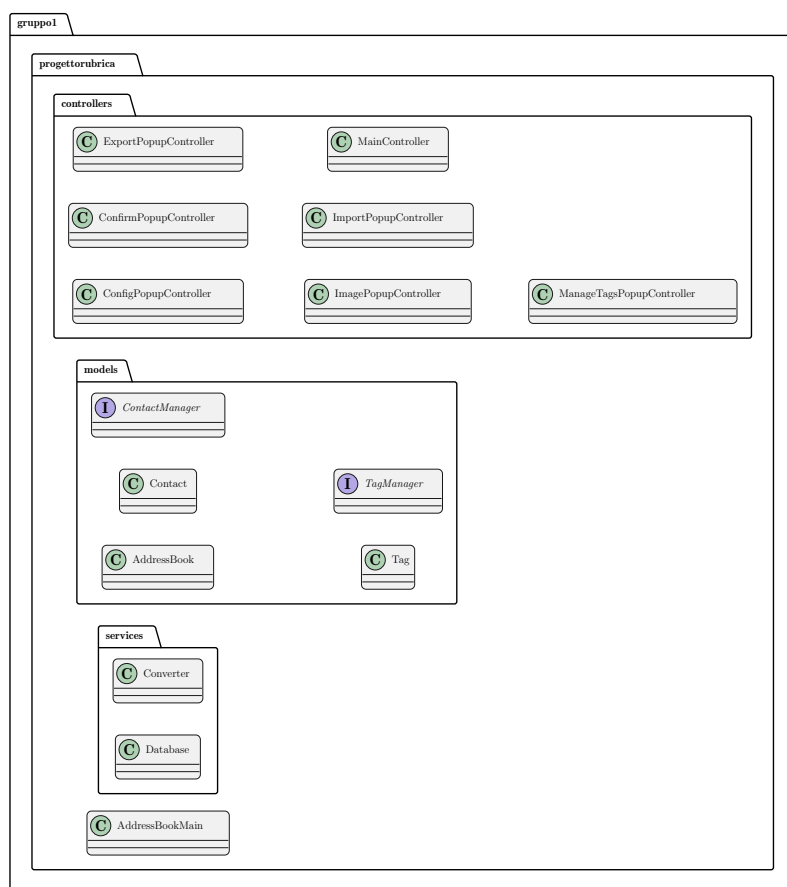


Figure 16: Diagramma package testing

Figure 15: Diagramma package implementazione

Package controllers

Questo Package è dedicato esclusivamente alla gestione dell'interfaccia grafica dell'applicazione. Al suo interno sono presenti classi Controller ognuna delle quali gestisce un aspetto specifico dell'interfaccia grafica, come l'esportazione (*ExportPopupController*), importazione (*ImportPopupController*) o schermata di conferma per l'eliminazione di un contatto o tag (*ConfirmPopupController*).

Questo consente una separazione delle responsabilità, con i controller che si occupano esclusivamente della logica di interazione con l'utente.

La presenza di diverse classi, ciascuna con un compito preciso, segue il principio di singola responsabilità.

Package models

Questo Package contiene le classi che rappresentano i modelli del sistema e la logica.

Definendo *AddressBook*, *Contact* e *Tag* si definiscono i modelli.

Definendo le interfacce *ContactManager* e *TagManager* si separa la gestione dei modelli e le interazioni con questi.

Tali interfacce offrono dei metodi per *ExportPopupController*, *ImportPopupController* e *ManageTagsPopupController* per la gestione dei contatti e dei tag.

Package services

In questo Package sono presenti le classi *Converter* e *Database*, che gestiscono operazioni generali come la conversione di dati e l'interazione con il database.

Tali classi si occupano del salvataggio dei dati in remoto e la trasformazione dei formati. Questo riduce la duplicazione del codice e quindi migliora la manutenibilità.

Package di test

I Package per i test sono relativi ai modelli e ai servizi e sono in un package separato.

Le classi di test *AddressBookTest*, *ContactTest* e *TagTest* sono associati ai rispettivi modelli, mentre *ConverterTest* e *DatabaseTest* sono dedicati ai test dei servizi.

Questo facilita l'individuazione di eventuali bug in fase di implementazione e refactoring.

3.2 UTC 1 - AddressBook

UTC 1.1	Test AddressBook - getInstance
Test items	<pre>getInstance(); AddressBook(); getAllContacts(); getAllTags(); getDBUrl(); a.getDB(); getAllContacts() getAllTags()</pre>
Input	<code>a = AddressBook.getInstance();</code>
Oracle	<pre>a.getAllContacts().isEmpty()==TRUE a.getAllTags().isEmpty()==TRUE a.getDBUrl().isEmpty()==TRUE a.getDB()==NULL File(pathConfig).exists()==FALSE File(pathData).exists()==FALSE</pre>

UTC 1.2	Test AddressBook - getInstance
Test items	<pre>getInstance(); AddressBook(); getAllContacts(); getAllTags(); getDBUrl(); a.getDB(); loadOBJ(); getAllContacts() getAllTags()</pre>
Input	<pre>genero una lista di contatti casuali: contactsProva genero una lista di Tag casuali: tagsProva oos.writeObject(contactsProva); in Data.bin oos.writeObject(tagsProva); in Data.bin a = AddressBook.getInstance();</pre>
Oracle	<pre>a.getAllTags().containsAll(tagsProva)==TRUE a.getAllContacts().containsAll(contactsProva)==TRUE a.getDBUrl().isEmpty()==TRUE a.getDB()==NULL File(pathConfig).exists()==FALSE File(pathData).exists()==TRUE</pre>

UTC 1.8	Test AddressBook - addManyContacts
Test items	<pre>getInstance(); AddressBook(); addManyContacts(Collection<Contact> c) saveOBJ(); loadConfig(); addManyTags(...); getAllContacts();</pre>
Input	<pre>genero una lista di contatti casuali: contactsProva database = new Database(url); genero Config.bin con url a = AddressBook.getInstance(); a.addManyContacts(contactsProva); contattiLetti = database.getAllContacts();</pre>
Oracle	<pre>contattiLetti, a.getAllContacts() File(pathData).exists()==FALSE</pre>

UTC 1.4	Test AddressBook - getInstance
Test items	<pre> getInstance(); AddressBook(); loadConfig(); initDB(); loadFromDB(); dataToDB(); getDbUrl(); a.getDB(); getAllContacts() getAllTags() </pre>
Input	<p>genero Config.bin con url</p> <p>genero una lista di contatti casuali: contactsProvaDb genero una lista di Tag casuali: tagsProvaData genero una lista di contatti casuali: contactsProvaDb genero una lista di Tag casuali: tagsProvaData</p> <pre> database = new Database(url); database.insertManyContacts(contactsProvaDb); database.insertManyTags(tagsProvaDb); oos.writeObject(contactsProvaData); in Data.bin oos.writeObject(tagsProvaData); in Data.bin a = AddressBook.getInstance(); </pre>
Oracle	<pre> File(pathData).exists()==FALSE File(pathConfig).exists()==TRUE contactsProvaData.size() == a.getAllContacts().size() tagsProvaData.size() == a.getAllTags().size() contactsProvaData == a.getAllContacts() tagsProvaData == a.getAllTags() url == a.getDbUrl() a.getDB() != NULL </pre>

UTC 1.9	Test AddressBook - addManyContacts
Test items	<pre> getInstance(); AddressBook(); loadConfig() initDB() loadFromDB() addManyContacts(Collection<Contact> c) saveOBJ(); a.getAllContacts() </pre>
Input	<p>genero Config.bin con url database = new Database(url);</p> <p>genero una lista di contatti casuali: contactsProva a = AddressBook.getInstance(); a.addManyContacts(contactsProva);</p> <p>urlField.set(a, "urlNonValido.it"); genero una lista di contatti casuali: contactsProva a.addManyContacts(contactsProva);</p> <p>urlField.set(a, url); genero una lista di contatti casuali: contactsProva a.addManyContacts(contactsProva);</p> <p>contattiLetti = database.getAllContacts()</p>
Oracle	<pre> File(pathConfig).exists()==TRUE File(pathData).exists()==FALSE contattiLetti==a.getAllContacts() </pre>

UTC 1.10	Test AddressBook - removeContact(Contact c)
Test items	<pre> getInstance(); AddressBook(); loadOBJ(); removeContact(Contact c); getAllContacts() getAllTags() </pre>
Input	<pre> genero Data.bin con un singolo contatto casuale a = AddressBook.getInstance(); a.removeContact(c); </pre>
Oracle	<pre> a.getAllContacts().isEmpty()==TRUE a.getAllTags().isEmpty()==TRUE File(pathData).exists()==FALSE </pre>

UTC 1.12	Test AddressBook - getTag(String descrizione)
Test items	<pre> getInstance(); AddressBook(); addManyTags(...); a.getTag(String descrizione); </pre>
Input	<pre> genero una lista di Tag casuali: tagsProva a = AddressBook.getInstance(); a.addManyTags(tagsProva); Tag tagPrelevato1 = a.getTag(tagsProva(1).getDescription()); Tag tagPrelevato2 = a.getTag(tagsProva(2).getDescription()); </pre>
Oracle	<pre> tagsProva(1) == tagPrelevato1 tagsProva(2) == tagPrelevato2 </pre>

UTC 1.14	Test AddressBook - getTag(int id)
Test items	<pre> getInstance(); AddressBook(); addManyTags(...); </pre>
Input	<pre> genero un vettore di Tag casuali: tagVett[] tagVett[0]=111 tagVett[1]=222 tagVett[2]=333 a = AddressBook.getInstance(); a.addManyTags(Arrays.asList(tagVett)); Tag tagPrelevato = a.getTag(222); </pre>
Oracle	<pre> tagVett[1] == tagPrelevato </pre>

UTC 1.15	Test AddressBook - getTag(int id)
Test items	<pre> getInstance(); AddressBook(); addManyTags(...); </pre>
Input	<pre> genero un vettore di Tag casuali: tagVett[] tagVett[0]=111 tagVett[1]=222 tagVett[2]=333 a = AddressBook.getInstance(); a.addManyTags(Arrays.asList(tagVett)); Tag tagPrelevato = a.getTag(123); </pre>
Oracle	<pre> tagPrelevato==NULL </pre>

3.3 UTC 2 - Contact

UTC 2.16	Test Contact - removeTagIndex(Integer tagIndex)
Test items	addTagIndex(Integer tagIndex); getAllTagIndexes(); removeTagIndex(Integer tagIndex);
Input	inserisco 2 tag c.addTagIndex(2); c.addTagIndex(4); ottengo la collezione di tag c.getAllTagIndexes(); rimuovo 1 tag presente ed 1 assente
Oracle	c.removeTagIndex(2)==TRUE c.removeTagIndex(5)==FALSE tags2.contains(2)==TRUE

UTC 2.17	Test Contact - addTagIndex(Integer tagIndex)
Test items	addTagIndex(Integer tagIndex); getAllTagIndexes();
Input	inserisco 3 tag c.addTagIndex(1); c.addTagIndex(2); c.addTagIndex(3); ottengo la collezione di tag c.getAllTagIndexes();
Oracle	tags.contains(1)==TRUE tags.contains(2)==TRUE tags.contains(3)==TRUE tags.contains(4)==FALSE

UTC 2.18	Test Contact - equals()
Test items	equals(); setNumbers(String[] numbers) setEmails(String[] emails) setProfilePicture(Byte[] profilePicture) addTagIndex(Integer tagIndex)
Input	creo una stringa creo un contatto con stesso nome e cognome del Contact c creo 2 vettori contenenti stessi numeri di cellulare c1.setNumbers(numbers1); c.setNumbers(numbers2); creo 2 vettori contenenti stesse email c.setEmails(emails1); c1.setEmails(emails2); c.addTagIndex(2); c1.addTagIndex(2); creo 2 vettori contenenti stessi byte dell'immagine profilo c.setProfilePicture(image1); c1.setProfilePicture(image2);
Oracle	c.equals(null)==FALSE c.equals(c)==TRUE c.equals(s)==FALSE c.equals(c1)==TRUE

3.4 UTC 3 - Converter

UTC 3.1	Test Converter- ParseCSV
Test items	parseCsv();
Input	contacts = Converter.parseCSV(csvFile);
Oracle	contacts != NULL contacts.size() == 2

UTC 3.2	Test Converter - ParseCSV
Test items	parseCsv();
Input	contacts = Converter.parseCSV(csvFile);
Oracle	contact1.getName() == Luca contact1.getSurname() == Rossi contact1.getNumbers() == {"1234567890", "0987654321", "1122334455"} contact1.getEmail() == {"l.rossi@gmail.com", "rossil@outlook.com", "lucarossi@alice.it"} contact1.getProfilePicture() == "Hello world".getBytes() contact2.getName() == Mario contact2.getSurname() == Grigi contact2.getNumbers() == {"2233445566", "6655443322", "7788990011"} contact2.getEmail() == {"m.grigi@gmail.com", "grigimar@outlook.com", ""} contact2.getProfilePicture() == "Hello world".getBytes()

UTC 3.4	Test Converter - ParseCSV
Test items	parseCsv();
Input	contacts = Converter.parseCSV(invalidFile);
Oracle	contacts != NULL contacts.isEmpty() == TRUE

UTC 3.5	Test Converter - ParseVCard
Test items	parseVCard();
Input	contacts = Converter.parseVCard(vcfFile);
Oracle	contacts != NULL contacts.size() == 2

UTC 3.6	Test Converter - ParseVCard
Test items	parseVCard();
Input	contacts = Converter.parseVCard(vcfFile);
Oracle	contact1.getName() == Luca contact1.getSurname() == Rossi contact1.getNumbers() == {"1234567890", "0987654321", "1122334455"} contact1.getEmail() == {"l.rossi@gmail.com", "rossil@outlook.com", "lucarossi@alice.it"} contact1.getProfilePicture() == "Hello world".getBytes() contact2.getName() == Mario contact2.getSurname() == Grigi contact2.getNumbers() == {"2233445566", "6655443322", "7788990011"} contact2.getEmail() == {"m.grigi@gmail.com", "grigimar@outlook.com", ""} contact2.getProfilePicture() == "Hello world".getBytes()

UTC 3.8	Test Converter - ParseVCard
Test items	parseVCard();
Input	contacts = Converter.parseVCard(invalidFile);
Oracle	contacts != NULL contacts.isEmpty() == TRUE

UTC 3.9	Test Converter - onExportCSV
Test items	onExportVCard();
Input	tempFile = File.createTempFile("contacts", ".vcf"); genero dei contatti : createContacts() List<String> lines = Files.readAllLines(tempFile.toPath())
Oracle	tempFile != NULL

UTC 3.10	Test Converter - onExportCSV
Test items	onExportCSV();
Input	tempFile = File.createTempFile("contacts", ".csv"); genero dei contatti : createContacts() esporto i contatti generati List<String> lines = Files.readAllLines(tempFile.toPath())
Oracle	lines.get(0).contains("Name, Surname, TEL1, TEL2, TEL3, EMAIL1, EMAIL2, EMAIL3, PHOTO,") == TRUE lines.get(1).contains("Luca", "Rossi", numeri, emails + fotoProfilo + ",") == TRUE lines.get(2).contains("Mario", "Grigi", numeri, emails + fotoProfilo + ",") == TRUE

UTC 3.11	Test Converter - onExportVCard
Test items	onExportVCard();
Input	tempFile = File.createTempFile("contacts", ".vcf"); genero dei contatti : createContacts() List<String> lines = Files.readAllLines(tempFile.toPath())
Oracle	tempFile != NULL

UTC 3.12	Test Converter - onExportVCard
Test items	onExportVCard();
Input	tempFile = File.createTempFile("contacts", ".vcf"); genero dei contatti : createContacts() List<String> lines = Files.readAllLines(tempFile.toPath())
Oracle	Controllo di tutte le righe del file .VCard

UTC 3.13	Test Converter - StringToByteArrayToString
Test items	stringToByteArray(); byteArrayToString();
Input	encodedString = Base64.getEncoder().encodeToString(original.getBytes()); Byte[] res = Converter.stringToByteArray(encodedString); String result = Converter.byteArrayToString(res);
Oracle	result = encodedString

UTC 3.14	Test Converter - toWrapper
Test items	toWrapper()
Input	byte[] byteArray = {1, 2, 3, 4, 5}; Byte[] expected = {1, 2, 3, 4, 5}; Byte[] result = Converter.toWrapper(byteArray);
Oracle	expected == result

UTC 3.16	Test Converter - toPrimitive
Test items	toPrimitive()
Input	Byte[] byteObjectArray = {1, 2, 3, 4, 5}; byte[] expected = {1, 2, 3, 4, 5}; byte[] result = Converter.toPrimitive(byteObjectArray);
Oracle	expected == result

UTC 3.18	Test Converter - ImageViewToByteArray
Test items	imageViewToByteArray();
Input	inizializzo image con un'immagine; ImageView imageView = new ImageView(image); Byte[] byteArray = Converter.imageViewToByteArray(imageView);
Oracle	byteArray != null byteArray.length > 0

UTC 3.19	Test Converter - ImageViewToByteArray
Test items	imageViewToByteArray();
Input	<pre> inializzo image con un'immagine; ImageView imageView = new ImageView(image); Byte[] byteArray = Converter.imageViewToByteArray(imageView); byte[] primitiveArray = Converter.toPrimitive(byteArray); ByteArrayInputStream bais = new ByteArrayInputStream(primitiveArray); BufferedImage bufferedImage = ImageIO.read(bais); </pre>
Oracle	bufferedImage != null

3.5 UTC 4 - Database

UTC 4.2	Test Database - verifyDBUrl
Test items	verifyDBUrl(Sting url);
Input	Fornisco url valido
Oracle	result == true

UTC 4.3	Test Database - verifyDBUrl
Test items	verifyDBUrl(Sting url);
Input	Fornisco url non valido
Oracle	result == false

UTC 4.4	Test Database - verifyDBUrl
Test items	verifyDBUrl(Sting url);
Input	Fornisco url nullo
Oracle	result == false

UTC 4.5	Test Database - verifyDBUrl
Test items	verifyDBUrl(Sting url);
Input	Fornisco url vuoto
Oracle	result == false

UTC 4.6	Test Database - insertContact
Test items	insertContact(Contact c);
Input	<pre> Creo un contatto di prova con tutti i campi valorizzati: c insertContact(c); </pre>
Oracle	getAllContacts().contains(c) == true;

UTC 4.7	Test Database - removeContact
Test items	removeContact(Contact c);
Input	<pre> Creo un contatto di prova con tutti i campi valorizzati insertContact(c); removeContact(c); </pre>
Oracle	getAllContacts().contains(c) == false;

UTC 4.8	Test Database - insertTag
Test items	insertTag(Tag tag);
Input	<pre> Creo un tag: t insertTag(t); removeTag(t); </pre>
Oracle	getAllTags().contains(t) == true;

UTC 4.9	Test Database - removeTag
Test items	removeTag(Tag tag);
Input	Creo un tag: t insertTag(t); removeTag(t);
Oracle	getAllTags().contains(t) == false;
UTC 4.10	Test Database - getAllContacts
Test items	getAllContacts();
Input	Creo 3 contatti: c1,c2,c3 e li inserisco nel database
Oracle	getAllContacts().contains(c1) == true; getAllContacts().contains(c2) == true; getAllContacts().contains(c3) == true;
UTC 4.11	Test Database - getAllTags
Test items	getAllTags();
Input	Creo 3 tag: t1,t2,t3 e li inserisco nel database
Oracle	getAllTags().contains(t1) == true; getAllTags().contains(t2) == true; getAllTags().contains(t3) == true;
UTC 4.12	Test Database - getAllTags
Test items	getAllTags();
Input	Creo 3 tag: t1,t2,t3 e li inserisco nel database
Oracle	getAllTags().contains(t1) == true; getAllTags().contains(t2) == true; getAllTags().contains(t3) == true;
UTC 4.13	Test Database - insertManyContacts
Test items	insertManyContacts(Collection<Contact> contacts);
Input	Creo 3 contatti: c1,c2,c3 Inserisco i tre contatti in Collection<Contact> lista insertManyContacts(lista);
Oracle	getAllContacts().contains(c1) == true; getAllContacts().contains(c2) == true; getAllContacts().contains(c3) == true;
UTC 4.14	Test Database - insertManyTags
Test items	insertManyTags(Collection<Tag> tags);
Input	Creo 3 tag: t1,t2,t3 Inserisco i tre tag in Collection<Tag> lista insertManyTags(lista);
Oracle	getAllTags().contains(t1) == true; getAllTags().contains(t2) == true; getAllTags().contains(t3) == true;
UTC 4.15	Test Database - deleteAllContacts
Test items	deleteAllContacts();
Input	Creo 3 contatti: c1,c2,c3 Inserisco i tre contatti nel database deleteAllContacts();
Oracle	getAllContacts().contains(c1) == false; getAllContacts().contains(c2) == false; getAllContacts().contains(c3) == false;
UTC 4.16	Test Database - deleteAllTags
Test items	deleteAllTags();
Input	Creo 3 tag: t1,t2,t3 Inserisco i tre tag nel database deleteAllTags();
Oracle	getAllTags().contains(t1) == false; getAllTags().contains(t2) == false; getAllTags().contains(t3) == false;

UTC 4.17	Test Database - contactToDocument
Test items	contactToDocument(Contact c);
Input	Creo un contatto con tutti i campi valorizzati: c Document doc = database.contactToDocument(c);
Oracle	I campi di doc corrispondono a quelli di c

UTC 4.18	Test Database - documentToContact
Test items	documentToContact(Document d);
Input	Creo un documento con i campi di contatto: doc Contact c = database.documentToContact(doc);
Oracle	I campi di doc corrispondono a quelli di c

UTC 4.19	Test Database - tagToDocument
Test items	tagToDocument(Tag tag);
Input	Creo un tag: t Document doc = database.tagToDocument(t);
Oracle	I campi di doc corrispondono a quelli di t

UTC 4.20	Test Database - documentToTag
Test items	documentToTag(Document d);
Input	Creo un documento con i campi di tag: doc Tag t = database.documentToTag(doc);
Oracle	I campi di doc corrispondono a quelli di t

3.6 UTC 5 - Tag

UTC 5.2	Test Tag - Constructor
Test items	Tag(String description, int id)
Input	Tag tag = new Tag("tag",10)
Oracle	tag != NULL tag.getId() == 10

UTC 5.4	Test Tag - getDescription
Test items	getDescription() Tag(String tag)
Input	Tag tag = new Tag("tag")
Oracle	tag.getDescription == "tag"

UTC 5.6	Test Tag - setIndex
Test items	setIndex(int index) Tag tag = new Tag("tag")
Input	Tag.setIndex(11); Tag tag = new Tag("tag")
Oracle	tag.getId() == 11

UTC 5.7	Test Tag - equals
Test items	Tag tag = new Tag("tag") Tag tag = new Tag("tag",10)
Input	Tag tag1 = new Tag("tag") Tag tag2 = new Tag("tag") Tag tag3 = new Tag("tag") Tag tag4 = new Tag("tag")
Oracle	tag1 == tag2 tag3 == tag4