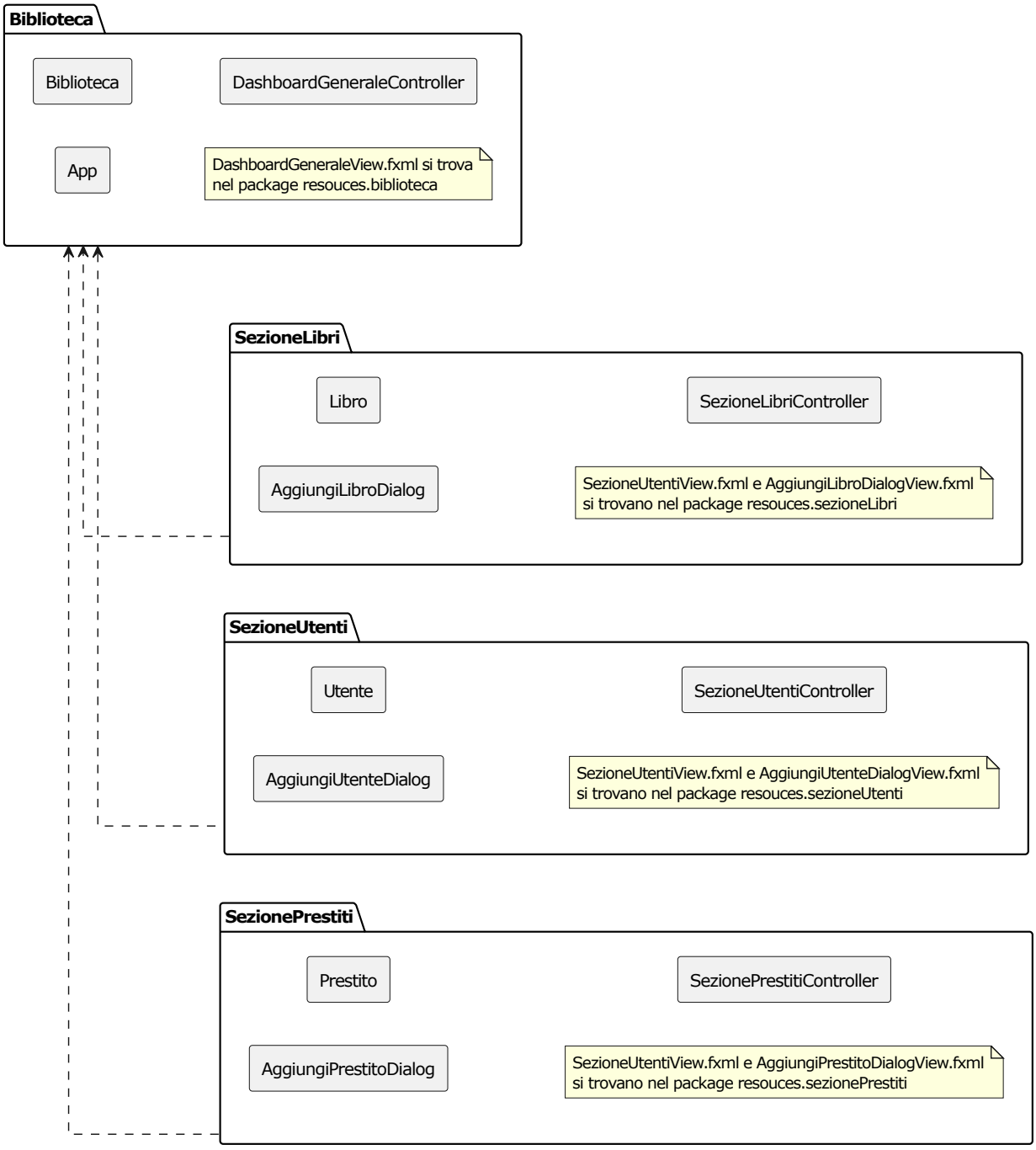
Documento di Progettazione del Software
Biblioteca Universitaria

ANGELO MASTANDREA, VINCENZO PASCARIELLO,
SIMONE PELLECCHIA, MARTINA TURI

8 dicembre 2025

Indice

1 Architettura del Sistema



Si decide di utilizzare il Design Pattern MVC per una divisione ottimale delle funzionalità in moduli specifici.
Si caratterizzano quindi le classi dell'applicazione in 3 sezioni: Model, View e Controller.

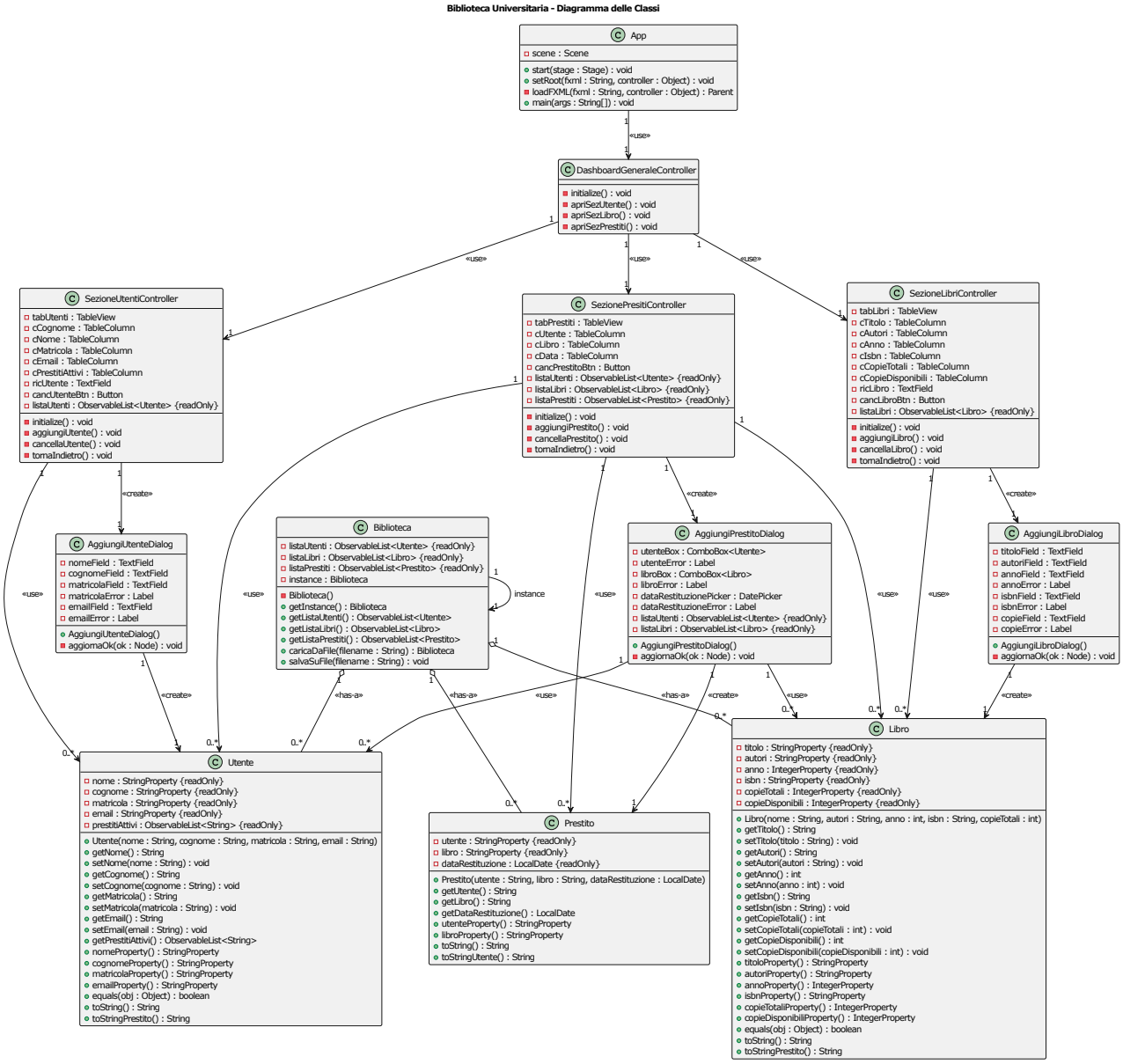
- All'interno del Model sono presenti le rappresentazioni dei dati utilizzati dall'applicazione, quindi esso comprenderà la classe Utente, Libro, Prestito e Biblioteca.
- La View è gestita attraverso quattro layout FXML che rappresentano il modo di visualizzare l'interfaccia associata ad ogni sezione del Sistema. Sono previste quindi una view per la DashBoard generale e una per ognuna delle tre sezioni Utenti, Libri e Prestiti. Si prevede per la sola funzione di inserimento l'utilizzo di finestre di dialogo per ogni sezione, le quali sono piccole interfacce aggiuntive. Ogni finestra di dialogo interagisce direttamente con i relativi "Controller" previsti dal pattern MVC inviando i dati sotto forma di istanze di classi presenti nella sezione "Model".

- Per i Controller sono quindi previste 4 classi: ”DashboardGeneraleController”, ”SezioneLibriController”, ”SezioneUtentiController”, ”SezionePrestitiController”.

Il compito di ogni Controller è quello di operare su ogni lista di Sezione per aggiungere, modificare, e rimuovere un elemento da essa. È suo compito ricevere i dati dalla finestra di dialogo, richiamata solo per le operazioni di inserimento, la quale si occupa anche di controllare la correttezza del formato dei dati inseriti. Il Controller infine si preoccupa di controllare la presenza di eventuali duplicati e, in loro assenza, inserire il nuovo oggetto nella struttura dati dedicata, il cui riferimento è situato all'interno del Controller stesso.

Per un'ottimale gestione dei dati e per permettere un'efficace comunicazione tra model e controller, si sceglie di implementare una classe Biblioteca tramite il design pattern del “Singleton”. Utilizzando questo pattern è possibile, per ogni Controller, accedere facilmente alle strutture dati su cui dovrà operare, le quali saranno tutte contenute in Biblioteca. Trattando Biblioteca come Singleton, infatti, sarà possibile istanziare un solo oggetto di quest'ultima per tutta la durata del programma e questa istanza sarà sempre accessibile in qualsiasi parte di esso. Sebbene l'utilizzo di tale pattern porti una relativa semplicità e facilità implementativa, il Singleton viene trattato effettivamente come una “variabile globale” rendendo accessibili i riferimenti di tutte le liste a tutti i Controller. Si supera questa vulnerabilità accedendo a “Biblioteca” solo dal metodo “initialize()” dei Controller e solo per recuperare il riferimento alla sola lista utile allo specifico Controller. L'istanza di Biblioteca viene recuperata ogni volta che viene eseguito il software leggendola dal file contenente tutto l'archivio della reale Biblioteca. L'istanza di Biblioteca si occuperà, appena verrà modificata una qualsiasi delle strutture in essa contenuta, di aggiornare il file dell'archivio.

2 Modello Statico



2.1 Biblioteca

La classe Biblioteca contiene tutte le strutture dati e fornisce i metodi per caricare queste ultime da file o per salvarle.
La coesione della classe è **Funzionale** perché gli attributi si occupano soltanto di mantenere i dati e le operazioni di caricaDaFile() e salvaSuFile() sono gestite da due metodi differenti.
Per quanto riguarda l'accoppiamento, Biblioteca **non prende dati da nessun altro oggetto**, ma è questa classe a fornire a tutti i controller i riferimenti alle strutture dati, generando accoppiamento.

2.2 Libro

La classe Libro ha la sola responsabilità di mantenere le informazioni necessarie all'identificazione di un Libro e alle sue copie disponibili. Anche in questo caso la coesione è **Funzionale**.
Oltre agli attributi per l'identificazione di un Libri, come ISBN, titolo, anno ed autori, sono presenti gli attributi copieTotali e copieDisponibili, quest'ultimo è aggiornato in fase di inserimento di un nuovo prestito o di rimozione, quindi è acceduto da SezionePrestitiController costituendo un accoppiamento sui Contenuti mitigato accendendo con un metodo setter.
La classe Libro **non è accoppiata** direttamente con altre classi in quanto non dipende e non usa metodi di altre classi o riferimenti ad altri componenti.

2.3 Utente

La classe Utente ha la sola responsabilità di mantenere le informazioni necessarie all'identificazione di un utente e ai suoi prestiti attivi. Anche in questo caso la coesione è **Funzionale**.
Oltre agli attributi per l'identificazione di un utente, come nome,cognome,matricola ed email, è presente un attributo prestitiAttivi di tipo ObservableList<String>, questa lista è aggiornata in fase di inserimento di un nuovo prestito quindi acceduta da SezionePrestitiController costituendo un accoppiamento sui Contenuti mitigato accendendo con un metodo setter.
La classe Utente **non è accoppiata** direttamente con altre classi in quanto non dipende e non usa metodi di altre classi o riferimenti ad altri componenti.

2.4 Prestito

La classe Prestito è progettata in modo da rappresentare unicamente i dati.
Al fronte di ciò essa **non è accoppiata** direttamente con altre classi in quanto non dipende e non usa metodi di altre classi o riferimenti ad altri componenti.
Inoltre la coesione è **Funzionale** in quanto si implementa, attraverso i metodi, la singola funzionalità di rappresentare un oggetto.

2.5 AggiungiLibroDialog

La classe AggiungiLibroDialog ha il compito di controllare la validità del formato dei dati inseriti, creare un oggetto di tipo Libro, e passarlo poi al controller SezioneLibriController che si preoccuperà dell'aggiunta.
Il metodo principale è AggiungiLibroDialog() che istanzia l'oggetto Libro e lo passa al Controller.
Questa classe ha una coesione **Funzionale** perché si occupa appunto soltanto del compito elementare di creare un oggetto libro con i dati inseriti in input e di passarlo al controller che poi compie effettivamente l'operazione di aggiunta in lista.
Essa **non è accoppiata** direttamente con altre classi in quanto non dipende e non usa metodi di altre classi o riferimenti ad altri componenti.

2.6 SezioneLibriController

La classe SezioneLibriController si occupa quindi di effettuare operazioni di inserimento, modifica o rimozione di Libri all'interno della Struttura dati.
L'attributo principale è proprio il riferimento alla struttura dati listaLibri di tipo ObservableList<Libro> che verrà utilizzato nei metodi di SezioneLibriController per aggiungere o cancellare un libro. Siccome ogni operazione è svolta da metodi differenti, compresa la modifica che viene gestita soltanto tramite binding e quindi senza neanche il bisogno di aggiungere un metodo, la coesione è **Funzionale**.
Essendo l'attributo listaLibri passato dalla classe Biblioteca, si ha un accoppiamento per **Timbro**. Effettivamente usando il design pattern del Singleton è possibile accedere a tutti gli attributi di Biblioteca, tuttavia l'accesso a Biblioteca è estremamente controllato e relegato un'unica volta soltanto alla fase di initialize() del controller e soltanto per l'attributo su cui bisogna strettamente lavorare, in questo caso listaLibri. In questo modo non si accede a Biblioteca all'interno di ogni metodo, ma solo un'unica volta per memorizzare il riferimento alla struttura dati diminuendo l'accoppiamento. In seguito ogni operazione verrà fatta sull'attributo quindi non si accederà più al singleton Biblioteca.
Nonostante la lista di Libri è strettamente necessaria al Controller per le operazioni di aggiunta, modifica e cancellazione, questo non è un accoppiamento sui dati perché effettivamente per svolgere i controlli necessari basterebbe soltanto l'isbn che è il codice identificativo univoco per ogni libro.
SezioneLibriController avrà anche un accoppiamento sui **Dati** con AggiungiLibroDialog poiché la finestra di dialogo passa solo i dati strettamente necessari al Controller per effettuare l'operazione di inserimento. In questo caso passa il nuovo oggetto Libro inserito in input dal Bibliotecario per l'aggiunta.

2.7 AggiungiUtenteDialog e SezioneUtentiController

È duale rispetto alla sezioneLibri il caso di AggiungiUtenteDialog e SezioneUtenteController.
L'attributo principale è sempre listaUtenti di tipo ObservableList<Utente>.
La coesione rimane **Funzionale** per la gestione in metodi differenti di ogni operazione. Lo stesso vale per AggiungiUtenteDialog.
L'accoppiamento con Biblioteca rimane per **Timbro** perché la listaUtenti è necessaria al Controller per lavorare su di

essa, ma si potrebbe utilizzare solo la matricola di un Utente per identificarlo univocamente anziché tutto l'oggetto.
L'accoppiamento tra il controller e la dialog rimane per **Dati**.

2.8 AggiungiPrestitoDialog

La classe AggiungiPrestritoDialog ha il compito principale di raccogliere dati in input per un nuovo prestito da aggiungere e di passarlo al controller SezionePrestitiController per l'aggiunta effettiva alla lista.
Per fare ciò la Dialog ha accesso anche alla lista dei Libri e degli Utenti, questo perché deve accertarsi che l'Utente e il Libro da relazionare per un nuovo prestito siano effettivamente presenti in archivio.
Avendo accesso ai riferimenti di tali strutture dati AggiungiPrestitoDialog ha un accoppiamento per **Timbro** con la classe Biblioteca esattamente come avviene per i Controller.
Il metodo principale è AggiungiPrestitoDialog(listaUtenti, listaLibri) che istanzia l'oggetto nuovo Prestito e lo passa al Controller.
Occupandosi solo di questo compito elementare la coesione è **Funzionale**.

2.9 SezionePrestitiController

La classe SezioniPrestitiController si occupa delle operazioni da effettuare sulla struttura dati contenente i prestiti.
Gli attributi principali sono quindi i riferimenti a tutte le strutture dati: listaUtenti ObservableList<Utente>, listaLibri ObservableList<Libro>, listaPrestiti ObservableList<Prestito>.
Essi sono utilizzati dai metodi principali aggiungiPrestito() e cancellaPrestito() per controllare l'esistenza dell'utente e l'esistenza/disponibilità del libro oltre che eventualmente per modificare il numero di copie di quest'ultimo. Gestendo sempre tutto in metodi differenti, la coesione è **Funzionale**.
Avendo quindi i riferimenti alle strutture dati, anche SezionePrestitiController avrà un accoppiamento sul **Timbro** con Biblioteca, alleviato accedendovi solo nel metodo initialize(), così come avveniva anche per i controller delle altre due sezioni.
Come abbiamo già detto SezioniPrestitiController è accoppiata sui **Contenuti** con le classi Utente e Libro, dato che si accede, anche se tramite metodi getter e setter, agli attributi prestitiAttivi di Utente e copieDisponibili di Libro.

2.10 Principi di buona progettazione

- **KISS - Keep It Simple, Stupid!**: Le classi che devono svolgere operazioni, su tutte in particolare i Controller, sono caratterizzate soltanto dalla disponibilità di poche funzioni come ad esempio l'aggiunta, la modifica e la cancellazione, le quali vengono tutte gestite da metodi differenti per rendere ancora più comprensibile il funzionamento generale della classe. In particolare la funzione di aggiunta, che tra le tre è la più complessa, viene ulteriormente semplificata attraverso la divisione delle operazioni da eseguire tra oggetti delle classi Dialog e delle classi Controller. Per quanto riguarda le strutture utilizzate, sono presenti semplici liste di oggetti del model.

- **SINE - Simple Is Not Easy**: Si è scelto di non utilizzare una facile interfaccia per l'inserimento dei dati, ma delle finestre di dialogo che mostrano un'interfaccia aggiuntiva e che effettuano già i controlli sul formato dei dati, non demandando il compito a controller più generali che diverrebbero saturi di operazioni da svolgere.
  Per quanto riguarda la scelta nell'utilizzo del singleton Biblioteca, senza il suo utilizzo le liste contenenti i dati sarebbero state distribuite in più classi all'interno del software, invece che essere tutte accessibili da una sola zona di memoria. Le conseguenze della scelta presa, a discapito della facilità di progettazione, sono una maggiore semplicità nel salvataggio e nel caricamento dei dati da file ed incrementi di accoppiamento tra le classi evitati. Infatti senza seguire il Design Pattern del Singleton la classe SezioniPrestitiController avrebbe dovuto accedere a tutte le strutture dati e quindi generare accoppiamento con tutti gli altri Controller avendo così un impatto negativo in termini di manutenibilità e dipendenza.

- **DRY - Don't Repeat Yourself**: Il Software non presenta ripetizioni di funzioni, ognuna di esse è diversa dalle altre per mansioni svolte o dati trattati.

- **YAGNI - You Aren't Going to Need It**: Il Software si concentra a implementare solo le funzionalità strettamente necessarie per la Biblioteca, senza trattare dettagli non richiesti.

- **Separazione delle preoccupazioni (separation of concerns)**: La divisione dell'intero Sistema in Sezioni permette la possibilità di affrontare i vari compiti da svolgere in moduli diversi.

- **Ortogonalità**: Si è tenuti separati nel Sistema tutti gli aspetti che sono già separati concettualmente. Ad esempio non c'è alcun legame tra il modo in cui salvare un dato e il dato stesso contenuto in questo caso in Biblioteca. A prescindere dalle modifiche che vengono apportate all'archivio esso sarà sempre salvato allo stesso modo. La funzione di salvataggio su file è indipendente dal fatto che venga effettuata un'operazione su un Utente, un Libro o un Prestito.
- **Prinicpio della minima sorpresa**: Il codice segue sempre lo stesso pattern per i Controller e le Dialog, le operazioni da compiere sono nominate sempre allo stesso modo e seguono sempre lo stesso ordine. L'unica cosa che cambia è ovviamente l'implementazione a causa della necessità delle diverse sezioni di lavorare con oggetti differenti.
- **Principio della singola responsabilità**: Sono completamente divise le responsabilità principali del Sistema: rappresentare i dati (Utente, Libro, Prestiti), operare sui dati (Controller), mantenere e salvare i dati (Biblioteca).
- **Principio aperto/chiuso**: Gli attributi delle classi sono tutti privati e accessibili tramite Getter. Il sistema rimane aperto all'estensione qualora si volesse specializzare una qualsiasi delle classi Utente, Libro, Prestito con ulteriori attributi e metodi.
- **Principio di sostituzione di Liskov**: Non si è utilizzata l'ereditarietà per organizzare le classi che compongono il Sistema.
- **Principio di segregazione delle interfacce**: Non si sono utilizzate Interfacce per organizzare le classi che compongono il Sistema.
- **Principio di inversione delle dipendenze**: Alcuni componenti di livello superiore come i Controller, dipendono dalle Dialog senza le quali non potrebbero svolgere le operazioni di aggiunta.
- **Privilegiare l'associazione rispetto all'ereditarietà**: Non si è utilizzata l'ereditarietà, ma sono invece molto utilizzate le associazioni, ad esempio con riferimenti ad oggetti di altre classi.
- **Principio di robustezza**: Per aumentare la robustezza del Sistema vengono fatti controlli sul formato dei dati inseriti, come la lunghezza della matricola dell'utente o il prefisso standard dell'isbn di un libro, in modo tale che un utilizzatore del software venga avvisato qualora si sbagliasse nell'inserimento dei dati.

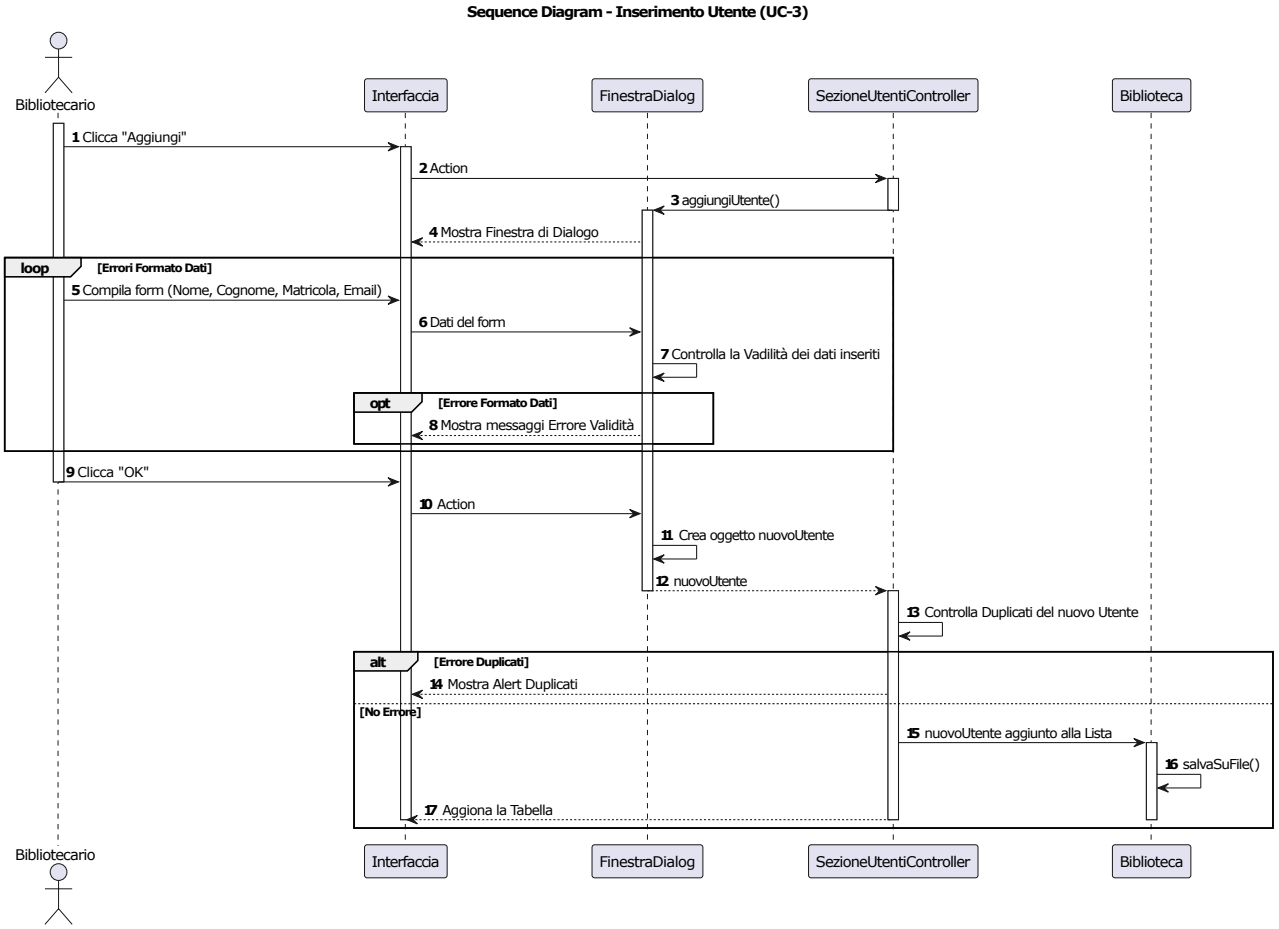
Il rispetto di questi Principi di Buona Progettazione, oltre a migliorare la qualità del Software finale, hanno soprattutto un impatto positivo su proprietà fondamentali come la manutenibilità e la riutilizzabilità.

3 Modello dinamico

3.1 Diagrammi di Sequenza

Per documentare come e quando le classi effettivamente interagiscono tra loro, si passa dall'analisi statica a quella dinamica del Sistema. In questo modello dinamico, attraverso i diagrammi di sequenza, si trattano tutti i casi d'uso più significativi e complessi in modo da rendere chiaro ogni aspetto del flusso del programma.
I Diagrammi di Sequenza scelti permettono di visualizzare cosa accade durante ognuno dei passi che costituiscono lo svolgimento di un caso d'uso, mostrando in ordine tutte le interazioni che si verificano tra oggetti delle diverse classi. Ogni diagramma riportato di seguito è accompagnato da un commento che spiega la logica e il funzionamento di ogni operazione.

3.1.1 Inserimento Utente (UC-3)

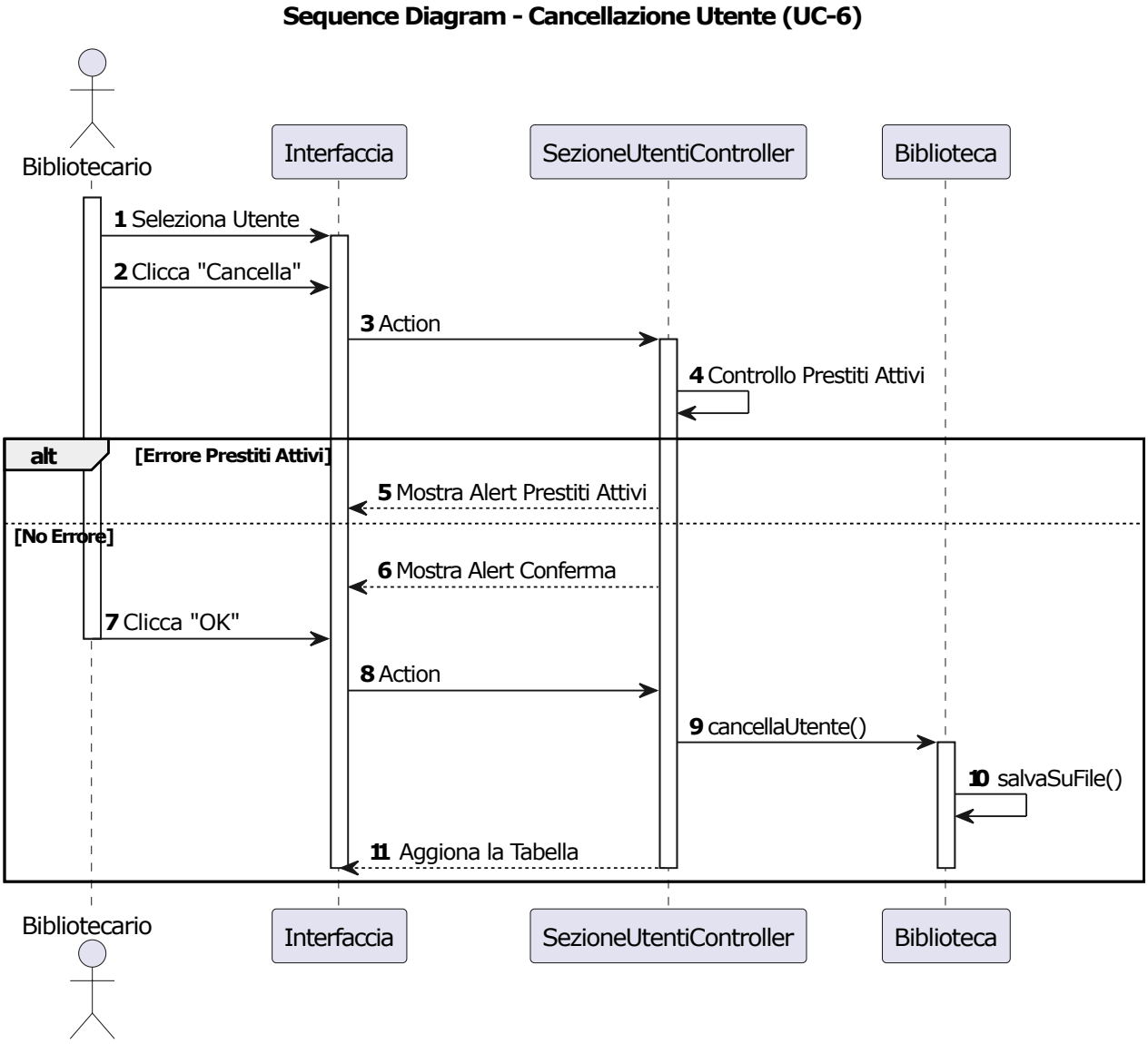


Il diagramma di sequenza mostrato in figura mostra le interazioni tra classi durante l'esecuzione del caso d'uso di Inserimento Utente (UC-3). L'attore principale è il Bibliotecario che avvia l'operazione di inserimento.
Flusso principale:

- Il Bibliotecario clicca "Aggiungi" tramite l'interfaccia che, grazie ad un'interazione con SezioneUtentiController, mostra la finestra di dialogo.
- Il Bibliotecario compila il form mostratogli dall'interfaccia con i dati: Nome, Cognome, Matricola ed Email.
- I dati vengono passati a FinestraDialog che controlla la validità del formato dei dati inseriti mostrando un Alert sull'interfaccia in caso di errore, facendo ripetere l'operazione di compilazione all'utente finchè i dati non risultino corretti.
- Il Bibliotecario clicca "Ok" tramite l'interfaccia, a questo punto FinestraDialog crea l'oggetto nuovoUtente, contenente i dati dell'Utente.

- Il flusso si sposta su SezioneUtentiController che riceve l'oggetto appena creato e, se esso non è un duplicato, lo inserisce correttamente nella lista, altrimenti mostra un Alert nell'Interfaccia.
- Dopo l'Inserimento nella Lista, il controllo passa a Biblioteca che ha il compito di salvare le modifiche apportate alla Lista all'interno del file mantenendo quest'ultimo sempre aggiornato.
- A questo punto, avendo aggiunto l'elemento alla Lista del Controller, la lista mostrata al Bibliotecario dall'interfaccia sarà aggiornata.

3.1.2 Cancellazione Utente (UC-6)

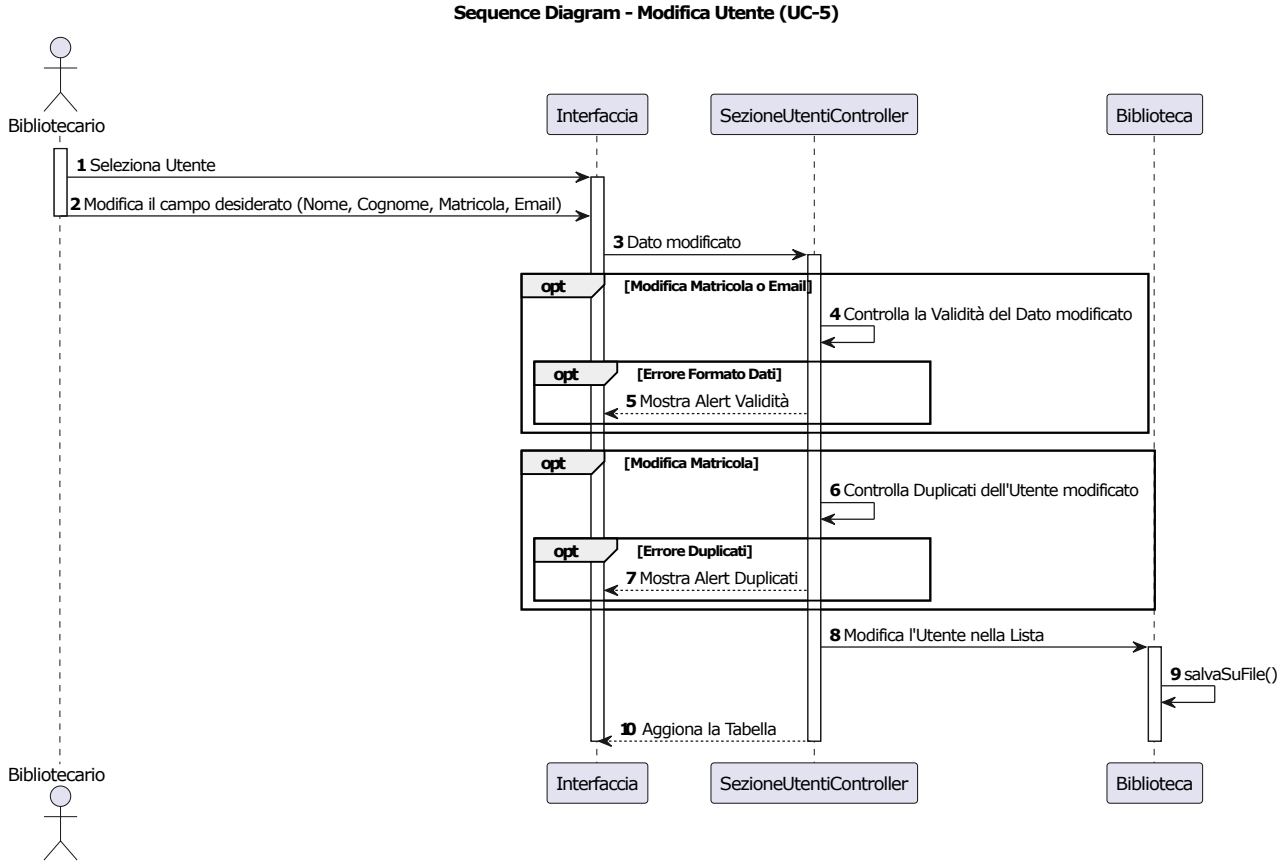


Il diagramma di sequenza mostrato in figura mostra le interazioni tra classi durante l'esecuzione del caso d'uso di Cancellazione Utente (UC-6). L'attore principale è il Bibliotecario che avvia l'operazione di cancellazione.
Flusso principale:

- Il Bibliotecario seleziona, tramite l'interfaccia che mostra la lista degli Utenti aggiornata, un Utente.
- Il Bibliotecario clicca "Cancella" tramite l'interfaccia.
- SezioniUtentiController effettua un controllo sui possibili Prestiti attivi dell'Utente selezionato vista l'impossibilità di cancellare un utente che presenta ancora Prestiti da estinguere.

- Verrà mostrata sull'Interfaccia grafica, in caso di presenza di Prestiti attivi per l'Utente che si intende eliminare, un Alert di errore terminando così l'esecuzione del caso d'uso. Altrimenti verrà semplicemente mostrato un Alert di conferma.
- A questo punto il Bibliotecario clicca "Ok".
- Il flusso si sposta su SezioneUtentiController che cancella correttamente l'Utente dalla lista.
- Dopo l'Inserimento nella Lista, il controllo passa a Biblioteca che ha il compito di salvare le modifiche apportate alla Lista all'interno del file mantenendo quest'ultimo sempre aggiornato.
- A questo punto, avendo aggiunto l'elemento alla Lista del Controller, la lista mostrata al Bibliotecario dall'interfaccia sarà aggiornata.

3.1.3 Modifica Utente (UC-5)



Il diagramma di sequenza mostrato in figura mostra le interazioni tra classi durante l'esecuzione del caso d'uso di Modifica Utente (UC-5). L'attore principale è il Bibliotecario che avvia l'operazione di Modifica.
Flusso principale:

- Il Bibliotecario seleziona, tramite un interfaccia che mostra la lista degli Utenti aggiornata, un Utente.
- Il Bibliotecario modifica direttamente tramite l'interfaccia il campo desiderato.
- SezioniUtentiController effettua vari controlli in base a quale campo è stato cambiato.
- Nel caso in cui si modifichi la matricola o l'email, verrà effettuato un controllo sul formato dei dati inseriti mostrando un Alert sull'interfaccia in caso d'errore.
- Nel caso in cui si modifichi la matricola, verrà effettuato un ulteriore controllo sui duplicati in modo da non permettere la modifica del campo matricola di un Utente con un'altra matricola già presente nel sistema, mostrando un Alert sull'interfaccia in caso di presenza di duplicati.

- Nel caso di modifica dei campi Nome o Cognome, non verrà effettuato nessun controllo da parte di SezioneUtenti-Controller.
- A questo punto SezioneUtentiController modifica correttamente il campo desiderato dell'Utente nella lista.
- Dopo la modifica nella Lista, il controllo passa a Biblioteca che ha il compito di salvare le modifiche apportate alla Lista all'interno del file mantenendo quest'ultimo sempre aggiornato.
- A questo punto, avendo aggiunto l'elemento alla Lista del Controller, la lista mostrata al Bibliotecario dall'interfaccia sarà aggiornata.

I precedenti diagrammi di sequenza mostrati sono rappresentativi anche dei casi d'uso relativi ad Inserimento Libro (UC-9), Cancellazione Libro (UC-12), Modifica Libro (UC-11).

3.1.4 Inserimento Libro (UC-9)

Si evidenziano le differenze con il duale diagramma di sequenza di Inserimento Utente (UC-3).
Il controllo di validità dei dati inseriti (passo 7) sarà effettuato sull'ISBN, sull'anno e sul numero di copie totali.
Allo stesso modo, il controllo dei duplicati effettuato da SezioneUtentiController (passo 13) verrà fatto sull'ISBN.

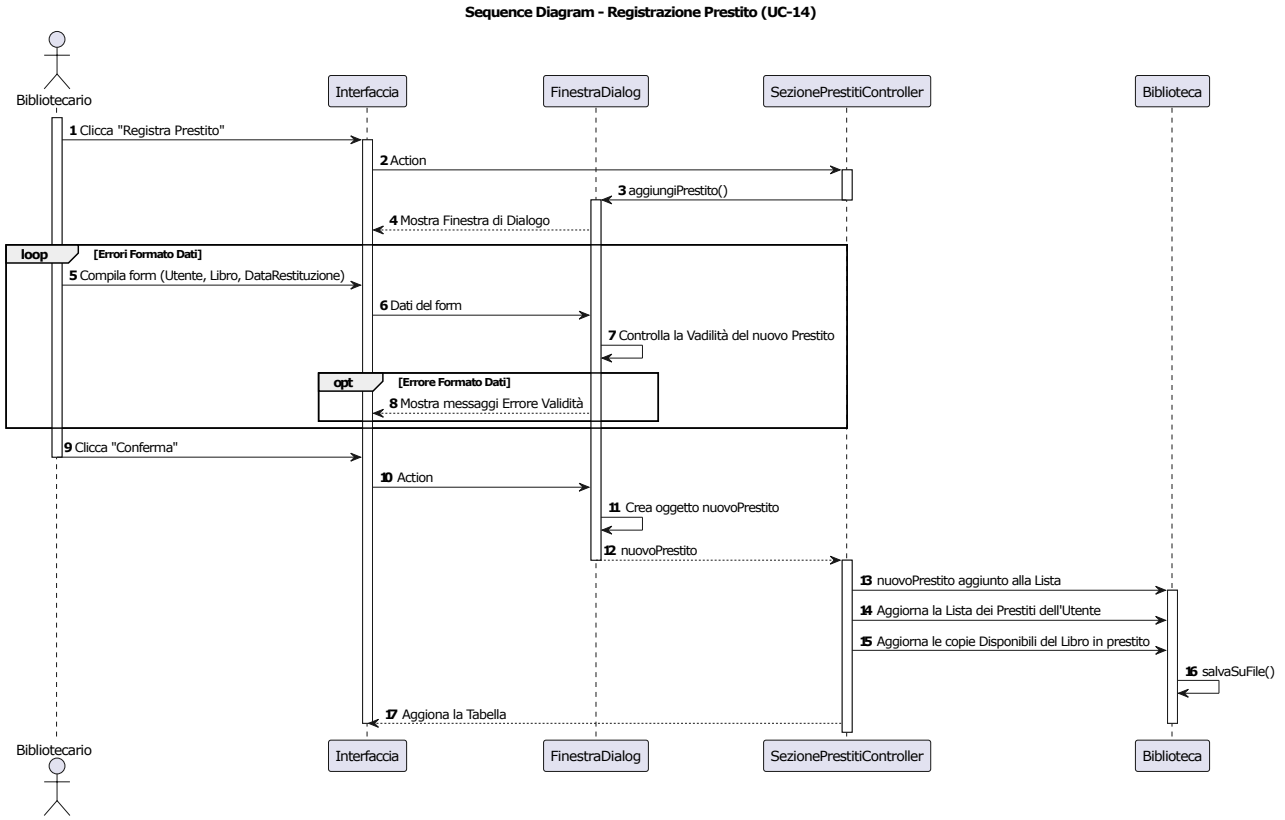
3.1.5 Cancellazione Libro (UC-12)

Si evidenziano le differenze con il duale diagramma di sequenza di Cancellazione Utente (UC-6).
Il controllo effettuato dal controller al passo 4 dovrà verificare, prima di cancellare il libro, che le copie disponibili siano uguali alle copie totali. In altre parole questo è per accertarsi di non cancellare un libro se ci sono ancora copie coinvolte in prestiti. Il resto del flusso rimane invariato.

3.1.6 Modifica Libro (UC-11)

Si evidenziano le differenze con il duale diagramma di sequenza di Modifica Utente (UC-5).
I controlli effettuati dal controller varieranno solo in base agli attributi di Libro.
Se si sceglie di modificare ISBN, anno o copieTotali, si effettuerà un controllo di validità sul formato dei dati.
In seguito, se il campo modificato è l'ISBN, si dovrà effettuare un ulteriore controllo per verificare che l'ISBN non sia già presente all'interno della lista, in modo che la modifica del libro non comporti l'introduzione di un duplicato.
Se il campo modificato è invece il numero di copieTotali, si dovrà verificare che il nuovo numero di copieTotali sia maggiore del numero di copie in Prestito (differenza tra il vecchio valore di copieTotali e copieDisponibili). Questo controllo assicura che il numero di copieTotali sia almeno uguale al numero di copie in prestito.
Le modifiche ai campi titolo o autore non introducono controlli, esattamente come per Nome e Cognome nel diagramma di sequenza duale di Modifica Utente. Il resto del flusso rimane invariato.

3.1.7 Registrazione Prestito (UC-14)



Il diagramma di sequenza mostrato in figura mostra le interazioni tra classi durante l'esecuzione del caso d'uso di Registrazione Prestito (UC-14). L'attore principale è il Bibliotecario che intende registrare un Prestito.
Flusso principale:

- Il Bibliotecario clicca "Registra Prestito" tramite l'interfaccia che, grazie ad un'interazione con SezionePrestitiController, mostra la finestra di dialogo.

- Il Bibliotecario compila il form visualizzato sull'interfaccia con i dati: Utente, Libro, DataRestituzione.

- I dati vengono passati a FinestraDialog che controlla la validità dei dati inseriti, in particolare: l'Utente dovrà essere registrato presso la Biblioteca; il Libro dovrà essere presente nel catalogo; la Data di Restituzione non deve far riferimento ad una passata. In caso di errore sulla validità, la FinestraDialog mostrerà un messaggio specifico per ogni campo e farà ripetere l'operazione di compilazione all'Utente finchè i dati non risultino corretti.

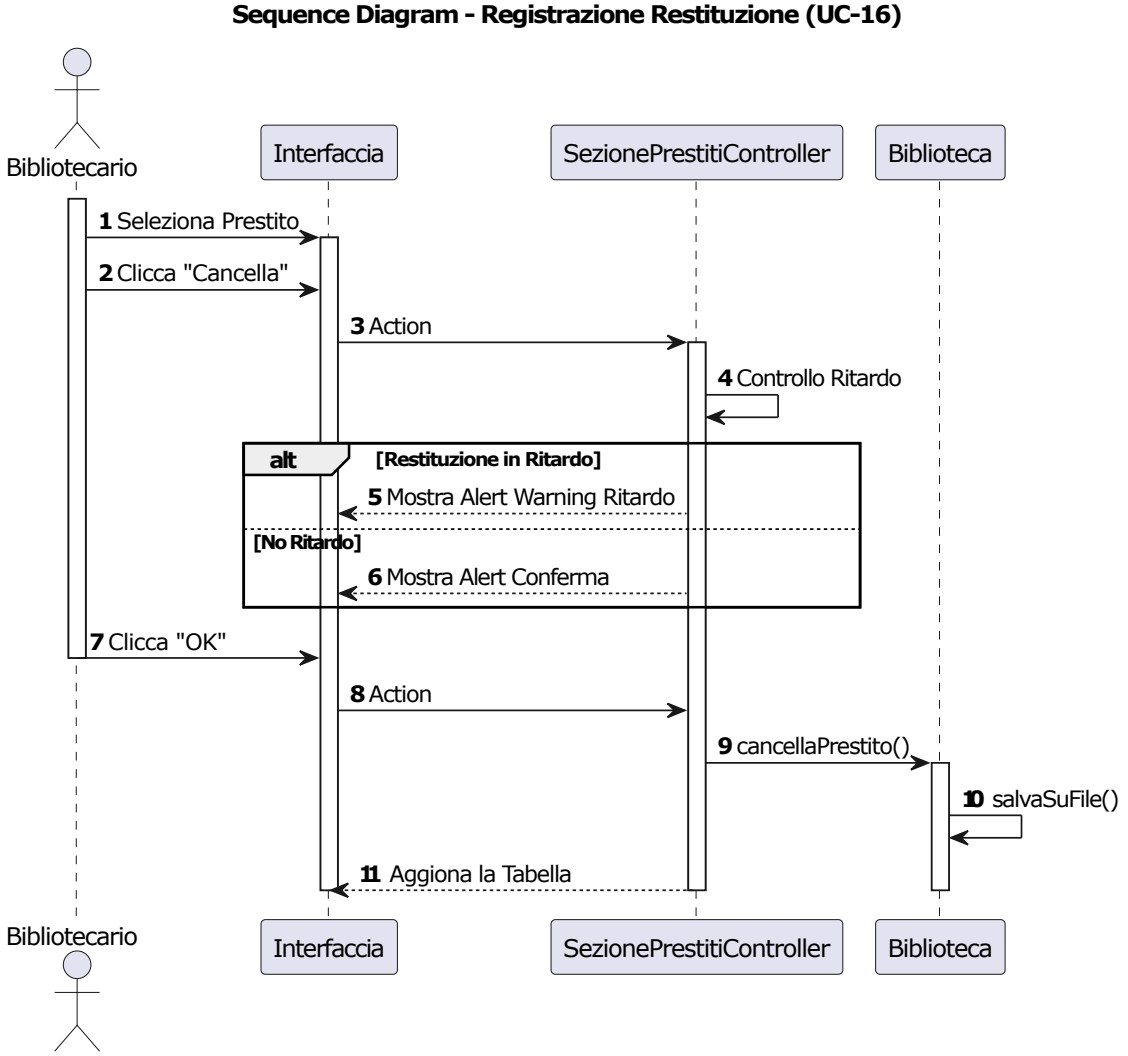
- A questo punto il Bibliotecario potrà cliccare "Conferma" e conseguentemente FinestraDialog creerà un oggetto nuovoPrestito contenente i dati inseriti e lo passerà a SezionePrestitiController.

- Quest'ultimo aggiungerà nuovoPrestito alla lista dei Prestiti, aggiornando la lista dei Prestiti attivi dell'Utente specificato ed il numero di copie Disponibili del Libro preso in prestito.

- Il controllo passa a Biblioteca che ha il compito di salvare le modifiche apportate all'interno del file mantenendo quest'ultimo sempre aggiornato.

- Infine, il Bibliotecario potrà visualizzare dall'interfaccia la lista dei Prestiti attivi aggiornata.

3.1.8 Registrazione Restituzione (UC-16)



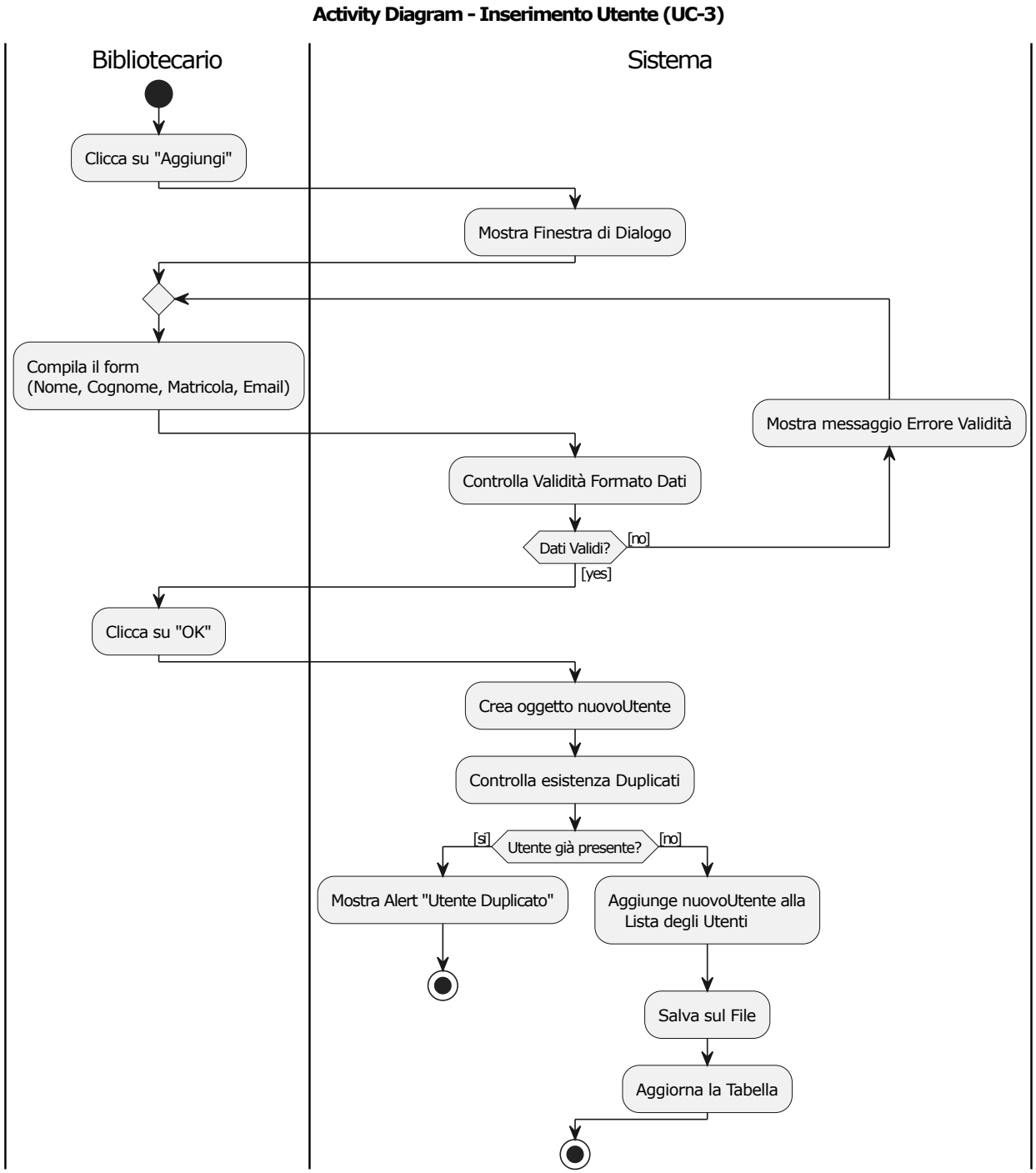
Il diagramma di sequenza mostrato in figura mostra le interazioni tra classi durante l'esecuzione del caso d'uso di Registrazione Restituzione (UC-16). L'attore principale è il Bibliotecario che intende registrare la restituzione di un prestito e la sua conseguente cancellazione.
Flusso principale:

- Il Bibliotecario seleziona un prestito specifico dalla lista e clicca "Cancella" tramite l'Interfaccia. Quest'ultima comunica l'evento a SezionePrestitiController.
- SezionePrestitiController eseguirà una verifica per determinare se la restituzione sta avvenendo entro i termini previsti.
- Nel caso in cui la Restituzione risulti in ritardo, il Controller mostrerà sull'Interfaccia un "Alert Warning Ritardo" contenente un messaggio specifico per il ritardo, ma permetterà comunque di effettuare la restituzione.
- Nel caso in cui la Restituzione non risulta in ritardo, viene mostrato un "Alert Conferma" standard.
- Quando il Bibliotecario clicca "OK" l'Interfaccia trasmette l'azione di conferma al SezionePrestitiController.
- Quest'ultimo rimuoverà effettivamente il Prestito dal Sistema e passerà il controllo a Biblioteca che che ha il compito di salvare le modifiche apportate all'interno del file mantenendo quest'ultimo sempre aggiornato.
- Infine, il Bibliotecario potrà visualizzare dall'interfaccia la lista dei Prestiti attivi aggiornata.

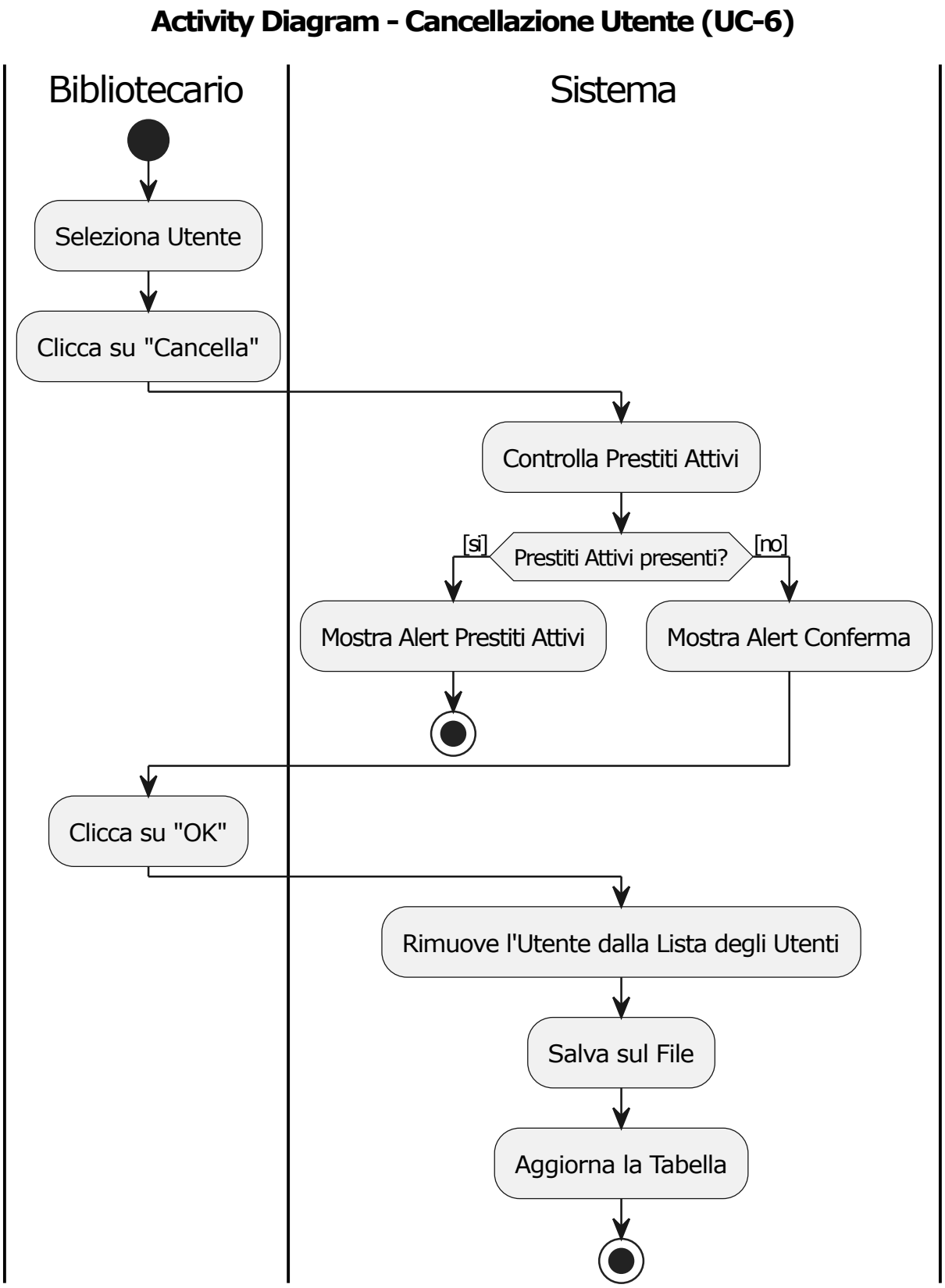
3.2 Diagrammi di Attività

Dopo aver analizzato nel dettaglio i diagrammi di sequenza, che ci hanno permesso di osservare nel dettaglio le interazioni tra classi, abbiamo spostato la nostra attenzione su una prospettiva complementare: i Diagrammi di Attività.
Possiamo vederli come una versione più immediata: se prima ci preoccupavamo di chi faceva cosa, ora ci interessa solo cosa succede e in che ordine. La grande utilità di questi diagrammi risiede nella loro capacità di astrazione: essi ci liberano dalla necessità di conoscere i dettagli tecnici, come quale specifica classe o controller stia gestendo un'operazione. Ciò rende il diagramma di attività lo strumento ideale per descrivere in modo semplice il flusso logico del processo, nascondendo tutta la complessità interna del sistema.

3.2.1 Inserimento Utente (UC-3)

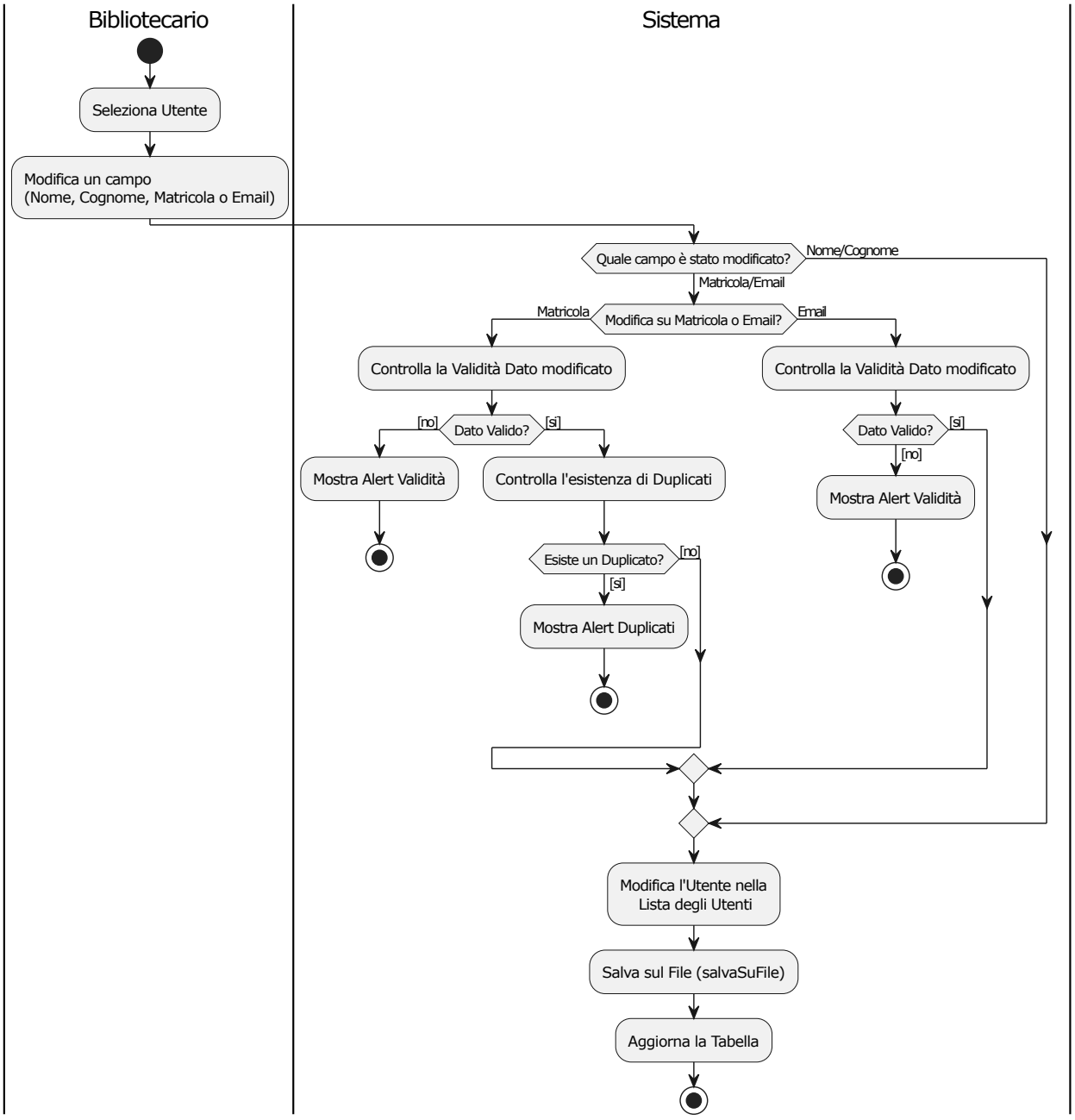


3.2.2 Cancellazione Utente (UC-6)



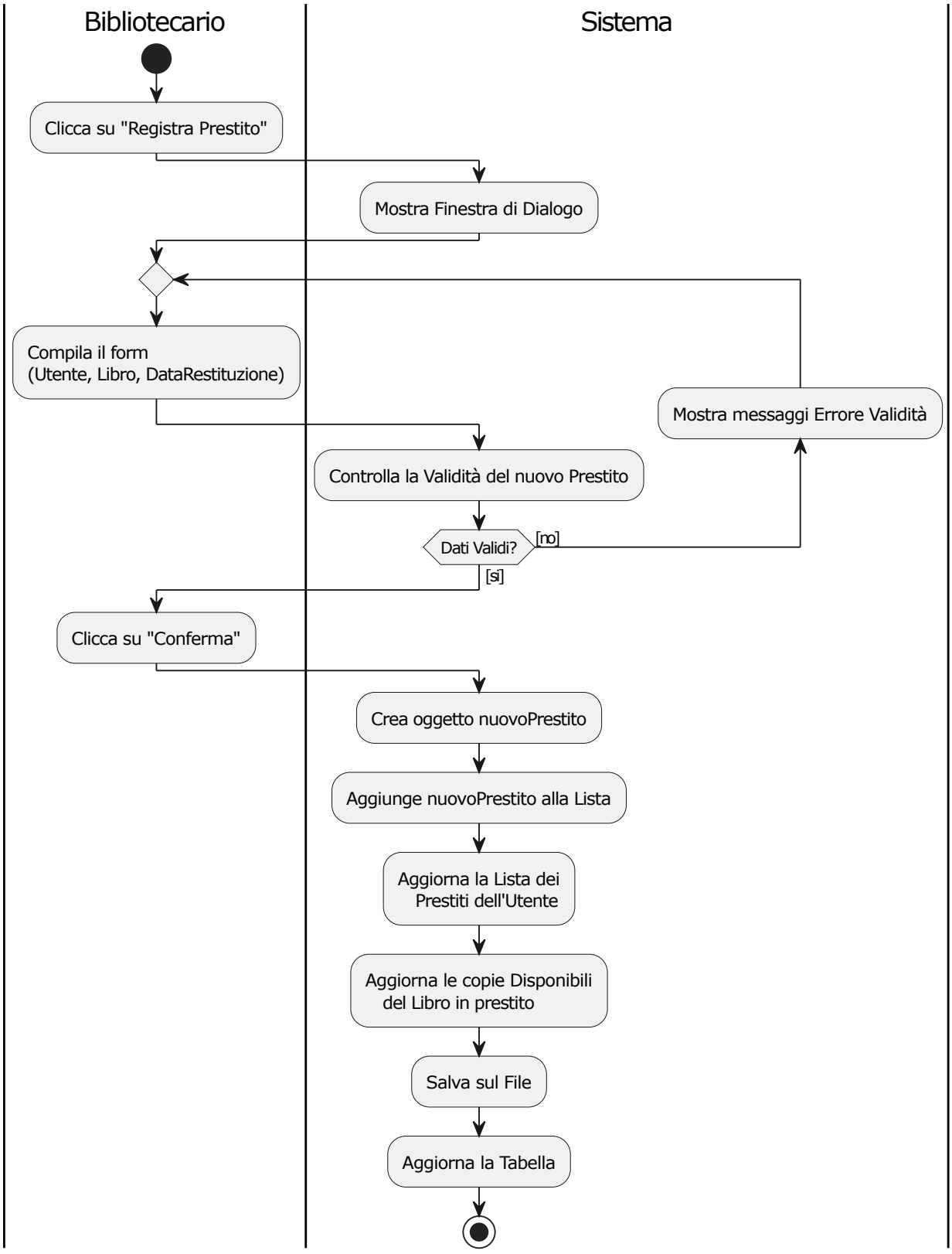
3.2.3 Modifica Utente (UC-5)

Activity Diagram - Modifica Utente (UC-5)

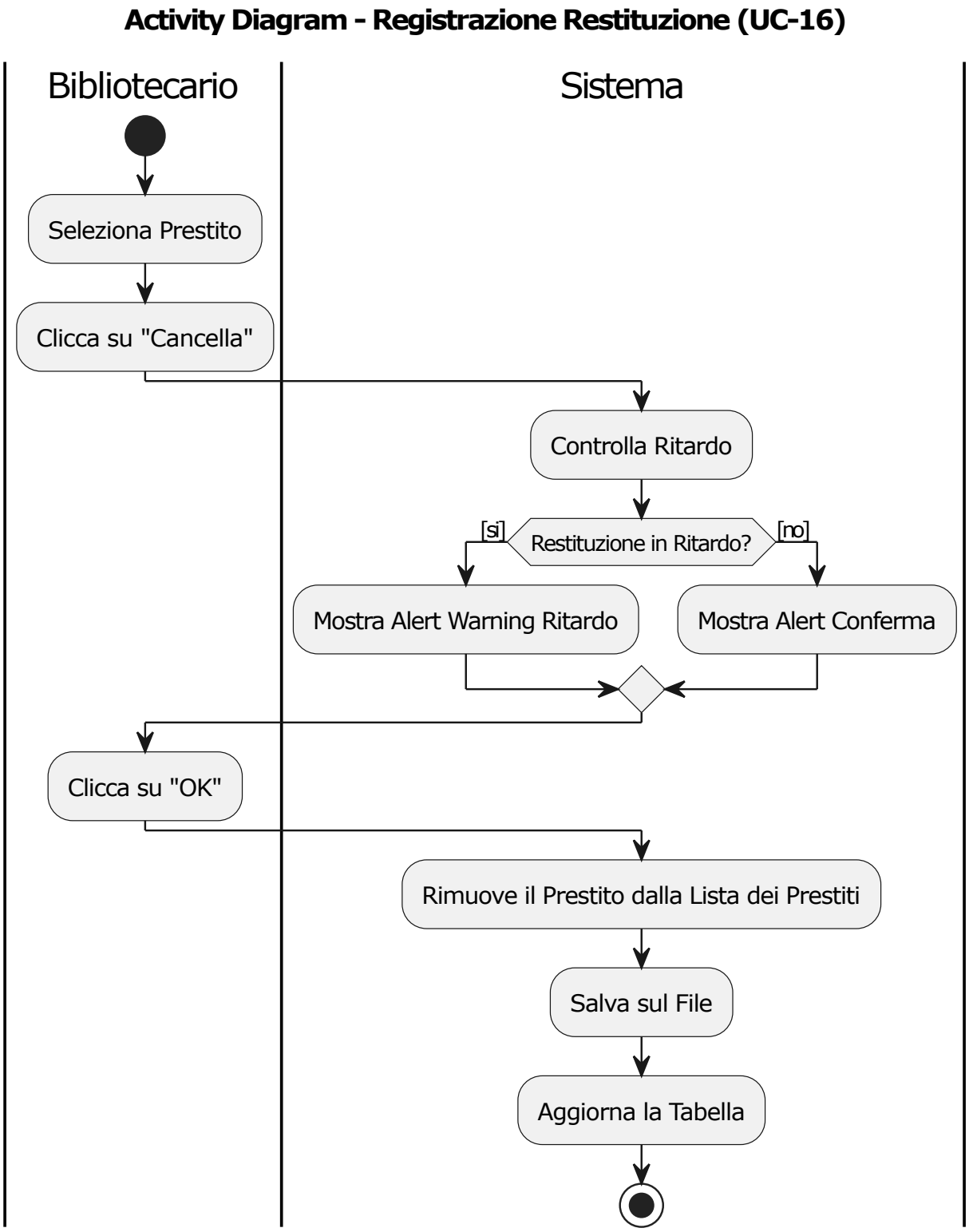


3.2.4 Registrazione Prestito (UC-14)

Activity Diagram - Registrazione Prestito (UC-14)



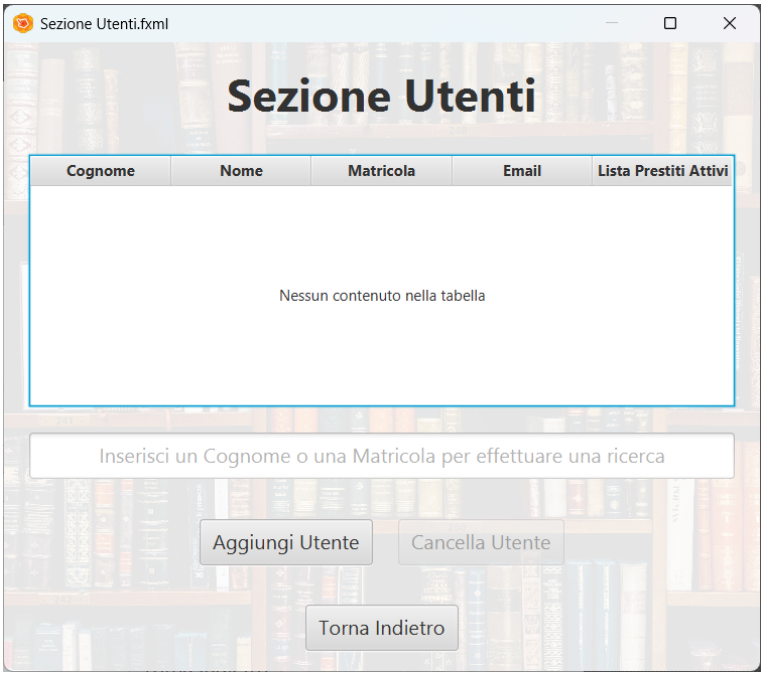
3.2.5 Registrazione Restituzione (UC-16)



4 Design dell'interfaccia grafica

4.1 Dashboard Generale

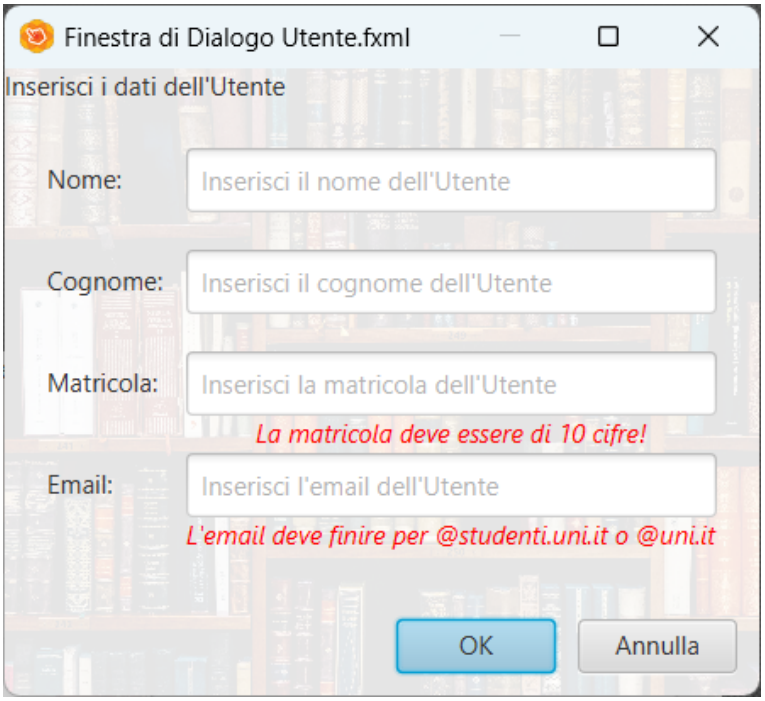
4.2 Sezione Utenti



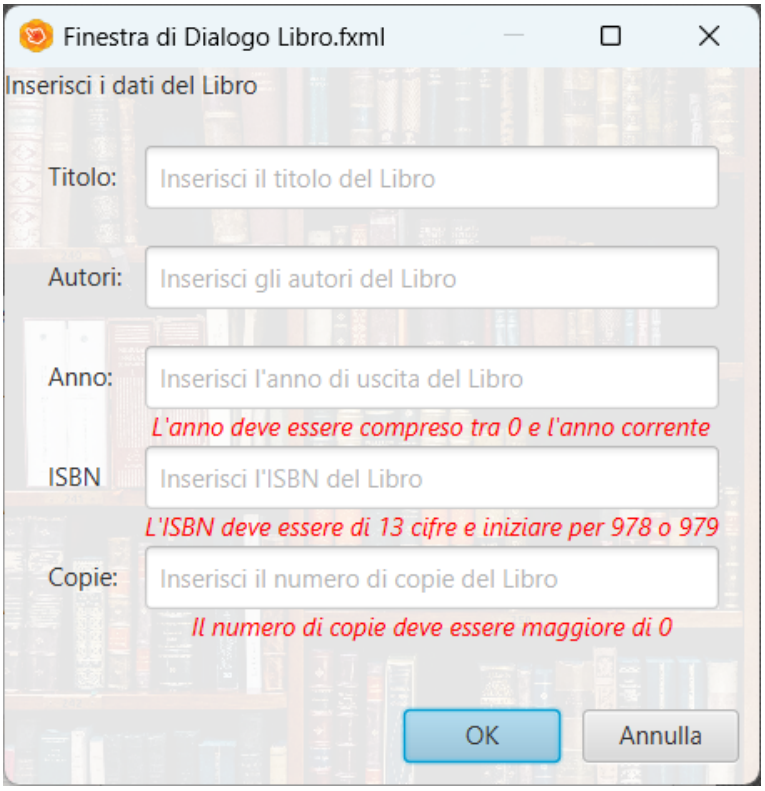
4.3 Sezione Libri

4.4 Sezione Prestiti

4.5 Finestra di Dialogo Utente



4.6 Finestra di Dialogo Libro



4.7 Finestra di Dialogo Prestito