

INGEGNERIA DEL SOFTWARE - A.A. 2025/2026

Applicazione per la gestione di una Biblioteca universitaria

# Progettazione del Software

*Luisa Genovese*

*Erica Brancaccio*

*Paolo Alfé*

*Francesco Altieri*

15 dicembre 2025

## Indice

1 Progettazione del Software	2
1.1 Decomposizione in moduli . . . . .	2
1.1.1 Descrizione delle classi individuate . . . . .	2
1.1.2 Diagramma delle classi . . . . .	4
1.1.3 Diagramma dei package (essenziale) . . . . .	5
1.1.4 Diagramma dei package (completo) . . . . .	6
1.2 Diagrammi di sequenza . . . . .	7
1.2.1 C4: Inserimento di un libro . . . . .	7
1.2.2 C5: Modifica di un libro . . . . .	8
1.2.3 C6: Rimozione di un libro . . . . .	9
1.2.4 C7: Inserimento di un utente . . . . .	10
1.2.5 C8: Modifica di un utente . . . . .	11
1.2.6 C9: Rimozione di un utente . . . . .	12
1.2.7 C10: Ricerca di un libro . . . . .	13
1.2.8 C12: Ricerca di un utente . . . . .	14
1.2.9 C13: Registrazione di un prestito . . . . .	15
1.2.10 C14: Registrazione di una restituzione . . . . .	16
1.2.11 C15: Estensione di un prestito . . . . .	17
1.3 Principi di buona progettazione . . . . .	18
1.3.1 Livelli di Coesione . . . . .	18
1.3.2 Livelli di Accoppiamento . . . . .	19
1.3.3 Analisi della scalabilità . . . . .	20
1.3.4 Strategie per la riduzione dell'Accoppiamento . . . . .	20

# 1 Progettazione del Software

## 1.1 Decomposizione in moduli

### 1.1.1 Descrizione delle classi individuate

Il diagramma delle classi rappresenta il sistema della biblioteca, con le funzionalità di gestione (ovvero di aggiunta, modifica e rimozione dall'archivio) dei libri, degli utenti e dei prestiti, nonché quella di lettura/salvataggio di dati all'atto dell'apertura/chiusura dell'applicazione.

- **Biblioteca:** classe centrale dell'applicazione, che modella una biblioteca. Mantiene e gestisce le tre collezioni principali mediante gli attributi **libri**, **utenti** e **prestiti**. Fornisce i metodi per la serializzazione e deserializzazione (salvataggio e lettura) dei dati su file;
- **Prestiti:** classe che modella un insieme di prestiti. Possiede l'attributo **prestiti** (una collezione di oggetti di tipo **Prestito**). Implementa l'interfaccia **Archiviabile**, ereditando i metodi fondamentali per la manipolazione della collezione. Fornisce un metodo per la visualizzazione dei prestiti (getter) e un metodo di filtraggio per visualizzare solo i prestiti che rispettano il criterio selezionato;
- **Utenti:** classe che modella un insieme di utenti. Possiede gli attributi **chiaviMatricole** e **utenti** (due collezioni di oggetti di tipo **Utente**). Implementa le interfacce **Archiviabile** e **Mappabile**, ereditando i metodi fondamentali per la manipolazione della collezione. Fornisce un metodo per la visualizzazione degli utenti (getter) e un metodo per effettuare una ricerca all'interno della collezione;
- **Libri:** classe che modella un insieme di libri. Possiede gli attributi **chiaviISBN** e **libri** (due collezioni di oggetti di tipo **Libro**). Implementa le interfacce **Archiviabile** e **Mappabile**, ereditando i metodi fondamentali per la manipolazione della collezione. Fornisce un metodo per la visualizzazione dei libri (getter) e un metodo per effettuare una ricerca all'interno della collezione;
- **Archiviabile:** interfaccia parametrizzata implementata dalle classi **Prestiti**, **Utenti** e **Libri**. Definisce i tre metodi per le operazioni fondamentali di aggiunta, modifica e rimozione dell'elemento passato come parametro;
- **Mappabile:** interfaccia parametrizzata implementata dalle classi **Utenti** e **Libri**. Definisce i metodi per l'accesso e la ricerca di un valore attraverso una chiave;
- **Filtro:** classe enumerativa che definisce tre criteri di filtraggio: **TUTTI**, **ATTIVI** e **CONCLUSI**. Questa classe è utilizzata per l'applicazione di criteri di selezione sull'insieme dei prestiti;
- **Prestito:** classe che modella un prestito. I suoi attributi sono: **matricolaUtente**, **codiceISBNLibro**, **dataInizio**, **dataScadenza** e **dataRestituzione**. Fornisce vari metodi per la modifica e visualizzazione di questi ultimi (getter e setter) e per la verifica della correttezza del formato dei dati;
- **Utente:** classe che specializza **Persona**. Modella un utente del sistema. I suoi attributi sono: **matricolaUtente**, **email** e **prestitiAttivi** (una collezione di oggetti di tipo **Prestito**). Fornisce vari metodi per la modifica e visualizzazione di questi ultimi (getter e setter), per la gestione della collezione di prestiti e per la verifica della correttezza del formato dei dati;
- **Libro:** classe che modella un libro. I suoi attributi sono: **titolo**, **autori** (una collezione di oggetti di tipo **Autore**), **annoPubblicazione**, **codiceISBNLibro** e **copieDisponibili**.

Fornisce vari metodi per la modifica e visualizzazione di questi ultimi (getter e setter), per la gestione della collezione di autori e per la verifica della correttezza del formato dei dati;

- **Matricola:** classe che modella una matricola (ovvero il codice identificativo univoco) associata ad un utente. Possiede l'attributo `matricola`, il metodo per la sua visualizzazione (getter) e un metodo per verificare la correttezza del proprio formato;
- **ISBN:** classe che modella un codice ISBN (ovvero il codice identificativo univoco) associato ad un libro. Possiede l'attributo `codiceISBN`, il metodo per la sua visualizzazione (getter) e un metodo per verificare la correttezza del proprio formato;
- **Autore:** classe che specializza Persona. Modella l'entità autore associata ad un libro.
- **Persona:** classe astratta che funge da superclasse per Utente e Autore. Definisce gli attributi fondamentali di una persona, ovvero `nome` e `cognome`. Fornisce metodi per la loro gestione (getter e setter) e per la verifica della correttezza del loro formato.

All'interno del diagramma viene omessa l'implementazione dell'interfaccia `Serializable` al fine di aumentare la propria leggibilità. Sudetta interfaccia dovrà essere implementata dalle classi: Biblioteca, Prestiti, Libri, Utenti, Prestito, Libro, Utente, Autore, Matricola e ISBN.

Per lo stesso motivo, all'interno del diagramma viene omessa anche l'implementazione dell'interfaccia `Comparable`, che sarà implementata dalle classi: Prestito, Libro, Matricola, ISBN e Persona (con le proprie specializzazioni).

### 1.1.2 Diagramma delle classi

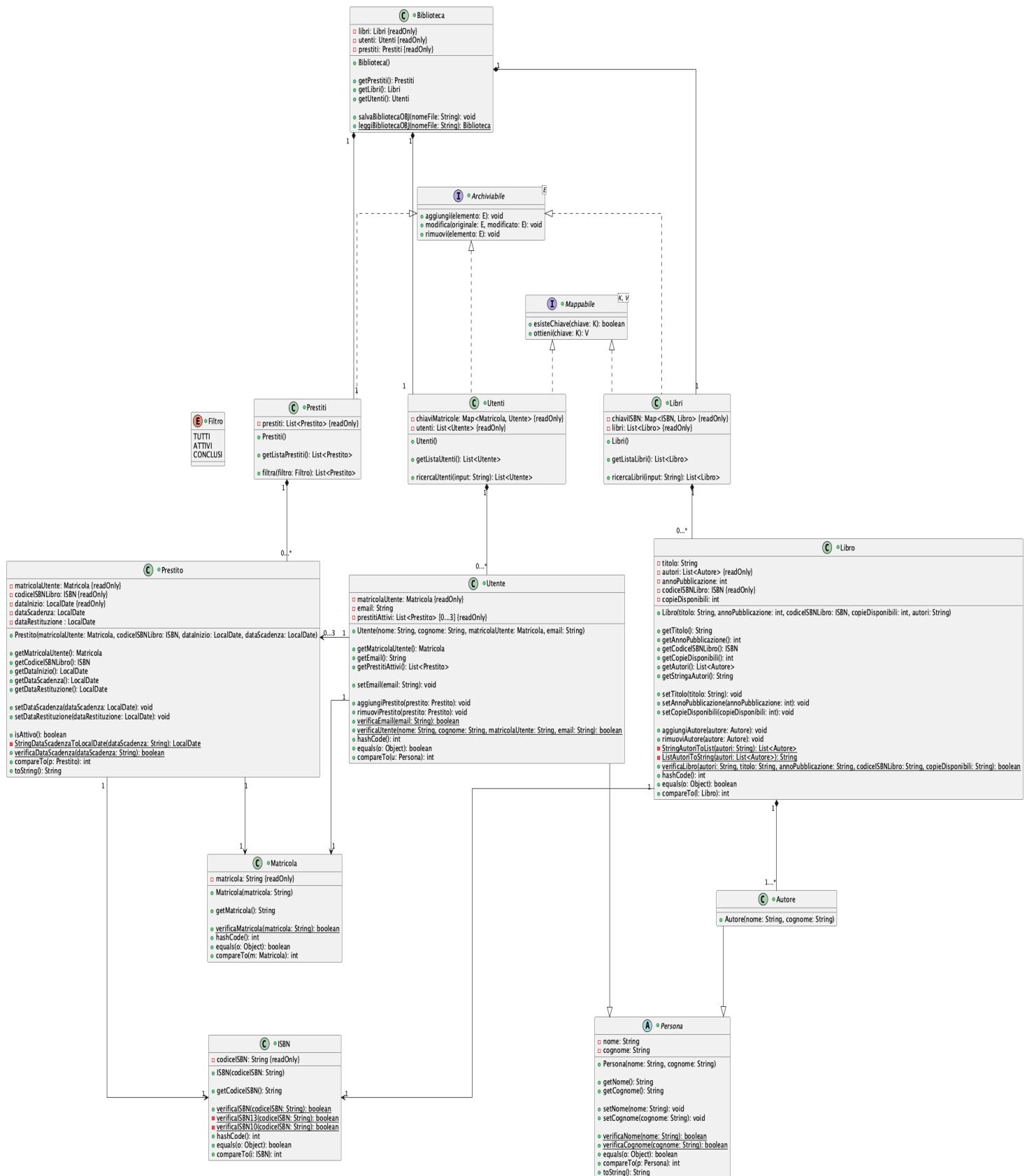


Figura 1: Diagramma delle Classi

### 1.1.3 Diagramma dei package (essenziale)

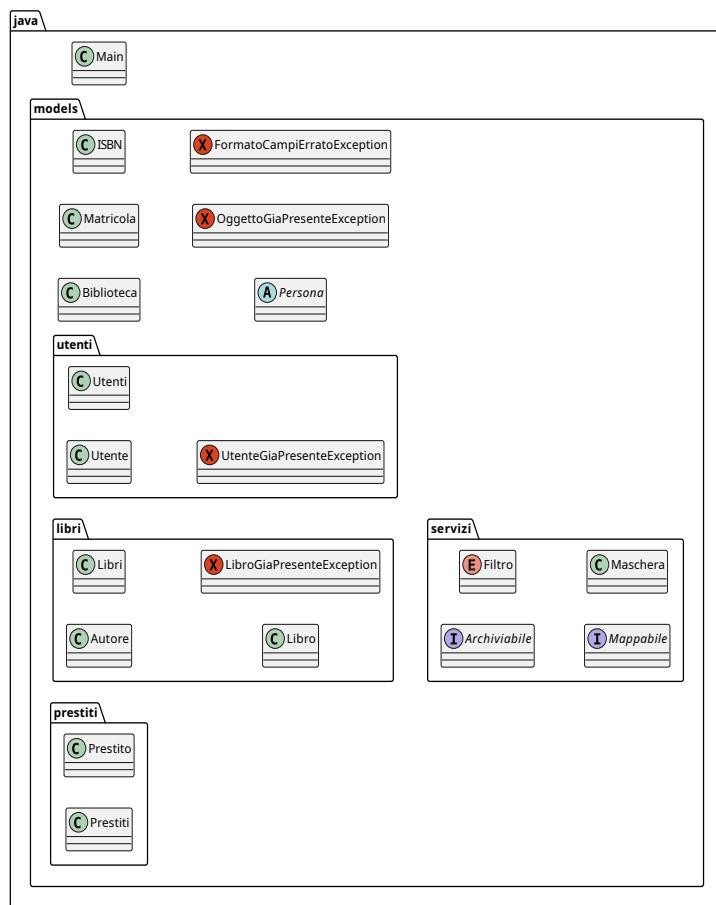


Figura 2: Diagramma dei Package relativo ai modelli

### 1.1.4 Diagramma dei package (completo)

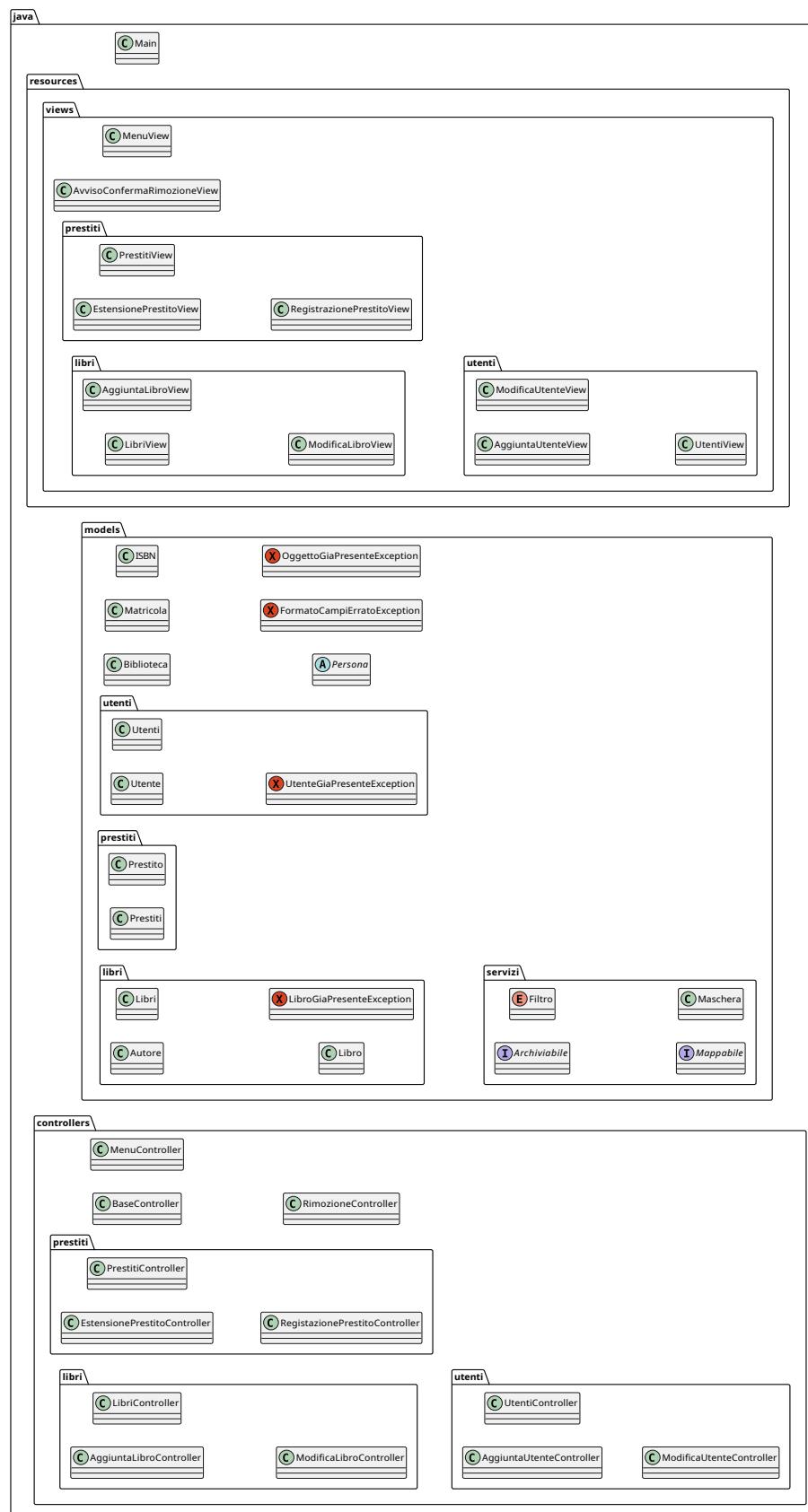


Figura 3: Diagramma dei Package completo (inклuse le classi Controller e View)

## 1.2 Diagrammi di sequenza

Di seguito vengono presentati dei diagrammi di sequenza che descrivono i flussi esecutivi dei casi d'uso più rilevanti, utili per la comprensione delle funzionalità principali del sistema. Questi diagrammi descrivono l'interazione tra l'attore *Amministratore* e gli oggetti del sistema coinvolti, ai fini di soddisfare i requisiti specificati.

### 1.2.1 C4: Inserimento di un libro

Il seguente diagramma descrive l'esecuzione del caso d'uso C4.

Per migliorare la leggibilità del diagramma, è stato omesso il flusso alternativo di annullamento dell'operazione tramite la pressione del pulsante "Annulla".

Per il flusso alternativo di inserimento errato o incompleto, è stata utilizzata la sintassi **loop-break**. Invece, per la descrizione del flusso alternativo relativo all'esistenza pregressa del libro all'interno dell'archivio, è stata utilizzata la sintassi **alt**.

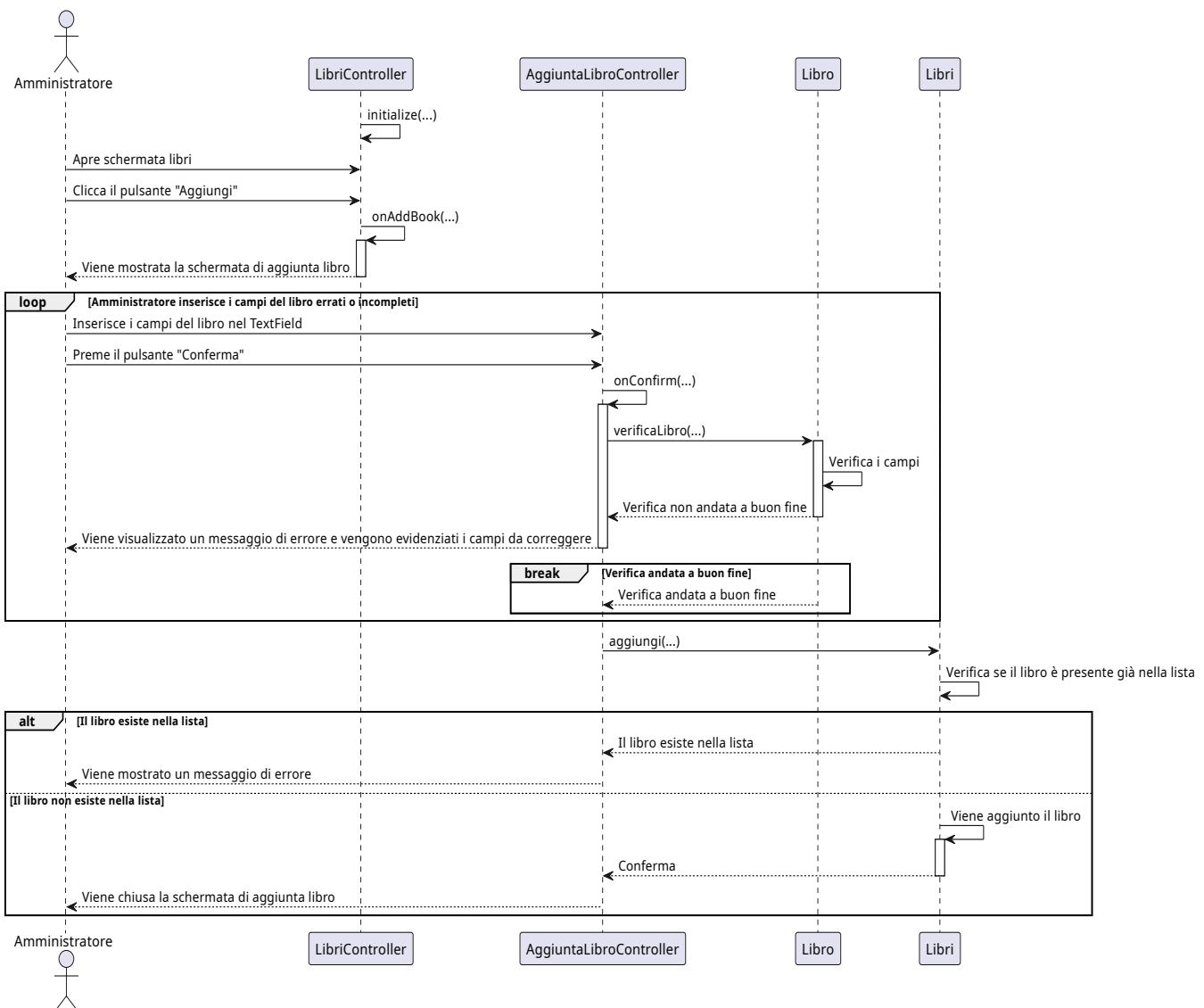


Figura 4: Diagramma di sequenza C4

### 1.2.2 C5: Modifica di un libro

Il seguente diagramma descrive l'esecuzione del caso d'uso C5.

Per migliorare la leggibilità del diagramma, è stato omesso il flusso alternativo di annullamento dell'operazione tramite la pressione del pulsante "Annulla".

Per il flusso alternativo relativo ad una modifica errata o incompleta è stata utilizzata la sintassi **loop-break**.

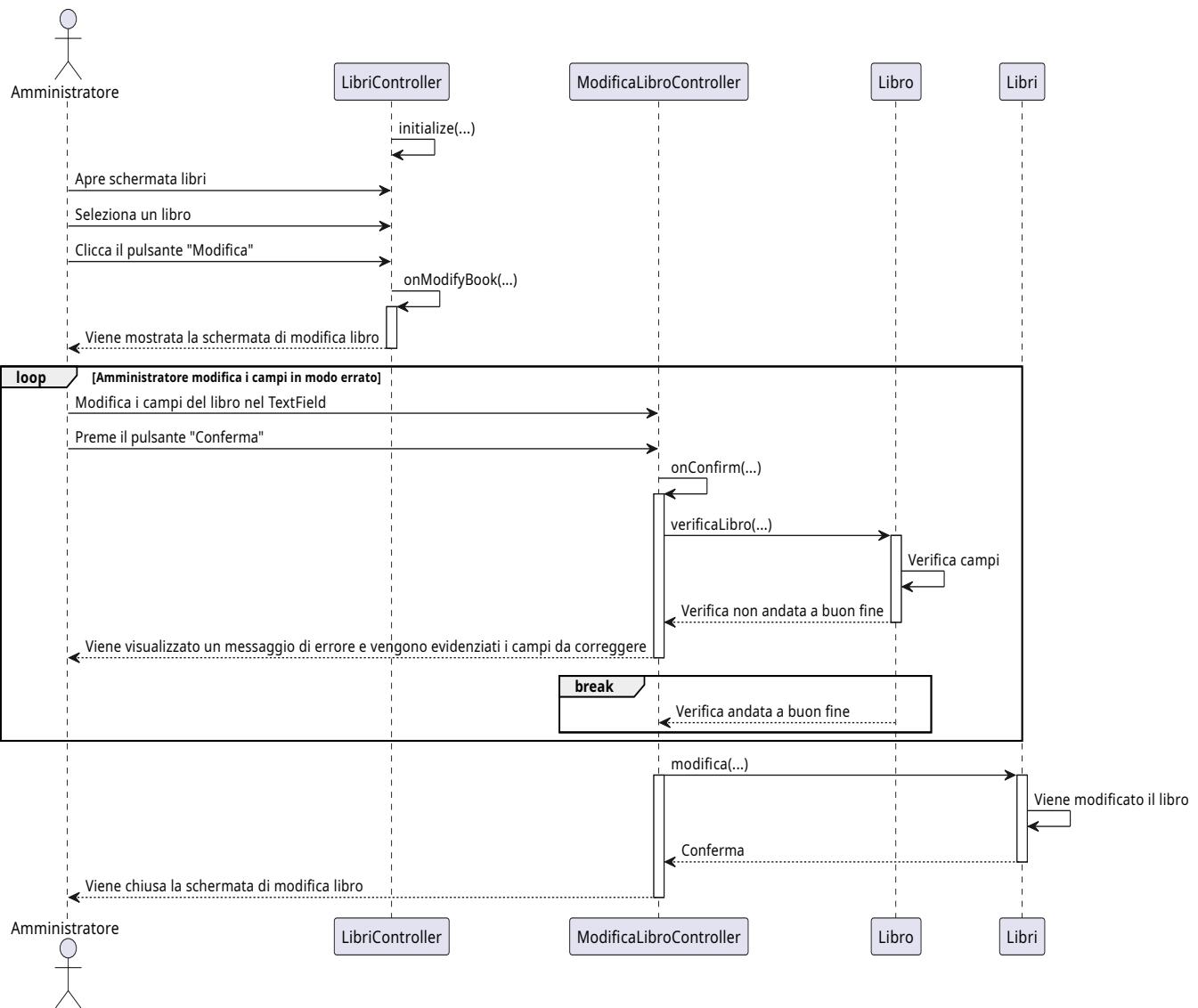


Figura 5: Diagramma di sequenza C5

### 1.2.3 C6: Rimozione di un libro

Il seguente diagramma descrive l'esecuzione del caso d'uso C6.

Per il flusso alternativo di annullamento dell'operazione tramite la pressione del pulsante "No" è stata utilizzata la sintassi `alt`.

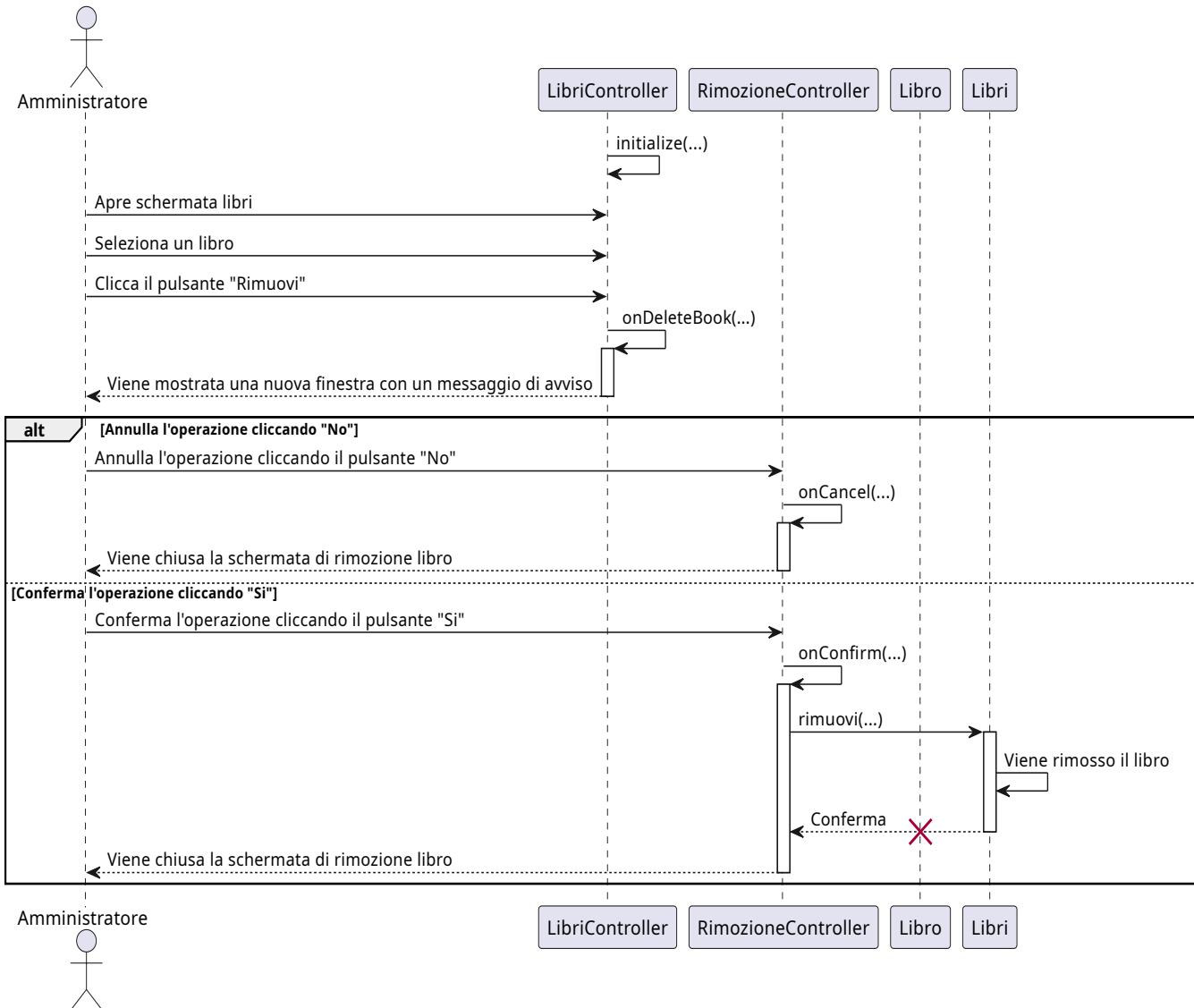


Figura 6: Diagramma di sequenza C6

### 1.2.4 C7: Inserimento di un utente

Il seguente diagramma descrive l'esecuzione del caso d'uso C7.

Per migliorare la leggibilità del diagramma, è stato omesso il flusso alternativo di annullamento dell'operazione tramite la pressione del pulsante "Annulla".

Per il flusso alternativo di inserimento errato o incompleto, è stata utilizzata la sintassi **loop-break**. Invece, per la descrizione del flusso alternativo relativo all'esistenza pregressa dell'utente all'interno dell'archivio, è stata utilizzata la sintassi **alt**.

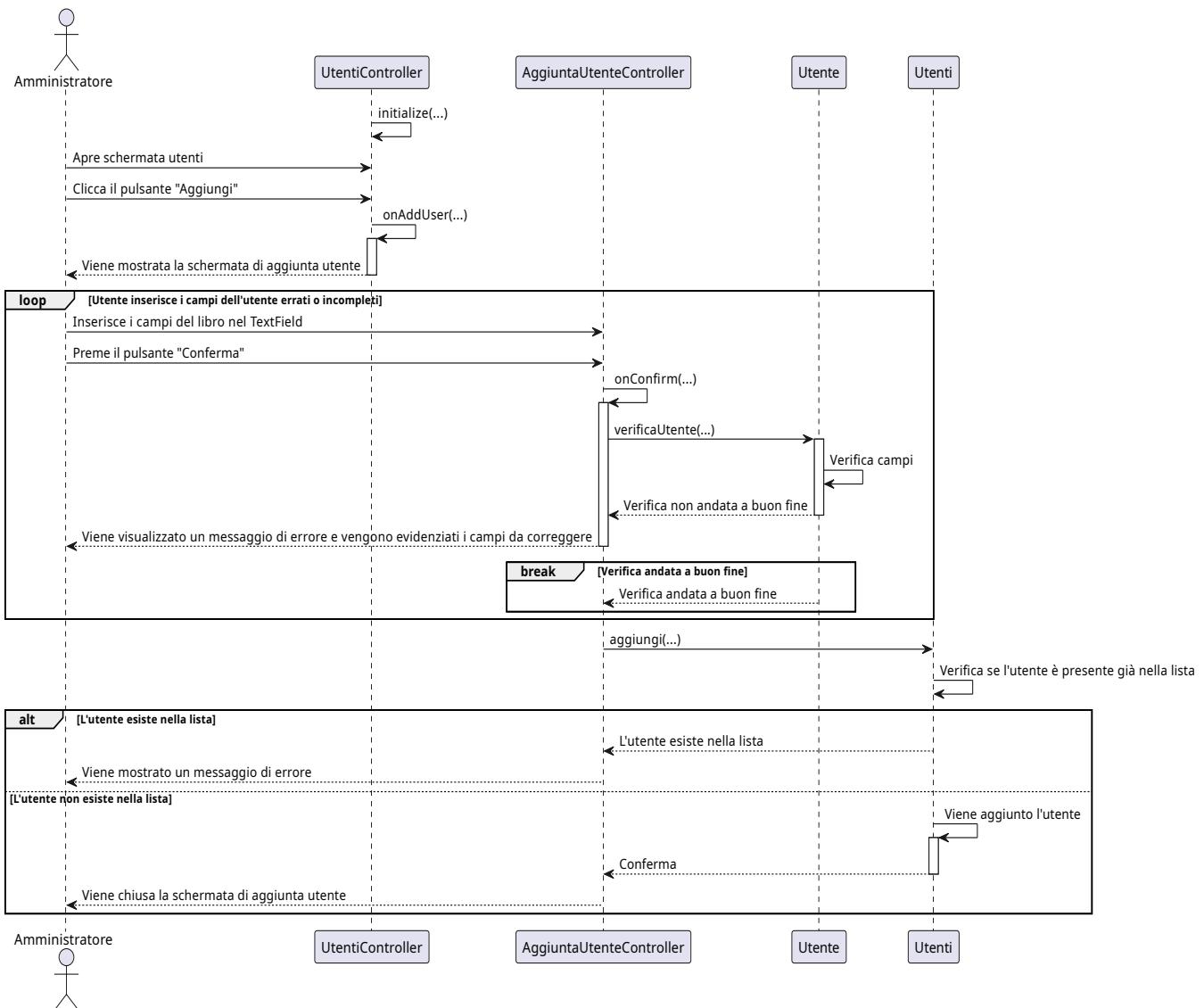


Figura 7: Diagramma di sequenza C7

### 1.2.5 C8: Modifica di un utente

Il seguente diagramma descrive l'esecuzione del caso d'uso C8.

Per migliorare la leggibilità del diagramma, è stato omesso il flusso alternativo di annullamento dell'operazione tramite la pressione del pulsante "Annulla".

Per il flusso alternativo relativo ad una modifica errata o incompleta è stata utilizzata la sintassi **loop-break**.

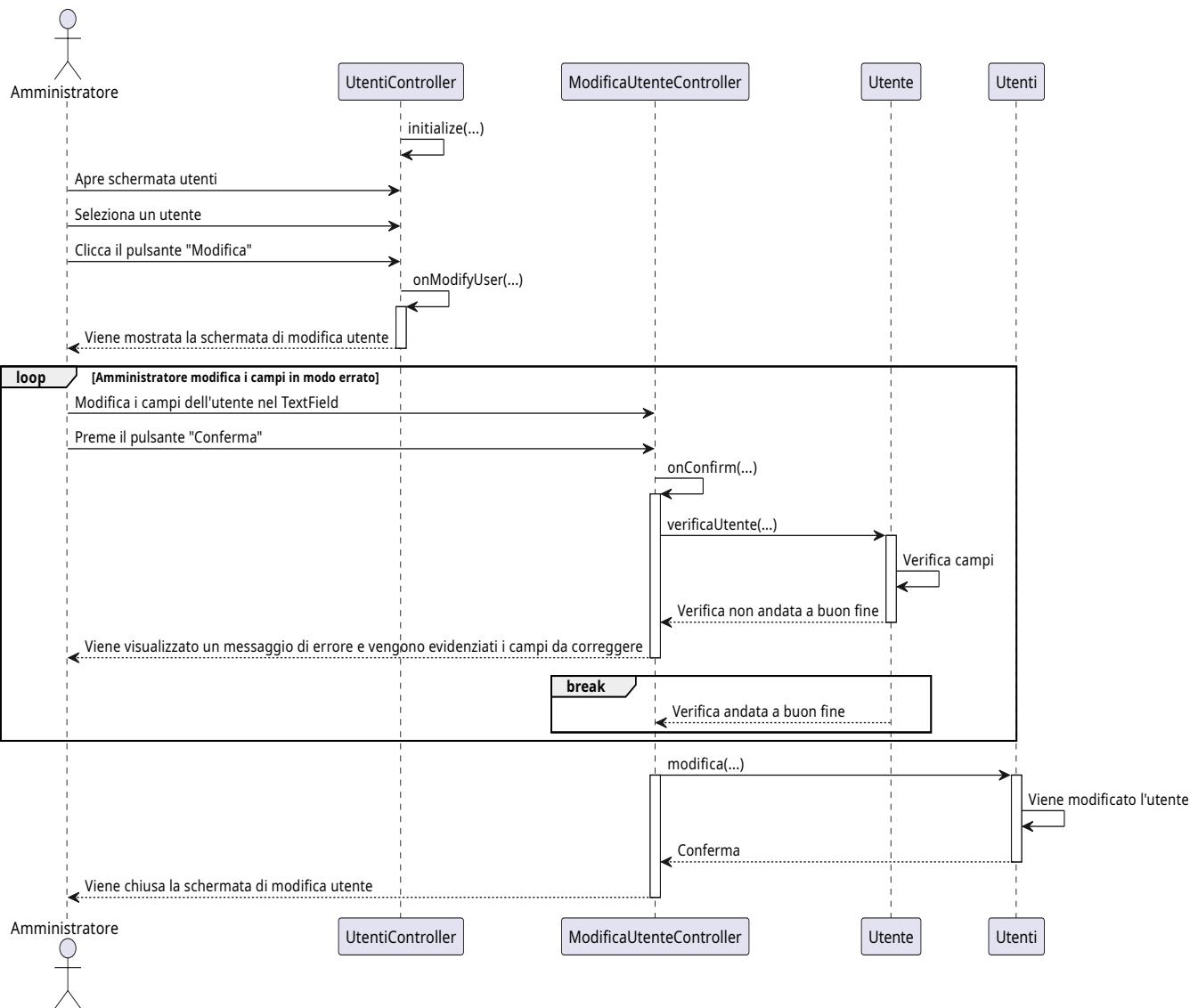


Figura 8: Diagramma di sequenza C8

### 1.2.6 C9: Rimozione di un utente

Il seguente diagramma descrive l'esecuzione del caso d'uso C9.

Per il flusso alternativo di annullamento dell'operazione tramite la pressione del pulsante "No" è stata utilizzata la sintassi `alt`.

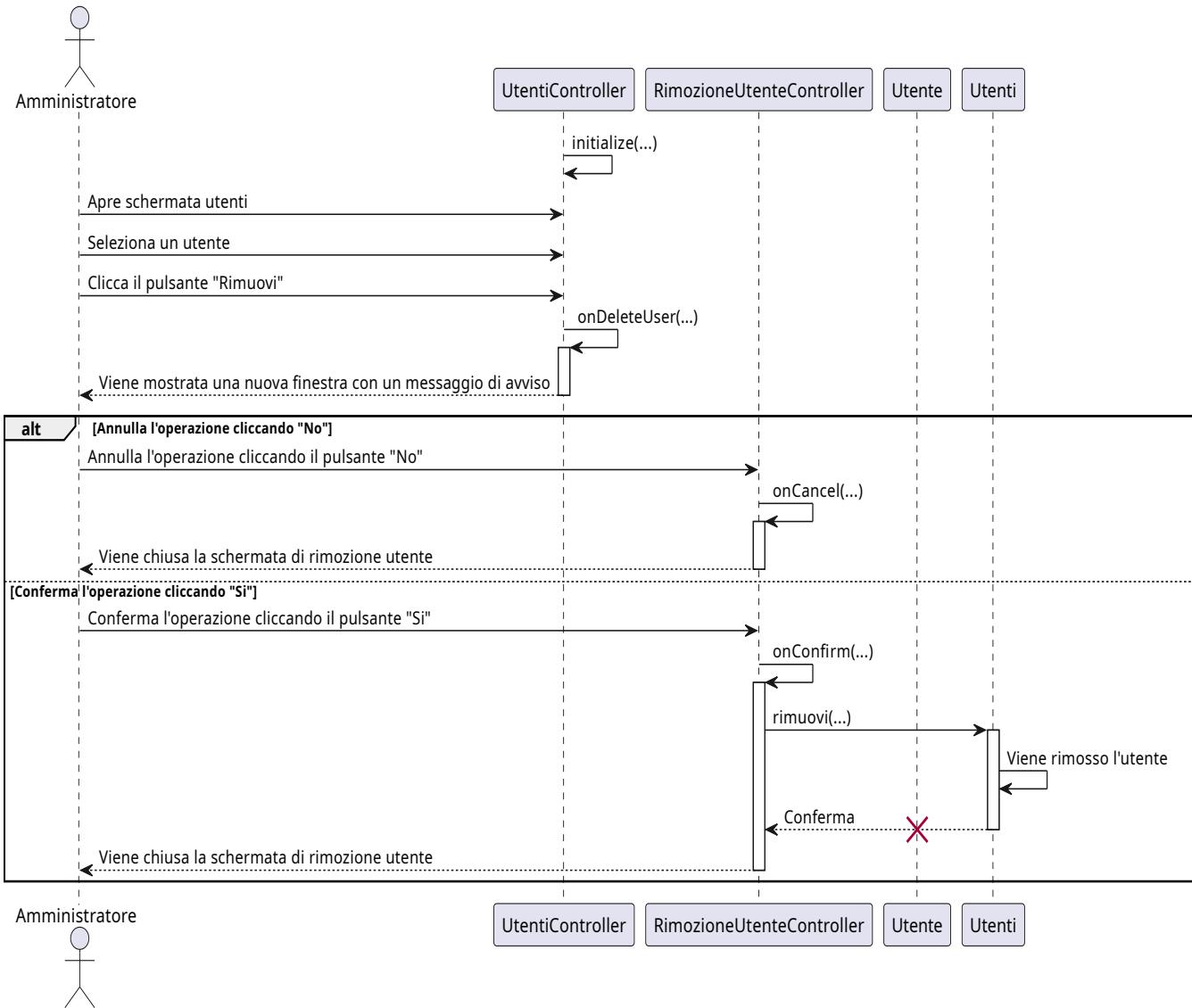


Figura 9: Diagramma di sequenza C9

### 1.2.7 C10: Ricerca di un libro

Il seguente diagramma descrive l'esecuzione del caso d'uso C10.

All'interno del diagramma è stato scelto di utilizzare la sintassi **loop-break** per simulare la ricerca mediante l'inserimento di una sottostringa nella barra di ricerca.

Ogni volta che viene aggiunto un carattere alla stringa, verrà effettuata una ricerca in tempo reale e verranno visualizzati tutti i libri i cui campi presenteranno almeno una corrispondenza con la sottostringa inserita.

L'operazione terminerà quando l'amministratore selezionerà un libro.

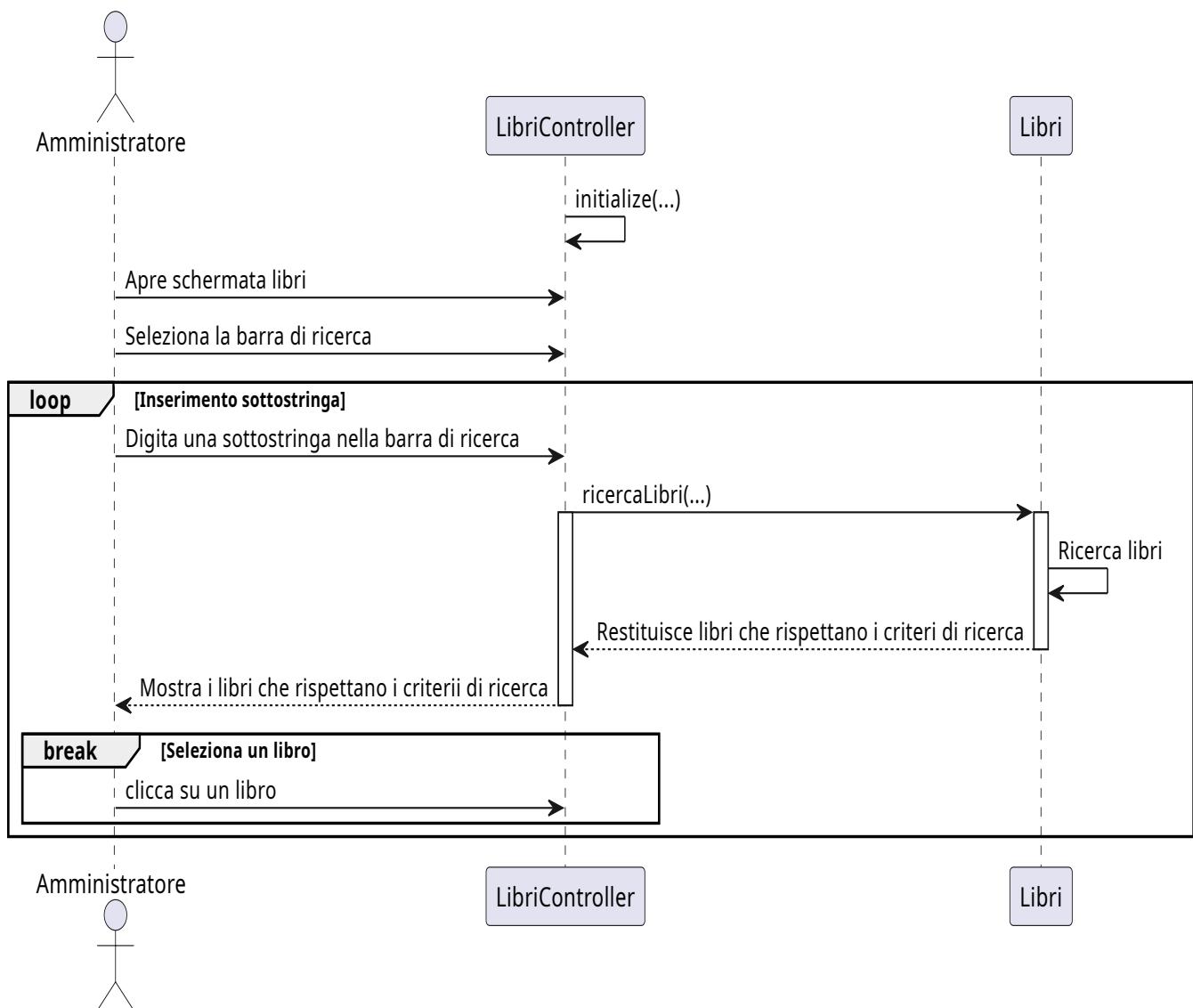


Figura 10: Diagramma di sequenza C10

### 1.2.8 C12: Ricerca di un utente

Il seguente diagramma descrive l'esecuzione del caso d'uso C12.

All'interno del diagramma è stato scelto di utilizzare la sintassi `loop-break` per simulare la ricerca mediante l'inserimento di una sottostringa nella barra di ricerca.

Ogni volta che viene aggiunto un carattere alla stringa, verrà effettuata una ricerca in tempo reale e verranno visualizzati tutti gli utenti i cui campi presenteranno almeno una corrispondenza con la sottostringa inserita.

L'operazione terminerà quando l'amministratore selezionerà un utente.

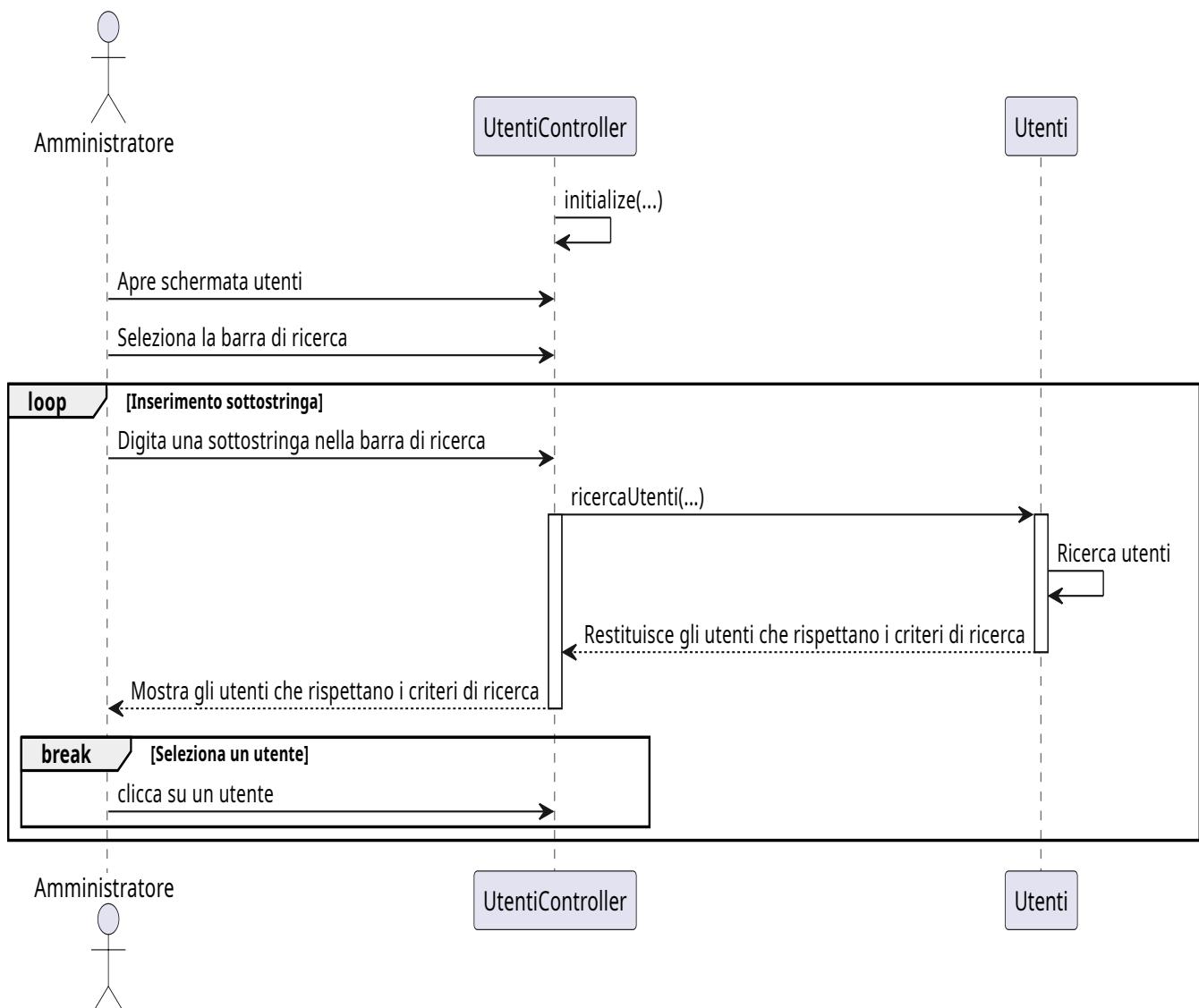


Figura 11: Diagramma di sequenza C12

### 1.2.9 C13: Registrazione di un prestito

Il seguente diagramma descrive l'esecuzione del caso d'uso C13.

Per migliorare la leggibilità del diagramma, è stato omesso il flusso alternativo di annullamento dell'operazione tramite la pressione del pulsante "Annulla".

Per il flusso alternativo di inserimento errato o incompleto è stata utilizzata la sintassi **loop-break**. Invece, per la descrizione del flusso alternativo relativo all'inesistenza dell'utente e del libro all'interno dell'archivio, è stata utilizzata la sintassi **alt**.

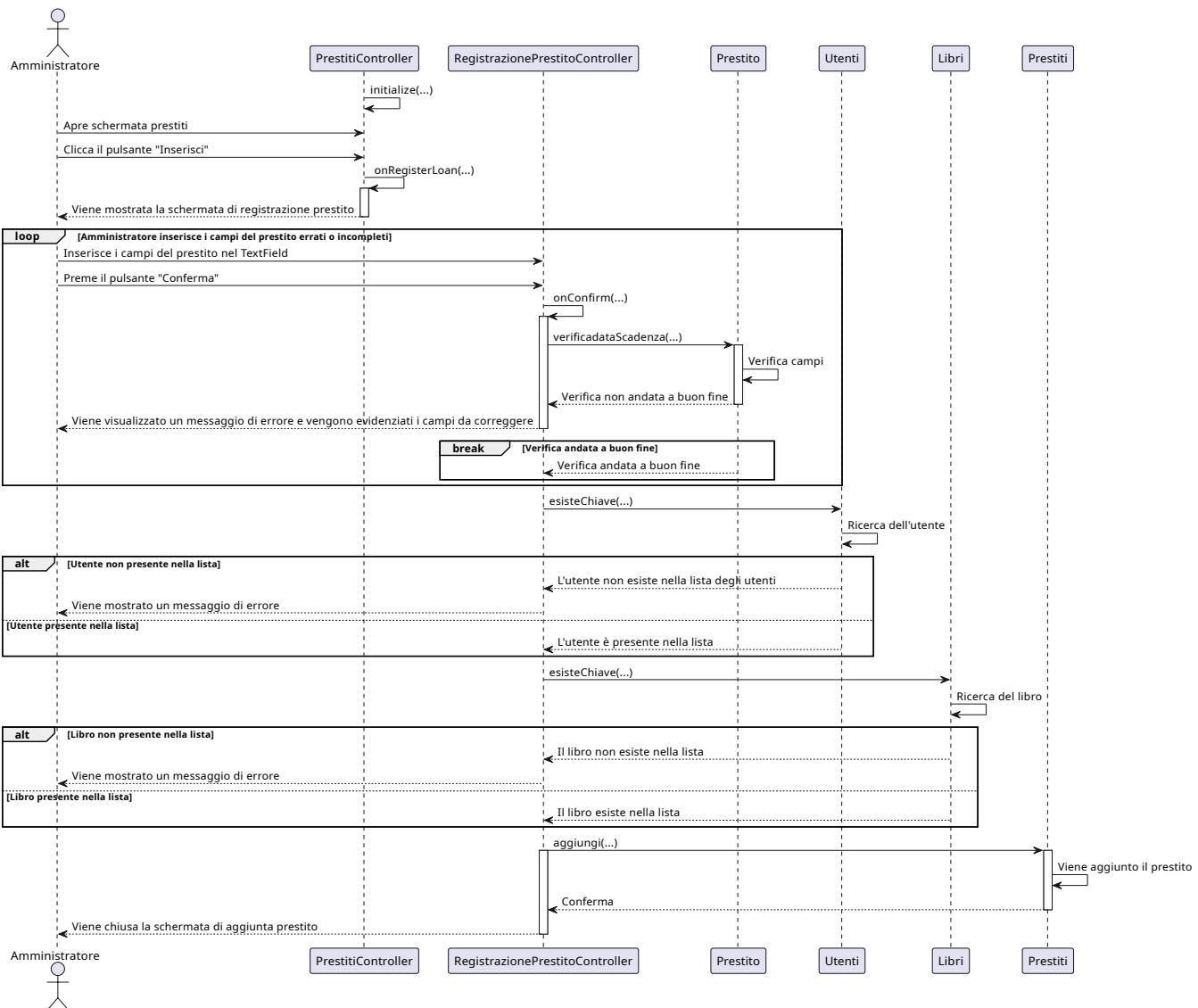


Figura 12: Diagramma di sequenza C13

### 1.2.10 C14: Registrazione di una restituzione

Il seguente diagramma descrive l'esecuzione del caso d'uso C14.

Per il flusso alternativo di annullamento dell'operazione tramite la pressione del pulsante "No" è stata utilizzata la sintassi `alt`.

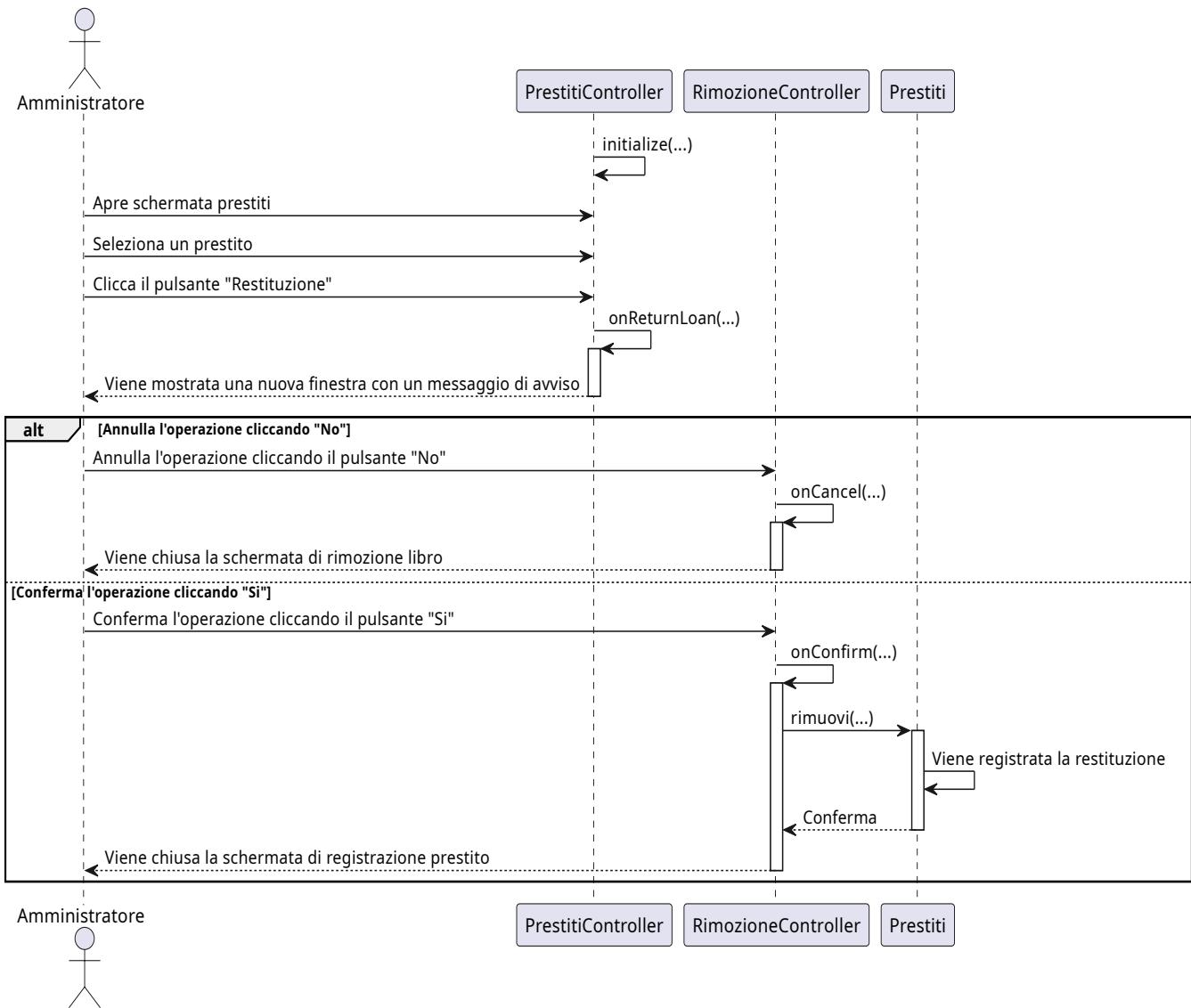


Figura 13: Diagramma di sequenza C14

### 1.2.11 C15: Estensione di un prestito

Il seguente diagramma descrive l'esecuzione del caso d'uso C15.

Per migliorare la leggibilità del diagramma, è stato omesso il flusso alternativo di annullamento dell'operazione tramite la pressione del pulsante "Annulla".

Per il flusso alternativo relativo ad una modifica errata o incompleta è stata utilizzata la sintassi **loop-break**.

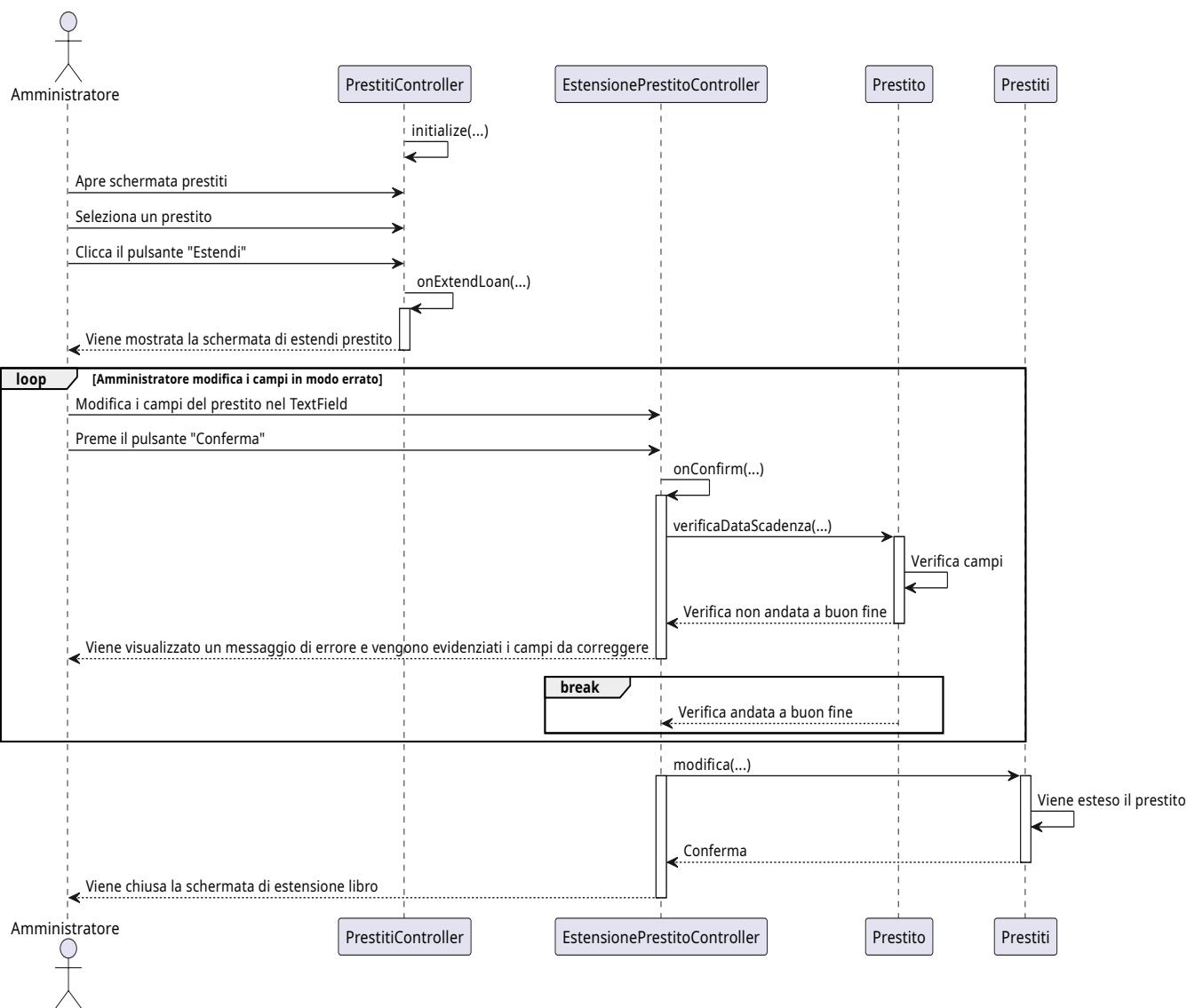


Figura 14: Diagramma di sequenza C15

## 1.3 Principi di buona progettazione

La progettazione di dettaglio dell'applicazione è stata portata a termine tenendo a mente i principi di buona progettazione, garantendo un'alta ortogonalità, manutenibilità, modularità e chiarezza.

Di seguito riportiamo le tabelle con i livelli di coesione e di accoppiamento individuati tra i moduli del nostro sistema:

### 1.3.1 Livelli di Coesione

Nome Classe	Livello di coesione	Nota
Biblioteca	<i>Coesione Comunicazionale</i>	La classe Biblioteca si occupa soltanto delle operazioni di salvataggio e lettura. Queste due funzionalità lavorano sullo stesso dato.
Libri	<i>Coesione Funzionale</i>	Tutti i metodi contenuti nella classe Libri si richiamano a vicenda per portare a termine un unico compito, ovvero la manipolazione di una lista di oggetti di tipo Libro.
Utenti	<i>Coesione Funzionale</i>	Tutti i metodi contenuti nella classe Utenti si richiamano a vicenda per portare a termine un unico compito, ovvero la manipolazione di una lista di oggetti di tipo Utente.
Prestiti	<i>Coesione Funzionale</i>	Tutti i metodi contenuti nella classe Prestiti si richiamano a vicenda per portare a termine un unico compito, ovvero la manipolazione di una lista di oggetti di tipo Prestito.
Libro	<i>Coesione Funzionale</i>	Tutti i metodi contenuti nella classe Libro servono per la visione e la modifica degli attributi e la verifica della correttezza del loro formato.
Utente	<i>Coesione Funzionale</i>	Tutti i metodi contenuti nella classe Utente servono per la visione e la modifica degli attributi e la verifica della correttezza del loro formato.
Prestito	<i>Coesione Funzionale</i>	Tutti i metodi contenuti nella classe Prestito servono per la visione e la modifica degli attributi e la verifica della correttezza del loro formato.
Autore	<i>Coesione Funzionale</i>	Essendo identica alla sua superclasse Persona, eredita il suo livello di coesione.
ISBN	<i>Coesione Funzionale</i>	Tutti i metodi contenuti nella classe ISBN servono per la visione e la modifica degli attributi.
Matricola	<i>Coesione Funzionale</i>	Tutti i metodi contenuti nella classe Matricola servono per la visione e la modifica degli attributi.
Persona	<i>Coesione Funzionale</i>	Tutti i metodi contenuti nella classe Persona servono per la visione e la modifica degli attributi.

Figura 15: Tabella dei livelli di coesione dei moduli

### 1.3.2 Livelli di Accoppiamento

Nome Classe	Nome Classe	Livello di accoppiamento	Nota
Biblioteca	Libri	<i>Accoppiamento per dati</i>	La classe Biblioteca ha un attributo di tipo Libri. Dato che la relazione che lega le classi è di composizione, l'oggetto di tipo Libri verrà istanziato all'interno della classe Biblioteca, richiamandone il costruttore.
Biblioteca	Utenti	<i>Accoppiamento per dati</i>	La classe Biblioteca ha un attributo di tipo Utenti. Dato che la relazione che lega le classi è di composizione, l'oggetto di tipo Utenti verrà istanziato all'interno della classe Biblioteca, richiamandone il costruttore.
Biblioteca	Prestiti	<i>Accoppiamento per dati</i>	La classe Biblioteca ha un attributo di tipo Prestiti. Dato che la relazione che lega le classi è di composizione, l'oggetto di tipo Prestiti verrà istanziato all'interno della classe Biblioteca, richiamandone il costruttore.
Libri	Libro	<i>Accoppiamento per dati</i>	La classe Libri contiene una collezione di oggetti di tipo Libro, che vengono istanziati al suo interno. Usufruisce dei metodi di Libro per garantire il funzionamento della classe.
Utenti	Utente	<i>Accoppiamento per dati</i>	La classe Utenti contiene una collezione di oggetti di tipo Utente, che vengono istanziati al suo interno. Usufruisce dei metodi di Utente per garantire il funzionamento della classe.
Prestiti	Prestito	<i>Accoppiamento per dati</i>	La classe Prestiti contiene una collezione di oggetti di tipo Prestito, che vengono istanziati al suo interno. Usufruisce dei metodi di Prestito per garantire il funzionamento della classe.
Libro	ISBN	<i>Accoppiamento per dati</i>	La classe Libro mantiene come attributo un riferimento ad un oggetto di tipo ISBN e lo utilizza all'interno dei propri metodi.
Libro	Autore	<i>Accoppiamento per dati</i>	La classe Libro mantiene come attributo un riferimento a una lista di oggetti di tipo Autore e lo utilizza all'interno dei propri metodi.
Utente	Matricola	<i>Accoppiamento per dati</i>	La classe Utente mantiene come attributo un riferimento ad un oggetto di tipo Matricola e lo utilizza all'interno dei propri metodi.
Utente	Persona	<i>Accoppiamento per contenuti</i>	Dato che Utente è una specializzazione di Persona, la sua implementazione dipenderà direttamente da quella della sua superclasse.
Utente	Prestito	<i>Accoppiamento per dati</i>	La classe Utente mantiene come attributo un riferimento a una lista di oggetti di tipo Prestito e lo utilizza all'interno dei propri metodi.
Prestito	Matricola	<i>Accoppiamento per dati</i>	La classe Prestito mantiene come attributo un riferimento ad un oggetto di tipo Matricola e lo utilizza all'interno dei propri metodi.
Prestito	ISBN	<i>Accoppiamento per dati</i>	La classe Prestito mantiene come attributo un riferimento ad un oggetto di tipo ISBN e lo utilizza all'interno dei propri metodi.
Autore	Persona	<i>Accoppiamento per contenuti</i>	Dato che Autore è una specializzazione di Persona, la sua implementazione dipenderà direttamente da quella della sua superclasse.

Figura 16: Tabella dei livelli di accoppiamento tra i moduli

Tutte le classi sono state progettate in accordo con il principio di *Separazione delle preoccupazioni*: ogni modulo contiene solo ed esclusivamente attributi e metodi che hanno a che fare con la propria funzionalità.

Ciò garantisce di ottenere, nella maggior parte dei casi, il più alto livello di coesione possibile (*coesione funzionale*).

Fin dalle prime fasi della progettazione, vista la natura stessa dell'applicazione è emerso un insieme di funzionalità comuni alle principali collezioni di dati (Prestiti, Libri ed Utenti).

Per evitare ridondanze, e rispettare quanto affermato dal principio di progettazione *DRY (Don't Repeat Yourself)*, tali funzionalità sono state astratte ed inserite nelle interfacce *Archiviabile* e *Mappabile*.

Le classi che implementano tali interfacce si preoccupano di ridefinire il comportamento specifico dei metodi ereditati da esse, mentre la loro funzionalità di base resterà sempre la stessa. In questo modo è possibile ottenere un codice chiaro (secondo quanto affermato dal principio *KISS - Keep It Simple*) e facilmente manutenibile.

### 1.3.3 Analisi della scalabilità

Per aumentare la scalabilità del sistema, sarebbe stato possibile rendere la classe *Utente* astratta, e successivamente specializzarla nelle diverse tipologie di utenti contemplati dall'applicazione.

È stato scelto di non adottare questa strategia per due motivi:

1. Un'ulteriore relazione di specializzazione avrebbe comportato un alto livello di accoppiamento;
2. Attualmente, il nostro sistema è in grado di gestire un'unica tipologia di utenti, ovvero gli studenti, quindi sarebbe stata un'aggiunta pressoché inutile (è stato seguito il principio *YAGNI - You Aren't Going to Need It*).

### 1.3.4 Strategie per la riduzione dell'Accoppiamento

Utilizzando ampliamente meccanismi come l'*incapsulamento* e l'*astrazione* mediante l'introduzione di interfacce, è stato possibile ridurre al minimo possibile l'accoppiamento tra i moduli, nonché aumentare la robustezza generale del sistema.

Uno degli aspetti su cui sarebbe possibile effettuare delle ottimizzazioni è l'uso dell'*ereditarietà*: per questioni semantiche (e dettate dalla natura del linguaggio di programmazione utilizzato, ovvero *Java*), la struttura del nostro sistema prevede due relazioni di specializzazione (*Utente* → *Persona* e *Autore* → *Persona*), che portano con sé un livello di accoppiamento estremamente elevato (*accoppiamento per contenuti*).

Una possibile soluzione potrebbe essere quella di predilire l'*associazione* all'*ereditarietà*: anziché far dipendere l'implementazione di *Utente* interamente da quella di *Persona*, si potrebbe pensare di far mantenere a *Utente* un riferimento ad una classe avente come attributi *nome* e *cognome* (e che potrebbe prendere il nome di "Anagrafica"), riducendo di molto l'accoppiamento.