Sistema di Gestione Biblioteca Universitaria

Design

Data: 14 Dicembre 2025
Versione: 2.0
Autori:

Alfonso Aufiero
Mattia Gerardo Bavaro
Andrea Botta
Christian Salvatore Bove

IV. Introduzione al Design

1 Scopo del documento

Il presente documento descrive la progettazione del *Sistema di Gestione di una Biblioteca Universitaria,* sulla base dei requisiti definiti nello **SRS - Specifica dei Requisiti Software.**
L'obiettivo è la definizione, in modo strutturato e verificabile, delle soluzioni architetturali e di dettaglio necessarie alla realizzazione del sistema, assicurando coerenza con i requisiti funzionali, non-funzionali, i casi d'uso e i vincoli tecnologici stabiliti.

Il documento è organizzato secondo due livelli di dettaglio:

- **Progettazione Architetturale (*High-Level Design - HLD*):**
  Definisce la vista d'insieme del sistema, i macro-componenti, le loro responsabilità e le interazioni principali.
  Questa fase è guidata principalmente dai requisiti non funzionali e dagli attributi di qualità attesi.

- **Progettazione di Dettaglio (Low-Level Design - LLD):**
  Approfondisce ogni componente, descrive le classi, gli attributi, le operazioni, le relazioni e i contratti formali.
  Questa fase è guidata dai requisiti funzionali e dai casi d'uso.

Il documento è destinato a sviluppatori, progettisti software e verificatori, e costituisce la base formale per l'implementazione del sistema oltre che per la successiva fase di testing.

V. High-Level Design (HLD)

2 Attributi di qualità (QA)

La seguente sezione elenca gli attributi di qualità che il nostro sistema dovrebbe rispettare.

2.1 QA Esterni

Gli attributi di qualità esterni riguardano le proprietà visibili agli utilizzatori del sistema.

- **Disponibilità:** Il sistema deve essere sempre accessibile e operativo.
- **Efficienza:** Il sistema deve rispettare i vincoli di risposta.
- **Safety:** Il sistema garantisce l'integrità dei dati.
- **Usabilità:** Il sistema deve garantire un flusso di operazioni scorrevole e user-friendly.

2.1 QA Interni

Gli attributi di qualità interni riguardano le proprietà di interesse degli sviluppatori del sistema.

- **Manutenibilità:** il sistema deve garantire la possibilità di ricevere aggiornamenti e miglioramenti.
- **Modularità:** il sistema deve essere composto da moduli funzionali garantendo che la modifica del singolo abbia il minimo impatto sugli altri.
- **Riusabilità:** i componenti del sistema devono poter essere riutilizzati in un sistema diverso con minima difficoltà.
- **Portabilità:** il sistema deve supportare l'esecuzione su sistemi operativi diversi.
- **Testabilità:** i moduli che compongono il sistema devono essere facilmente testabili singolarmente o nell'insieme.

3.1 Project Skeleton (Maven-style)

```
CSS
java.swe.group04.libraryms
├── Main.java
├── models
│   ├── LibraryArchive.java
│   └── Book.java
│   └── User.java
│   └── Loan.java
├── exceptions
│   ├── MandatoryFieldsException.java
│   ├── InvalidEmailException.java
│   ├── InvalidIsbnException.java
│   ├── NoAvailableCopiesException.java
│   ├── MaxLoansReachedException.java
│   └── UserHasActiveLoansException.java
├── service
│   ├── LibraryArchiveService.java
│   ├── BookService.java
│   ├── ServiceLocator.java
│   ├── UserService.java
│   └── LoanService.java
├── persistence
│   ├── FileService.java
│   └── ArchiveFileService.java
├── controllers
│   ├── MainController.java
│   ├── BookController.java
│   ├── BookDetailsController.java
│   ├── AddBookController.java
│   ├── UserListController.java
│   ├── UserDetailsController.java
│   ├── AddUserController.java
│   ├── LoanListController.java
└──└── RegisterLoanController.java

resources.swe.group04.libraryms
├── css
└── view
```

Il Project Skeleton *non definisce package logici*, ma la struttura dei file.

3.2 Diagramma dei Package

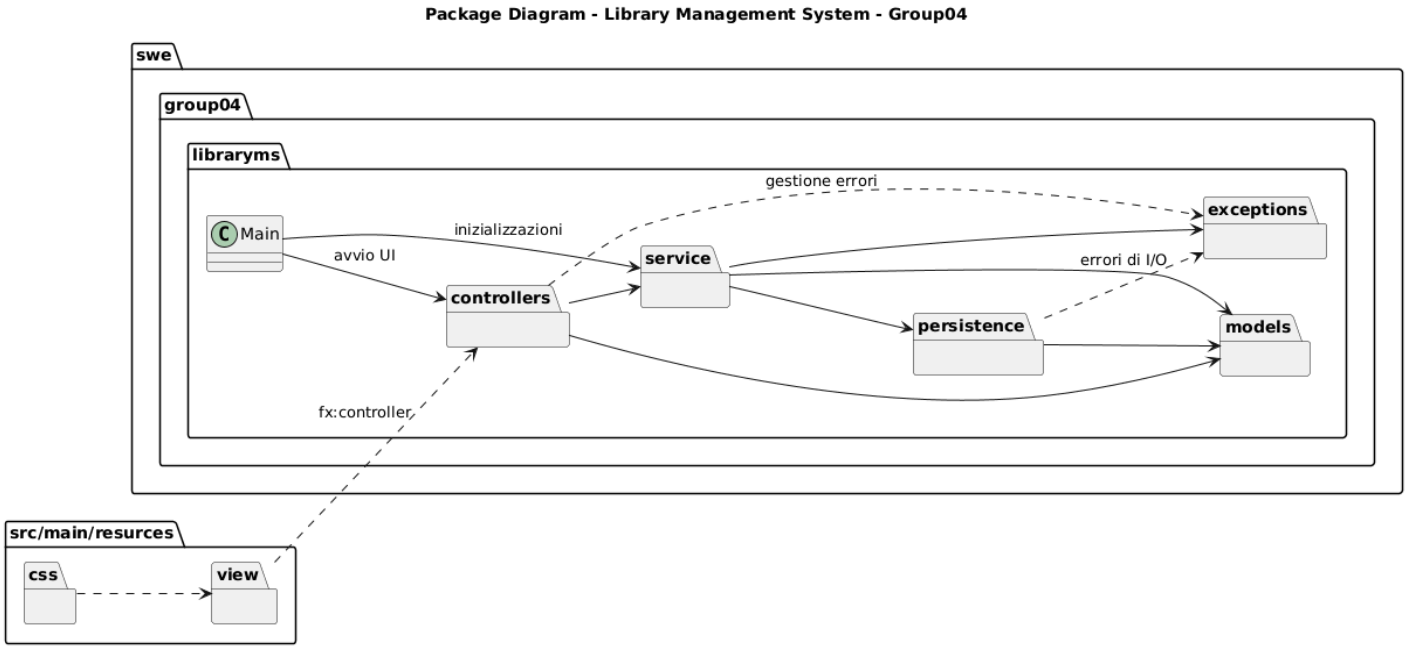

figura 1

Il diagramma dei package descrive la struttura del sistema attraverso le principali macro-funzioni implementate, organizzate secondo i principi di buona progettazione.

L'architettura rispetta il modello **MVC (*Model-View-Controller*)**, garantendo una chiara separazione tra logica applicativa, gestione della UI e rappresentazione dei dati.

Il principio di **Single Responsibility** è applicato assegnando a ciascun package un ruolo specifico:

- i *controller* gestiscono l'interazione con l'interfaccia utente,
- i *service* incapsulano la logica applicativa,
- i *modelli* rappresentano il dominio,
- il package di *persistenza* è responsabile della memorizzazione dei dati,
- le *view FXML* e i fogli di stile *CSS* definiscono la parte grafica dell'applicazione.

In accordo con il principio di **Separation of Concerns (SoC)**, problemi e responsabilità differenti sono organizzati in package differenti, rendendo il sistema più comprensibile, modulare e facilmente manutenibile.

Le dipendenze tra i package seguono un flusso gerarchico e unidirezionale, favorendo **basso accoppiamento** tra i componenti e **alta coesione** all'interno di ciascun package.

I *controller* dipendono dai *service*, questi ultimi dai *modelli* e dai componenti di *persistenza*, mentre le *view* dipendono esclusivamente dai *controller* tramite i riferimenti definiti nei file FXML.

Non si generano dipendenze cicliche e ogni livello interagisce solo con quello immediatamente sottostante, riducendo la complessità complessiva del sistema.

VI. Low-Level Design (LLD)

4.1 Diagramma delle classi

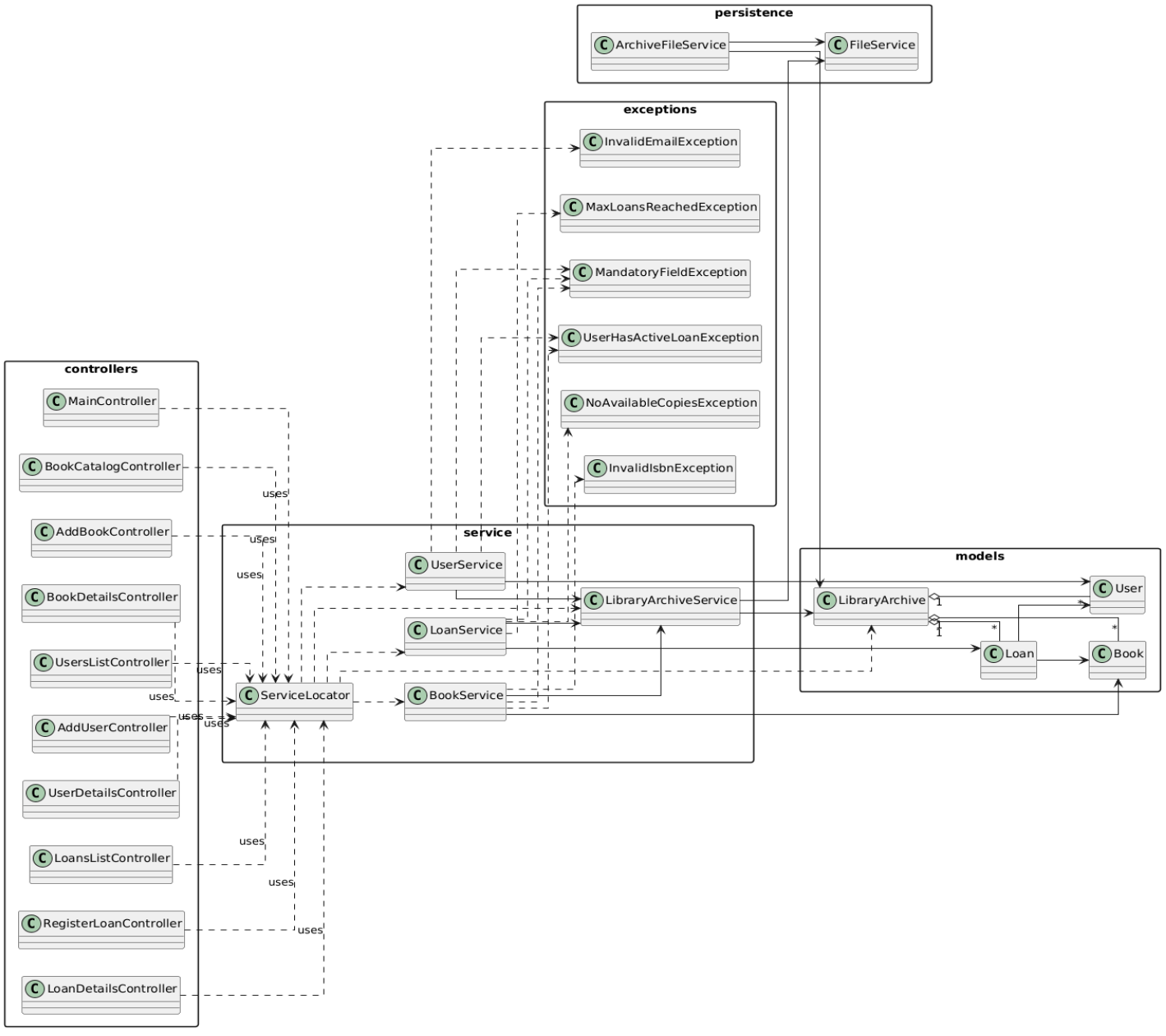

figura 2

La figura 2 rappresenta il diagramma completo delle classi. Per una scelta stilistica e di visibilità il diagramma è rappresentato a un livello di dettaglio ***concettuale*** e precede:

- diagrammi con un livello di dettaglio maggiore (3.1, 3.2);
- descrizione ed analisi approfondita delle classi dei package *models* e *service.*

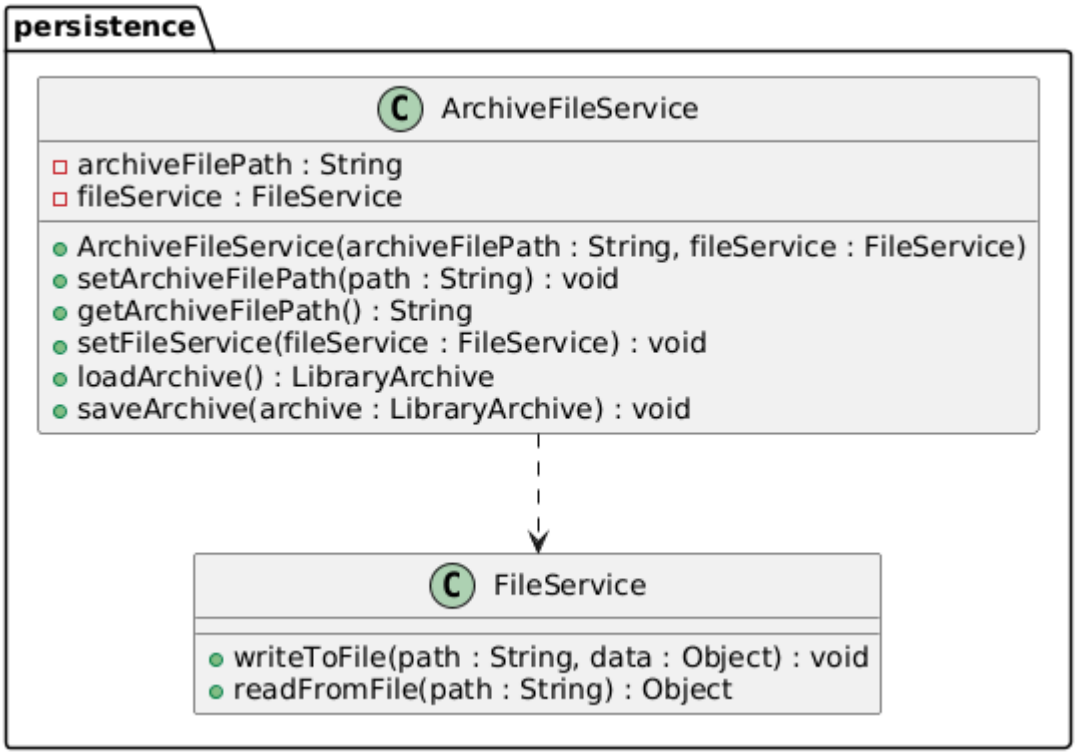

figura 3.1

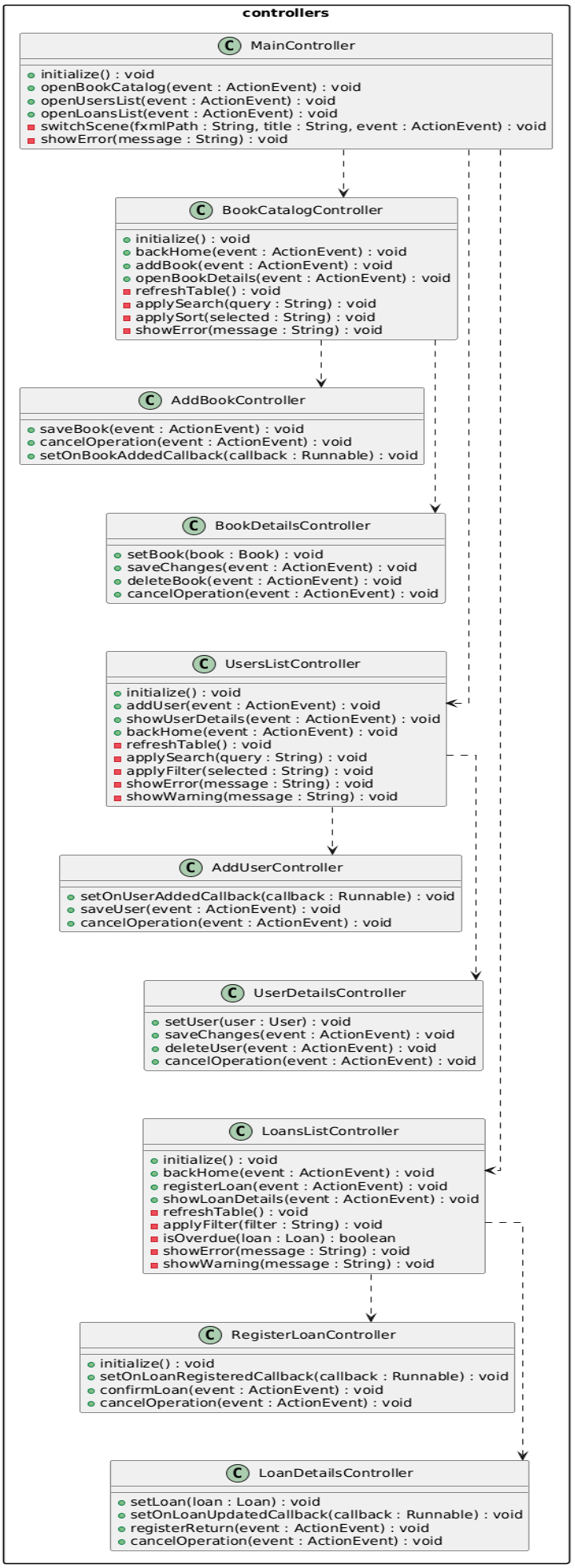
controllers
MainController
+ initialize() : void
+ openBookCatalog(event : ActionEvent) : void
+ openUsersList(event : ActionEvent) : void
+ openLoansList(event : ActionEvent) : void
- switchScene(fxmlPath : String, title : String, event : ActionEvent) : void
- showError(message : String) : void
BookCatalogController
+ initialize() : void
+ backHome(event : ActionEvent) : void
+ addBook(event : ActionEvent) : void
+ openBookDetails(event : ActionEvent) : void
- refreshTable() : void
- applySearch(query : String) : void
- applySort(selected : String) : void
- showError(message : String) : void
AddBookController
+ saveBook(event : ActionEvent) : void
+ cancelOperation(event : ActionEvent) : void
+ setOnBookAddedCallback(callback : Runnable) : void
BookDetailsController
+ setBook(book : Book) : void
+ saveChanges(event : ActionEvent) : void
+ deleteBook(event : ActionEvent) : void
+ cancelOperation(event : ActionEvent) : void
UsersListController
+ initialize() : void
+ addUser(event : ActionEvent) : void
+ showUserDetails(event : ActionEvent) : void
+ backHome(event : ActionEvent) : void
- refreshTable() : void
- applySearch(query : String) : void
- applyFilter(selected : String) : void
- showError(message : String) : void
- showWarning(message : String) : void
AddUserController
+ setOnUserAddedCallback(callback : Runnable) : void
+ saveUser(event : ActionEvent) : void
+ cancelOperation(event : ActionEvent) : void
UserDetailsController
+ setUser(user : User) : void
+ saveChanges(event : ActionEvent) : void
+ deleteUser(event : ActionEvent) : void
+ cancelOperation(event : ActionEvent) : void
LoansListController
+ initialize() : void
+ backHome(event : ActionEvent) : void
+ registerLoan(event : ActionEvent) : void
+ showLoanDetails(event : ActionEvent) : void
- refreshTable() : void
- applyFilter(filter : String) : void
- isOverdue(loan : Loan) : boolean
- showError(message : String) : void
- showWarning(message : String) : void
RegisterLoanController
+ initialize() : void
+ setOnLoanRegisteredCallback(callback : Runnable) : void
+ confirmLoan(event : ActionEvent) : void
+ cancelOperation(event : ActionEvent) : void
LoanDetailsController
+ setLoan(loan : Loan) : void
+ setOnLoanUpdatedCallback(callback : Runnable) : void
+ registerReturn(event : ActionEvent) : void
+ cancelOperation(event : ActionEvent) : void

figura 3.2

4.2 Descrizione ed analisi delle classi

Si analizzano le varie classi implementate in termini di coesione e accoppiamento. È bene ricordare che per ottenere una buona decomposizione è buona norma ottenere **alta coesione** e **basso accoppiamento**

4.2.1 Classe “Book”

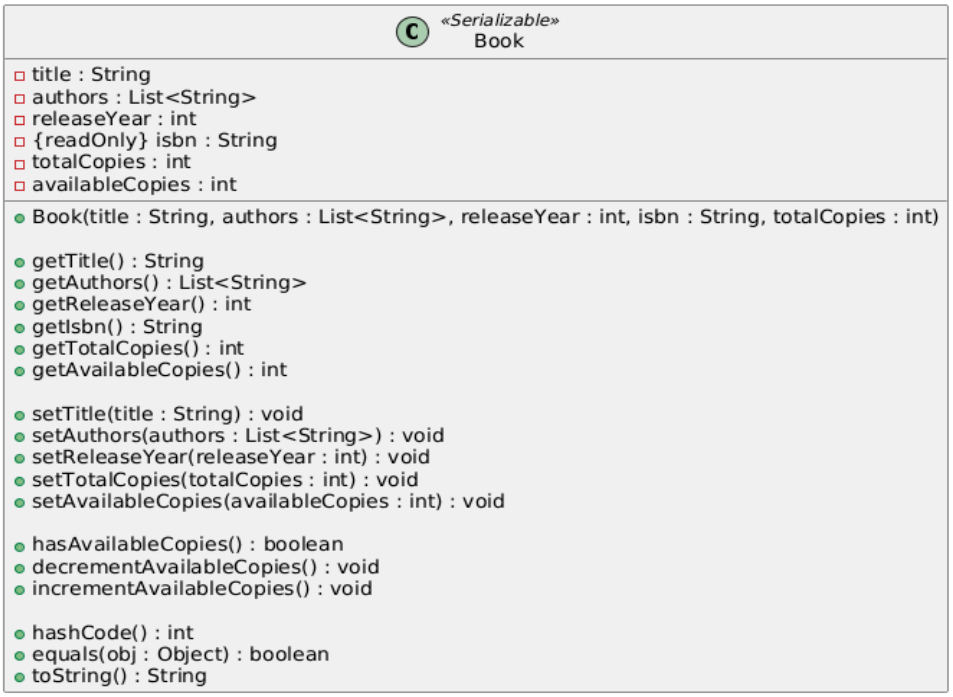

figura 4

La classe *Book* rappresenta il modello di dominio incaricato di descrivere un libro presente nell’archivio della biblioteca.
Gestisce esclusivamente i dati anagrafici e inventariali del libro (titolo, autori, anno di pubblicazione, ISBN, copie totali e copie disponibili).
Ogni metodo contribuisce all’obiettivo generale di mantenere lo stato coerente di un singolo libro, senza svolgere operazioni che coinvolgano altre entità: per questo motivo il livello di **coesione** della classe è **funzionale** (elevato).
La classe *Book* è **accoppiata** alle sole strutture dati contenute nei suoi attributi (*String, List<String>*), e non dipende da componenti esterne come controller, service o moduli di persistenza.
L’unico accoppiamento esplicito verso il dominio è rappresentato dalle relazioni con *Loan* e con *Library Archive*. Si tratta quindi di un **accoppiamento per dati**, limitato al minimo necessario per la rappresentazione del dominio, e classificabile come **basso**.

4.2.2 Classe “User”

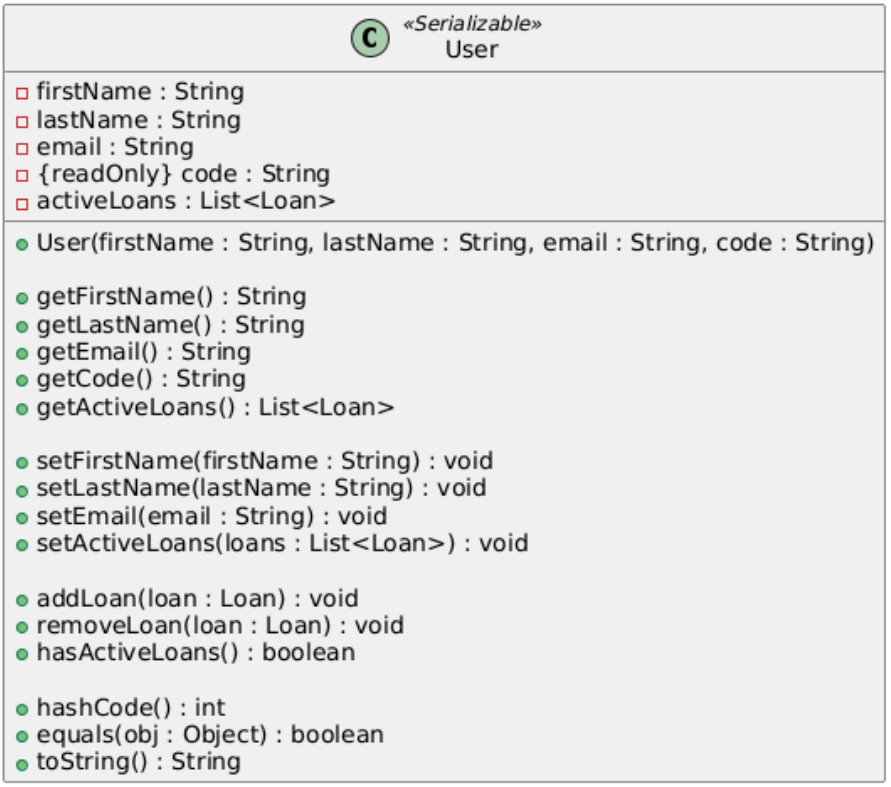

figura 5

La classe ***User*** modella un utente della biblioteca, identificato in modo univoco da un codice e descritto attraverso i suoi dati anagrafici essenziali.
Inoltre, mantiene l’elenco dei prestiti attivi associati all’utente, permettendo di verificarne rapidamente lo stato.
Le operazioni implementate sono tutte orientate alla gestione coerente di queste informazioni, motivo per cui la **coesione** della classe è **funzionale**: essa si occupa esclusivamente della rappresentazione dell’utente e del suo legame con i prestiti attivi, senza integrare logiche di business o responsabilità estranee.
L’**accoppiamento** della classe è **basso** e si limita alla dipendenza dal tipo *Loan*, necessaria per rappresentare l’associazione utente–prestito. Non esistono riferimenti a service, controller o componenti di persistenza, e la classe non interagisce con parti esterne del sistema se non attraverso i propri dati. Si tratta quindi di un **accoppiamento per dati**.

4.2.3 Classe “Loan”

figura 6

La classe ***Loan*** modella un prestito tra un utente e un libro, mantenendo *loanId*, riferimenti a *Book* e *User*, date prevista ed effettiva di restituzione e stato del prestito. I suoi metodi si limitano a leggere e aggiornare questi dati, senza alcuna logica di business: la responsabilità della classe è esclusivamente la rappresentazione coerente di un singolo prestito.

La **coesione** è quindi **funzionale** ed elevata. L’**accoppiamento** è **per dati (**basso) e consiste solo nelle dipendenze strutturali da *Book*, *User* e dai tipi necessari (*LocalDate, LoanStatus*), senza interazioni con controller, servizi o persistenza.

4.2.4 Classe “LibraryArchive”

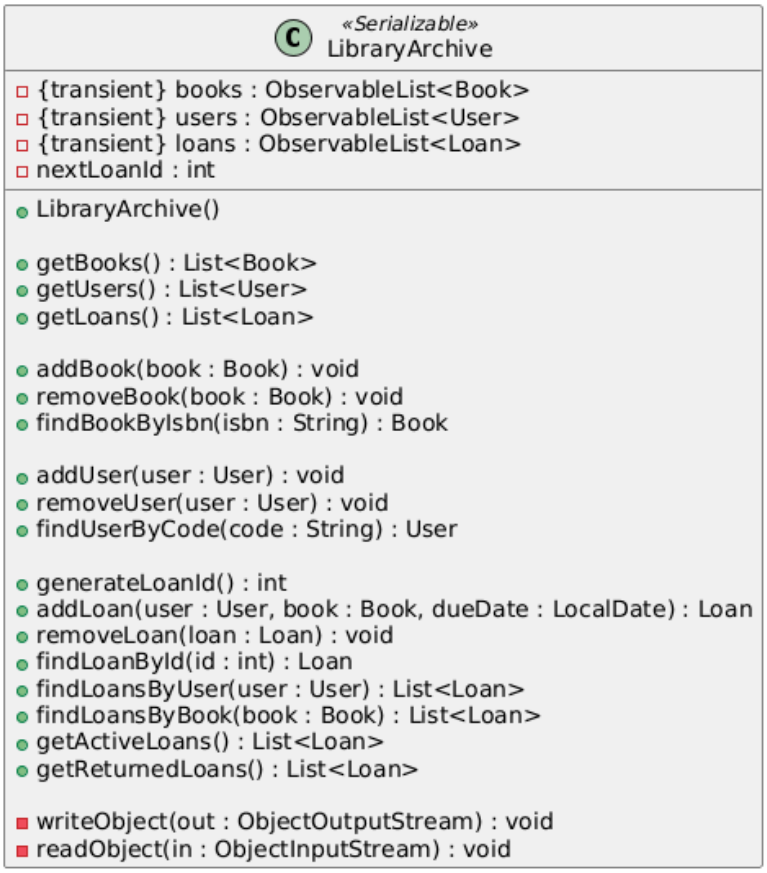

figura 7

La classe ***LibraryArchive*** rappresenta il contenitore centrale dei dati del sistema, mantenendo le liste di libri, utenti e prestiti, oltre a gestire la generazione incrementale degli identificativi dei prestiti.
Le sue operazioni consistono nell’aggiungere, rimuovere e ricercare entità all’interno delle collezioni, insieme all’estrazione di insiemi filtrati come i prestiti attivi o restituiti.
Non applica logiche di business complesse: si limita alla gestione strutturale delle liste e alla ricerca di elementi tramite attributi chiave (ISBN, codice utente, ID prestito).
La **coesione** è **funzionale** ed elevata, poiché la classe ha un’unica responsabilità ben definita, ovvero mantenere l’archivio in uno stato coerente.
L’**accoppiamento** è **per timbro** (basso–medio**)** e naturale per un contenitore di dominio: dipende da *Book*, *User* e *Loan* solo per mantenere le istanze, senza interagire con servizi, controller o persistenza. Il ruolo della classe rimane quindi confinato alla gestione dei dati, evitando qualsiasi forma di logica applicativa.

4.2.5 Classe “ArchiveFileService”

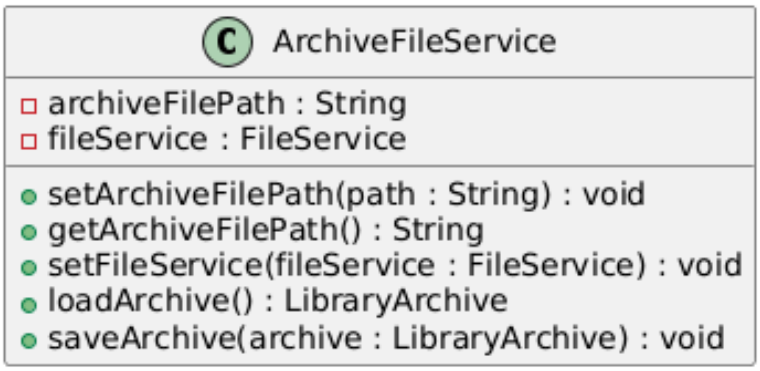

figura 8

La classe ***Archive File Service*** gestisce il salvataggio e il caricamento dell’archivio della biblioteca tramite un percorso di file configurabile e un componente *FileService* delegato alle operazioni di I/O. La responsabilità della classe è limitata a coordinare la persistenza dell’oggetto *LibraryArchive* senza definire alcuna logica interna di serializzazione, demandata invece a *FileService*; la **coesione** è quindi **funzionale** ed elevata. L’**accoppiamento** è **per timbro**: la classe dipende da *LibraryArchive* come oggetto da persistere e da *FileService* come servizio esterno necessario per leggere e scrivere su file, ma non interagisce con altri moduli del sistema.

4.2.6 Classe “FileService”

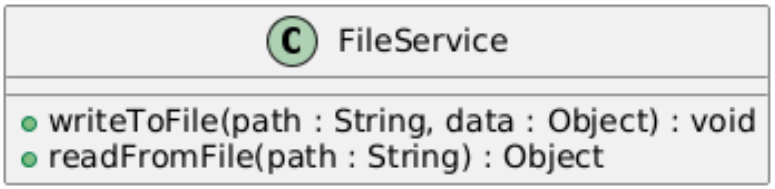

figura 9

La classe ***FileService*** fornisce un servizio generico di lettura e scrittura su file, utilizzando metodi parametrizzati tramite generics per gestire qualunque tipo di dato. La sua responsabilità è esclusivamente l’astrazione delle operazioni di I/O, senza definire dettagli specifici di persistenza, che vengono delegati ai componenti che la utilizzano. La **coesione** è quindi **funzionale** ed elevata. L’**accoppiamento** è **nullo**: la classe non dipende da nessun modello o servizio del sistema, offrendo un’interfaccia riusabile e indipendente dal dominio applicativo.

4.2.7 classe “BookService”

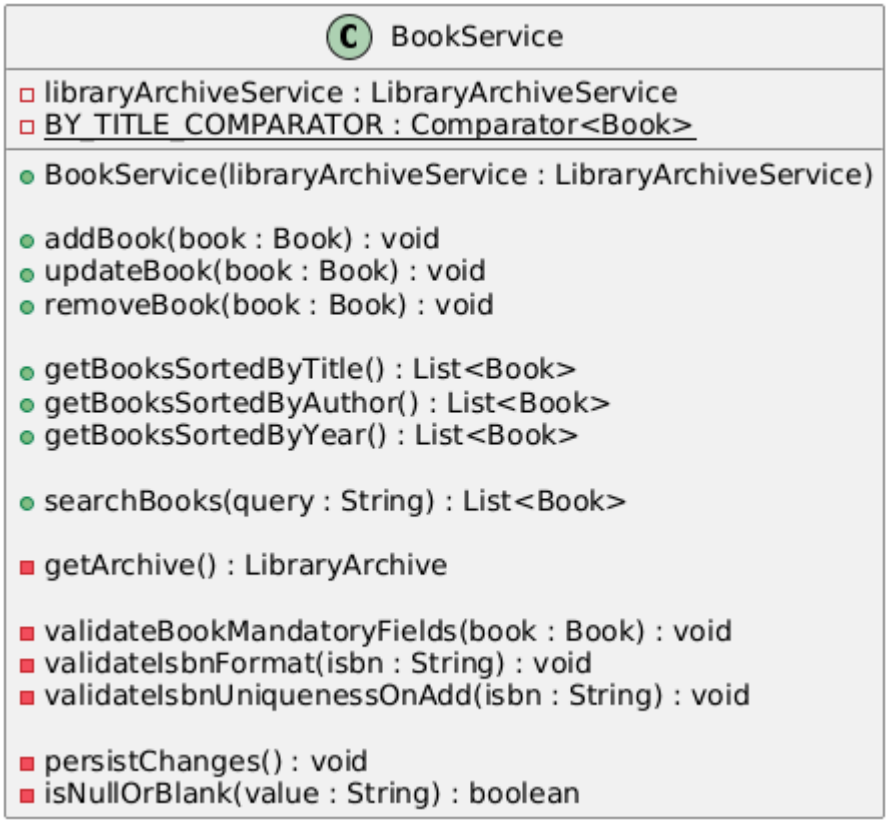

figura 10

La classe ***BookService*** incapsula la logica applicativa relativa alla gestione dei libri, operando sopra l’archivio di dominio (*LibraryArchive*) e, indirettamente, sulla sua persistenza tramite *LibraryArchiveService*.
I suoi metodi principali sono orientati alle operazioni CRUD sui libri, convalidando i dati tramite eccezioni specifiche e domandando la memorizzazione effettiva alle strutture dell’archivio. Inoltre, fornisce funzionalità di consultazione come l’ordinamento dei libri per titolo e la ricerca testuale, centralizzando in un unico punto le regole applicative legate alla gestione dei Book.
La **coesione** è **funzionale** ed elevata, poiché la classe si occupa esclusivamente della logica di business sui libri. L’**accoppiamento** è **per timbro**, appropriato al suo ruolo di servizio: dipende da *LibraryArchive*, *LibraryArchiveService*, dal modello Book e dal sottosistema delle eccezioni, ma non interagisce direttamente con controller o componenti UI, mantenendo chiara la separazione tra livelli architetturali.

4.2.8 classe "LibraryArchiveService"

figura 11

La classe ***Library Archive Service*** funge da strato di servizio che gestisce l'accesso all'archivio principale del sistema e coordina le operazioni di caricamento e salvataggio tramite *FileService*.
Fornisce metodi centralizzati per ottenere libri, utenti e prestiti, agendo come punto di accesso uniforme ai dati dell'applicazione. Non implementa logiche di dominio complesse, ma si limita a incapsulare il recupero dell'archivio e a delegarne la persistenza, mantenendo una **coesione elevata** grazie a una responsabilità unica e chiara.
L'**accoppiamento** è **per timbro**: la classe dipende dall'archivio, dai modelli *Book*, *User* e *Loan*, e da *FileService* per le operazioni di I/O, senza coinvolgere livelli superiori come controller o interfacce utente.

4.2.9 classe “LoanService”

figura 12

La classe ***LoanService*** incapsula la logica applicativa relativa al ciclo di vita dei prestiti, operando sull’archivio di dominio (*LibraryArchive*) e coordinandosi con *LibraryArchiveService* per l’accesso centralizzato ai dati.
I metodi principali gestiscono la registrazione di un nuovo prestito, applicando i vincoli di dominio tramite eccezioni specifiche, e la registrazione della restituzione, oltre a fornire funzionalità di consultazione e filtro sui prestiti attivi: recupero globale dei prestiti attivi, loro ordinamento, filtraggio per utente o per libro e verifica di eventuali ritardi.
La **coesione** è **funzionale** ed elevata, poiché la classe si occupa esclusivamente della logica di business legata ai Loan, senza inglobare responsabilità di presentazione o di persistenza di basso livello.
L’**accoppiamento** è **moderato** e coerente con il ruolo di servizio: *LoanService* dipende da *LibraryArchive*, *LibraryArchiveService*, dai modelli *Book*, *User* e *Loan* e dal sottosistema delle eccezioni, mantenendo comunque una chiara separazione dai controller e dagli strati di interfaccia utente.

4.9.10 classe “UserService”

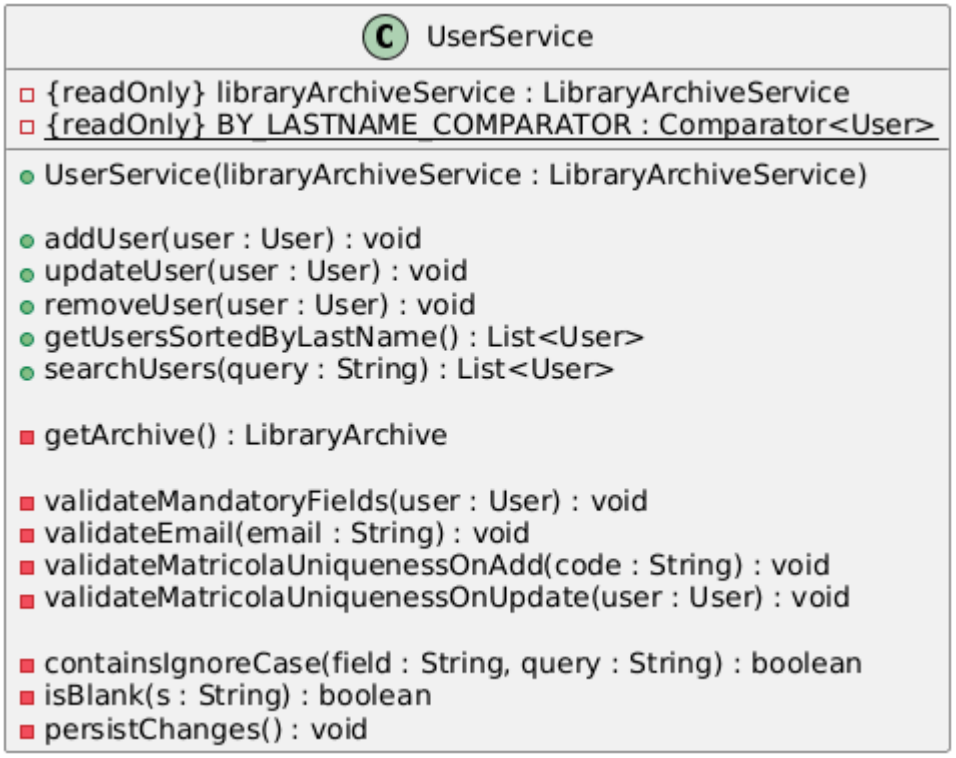

figura 13

La classe ***UserService*** gestisce la logica applicativa relativa agli utenti dell’archivio, offrendo operazioni di creazione, modifica e rimozione convalidando i dati tramite eccezioni specifiche.
Fornisce inoltre funzioni di consultazione, come l’ordinamento alfabetico degli utenti e la ricerca testuale. La **coesione** è **funzionale** ed elevata, poiché la classe si occupa esclusivamente delle regole di business degli utenti senza interferire con aspetti di presentazione o persistenza.
L’**accoppiamento** è **per timbro**: *UserService* **dipende** da *LibraryArchive* e *LibraryArchiveService* per accedere e aggiornare lo stato globale degli utenti, mantenendo comunque una chiara separazione dagli altri livelli architetturali

4.9.11 classe “ServiceLocator”

figura 14

Il diagramma rappresenta la classe **ServiceLocator**, responsabile di fornire un punto di accesso centralizzato ai servizi applicativi del sistema.La classe implementa il *Service Locator pattern*, istanziando una sola volta (come attributi static final) i principali servizi dell’applicazione e rendendoli disponibili tramite metodi di accesso statici. Tutti i servizi condividono la medesima istanza di LibraryArchiveService, garantendo coerenza operativa sull’archivio applicativo. Il ServiceLocator non si occupa del caricamento dell’archivio da persistenza, che viene delegato ad altri componenti dell’applicazione, e viene utilizzato esclusivamente in modalità statica, senza esposizione del costruttore.
Dal punto di vista dell’**accoppiamento**, il ServiceLocator riduce la dipendenza diretta dei controller dai singoli servizi e dalla loro istanziazione, centralizzando la gestione delle dipendenze e semplificando l’evoluzione dell’architettura. In termini di **coesione**, la classe presenta un’elevata coesione funzionale, in quanto tutte le sue responsabilità sono limitate esclusivamente alla fornitura e gestione degli accessi ai servizi.
Il ServiceLocator non contiene logica di business né di persistenza, che rimangono delegate ai rispettivi service.

5 Diagrammi di sequenza

In questa sezione vengono presentati i diagrammi di sequenza relativi ai principali casi d'uso. Tali diagrammi descrivono il comportamento dinamico del sistema, evidenziando l'ordine temporale dei messaggi scambiati tra gli attori coinvolti (operatore) e i diversi componenti software (strati di presentazione, logica applicativa e gestione dei dati).

Per non appesantire eccessivamente i diagrammi si è deciso di accorpare le entità "*View*" e "*Controller*" sotto un unico attore del tipo "*ViewController*" e di mostrare solamente le interazioni principali tra gli attori partecipanti, escludendo per esempio, alcuni valori di ritorno scontati o gli aggiornamenti espliciti della TableView.

5.1 Inserimento nuovo utente

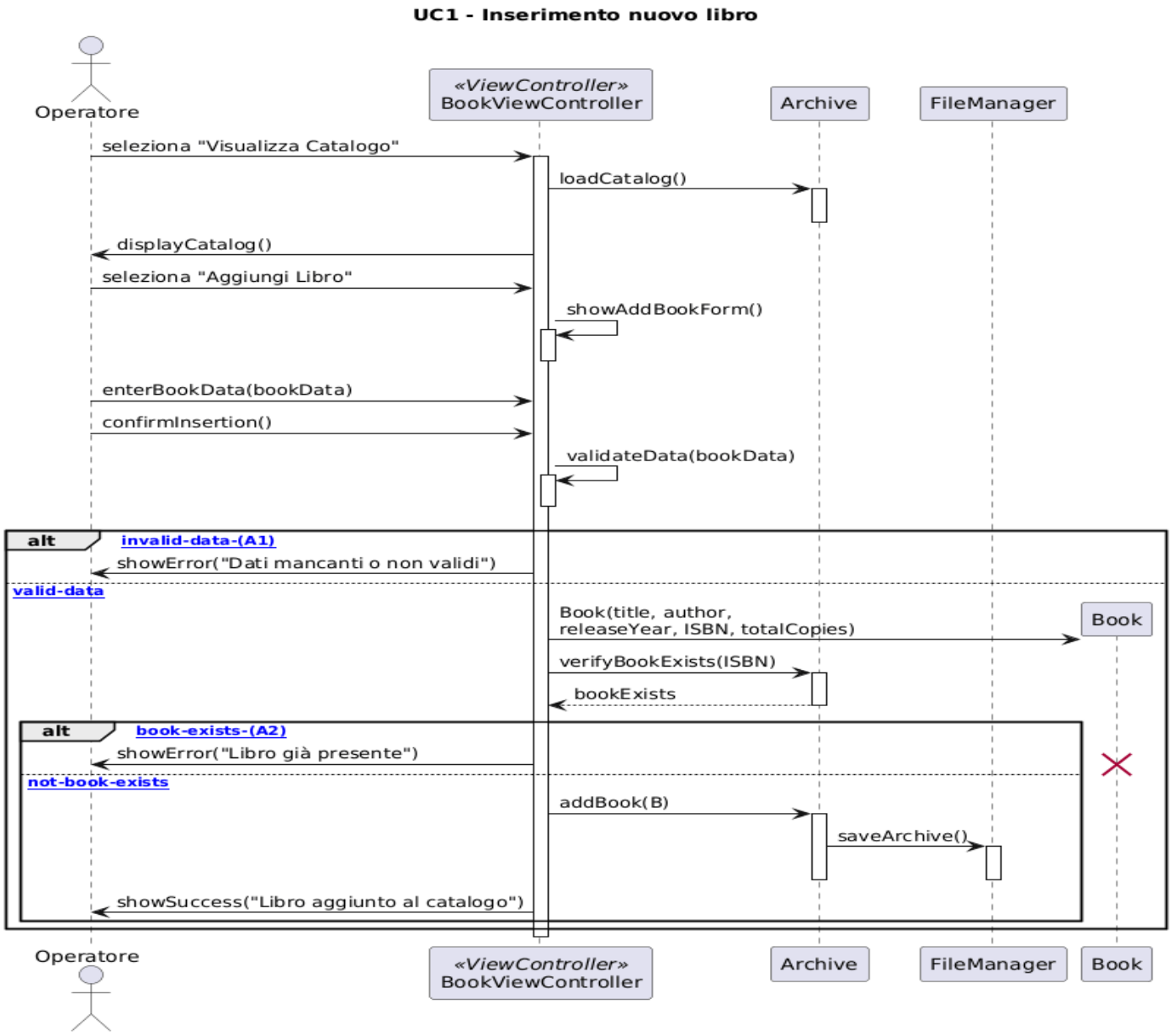

figura 15

Il diagramma in ***fig. 15*** mostra il flusso di sequenza del primo caso d'uso (**UC-1**). L'operatore apre il catalogo e seleziona "**Aggiungi Libro**", visualizzando il modulo di inserimento. Dopo aver compilato i campi richiesti e confermato, il sistema valida i dati. Se la validazione fallisce viene mostrato un errore; se è superata, viene creato un nuovo oggetto *Book* e viene verificata l'assenza di un libro con lo stesso ISBN. In caso di duplicato il sistema segnala l'errore e annulla l'inserimento; altrimenti aggiunge il libro all'archivio, salva l'aggiornamento e conferma il corretto inserimento all'operatore.

5.2 Modifica di un libro esistente

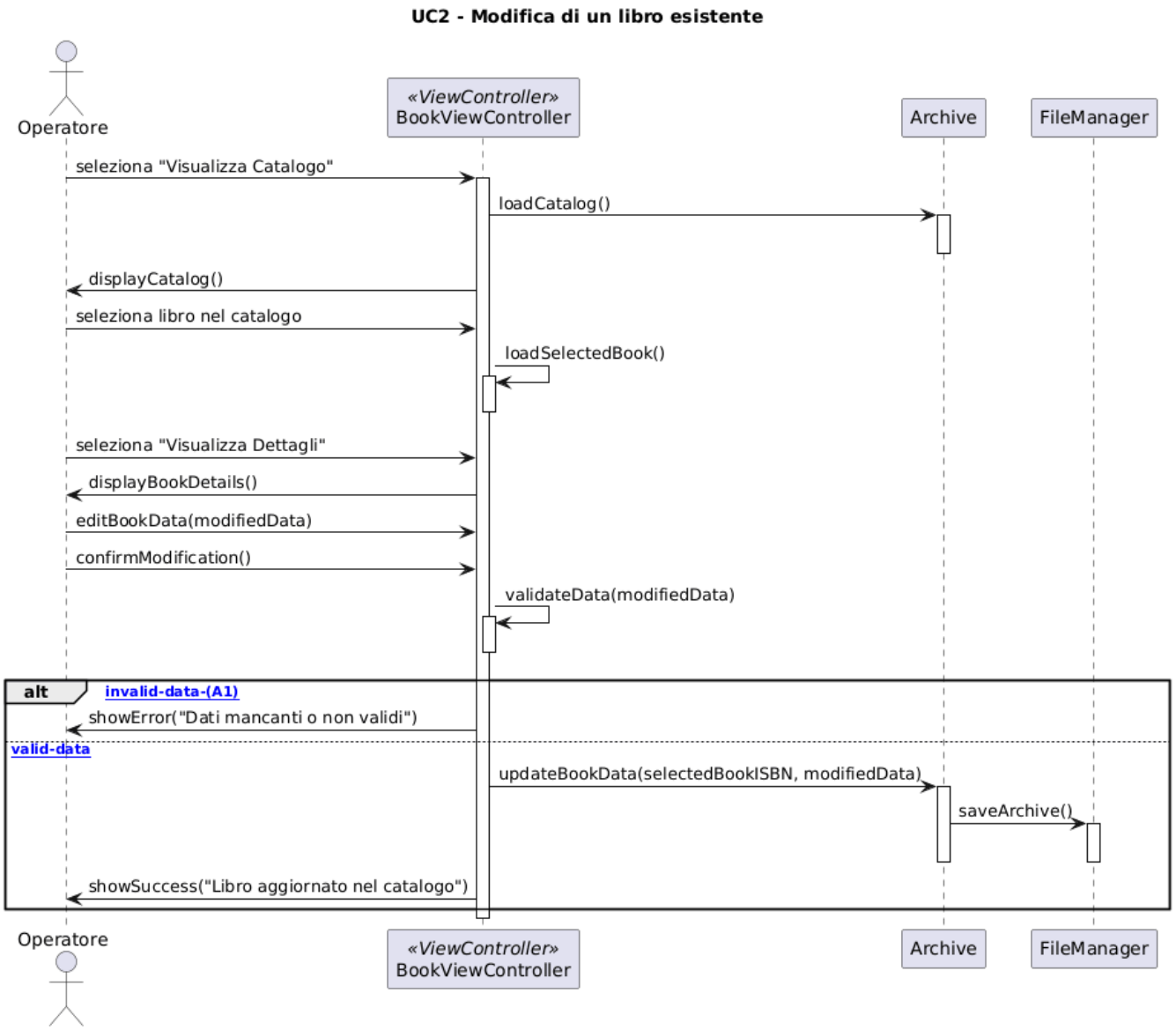

figura 16

Il diagramma in ***fig. 16*** mostra il flusso di sequenza del secondo caso d'uso (**UC-2**). L'operatore visualizza il catalogo, seleziona un libro e apre la schermata dei dettagli. Dopo aver modificato i campi desiderati e confermato la modifica, il sistema valida i dati inseriti. Se la validazione fallisce viene mostrato un errore; se supera i controlli, l'archivio viene aggiornato con i nuovi dati del libro e il sistema salva le modifiche. Infine viene mostrato un messaggio di conferma all'operatore.

5.3 Eliminazione di un libro

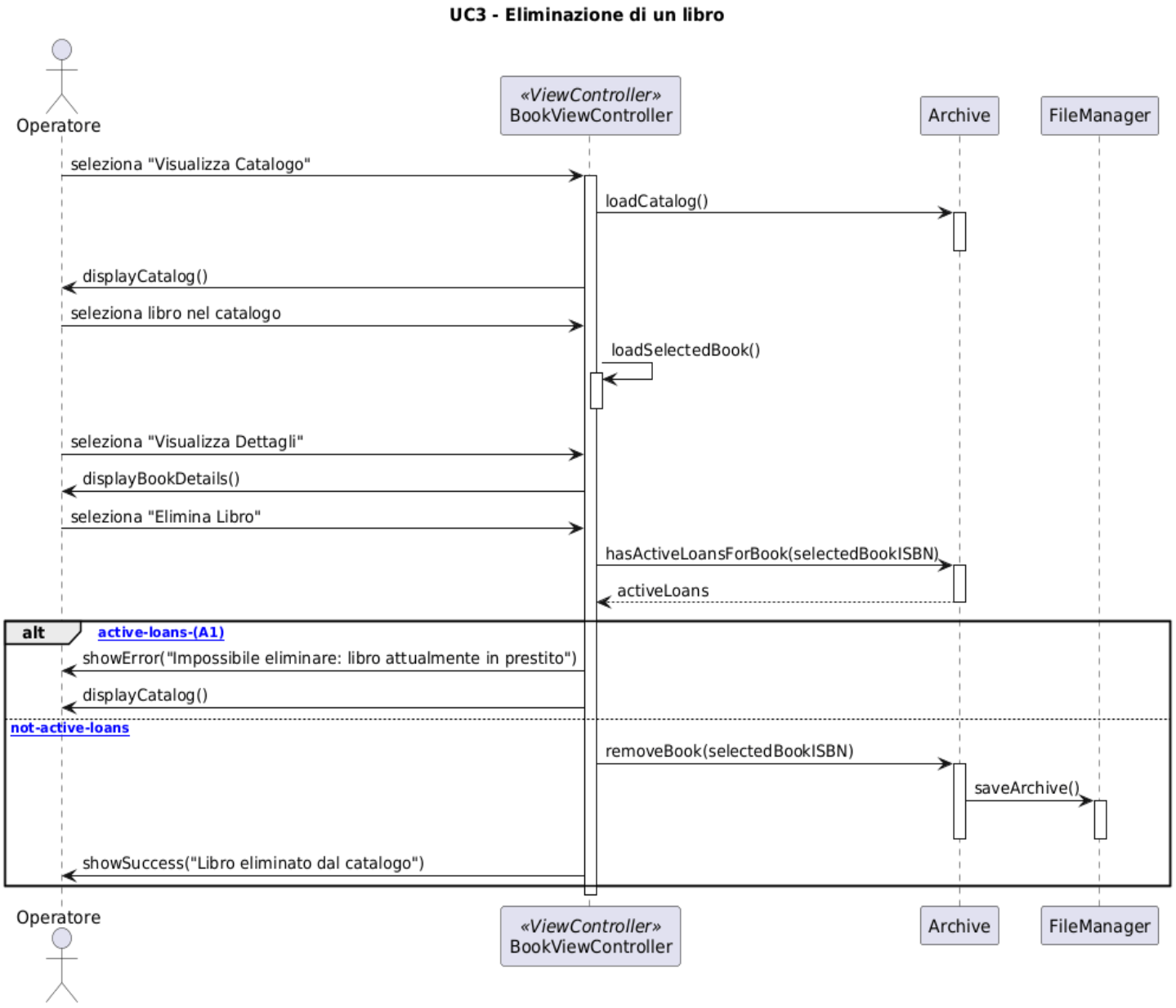

figura 17

Il diagramma in ***fig. 17*** **mostra** il flusso di sequenza del terzo caso d'uso (**UC-3**). L'operatore visualizza il catalogo, seleziona un libro e apre la schermata dei dettagli ("**Visualizza Dettagli**"). Quando richiede l'eliminazione, il sistema verifica se il libro è associato a prestiti attivi. Se risultano prestiti in corso, viene mostrato un messaggio di errore; in caso contrario, il libro viene rimosso dall'archivio, l'aggiornamento viene salvato e il sistema conferma l'eliminazione all'operatore.

5.4 Visualizzazione catalogo

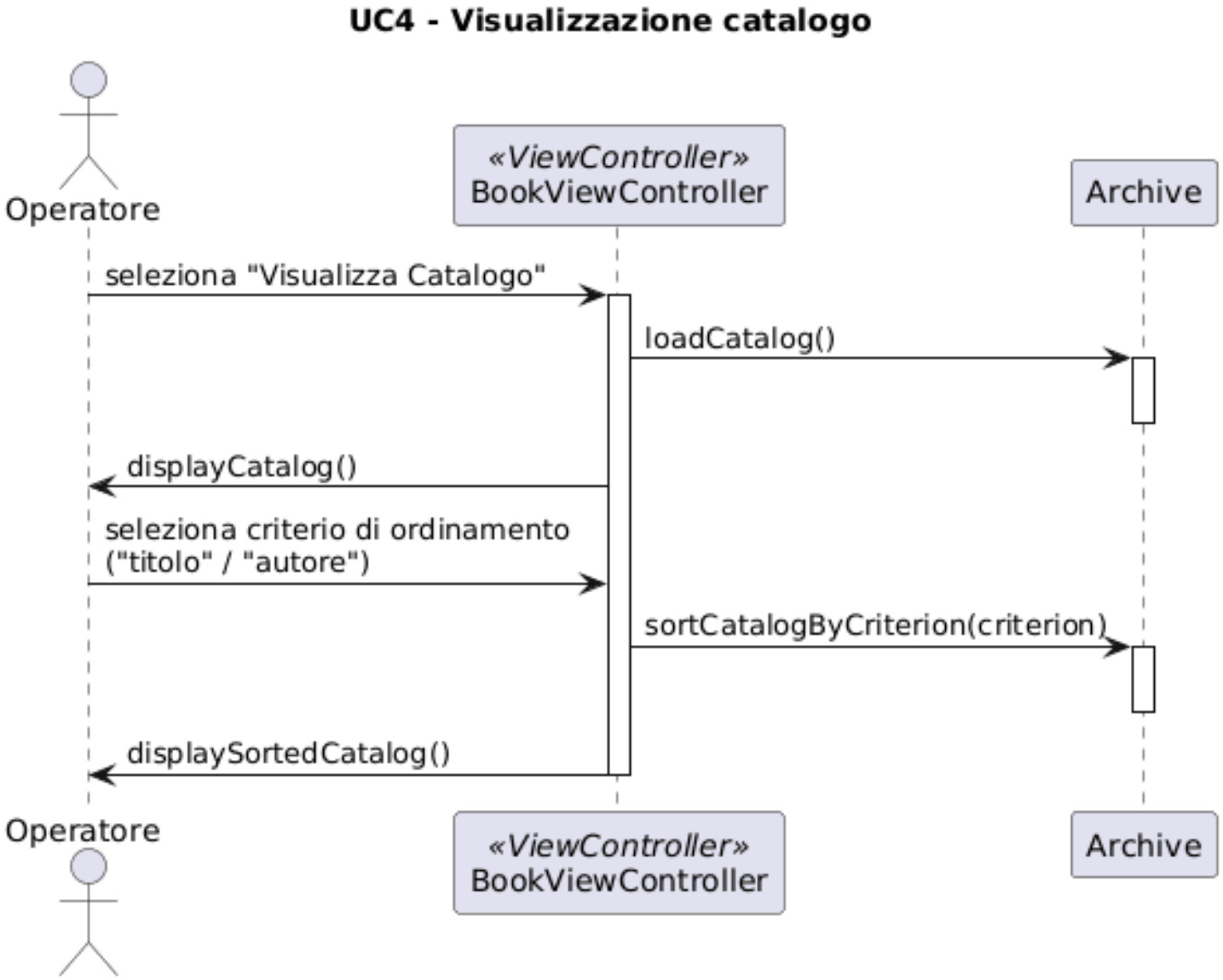

figura 18

Il diagramma in ***fig. 18*** mostra il flusso di sequenza del quarto caso d'uso (**UC-4**). L'operatore richiede la visualizzazione del catalogo ("**Visualizza Catalogo**") e il sistema mostra l'elenco dei libri presenti. Successivamente, l'operatore può scegliere un criterio di ordinamento, come titolo o autore, e il sistema aggiorna la visualizzazione applicando l'ordinamento selezionato.

5.5 Ricerca di un libro nel catalogo

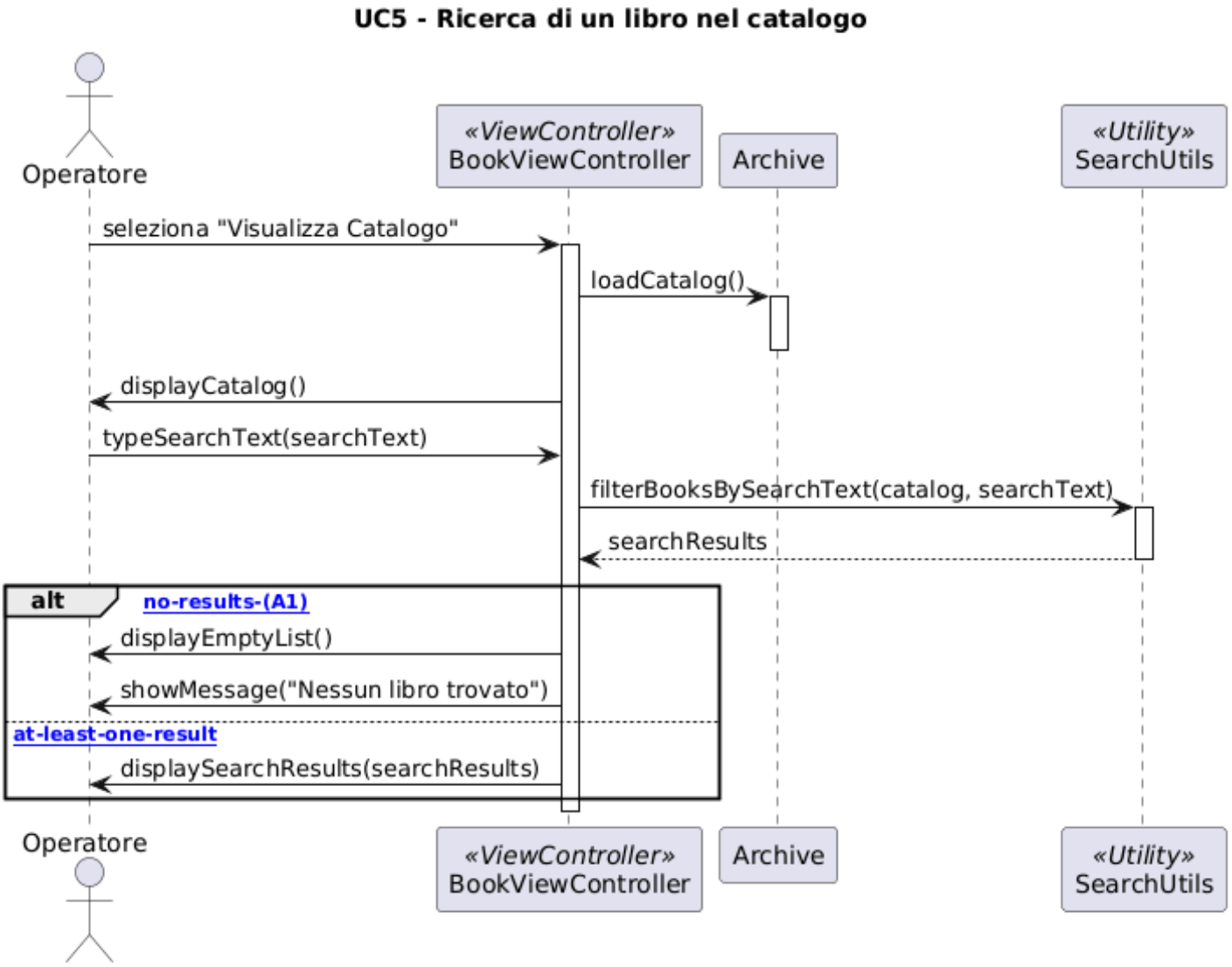

figura 19

Il diagramma in ***fig. 19*** mostra il flusso di sequenza del quinto caso d'uso (**UC-5**). L'operatore visualizza il catalogo e inserisce del testo nella barra di ricerca. Il sistema filtra dinamicamente i libri confrontando il testo inserito con i dati del catalogo e restituisce l'elenco dei risultati. Se nessun libro corrisponde alla ricerca, viene mostrato un elenco vuoto con un messaggio informativo; in caso contrario, il sistema presenta i risultati trovati.

5.6 Visualizzazione dei dettagli di un libro

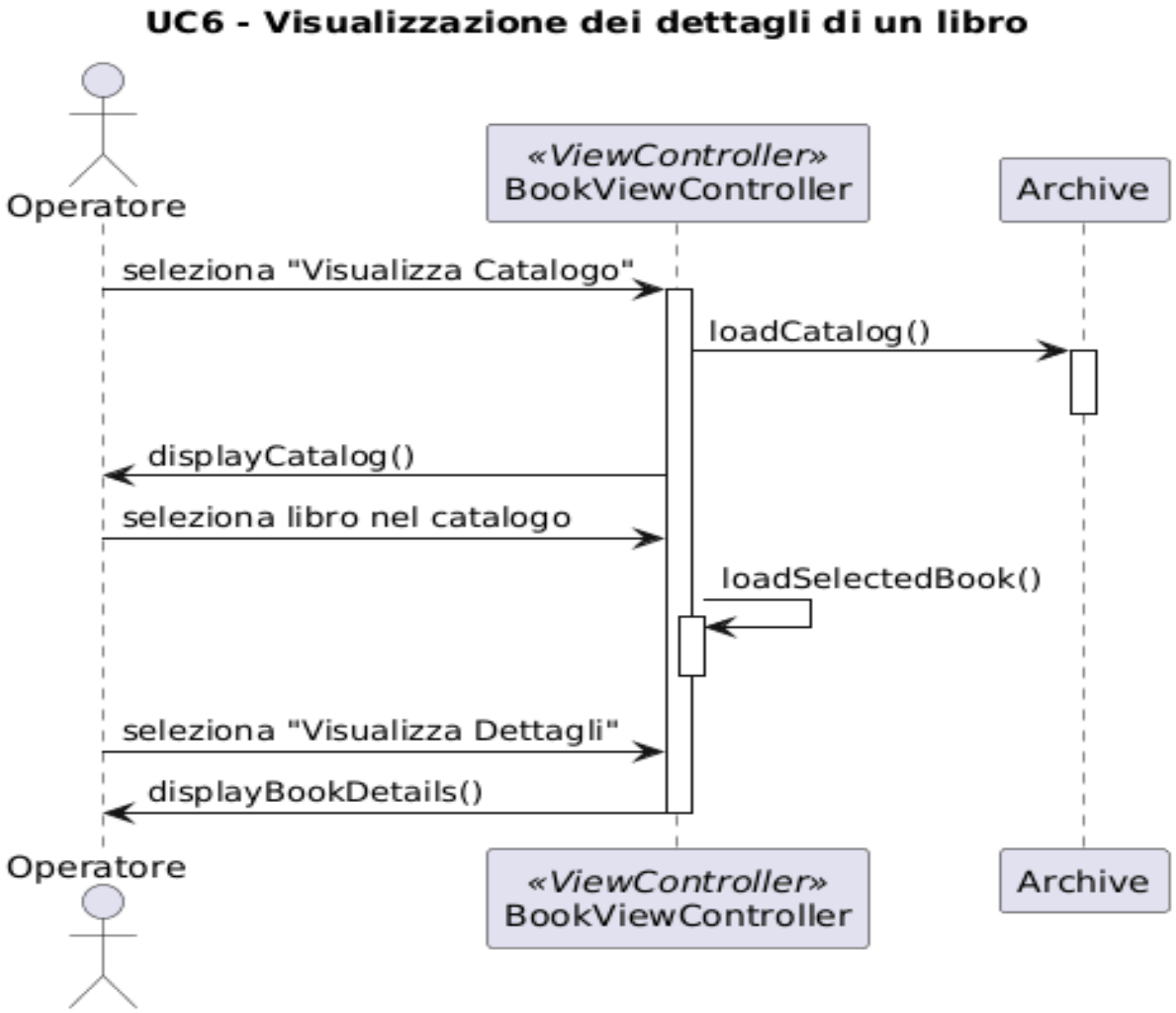

figura 20

Il diagramma in ***fig. 20*** mostra il flusso di sequenza del sesto caso d'uso (***UC-6***). L'operatore visualizza il catalogo dei libri e seleziona un elemento dall'elenco. Dopo aver richiesto la visualizzazione dei dettagli, il sistema recupera il libro selezionato e mostra tutte le informazioni associate, offrendo una consultazione completa dei suoi attributi.

5.7 Registrazione di un nuovo utente

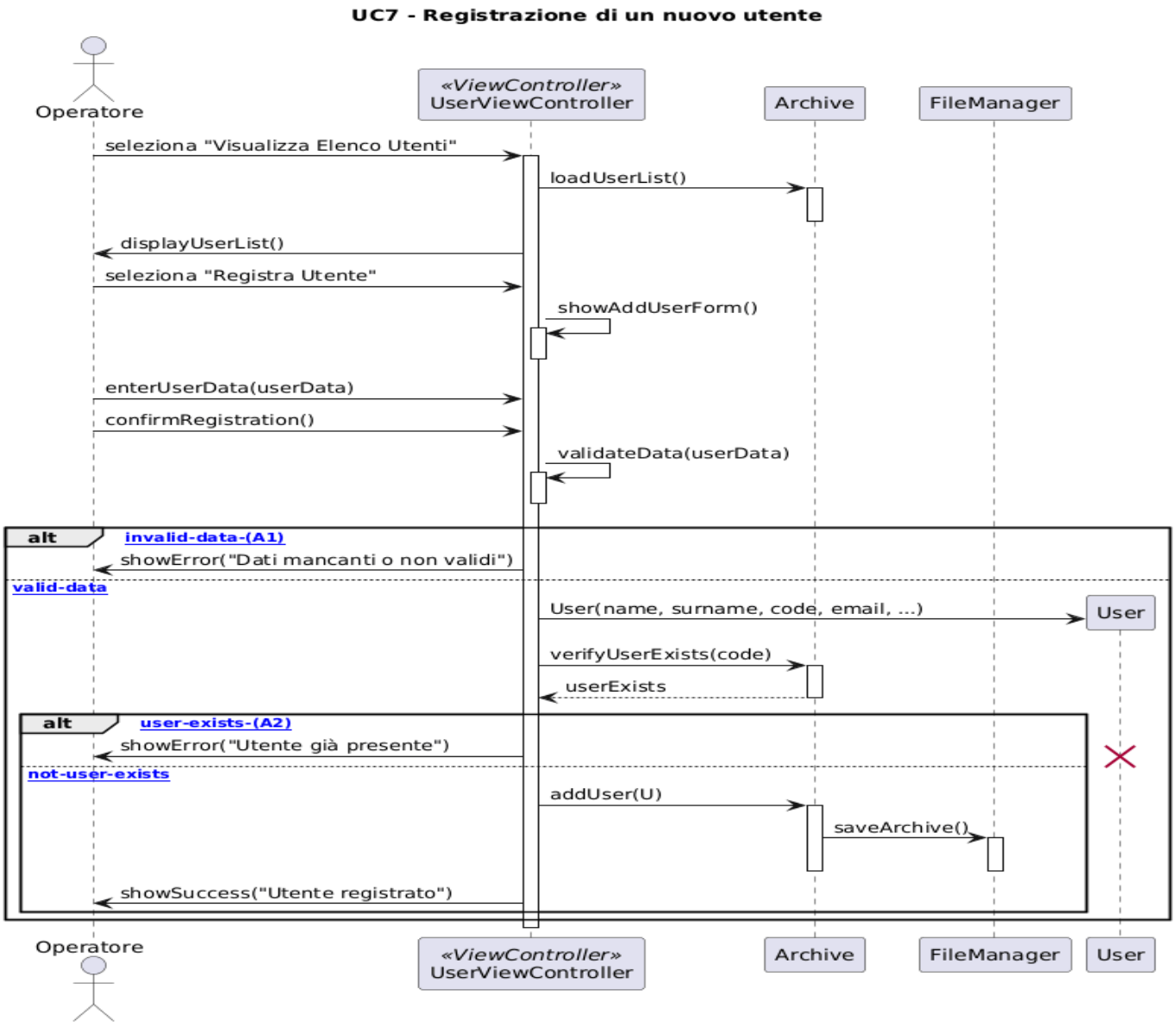

figura 21

Il diagramma in ***fig. 21*** mostra il flusso di sequenza del settimo caso d'uso (***UC-7***). L'operatore visualizza l'elenco degli utenti e avvia la procedura di registrazione, compilando il modulo dedicato. Il sistema valida i dati inseriti e, se risultano corretti, crea un nuovo oggetto *User* e verifica che non esista già un utente con lo stesso identificativo. In caso di duplicato viene segnalato un errore e l'operazione viene annullata; altrimenti l'utente viene aggiunto all'archivio, l'aggiornamento viene salvato e viene mostrata una conferma all'operatore.

5.8 Modifica di un utente

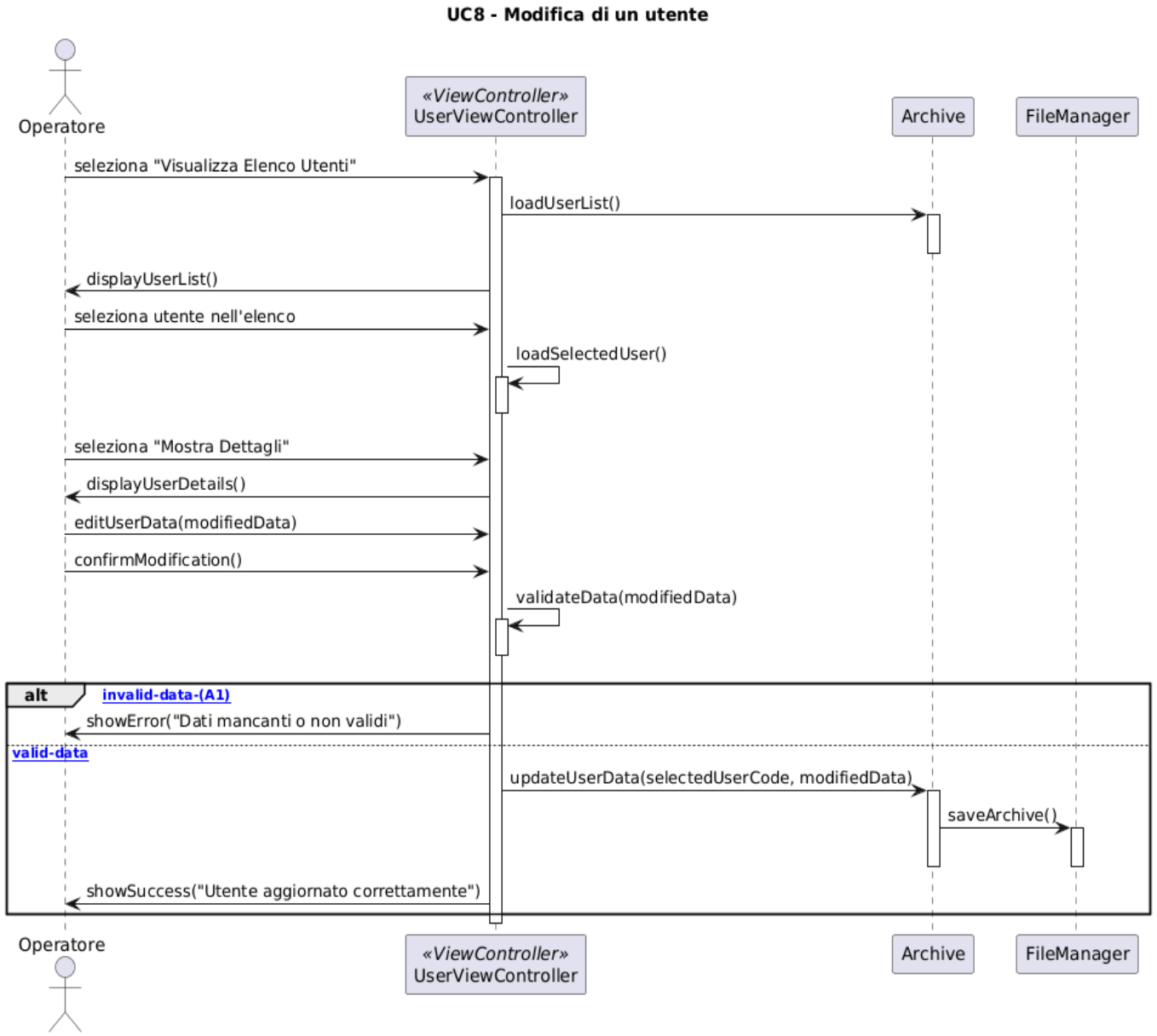

figura 22

Il diagramma in ***fig. 22*** mostra il flusso di sequenza dell'ottavo caso d'uso (***UC-8***). L'operatore visualizza l'elenco degli utenti, seleziona un utente e accede ai relativi dettagli. Dopo aver modificato i campi desiderati e confermato l'operazione, il sistema valida i dati inseriti. Se la validazione fallisce viene mostrato un errore; in caso di esito positivo, l'archivio viene aggiornato con le nuove informazioni dell'utente, salvato, e l'operatore riceve un messaggio di conferma.

5.9 Eliminazione di un utente

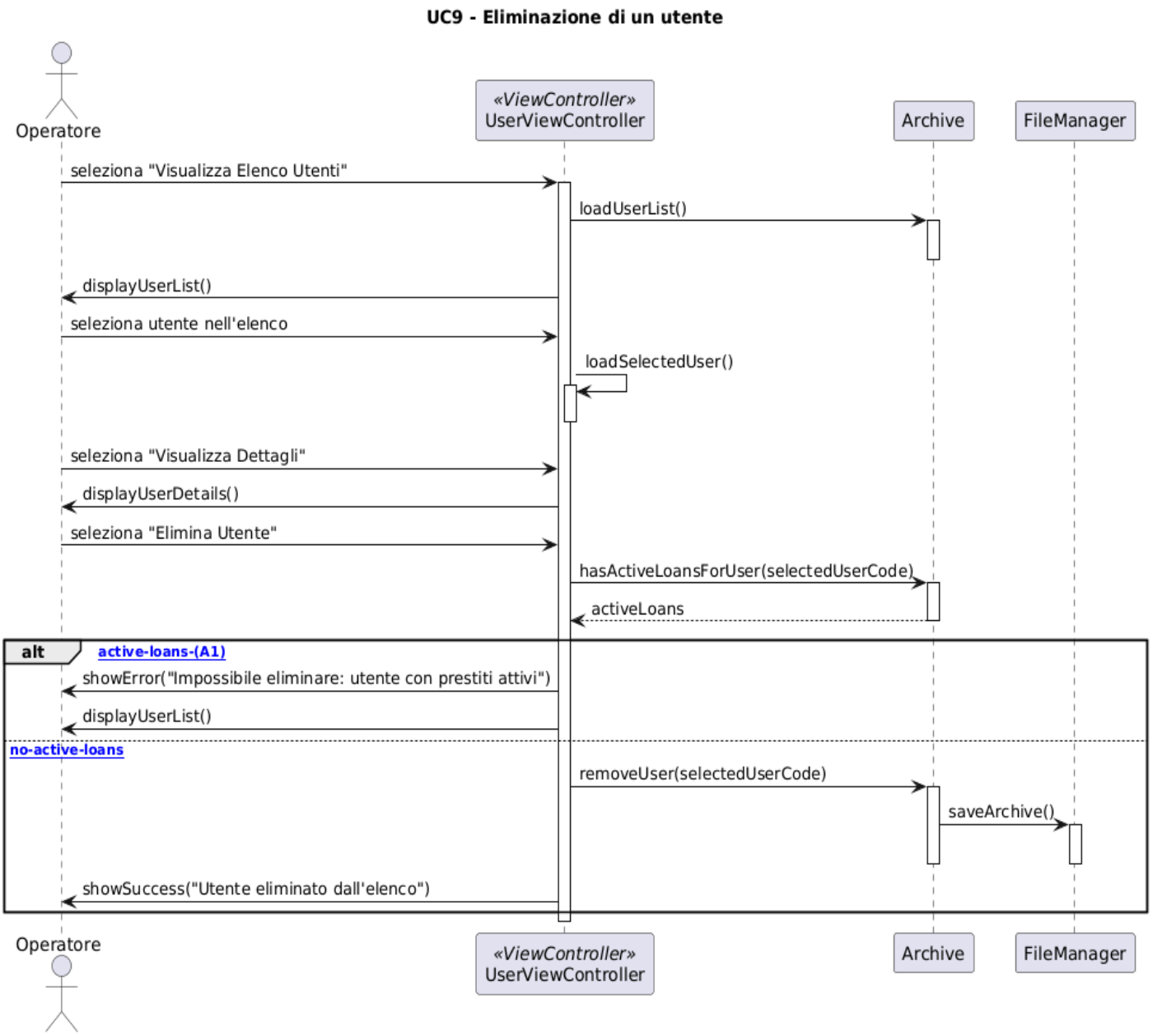

figura 23

Il diagramma in ***fig. 23*** mostra il flusso di sequenza del nono caso d'uso (***UC-9***). L'operatore visualizza l'elenco degli utenti, seleziona un utente e ne consulta i dettagli. Quando richiede l'eliminazione ("***Elimina Utente***"), il sistema verifica se l'utente possiede prestiti attivi. In presenza di prestiti attivi l'operazione viene bloccata e viene mostrato un messaggio di errore; se non ci sono prestiti pendenti, l'utente viene rimosso dall'archivio, le modifiche vengono salvate e il sistema conferma l'avvenuta eliminazione.

5.10 Visualizzazione elenco utenti

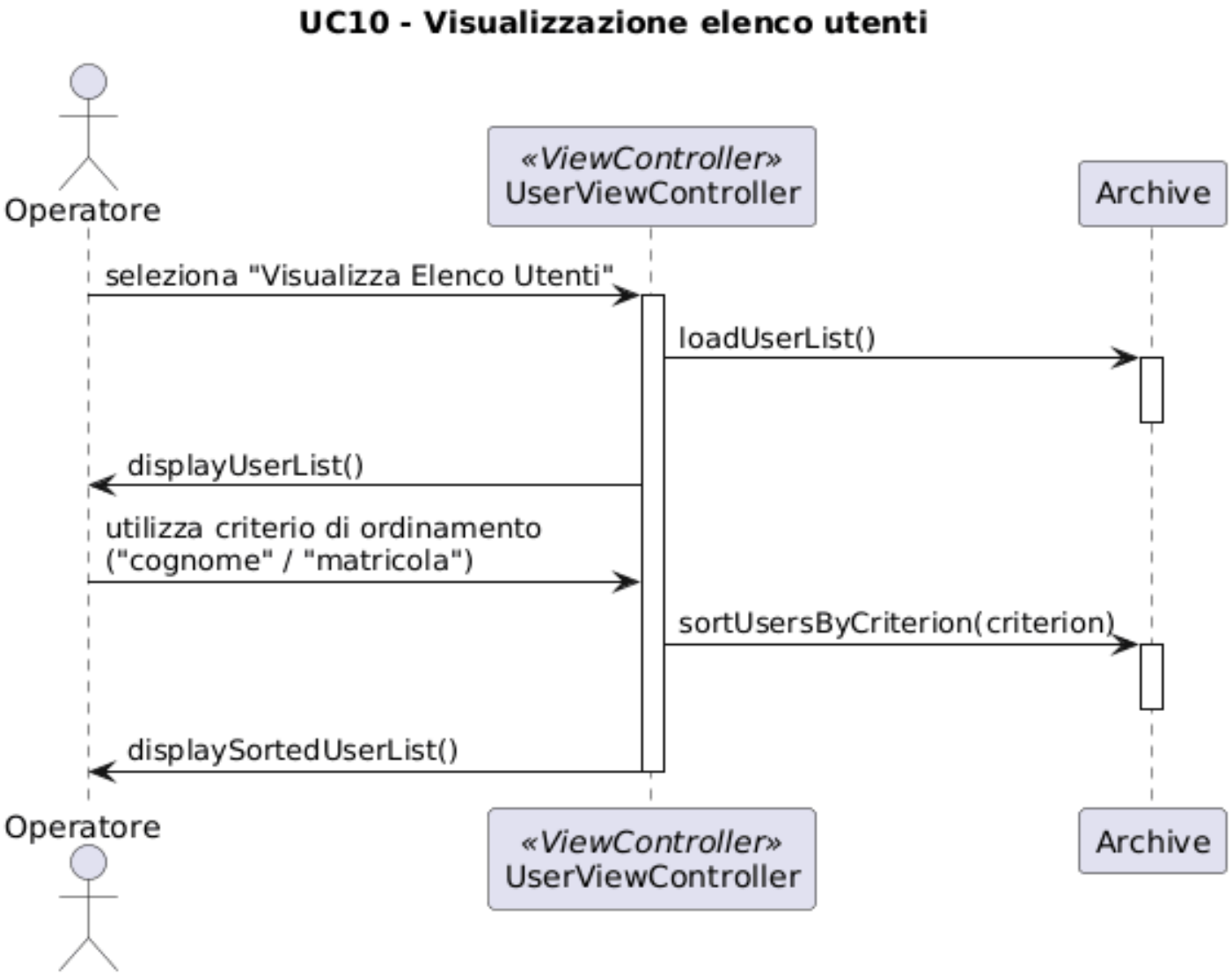

figura 24

Il diagramma in ***fig. 24*** mostra il flusso di sequenza del decimo caso d'uso (***UC-10***). L'operatore visualizza l'elenco degli utenti registrati e può applicare un criterio di ordinamento, come cognome o matricola. Il sistema ordina i dati secondo il criterio selezionato e aggiorna la visualizzazione mostrando l'elenco ordinato.

5.11 Ricerca di un utente registrato nell'elenco

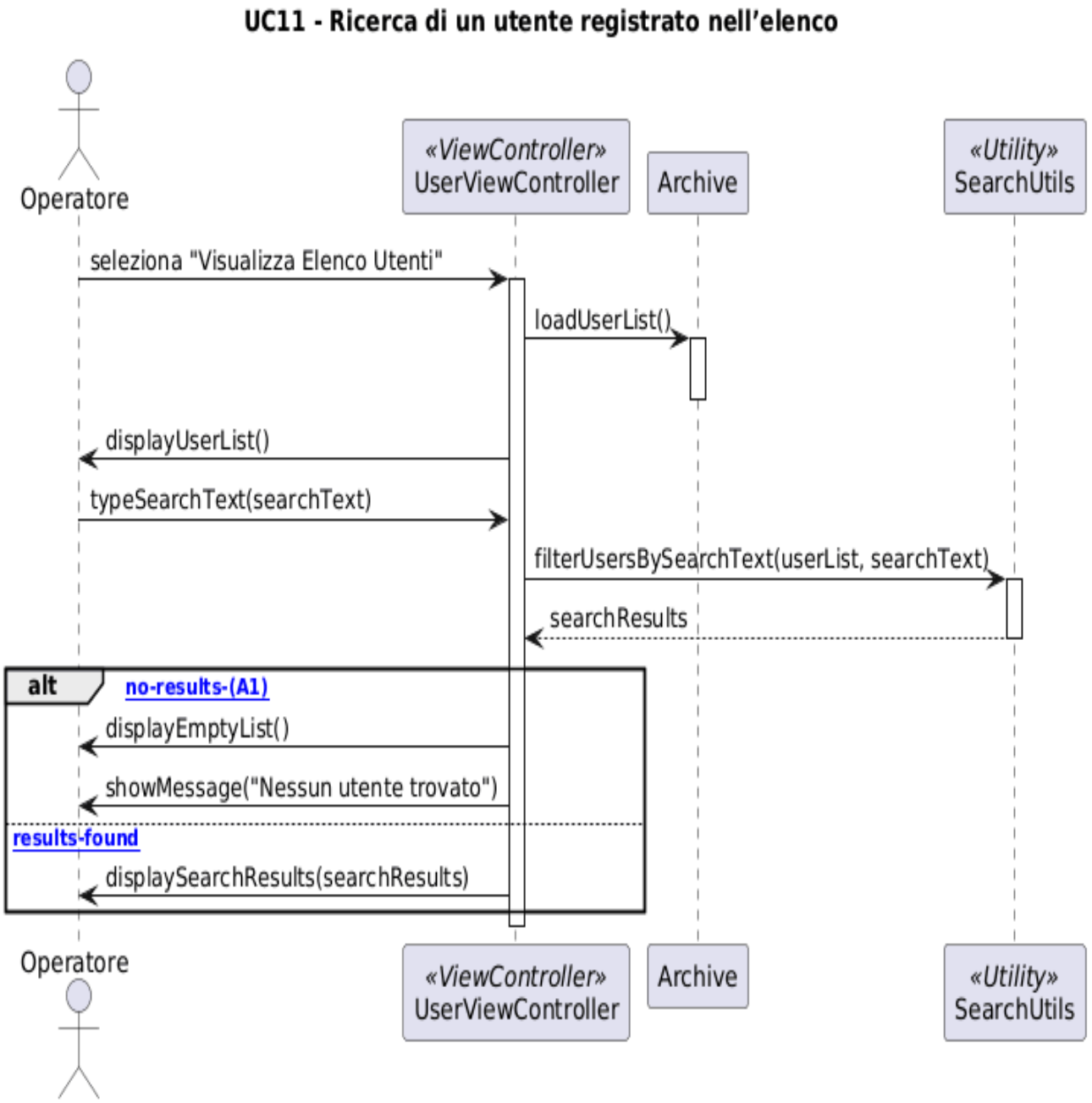

figura 25

Il diagramma in ***fig. 25*** mostra il flusso di sequenza dell'undicesimo caso d'uso (***UC-11***). L'operatore visualizza l'elenco degli utenti e inserisce del testo nella barra di ricerca. Il sistema filtra dinamicamente gli utenti in base ai dati inseriti, mostrando i risultati corrispondenti. Se nessun utente soddisfa i criteri di ricerca, viene mostrato un elenco vuoto accompagnato da un messaggio informativo.

5.12 Visualizzazione dettagli utenti

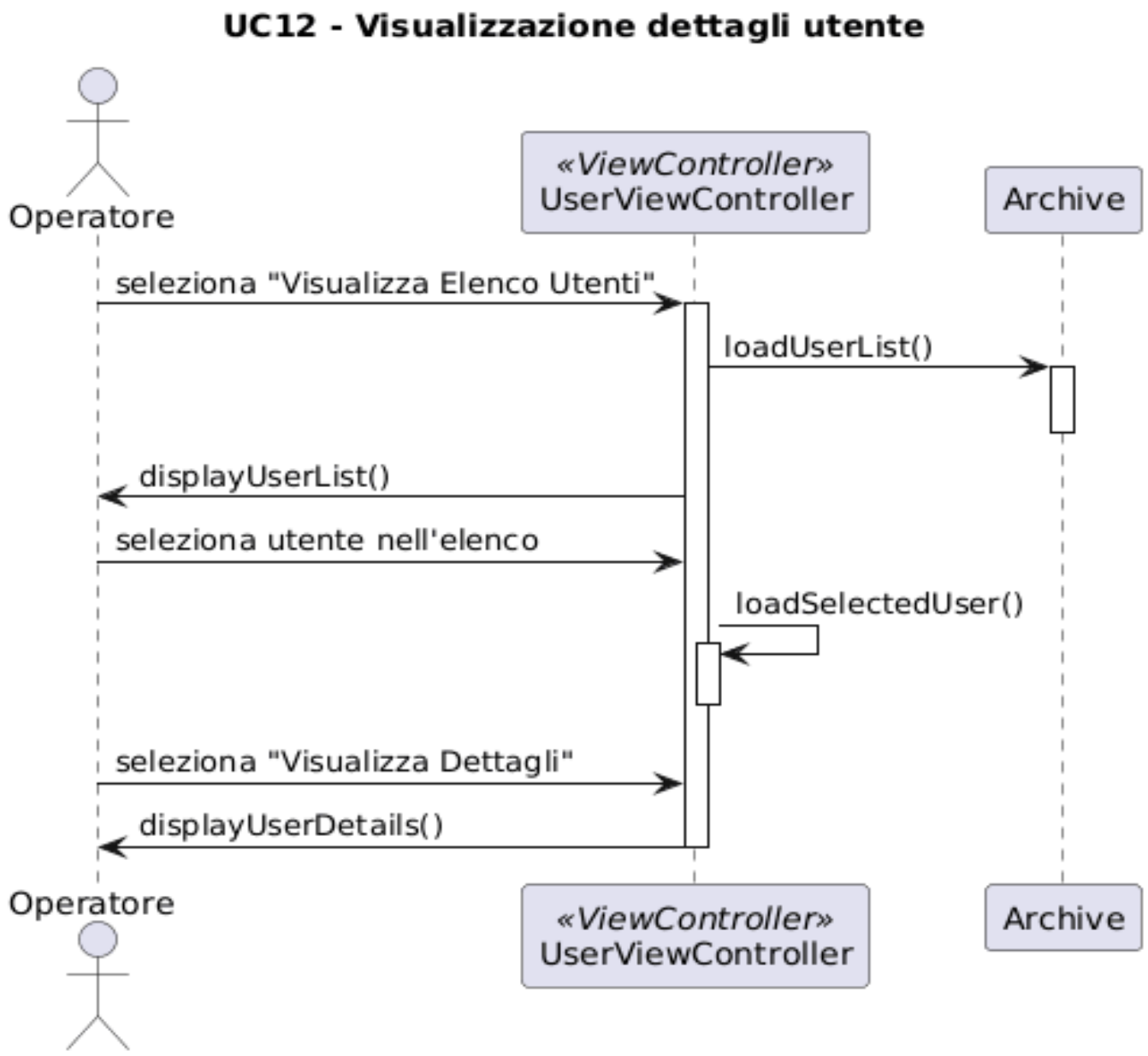

figura 26

Il diagramma in ***fig. 26*** mostra il flusso di sequenza del dodicesimo caso d'uso (***UC-12***). L'operatore visualizza l'elenco degli utenti, seleziona un utente e richiede la visualizzazione dei dettagli. Il sistema recupera l'utente selezionato e mostra tutte le informazioni associate, permettendo una consultazione completa dei suoi dati.

5.13 Visualizzazione prestiti attivi di un utente

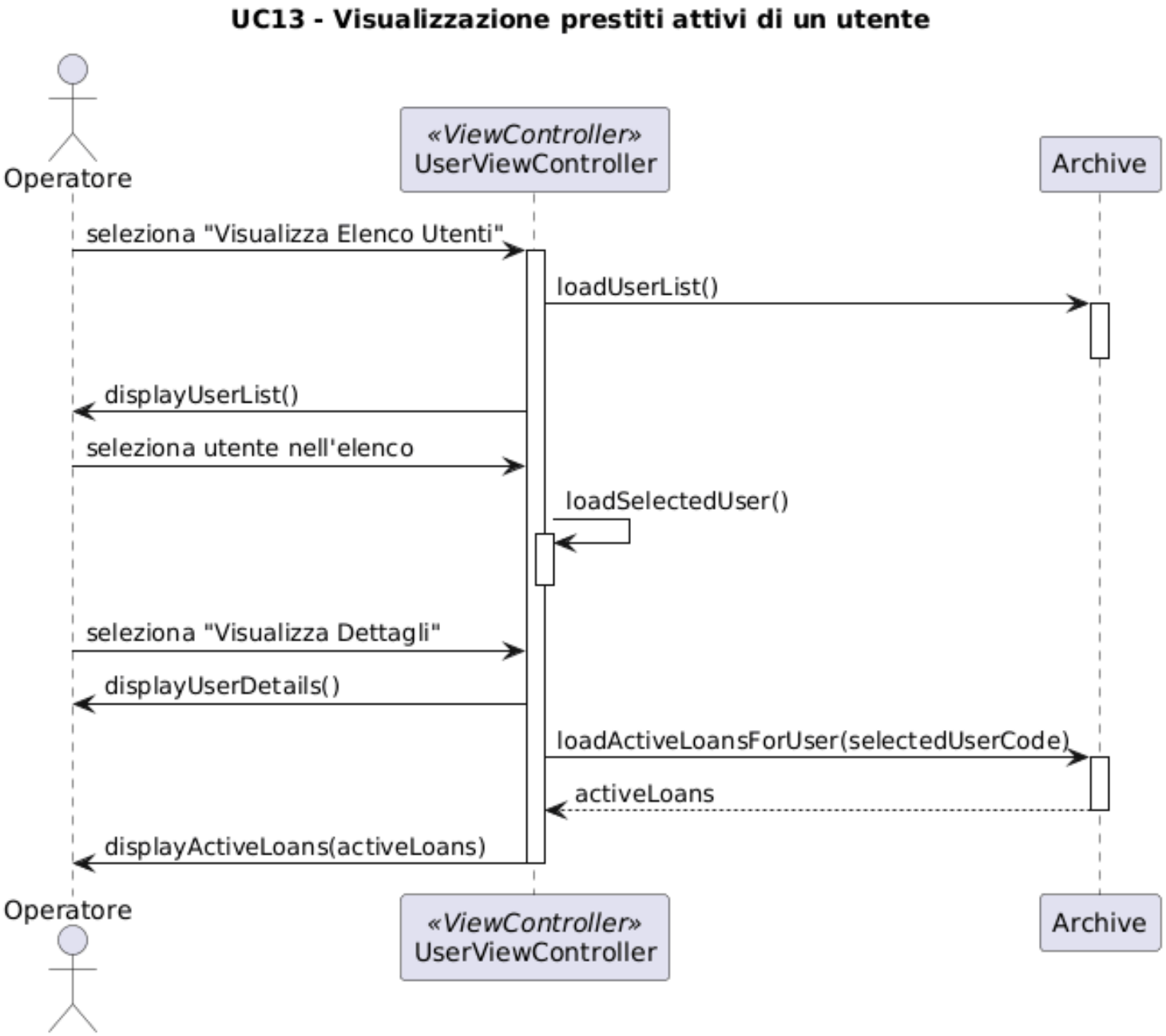

figura 27

Il diagramma in ***fig. 27*** mostra il flusso di sequenza del tredicesimo caso d'uso (***UC-13***). L'operatore visualizza l'elenco degli utenti, seleziona un utente e consulta i relativi dettagli. Il sistema recupera quindi tutti i prestiti attivi associati all'utente selezionato e li mostra all'operatore, permettendo una visione immediata delle operazioni di prestito attualmente in corso

5.14 Registrazione di un nuovo prestito

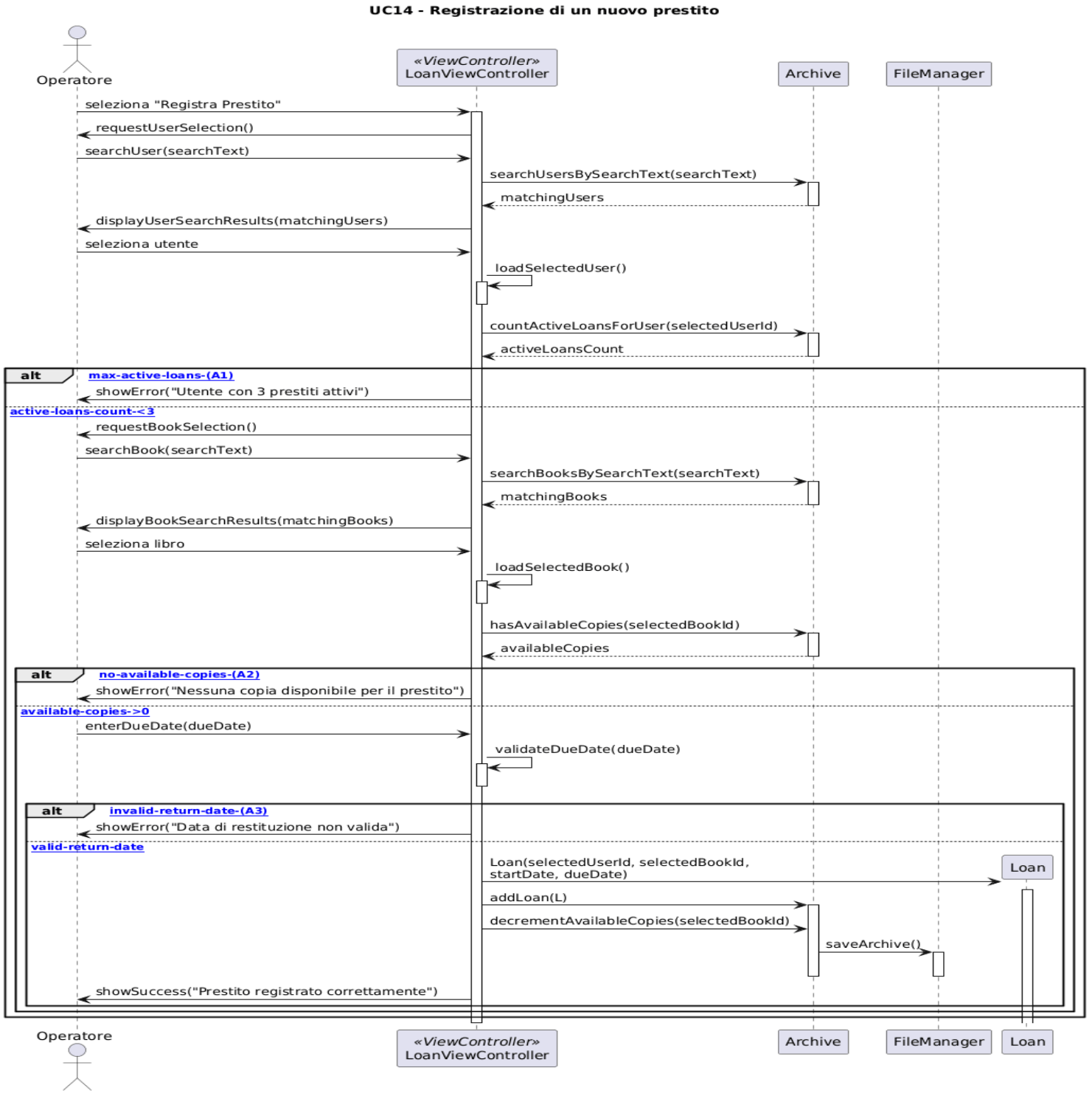

figura 28

Il diagramma in ***fig. 28*** mostra il flusso di sequenza del quattordicesimo caso d'uso (***UC-14***). L'operatore avvia la registrazione di un nuovo prestito ("**Registra Prestito**"), ricerca l'utente e lo seleziona. Il sistema verifica che l'utente non abbia già tre prestiti attivi; in caso contrario, l'operazione viene interrotta. Superato il controllo, l'operatore ricerca il libro e lo seleziona; il sistema verifica che siano disponibili copie per il prestito.
L'operatore inserisce quindi la data di restituzione prevista, che viene validata dal sistema. Se valida, viene creato un nuovo oggetto *Loan*, registrato nell'archivio; il numero di copie disponibili del libro viene aggiornato e l'archivio viene salvato. Infine il sistema conferma la corretta registrazione del prestito.

5.15 Visualizzazione prestiti attivi

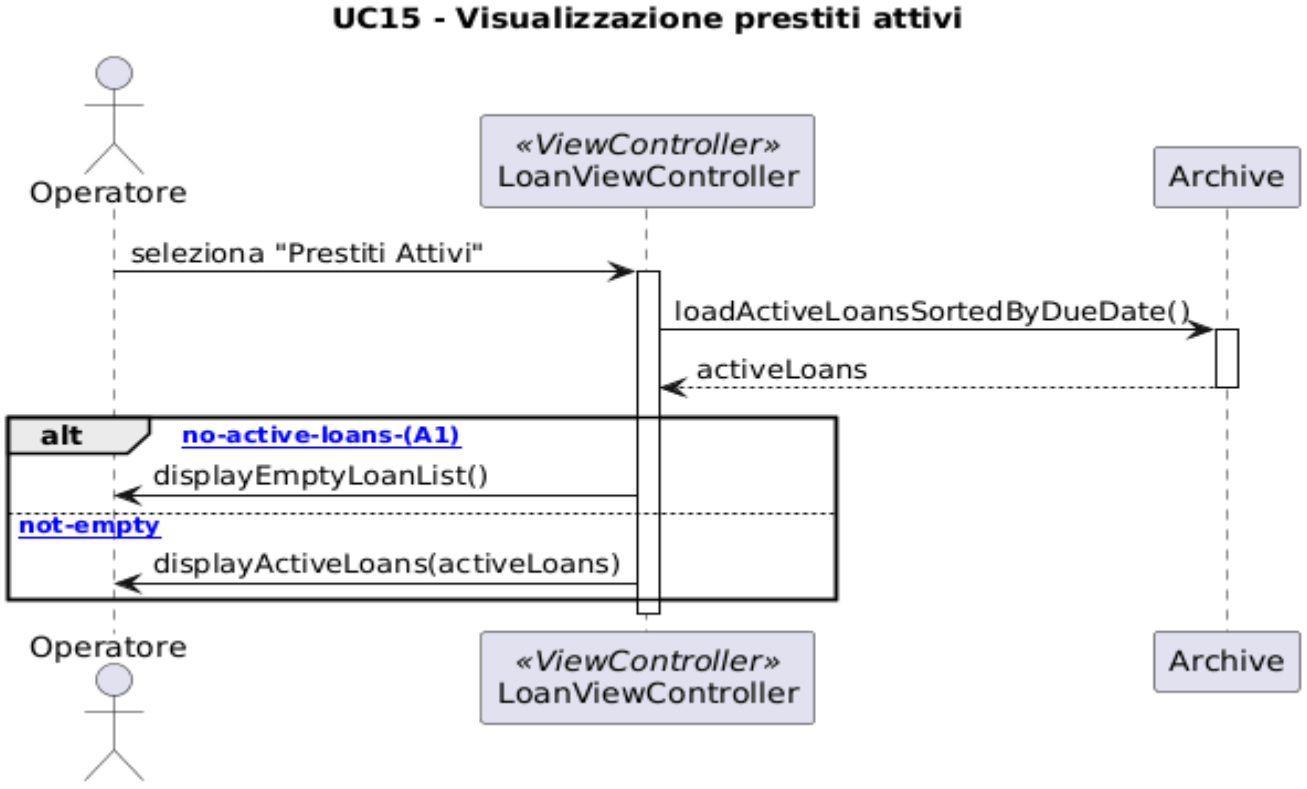

figura 29

Il diagramma in ***fig. 29*** mostra il flusso di sequenza del quindicesimo caso d'uso (***UC-15***). L'operatore richiede la visualizzazione dei prestiti attivi ("**Prestiti Attivi**") e il sistema recupera l'elenco ordinato per data di restituzione prevista. Se non sono presenti prestiti attivi viene mostrata una lista vuota; in caso contrario, il sistema presenta all'operatore tutti i prestiti attualmente in corso.

5.16 Evidenziazione dei prestiti scaduti

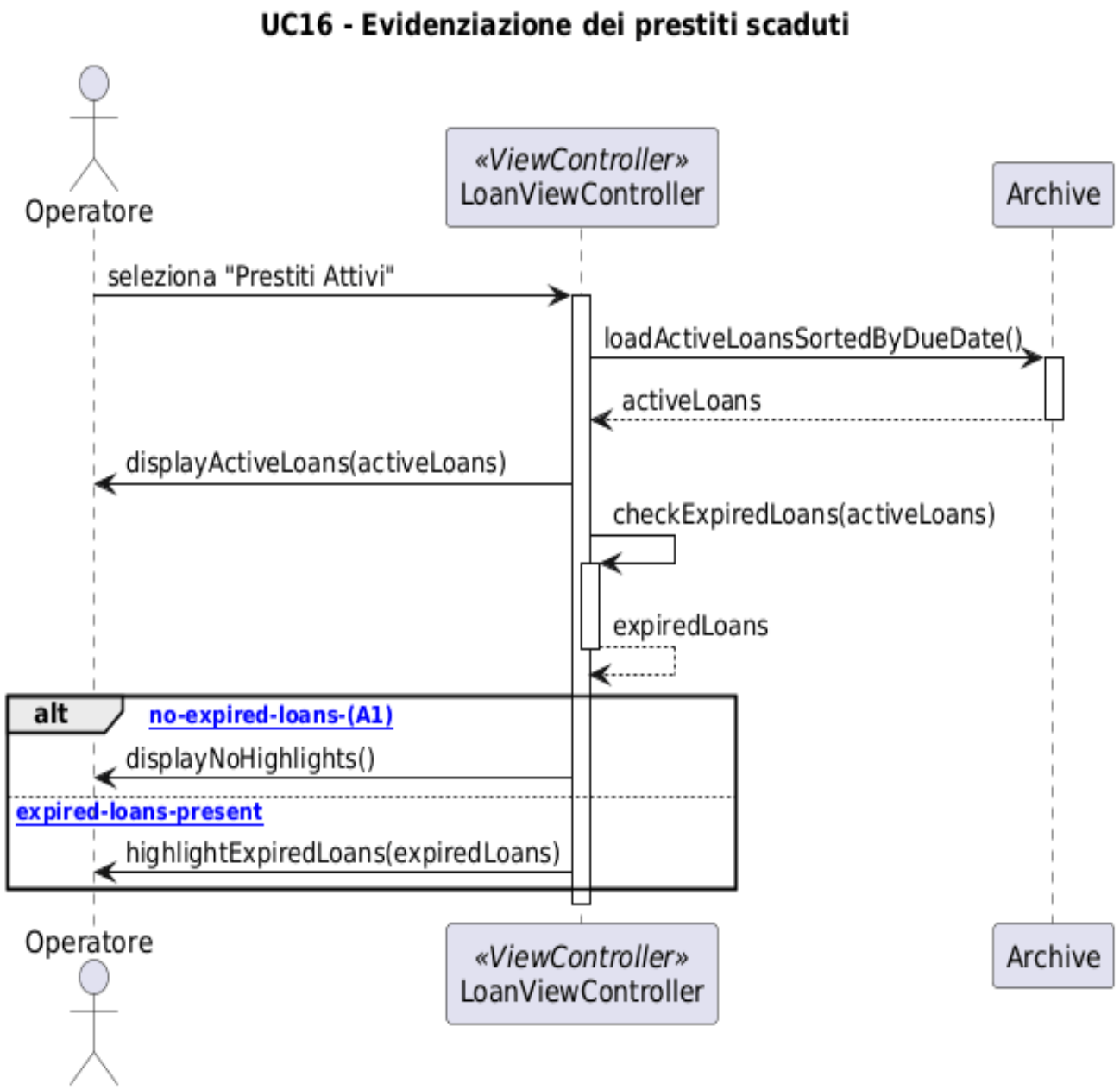

figura 30

Il diagramma in ***fig. 30*** mostra il flusso di sequenza del sedicesimo caso d'uso (***UC-16***). L'operatore visualizza l'elenco dei prestiti attivi, mentre il sistema confronta la data odierna con la data prevista di restituzione per ciascun prestito. I prestiti scaduti vengono individuati ed evidenziati graficamente; se non sono presenti prestiti oltre la data, l'elenco viene mostrato senza alcuna evidenziazione.

5.17 Registrare la restituzione di un libro

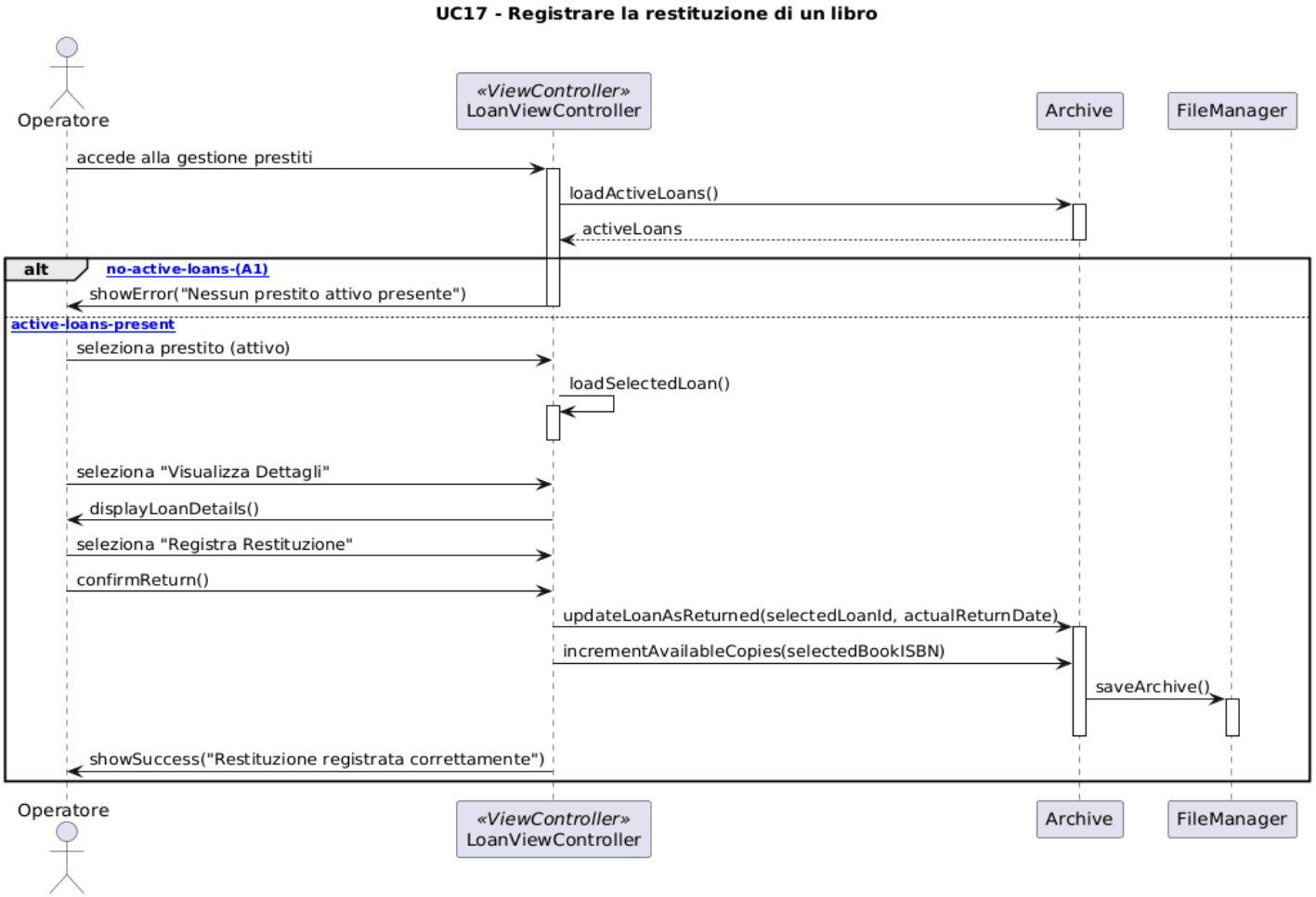

figura 31

Il diagramma in ***fig. 31*** mostra il flusso di sequenza del diciassettesimo caso d'uso (***UC-17***). L'operatore seleziona un prestito attivo e richiede la visualizzazione dei relativi dettagli("**Visualizza Dettagli**"); successivamente avvia la procedura di restituzione e ne conferma l'esecuzione. Il sistema aggiorna lo stato del prestito, registra la data di restituzione effettiva, incrementa il numero di copie disponibili del libro e salva l'archivio aggiornato. In assenza di prestiti attivi, viene mostrato un messaggio di errore.